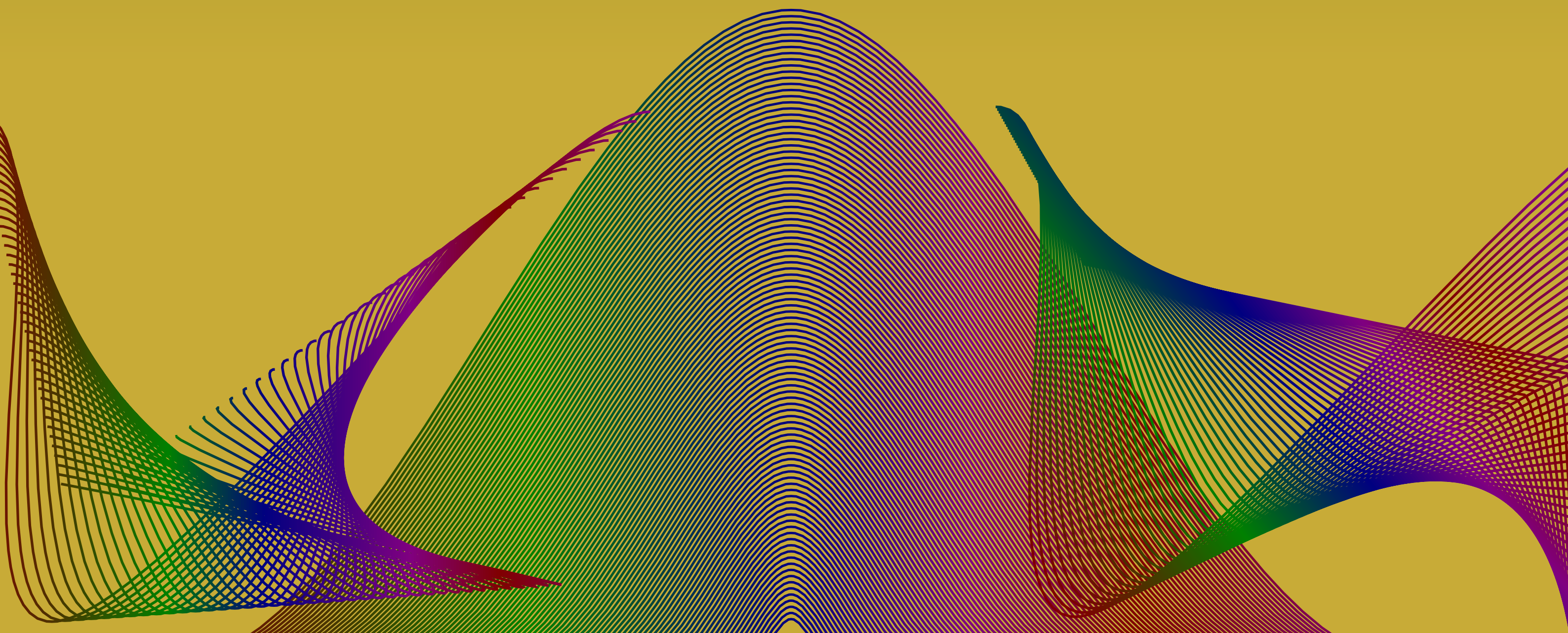
Hyderabad (Sind) National Collegiate Board's
Kishinchand Chellaram College
Churchgate, Mumbai - 20

Department of Statistics

under the aegis of DBT-STAR Status Scheme

Analyzing and Visualizing Data using free open-source software

Python Programming with case studies



Hyderabad (Sind) National Collegiate Board's

Kishinchand Chellaram College

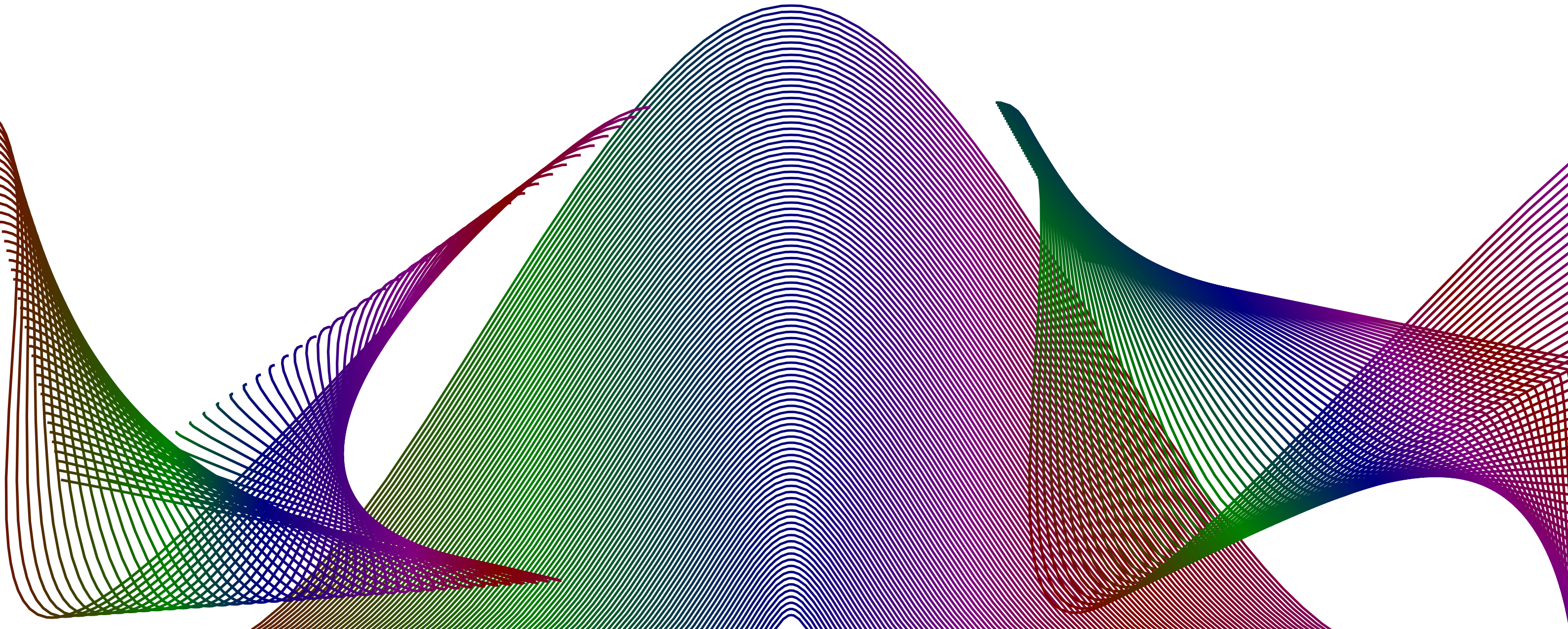
Churchgate, Mumbai - 20

Department of Statistics

under the aegis of DBT-STAR Status Scheme

Analyzing and Visualizing Data using free open-source software

Python Programming with case studies



Analysing and Visualizing Data using free open-source software

Python Programming with case studies

Editor

Dr. Asha Angnamal Jindal
Department of Statistics

Kishinchand Chellaram College

Vidyasagar Principal K.M. Kundnani Chowk,
124, Dinshaw Wachha Road,
Churchgate, Mumbai 400020.
Tel: +91-22-2285 5726; +91-22-6698 1000;
Fax: +91-22-2202 9092;
Email: office@kccollege.edu.in
Website: http://www.kccollege.edu.in/

Shailja Prakashan

57–P, Kunj Vihar–II,
Yashoda Nagar, Kanpur-11
Ph.: 0512-2633004
Email: shailjaprakashan@gmail.com

ISBN 978-93-80788-93-7

9 789380 788937 >

ISBN 978-93-80788-93-7

Preface

Python is a powerful programming language that is capable of conducting any kind of data analysis in the Sciences, Social Sciences, Corporate world and Management. Mathematics is generally thought to be the language of science whereas Data Analysis is the language of research. Research is important for human progress, so as long as there is research there will be a need to analyze data.

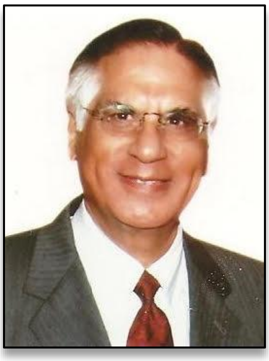
The present book is designed to meet this challenge of analysis of data. In our teaching, we generally say "Analyze this data using python, get the manuals or visit website". If you don't know how, then this book titled **"Analyzing and Visualizing Data using Free Open Source Software-Python Programming with case studies"** (Jupyter Notebook 6.0.3 and Python 3.7.6) is an answer to all such queries. It is designed in such a way that Python procedures become clear to the beginner. Also, it works as an effective reference for anyone conducting data analysis.

This involved the expansion of the original concept to include essentially all major statistical procedure as well as regression and advance models. This is the result of years of teaching/research experience of contributors that you now hold in your hands.

I, Dr. Asha A. Jindal, Associate Professor and Head, Department of Statistics wish to acknowledge all the contributors who put their creative efforts in writing the chapters. I sincerely acknowledge Principal Dr. Hemlata K. Bagla, K. C. College and Dr. Sagarika V. Damle, DBT-Star Status Coordinator for their support and encouragement.

I express sincere gratitude to Mr. Ashutosh Tiwari, Shailja Prakashan, Kanpur for wholeheartedly giving ISBN number for this book and encouraging this initiative. I thankfully acknowledge Mr. Roshan Khilani for the book cover page designing to final formatting of the present book, Mr. Shubham Niphadkar who contributed substantially to the design and format chapters of present book.

Dr. Asha Jindal
EDITOR



Message

It is heartening to learn that Dr. Asha Jindal, Head, Department of Statistics along with team of enthusiastic budding researchers has undertaken this exercise of preparing this handy book on free open source software "Python Programming with Case Studies", to meet various challenges, while analysing Datasets generated from different fields.

This book contains simple and doable data set and codes on how to prepare for analytics and what to do when caught in an analytical challenge . I am sure this crisp and informative document would enable the students and researcher community to prepare and manage analytics more effectively.

I am confident that it will also be widely useful for Educational institutions, Experts, Data Scientist and other Stakeholders as a ready recknor.

I congratulate Department of Statistics for introducing this book , under the aegis of DBT-Star College Status Scheme.

Dr. Kishu Mansukhani

President, H(S)NC Board

ISBN 978-93-80788-93-7

Message

"A good teacher can inspire hope, ignite the imagination, and instill a love of learning."

As Principal of K C College it gives me the greatest joy to see my teachers fulfilling their role and going beyond for the cause of education and students! A heartfelt congratulations to the Department of Statistics, under the aegis of DBT-Star College Status Scheme, for preparing a comprehensive list of articles, algorithm and codes in a book as well as e- book form titled "Python Programming with Case studies". We, as mentors, have a great responsibility of fortifying our students, making them capable by developing skills in them so that they are successful in their careers.

This skill oriented book will help teachers to be at par with the new methodology & technology - and this in turn will help them guide students and develop skills in them. The book keeps abreast of the latest technology, demands and needs of the industry and uses a detailed, step-by-step approach with the help of Information, Education and Communication (IEC) tools to enhance the skills of reader's.

The Book is a valuable addition untertaken by the Statistics Department to build good analysts and to empower their learning. In today's times, technological tools must be utilized to create audio-visual clips for quicker dissemination of knowledge through digital platforms!

I wish the publication of this book all success.

Dr. Hemlata K. Bagla

Principal, K. C. College

Foreword

The novelty of this book, “Analyzing and Visualizing Data using free open-source software Python Programming with case studies” emerges out of its context and from within its content. In contextual terms, Python is very popular and cutting-edge programming language which is widely used in big data analysis. This language is very user friendly and has wide applications with Python libraries. It can be easily integrated with other programming languages. The substantial practical relevance of this book is underlined through its purported attempt to introduce Python programming language in a very simple way to the beginners as well as advance learners.

In terms of the contents, this book certainly merits a leading position as there is a wide variety of applications of Python programming language with applications, and case studies are covered. The innovative qualities evidenced in various chapters are excellent and to my mind, this is demonstrated originality of a high order. This edited book consists of 16 chapters and 4 case studies. The first chapter introduces Python programming. Optimum strategies in Python are covered in the next couple of chapters. Subsequent chapters discuss graphical representation, descriptive statistics, correlation and regression, and probability distribution, which are covered in the introductory statistics course. Advanced statistical programming techniques using Python are discussed in the next few chapters, which include ANOVA, ANCOVA, Factor Analysis, Cluster Analysis, and Non-parametric test. The unique part of the book is the four case studies on Polio, Air pollution, Diabetes, and analysing happiness development index using Python. The book offers a unique insight into the linking of innovative ideas very much on the academic plane to the potential applications, as discussed in the case studies. Last but not the least, I will like to congratulate the Editor, Dr Asha Jindal for tackling such a refractory and complex area of Python programming with keen insight, modern tools, and fresh ideas to produce an excellent book. It is my fervent hope that this book will act as a torch bearer not only for the students of statistics but also for researchers in the area of big data.

Dr Kuldeep Kumar
PhD(Kent), FSS, C.Stat,
Professor, Bond Business School
Bond University, Gold Coast, Queensland 4229, AUSTRALIA

Index

<u>Case Studies</u>

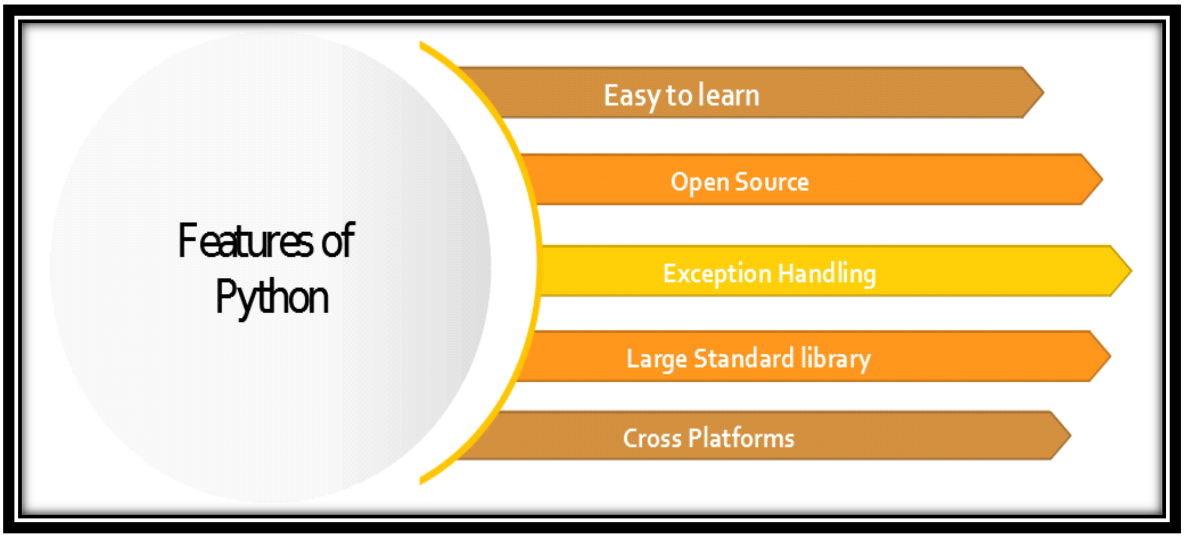
Download Resources used in the book from here.

Click here to DOWNLOAD

Chapter 1
Introduction to Python Programming

***Mr. Pravesh Tiwari**, L&T Financial Services, Manager*

Python is a very simple yet very powerful programming language. Python is developed by **Guido van Rossum**. Guido van Rossum started implementing Python in 1989**.**



1.1 What can we do with Python?

1. **Web framework like Django and Flask** are based on Python. They help you write server side code which helps you manage database, write backend programming logic, mapping urls etc.
2. There are many machine learning applications written in Python. **Face recognition** and **Voice recognition** in your phone is another example of machine learning.
3. **Data analysis** and **data visualization** in form of charts can also be developed using Python.
4. Scripting is writing small programs to **automate** simple tasks such as sending automated response emails etc.
5. You can develop **games** using Python.

1.2 Basic Syntax

1. Indentation is used in Python to delimit blocks. The number of spaces is variable, but all statements within the same block must be indented the same amount.

ISBN 978-93-80788-93-7

2. The header line for compound statements, such as if, while, def, and class should be terminated with a colon (:)

```
 1.f 5>2:
       print("True")
        print("Yes Its True")
       File "<1.python-1.nput-2-977590f094d0>", line 3
       print("Yes Its True")
       IndentationError: unindent does not match any outer indentation level
```

3. The semicolon (;) is optional at the end of statement.

4. Printing a statement to screen

```
print{"Hello"); print{"World")
   Hello
   World
```

5. Reading keyboard input

```
Name= input(“what is Your Name: '")
      what is Your Name: Pravesh
```

6. Comments

-Single Line

-Multiple Lines

```
 # single Line coment                      ‘’’Multiple
                                           Line
                                           Coments’’’
```

7. Python files have extension .py

1.3 Variables

Variables are containers for storing data values. A variable is created the moment you first assign a value to it.

```
 z=0.05                     x=6                        y=”storing value”
 print(z)                   print(x)                   print(y)
       0.05                      6                           storing value
```

Variables do not need to be declared with any particular type and can even change type after they have been set. Variables can change type, simply by assigning them a new value of a different type.

```
 x = 85
 print (“First value is:”, x)
 x =”new value”
 print(“changed value is:” + x)
       First value is: 85
       Changed value is: new value
```

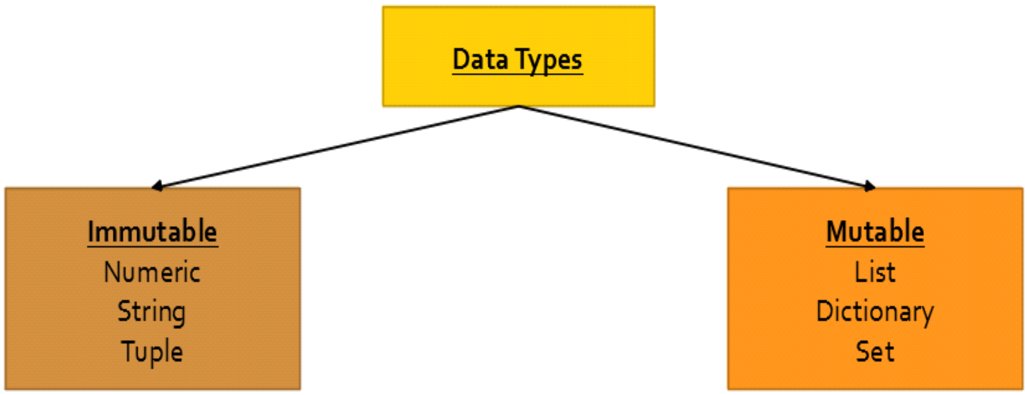
String variables can be declared either by using single or double quotes.

x = "John“ is the same as x = 'John’

Python allows you to assign a single value to several variables simultaneously.

```
 X = y = Z = 25
 print(x,y,z)
       25 25 25
```

ISBN 978-93-80788-93-7



You can also assign multiple objects to multiple variables.

```
x,y,z = 25, “string”, 86
print(x,y,z)
      25 string 86
```

Note:

A variable can have a short name (like x and y) or a more descriptive name (age, carname, total_volume).

Rules for Python variables:

A variable name must start with a letter or the underscore character.

A variable name cannot start with a number. A variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and _).

Variable names are case-sensitive (age, Age and AGE are three different variables).

1.4 Data Types in Python

Data type defines the type of the variable, whether it is an integer variable, string variable, tuple, dictionary, list etc. Python data types are divided in two categories, mutable data types and immutable data types.

The data stored in memory can be of many types. For example, a person's age is stored as a numeric value and his or her address is stored as alphanumeric characters. Python has various standard data types that are used to define the operations possible on them and the storage method for each of them.

To check the data type use **type()** function.

```
In [1]: a= [‘1’,2.2,3+1j]
In [2]: type(a)
Out [2]: list
```

Data Types in Python – Numbers:

Number data types store numeric values. They are immutable data types, means that changing the value of a number data type results in a newly allocated object.

There are three numeric types in Python:

1. **Integer** - Int, or integer, is a whole number, positive or negative, without decimals, of unlimited length.
2. **Float** - Float, or "floating point number" is a number, positive or negative, containing one or more decimals. Float can also be scientific numbers with an "e" to indicate the power of 10.
3. **Complex** - Complex numbers are written with a “i" as the imaginary part (a+bi).

You can convert from one type to another with the **int(), float(), and complex()** methods.

For e.g.:

```
In [5]: int(7.59)
Out[5]: 7
```

```
In [6]: float(8)
Out [6]: 8.0
```

```
In    [7]:    complex(8)
Out[7]: (8+0j)
```

Mathematical functions

Functions	Description
abs(x)	The absolute value of x: the (positive) distance between x and zero.
exp(x)	The exponential of x: ex
log(x)	The natural logarithm of x, for x> 0
pow(x, y)	The value of x**y.
round(x, n)	x rounded to n digits from the decimal point.
max(x1, x2,...)	The largest of its arguments: the value closest to positive infinity
min(x1, x2,...)	The smallest of its arguments: the value closest to negative infinity
sqrt(x)	The square root of x for x > 0

Data Types in Python – Strings:

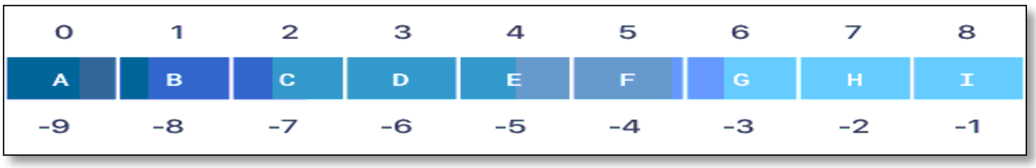
Python Strings are Immutable objects that cannot change their values.

You can update an existing string by (re)assigning a variable to another string.

```
In [6]: string= “Hello world”
        string[1]=”1”
Traceback (most recent call last):

      Flie”<ipython-input-6-e84b33767afb>”, line 2, in <module>
            String[1] = “1”
TypeError: ‘str’ object does not support item assignment
```

Python accepts single ('), double (") and triple (''' or """) quotes to denote string literals.

String indexes are starting at 0 in the beginning of the string and working their way from -1 at the end.



0	1	2	3	4	5	6	7	8
A	B	C	D	E	F	G	H	I
-9	-8	-7	-6	-5	-4	-3	-2	-1

Python accepts single ('), double (") and triple (''' or """) quotes to denote string literals.

```
example1 = ‘single quote’
example2 = “double quote”
example3 = ‘’’multiline
             quote example’’’
```

Common String Operators

Assume string variable **a** holds 'Hello' and variable **b** holds 'Python’

Operator	Description	Example
+	Concatenation - Adds values on either side of the operator	a+b will give HelloPython
*	Repetition - Creates new strings, concatenating multiple copies of the same string	a*2 will give -HelloHello
[ ]	Slice - Gives the character from the given index	a[1] will give e
[:]	Range Slice - Gives the characters from the given range	a[1:4] will give ell
in	Membership - Returns true if a character exists in the given string	H in a will give True
not in	Membership - Returns true if a character does not exist in the given string	M not in a will give True

Note: To convert a variable x to string use **str(x)**.

Data Types in Python – List:

A list in Python is an ordered group of items or elements, and these list elements don't have to be of the same type. Python Lists are **mutable** objects that can change their values.

A list contains items separated by commas and enclosed within **square brackets**.

List indexes like strings starting at 0 in the beginning of the list and working their way from -1 at the end.

Similar to strings, Lists operations include **slicing** ([ ] and [:]) , **concatenation** (+), **repetition** (*), and **membership** (in).

Access Items

```
In (7): templist = ["Cherry","88",98,"coin",88.568]
        print ("First 3 element:" ,templist[0])
        print ("Last element:” ,templist[-1])
First element: Cherry
Last element: 88.568
```

Range of Indexes

```
In [8]: templist = ["Cherry", "88", 98, "coin", 88.568]
        print ("First 3 elements : ", templist[0:3])
First elements: ["Cherry", ‘88’, 98]
```

Update List

```
In [14]: print(“Original list:”, templist)
         templist[0]=”Guava”
         print(“changed list:”, templist)
Original list: [‘Guava’, ‘88’, 98,’coin’, 88.568]
Changed list: [‘Guava’,’88’, 98,’coin’, 88.568]
```

Delete elements from List

```
In [17]: print("Origina1 1ist:", temp1ist)
         de1(temp1ist[2])
         print("Changed 1ist: ", temp1ist)
Origina1 1ist: [‘Cherry’, ‘88’, 98, ‘coin’, 88.568]
Changed 1ist: [‘Cherry’, ‘88’, ‘coin’, 88.568]
```

Built-in List Functions & Methods

Python includes the following list functions:

Function	Description

len(list)	Gives the total length of the list. for e.g.
	list1=[1,2,4,3]
	len(list1)

max(list)	Returns item from the list with max value. for e.g.
	list1=[1,2,4,3]
	max(list1)

min(list)	Returns item from the list with min value. for e.g.
	list1=[1,2,4,3]
	min(list1)

list(tuple)	Converts a tuple into list. for e.g.
	tuple=(1,2,4,3)
	list(tuple)

sum(list)	Add all elements of a list. for e.g.
	list1=[1,2,4,3]
	sum(list1)

Data Types in Python – Tuples:

A tuple is a collection which is ordered and **unchangeable**. In Python tuples are written with round brackets. Python Tuples are **Immutable** objects that cannot be changed once they have been created.

A tuple contains items separated by commas and enclosed in parentheses instead of square brackets. You can update an existing tuple by (re)assigning a variable to another tuple.
The rules for tuple indices are the **same as for lists** and they have the same operations, functions as well. To write a tuple containing a single value, you have to include a comma, even though there is only one value. For e.g. tup1 = (50,);

Python tuple method **tuple()** converts a list of items into tuples.

```
In [7]: list=(2,4,3)
        list[0]=5
Traceback (most recent ca11 1ast):
      Fi1e "<ipython-input-7-83ba3d25a7e6>", 1ine 3, in <modu1e>
      1ist1[0] = 5
TypeError:’tuple’ object does not support  item assignment
```

Data Types in Python – Dictionary:

A dictionary is a collection which is unordered, changeable and indexed. In Python dictionaries are written with curly brackets, and they have keys and values. Each key is separated from its value by a colon (:), the items are separated by commas, and the whole thing is enclosed in curly braces. An empty dictionary without any items is written with just two curly braces, like this: {}. Keys are unique within a dictionary while values may not be. The values of a dictionary can be of any type.

Creating Dictionary

```
In [8]: thisdict = {
       “brand”: “Ford”
       ”model1”:”Mustang”,
       ”year”:1964
       }
       print(thisdict)
(‘brand’:’Ford’, ’model’: ‘mustang’,’year’: 1964)
```

Slicing Dictionary

```
In [10]: print(“Brand name: ”, thisdict[“brand”])
(‘Brand name:’, ‘Ford’)

In [11]: print(“Brand name: ”, thisdict[“brand”])
         print(“”Year: ”, thisdict[“Year”])
(‘Brand name: ’, ‘Ford’)
(‘Year: ’, 1964)
```

Updating Dictionary

```
In [14]: print(thisdict)
         thisdict["brand"] ="Mercedes"; #update existing entry
         print(thisdict)
{‘brand’: ‘Ford’, ‘model’: ‘Mustang’, "year’: 1964}
{‘brand’: ‘Mercedes’, ‘model’: ‘Mustang’, ‘year’: 1964}
```

Adding new key-value in Dictionary

```
In [15]: print(thisdict)
         thisdict['Price'] = 50000
         print(thisdict)
{'brand': 'Mercedes'. 'model': 'Mustang'. 'year': 1964}
{'Price': 50000, 'brand': 'Mercedes', 'model': 'Mustang'. 'year': 1964}
```

ISBN 978-93-80788-93-7

Removing elements from Dictionary

```
In (28): print(thisdict)
        :del thisdict['Price']
       : print(thisdict)
{‘Price': 50000, 'brand": 'Mercedes', ‘model': mustang", 'year": 1964} {‘brand':
‘Mercedes’, 'model’: ‘mustang', "year": 1964}
```

thisdict.clear() can be used to delete all elements of dictionary and **del thisdict** can be used to delete entire dictionary.

1.5 Operators in Python

Operators are the constructs which can manipulate the value of operands. Consider the expression 4 + 5 = 9. Here, 4 and 5 are called operands and + is called operator.

Python language supports the following types of operators.

- Arithmetic Operators
- Comparison (Relational) Operators
- Assignment Operators
- Logical Operators
- Bitwise Operators
- Membership Operators
- Identity Operators

Operators in Python – Arithmetic Operators:

Assume variable a holds 10 and variable b holds 20.

Operator	**Description**	**Example**
+ Addition	Adds values on either side of the operator.	a + b = 30
- Subtraction	Subtracts right hand operand from left hand operand.	a – b = -10
* Multiplication	Multiplies values on either side of the operator	a * b = 200
/ Division	Divides left hand operand by right hand operand	b / a = 2
% Modulus	Divides left hand operand by right hand operand and returns remainder	b % a = 0
** Exponent	Performs exponential (power) calculation on operators	a**b =10 to the power 20
//Floor Division	The division of operands where the result is the quotient in which the digits after the decimal point are removed. But if one of the operands is negative, the result is floored, i.e.,	9//2 = 4 and 9.0//2.0 = 4.0, -11//3 = -4, -11.0//3 = -4.0

	rounded away from zero (towards negative infinity) –	

Operators in Python – Comparison Operators:

Assume variable a holds 10 and variable b holds 20

Operator	**Description**	**Example**
==	If the values of two operands are equal, then the condition becomes true.	(a == b) is not true.
!=	If values of two operands are not equal, then condition becomes true.	(a != b) is true.
<>	If values of two operands are not equal, then condition becomes true.	(a <> b) is true. This is similar to != operator.
>	If the value of left operand is greater than the value of right operand, then condition becomes true.	(a > b) is not true.
<	If the value of left operand is less than the value of right operand, then condition becomes true.	(a < b) is true.
>=	If the value of left operand is greater than or equal to the value of right operand, then condition becomes true.	(a >= b) is not true.
<=	If the value of left operand is less than or equal to the value of right operand, then condition becomes true.	(a <= b) is true.

Operators in Python – Assignment Operators:

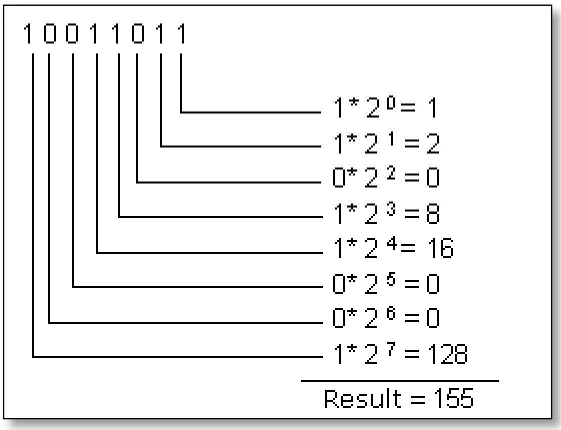
Assume variable a holds 10 and variable b holds 20

Operator	Description	Example
=	Assigns values from right side operands to left side operand	c = a + b assigns value of a + b into c
+= Add AND	It adds right operand to the left operand and assign the result to left operand	c += a is equivalent to c = c + a
-= Subtract AND	It subtracts right operand from the left operand and assign the result to left operand	c -= a is equivalent to c = c - a
*= Multiply AND	It multiplies right operand with the left operand and assign the result to left operand	c *= a is equivalent to c = c * a
/= Divide AND	It divides left operand with the right operand and assign the result to left operand	c /= a is equivalent to c = c / ac /= a is equivalent to c = c / a
%=Modulus AND	It takes modulus using two operands and assign the result to left operand	c %= a is equivalent to c = c % a
**= Exponent AND	Performs exponential (power) calculation on operators and assign value to the left operand	c **= a is equivalent to c = c ** a

ISBN 978-93-80788-93-7

//= Floor Division	It performs floor division on operators and assign value to the left operand	c //= a is equivalent to c = c // a

Operators in Python – Bitwise Operators:



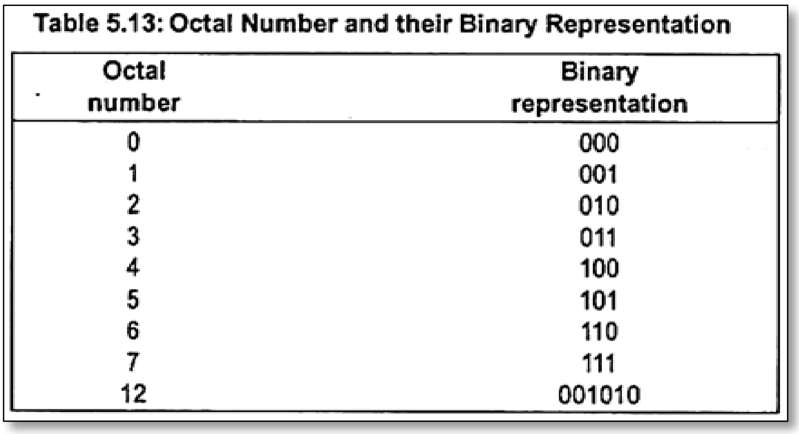


Table 5.13: Octal Number and their Binary Representation

Octal number	Binary representation
0	000
1	001
2	010
3	011
4	100
5	101
6	110
7	111
12	001010

Bitwise operator works on bits and performs bit by bit operation. Assume if **a = 60 and b = 13.** Now in binary format they will be as follows:

a = 0011 1100 and b = 0000 1101

Operator	**Description**	**Example**
& Binary AND	Operator copies a bit to the result if it exists in both operands	(a & b) (means 0000 1100)
| Binary OR	It copies a bit if it exists in either operand.	(a | b) = 61 (means 0011 1101)
^ Binary XOR	It copies the bit if it is set in one operand but not both.	(a ^ b) = 49 (means 0011 0001)
~ Binary Ones Complement	It is unary and has the effect of 'flipping' bits.	(~a) = -61 (means 1100 0011 in 2's complement form due to a signed binary number.
<< Binary Left Shift	The left operands value is moved left by the number of bits specified by the right operand.	a << 2 = 240 (means 1111 0000)

>> Binary Right Shift	The left operands value is moved right by the number of bits specified by the right operand.	a >> 2 = 15 (means 0000 1111)

Operators in Python – Logical, Identity, Membership:

Assume variable **a** holds 10 and variable **b** holds 20

Logical Operators	**Description**	**Example**
and	Returns True if both statements are true	a < 11 and b < 21
or	Returns True if one of the statements is true	a< 5 or b< 21
not	Reverse the result, returns False if the result is true	not(a < 5 and b < 10)

Identity Operators	**Description**	**Example**
is	Returns true if both variables are the same object	a is b
is not	Returns true if both variables are not the same object	a is not b

Membership Operators	**Description**	**Example**
in	Returns True if a sequence with the specified value is present in the object	a in [5,6,7,8, 10]
not in	Returns True if a sequence with the specified value is not present in the object	b not in [5,6,7,8, 10]

1.6 Control Structures in Python

In general, statements are executed sequentially: The first statement in a function is executed first, followed by the second, and so on. There may be a situation when you need to execute a block of code several number of times. Programming languages provide various control structures that allow for more complicated execution paths. A loop statement allows us to execute a statement or group of statements multiple times.

if,if…else and if…elif…else:

```
Syntax                Example

if test expression:   In [1]:  num = 3
      statement                  :if num > 0:
                                      print (num, "is positive number")
                      (3,'is a positive number')
Syntax                Example
if test expression:   In [2]:  num = -1
      statement_1           :if num > 0:
else:                                 print (num, "is positive number")
```

```
      statement_2           else:
                                  print(num"is not a positive number")
                       (-1,'is a positive number')
Syntax                 Example
if test expression:    num= -5
      statement_1      if num> 0:
elif:                        print(num, "is a positive number.")
      statement_2      elif num> 0:
else:                        print(num, "is a negative number.")
      statement_3      else:
                             print("number is zero.")
```

Python for loop:

The for loop in Python is used to iterate over a sequence (list,tuple,string) or other iterable objects. Iterating over a sequence is called traversal.

Syntax of for Loop

```
for val in  sequence:
      Body of for loop
```

Here, val is the variable that takes the value of the item inside the sequence on each iteration.

Loop continues until we reach the last item in the sequence. The body of for loop is separated from the rest of the code using indentation.

Examples

```
In [6]: Names = ["Vinay", "Pramod", "Richa", "Kanak"]
            for i in Names:
                  print(i)
Vinay
Pramod
Richa
Kanak
```

```
In [7]:  numbers = [6, 5, 3, 8, 4, 2, 5, 4, 11]
      sum =0
      for val in number:
      print(“the sum is”, sum)
(‘The sum is’, 48)
```

```
In [8]: digits =[0,1,5]
        for i in digits:
            print(i)
        else:
            print(“No number left”)
0
1
5
No number left
```

ISBN 978-93-80788-93-7

Python break statement:

The break statement terminates the loop containing it. Control of the program flows to the statement immediately after the body of the loop. If, break statement is inside a nested loop (loop inside another loop), break will terminate the innermost loop.

```
In [2] :for val in “string”:
            if val== "i":
                  break
            print(val)
            print("The end")
s
t
r
The end
```

Python continue statement:

The continue statement is used to skip the rest of the code inside a loop for the current iteration only. Loop does not terminate but continues on with the next iteration.

```
In [2] :for val in “string”:
            if val== "i":
                  continue
            print(val)
            print("The end")
s
t
r
n
g
The end
```

In Python programming, pass is a null statement. The difference between a comment and pass statement in Python is that, while the interpreter ignores a comment entirely, pass is not ignored.

```
In [7]: for num in range(1,6):
      If num==3:
            pass
      else:
            print(“Num = {}” .format(num))
Num =1
Num =2
Num =4
Num =5
```

1.7 References

1. https://www.programiz.com/python-programming
2. https://www.tutorialspoint.com/python/index.htm
3. https://www.w3resource.com/python/python-tutorial.php

Chapter 2
Functions, Modules, Advanced Operations and Pandas in Python

***Mr. Pravesh Tiwari**, L&T Financial Services, Manager*

2.1 Python Functions

A function is a block of organized, reusable code that is used to perform a single, related action. Functions provide better modularity for your application and a high degree of code reusing.

Defining a Function:

1. Keyword def marks the start of function header.
2. Parameters (arguments) through which we pass values to a function. They are optional.
3. A colon (:) to mark the end of function header.
4. Optional documentation string (docstring) to describe what the function does.
5. One or more valid python statements that make up the function body. Statements must have same indentation level.
6. An optional return statement to return a value from the function.

```
In [9]: def printme(name):
         “this prints a passed name with string into this function”
         print(“Hi”+ name + “, welcome in the class”)
In [10]: print(“Sumit”)
Hi Sumit, welcome in the class
```

You can send any data types of parameter to a function (string, number, list, dictionary etc.), and it will be treated as the same data type inside the function.

```
In [14]: def selinfo(infolist):
        print(“Name: ”, infolist[0])
        print(“Age: ”, infolist[1])
        print(“Hobby:”, infolist[2])
      info = [“Justin”, 25, “singing”]
Name: Justin
Age:  25
Hobby:Singing
```

```
 In [15]: def addition(x,y):        In [18]: def Operation(x,y):
                                                 add = x+y
                                                 sub = x-y
 c = x + y                                       product = x*y
                                                 divide = x/y
```

```
return c
In [16]: addition(5,6)
Out[16]: 11
In [19]: def Operation(x,y):
      add = x+y
      sub = x-y
      product = x*y
      divide = x/y
      return add, sub, product, divide
a,b,c,d  = operation(5,6)
```

```
         return  add,  sub,  product,
         divide
      Operation(5,6)
Out[18]:(11,-1,30,
0.8333333333333333333334)
```

2.2 Python Modules

A module is a file consisting of Python code. A module can define functions, classes and variables. A module can also include run-able code. We use modules to break down large programs into small manageable and organized files. Furthermore, modules provide reusability of code.

We can define our most used functions in a module and import it, instead of copying their definitions into different programs.

Python modules can be imported using **import** statement. For e.g. **import pandas.**

Python's *from* statement lets you import specific attributes from a module into the current namespace.

```
from modname import name1[, name2[, ... nameN]]
```

It is also possible to import all names from a module into the current namespace by using the following import statement.

```
from modname import *
```

A module can be installed using **pip** command for e.g. **pip install pandas**

```
def add(x,y):
      return(x+y)
def subtract(x, y):
      return(x-y)
```

```
import os
os.chdir
import calculation
#to Rename
from calculation import calc
```

2.3 Advanced Operations

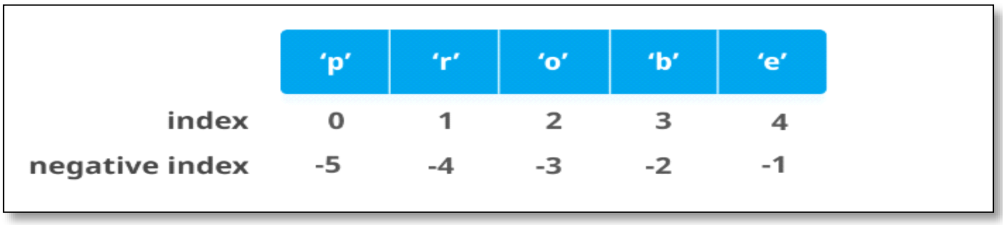
Some Basic Operations-Lists:

```
templist = []                                               #create emtylist
fruit = [“Apple ” , ”Orange” ,  “Guava” ,  “Mango” , “Banana” , “Kiwi”]
In [2] : print(fruits[0])                             #Slicing first element
Apple
In [3]: print(fruit[-1])                               #Slicing last element
Kiwi
In [4]: print(fruit[2:5])                                 #element 3rd to 5th
[‘Guava’, ‘Mango’ , ‘Banana’]
```

ISBN 978-93-80788-93-7

```
In [5]: print(fruit[:-2])                         #excluding last 2 element
['Apple ', 'orange', 'Guava', 'Mango']
In [6]: print(fruits[3:])                              #element 4th to end
['Mango', 'Banana', 'Kiwi']
```

Negative Indexing of List



	'p'	'r'	'o'	'b'	'e'
index	0	1	2	3	4
negative index	-5	-4	-3	-2	-1

Adding and Changing elements of list

We can use assignment operator (=) to change an item or a range of items.

```
 numbers= [2, 4, 5, 5, 0, 9, 7]          numbers= [2, 4, 5, 5, 0, 9, 7]
 numbers[1] = 55                         numbers[1:4] = [33,57,78]
 numbers                                 numbers
 [2, 55, 5, 5, 0, 9, 7]                  [2, 33,57,78, 0, 9, 7]
```

We can add one item to a list using **append()** method or add several items using **extend()** method.

```
 numbers= [2, 4, 5, 5, 0, 9, 7]          numbers= [2, 4, 5, 5, 0, 9, 7]
 numbers.extend([0.05,0.09,0.5])         numbers.append(0.05)
 numbers                                 numbers
 [2, 4, 5, 5, 0, 9, 7,0.05,0.09,0.5]    [2, 4, 5, 5, 0, 9, 7,0.05]
```

We can also use + operator to combine two lists. This is also called concatenation.

```
fruit =["Apple", "Orange",  "Guava"]
veggies = ["Cabbage", "Brinjal",  "Pumkin"]
combo= fruit + veggies
combo
["Apple", "Orange",  "Guava","Cabbage", "Brinjal",  "Pumkin"]
```

The * operator repeats a list for the given number of times.

```
In [18]: repeatlist = ["mean","variance",25,"67"]
         print(repeatlist*3)
["mean","variance",25,"67","mean","variance",25,"67","mean","variance",25,
"67"]
```

We can insert one item at a desired location by using the method **insert()** or insert multiple items by squeezing it into an empty slice of a list.

```
Temp = ["mean", "variance",25,"67"]
      Temp.insert(-1,"std deviation")
      Temp
['mean','variance',25,'std deviation','67']
```

```
Temp = ["mean", "variance",25,"67"]
      Temp[3:3]=[0,1,2]
      Temp
```

ISBN 978-93-80788-93-7

```
[‘mean’,’variance’,25,0,1,2,’67’]
```

Deleting and removing elements from the list

```
my_list=['p','r','o','b','l','e','m']  my_list=['p','r','o','b','l','e','m']
del my_list[2]                         del my_list[1:5]
my_list                                my_list
['p','r','b','l','e','m']              ['p','e','m']
```

```
In [33]: my_list = ['p', 'r','o','b,'l','e','m']
      del my_list
      my_list
Traceback(most recent call last):
      File"<ipython-input-33-f3fbb10993a>", line 3 ,in <module>
           my_list
NameError: name 'my_list' is not defined
```

We can use **remove()** method to remove the given item or **pop()** method to remove an item at the given index and **clear()** to completely remove all elements.

```
my_list=['p','r','o','b','l','e','m']  my_list=['p','r','o','b','l','e','m']
my_list.remove('p')                    my_list.pop(1)
my_list                                my_list
['r','o','b','l','e','m']              ['p','o','b','l','e','m']
my_list=['p','r','o','b','l','e','m']  my_list=['p','r','o','b','l','e','m']
my_list[2:4]=[]                        my_list.clear()
my_list                                my_list
['p','r','l','e','m']                  []
```

Other built in functions for list:

```
count()                                sort()
vowels=['a','e','i','o','u']           vowels=['e','a','u','o','i']
vowels.count('i')                      vowels.sort()
2                                      vowels
                                       ['a','e','i','o','u']
```

reverse()

```
months = ["January", "February", "March","April","May"]
months.reverse()
months
["May","April","March","February","January"]
```

Some Basic Operations-Dictionary:

```
my_dict= {}                                          #empty dictionary
```

```
my_dict={1:'apple',2: 'ball'}               #dictionary with integer keys
my_dict
{1: 'apple',2: 'ball'}
```

```
my_dict={'name':'John' , 1:[2,4,3]}        #dictionary with mixed keys
my_dict
{'name':'John' , 1:[2,4,3]}
```

ISBN 978-93-80788-93-7

```
my_dict = dict({1:'apple',2:'ball'})                        # using dict()
my_dict
{1:'apple',2:'ball'}
```

```
my_dict = {'name': 'Jack','age':26}            #slicing of dictionaries
my_dict['name']
'Jack'
```

```
my_dict.keys()                             #get a list of all the keys
dict_keys
(['name','age'])
```

```
my_dict.values()                             #get a list of all the values
dict_values
(['Jack',26])
```

Adding and Changing elements in a Dictionary

```
my_dict = {'name': 'Jack','age':26}
my_dict['age']=27
my_dict
{'name':'Jack','age':27}
```

```
my_dict = {'name': 'Jack','age':26}
my_dict['Hobby']="Singing"
my_dict
{'name':'Jack','age':26,'Hobby':'Singing'}
```

del keyword to remove individual items or the entire dictionary itself. All the items can be removed at once using the **clear()** method.

```
alphabets= {"a":11, "b":24, "c":32, "d":42, "e":16}
del alphabets["a"]
alphabets
{'b': 24, 'c': 32, 'd': 42, 'e': 16}
```

```
alphabets= {"a":11, "b":24, "c":32, "d":42, "e":16}
alphabets.clear()
alphabets
{}
```

```
In [21]: alphabets = {"a":11, "b":24, "c":32, "d"":42, "e":16}
        del alphabets
        alphabets
Traceback (most recent call last):
    File "<ipython-input-21-ed02c88cdd03>", line 3, in <module>
        alphabets
NameError: name 'alphabets' is not defined
```

The **update()** method inserts the specified items to the dictionary.

```
car= {
"brand": "Ford",
"model": "Mustang",
"year": 1964
}
car.update({"color": "White"})
car
{'brand': 'Ford', 'model': 'Mustang', 'year': 1964, 'color': 'White'}
```

Python Advance Operations-List and Dictionary:

Membership test for List and Dictionary

Dictionary

```
In [25]: squares = {1: 1, 3: 9, 5: 25,
7: 49, 9: 81}
      print(1 in squares)
      print(2 not in squares)
      print (49 in squares)
True
True
False
```

List

```
In [26]: my_list = ['p', 'r', 'o',
'b', 'l', 'e', 'm']
      print( 'p' in my_list)
      print(' a' in my_list)
      print('c' not in my_list)
True
False
True
```

Iterating over List and Dictionaries

Dictionary

```
In [27]: thisdict = {
            "Name": "Kartik",
            "College": "K.C.college",
            "Qualification": "B.sc"
            }
            for x in thisdict:
                  print(x)
Name
College
Qualification
```

```
In [29]: thisdict = {
            "Name": "Kartik",
            "College":
            "K.C.college",
            "Qualification": "B.sc"
            }
            for     x,     y     in
thisdict.items():
                  print(x,y)
Name Kartik
College K.C.college
Qualification B.sc
```

```
In [30]: thisdict = {
            "Name": "Kartik",
            "College": "K.C.college",
            "Qualification": "B.sc"
            }
            for x in thisdict:
                  print(thisdict[x])
Kartik
K.C.college
B.sc
```

List

```
In [31]: for fruit in ["apple","banana","mango"]:
       print(fruit)
apple
banana
```

```
mango
```

Python Advance Operations – Strings:

Slicing of String

```
In [4]: string= "Hello world"
      print(string[0])
      print(string(-1])
      print(string[2:7])
H
d
llo w
```

```
In [5]: strinc = "Hello world"
      print(strinc[25])
Traceback (most recent call last):
File"<ipython-input-5-cd88bcd3b787>"
,line 2,in <module>
      print(string[25])
IndexError: string index out of range
```

String Concatenation

```
a= "Hello"
b = "World"
a + b
"HelloWorld"
```

```
a= "Hello"
b = "World"
a + " " + b
'Hello World'
```

```
profession= "Developer"
txt= "My name is John, I am a" + profession
txt
"My name is John, I am a Developer"
```

The **format()** method takes the passed arguments, formats them, and pass them inside the placeholders **{}**.

```
In [10]: profession = “Analyst”
      txt = “My name is John, I am a {}”
      print(txt.format(profession))
My name is John, I am a Analyst
```

```
Name= "Shakir Khan"
Exp= 5
Hobby= "Reading Books"
text= "I am {}. I am working in TCS since last{} years, I like{}."
text.format(Name, Exp, Hobby)
'I am Shakir Khan. I am working in TCS since last5 years, I like Reading Books.'
```

To check if a certain phrase or character is present in a string, we can use the keywords **in** or **not in**.

```
txt = “The rain in Spain stays mainly in the plain”
x=”ain” in txt
x
True
```

Built in functions of strings

```
strip()
txt = "####### Goa Beaches##"
```

```
lower()
txt = "STRINGS"
```

```
txt.strip("#")
' Goa Beaches '
replace()
txt=   “   #GOA   #Beaches   #BestTemples
#AwesomeSeafood "
newtxt = txt.replacec("#","")
newtxt
'GOA Beaches BestTemples AwesomeSeafood’
```

```
txt.lower()
'strings'
upper()
txt = "strings"
txt.upper()
'STRINGS'
```

split()

```
txt = "Mean|Median|Mode|Geometric mean|Arithmetic mean"
newtxt = txt.split("|")
newtxt
['Mean', 'Median', 'Mode', 'Geometric mean', 'Arithmetic mean']
```

count()

```
txt = "I love apples, apple are my favorite fruit"
x = txt.count("apple")
x
2
```

join()

```
mytuple = ("John","Peter","Vicky")
x = "_".join(mytuple)
x
'John_Peter_Vicky'
```

isdigit()

```
txt = "50800"
x = txt.isdigit()
x
True
```

Escaping single quotes

```
In [21]: print('He said "what\'s there?"')
He said,"What's there?"
```

Escaping double quotes

```
In [22]: print('He said "\what's there?\"')
He said,"What's there?"
```

partition()

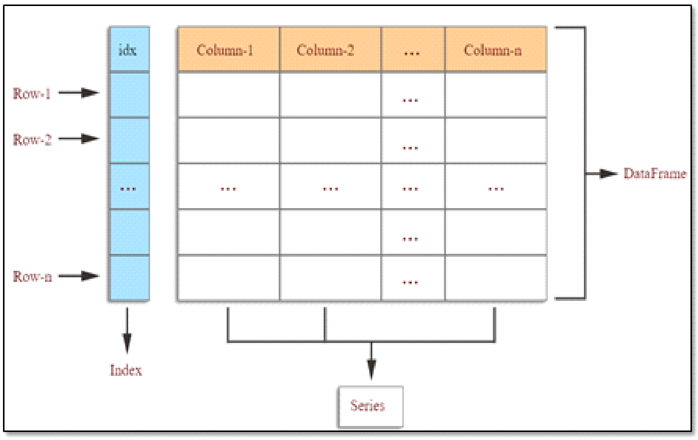
```
In [23]: txt = "I could eat Oranges all day"
      x= txt.partition("Oranges")
      print(x)
('I could eat',''Oranges','all day')
```

```
In [24]: txt = "I could eat Oranges all day"
      x= txt.partition("Apples")
      print(x)
('I could eat Oranges all day','','')
```

ISBN 978-93-80788-93-7

2.4 Introduction to Pandas

Pandas is an open-source Python Library providing high-performance data manipulation and analysis tool using its powerful data structures. It is fast, flexible, and expressive data structures designed to make working with 'relational' or 'labeled' data both easy and intuitive. It aims to be the fundamental high-level building block for doing practical, real world data analysis in Python.

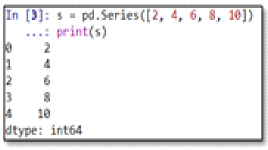


pandas is well suited for many different kinds of data:

- Tabular data with heterogeneously-typed columns, as in an SQL table or Excel spreadsheet.
- Ordered and unordered (not necessarily fixed-frequency) time series data.
- Arbitrary matrix data with row and column labels.
- Any other form of observational / statistical data sets.

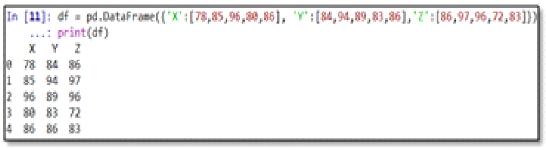
2.5 Basics of pandas Module

- Creating a data series in python



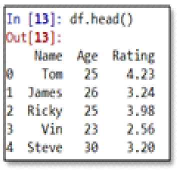
```
In [3]: s = pd.Series([2, 4, 6, 8, 10])
   ...: print(s)
0     2
1     4
2     6
3     8
4    10
dtype: int64
```

- Creating a Dataframe in python



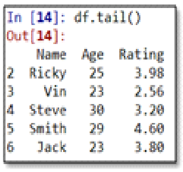
```
In [11]: df = pd.DataFrame({'X':[78,85,96,80,86], 'Y':[84,94,89,83,86],'Z':[86,97,96,72,83]})
    ...: print(df)
    X   Y   Z
0  78  84  86
1  85  94  97
2  96  89  96
3  80  83  72
4  86  86  83
```

- Inspecting datasets



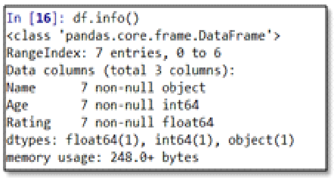
```
In [13]: df.head()
Out[13]:
     Name  Age  Rating
0     Tom   25    4.23
1   James   26    3.24
2   Ricky   25    3.98
3     Vin   23    2.56
4   Steve   30    3.20
```

.head() function will display first 5 rows of dataframe
Similarly, **.head(2)** function will display first 2 rows of dataframe



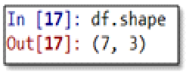
```
In [14]: df.tail()
Out[14]:
    Name  Age  Rating
2  Ricky   25    3.98
3    Vin   23    2.56
4  Steve   30    3.20
5  Smith   29    4.60
6   Jack   23    3.80
```

. tail() function will display last 5 rows of dataframe
Similarly, **.tail(2)** function will display last 2 rows of dataframe



```
In [16]: df.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7 entries, 0 to 6
Data columns (total 3 columns):
Name      7 non-null object
Age       7 non-null int64
Rating    7 non-null float64
dtypes: float64(1), int64(1), object(1)
memory usage: 248.0+ bytes
```

.info() will display structure of a dataframe i.e. datatype of each variable in dataframe.

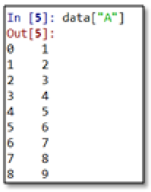


```
In [17]: df.shape
Out[17]: (7, 3)
```

.shape will show number of rows and columns in a dataframe

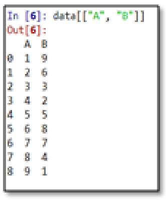
ISBN 978-93-80788-93-7

Slicing of Dataframe with Pandas



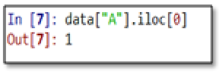
```
In [5]: data["A"]
Out[5]:
0     1
1     2
2     3
3     4
4     5
5     6
6     7
7     8
8     9
```

df[col] : Returns column with label col as Series



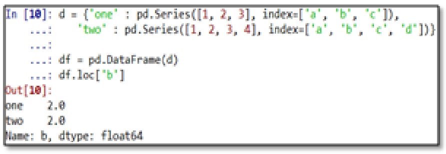
```
In [6]: data[["A", "B"]]
Out[6]:
   A  B
0  1  9
1  2  6
2  3  3
3  4  2
4  5  5
5  6  8
6  7  7
7  8  4
8  9  1
```

df[[col1, col2]]: Returns columns as a new DataFrame



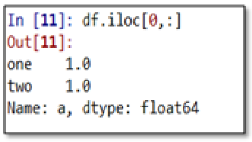
```
In [7]: data["A"].iloc[0]
Out[7]: 1
```

s.iloc[0] : Selection by position in series **s**



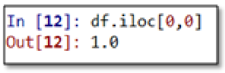
```
In [10]: d = {'one' : pd.Series([1, 2, 3], index=['a', 'b', 'c']),
    ...:      'two' : pd.Series([1, 2, 3, 4], index=['a', 'b', 'c', 'd'])}
    ...:
    ...: df = pd.DataFrame(d)
    ...: df.loc['b']
Out[10]:
one    2.0
two    2.0
Name: b, dtype: float64
```

s.loc['index_one'] : Selection by index



```
In [11]: df.iloc[0,:]
Out[11]:
one    1.0
two    1.0
Name: a, dtype: float64
```

df.iloc[0,:] : First row of dataframe df

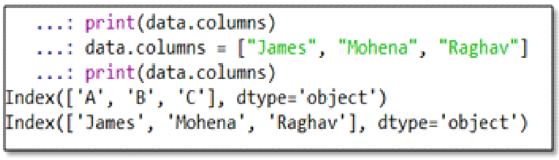


```
In [12]: df.iloc[0,0]
Out[12]: 1.0
```

df.iloc[0,0] : First element of first column

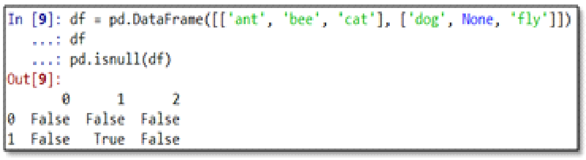
Advance Operations with Pandas

1) Renaming columns of dataframe.
   Syntax - df.columns = ['a','b','c']



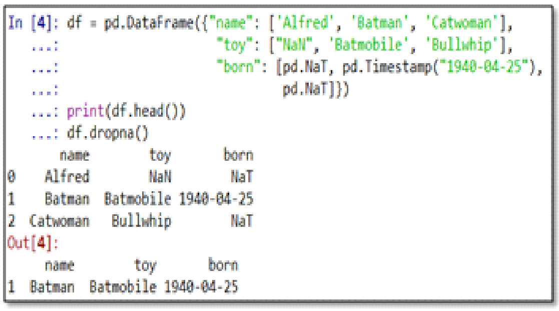
```
    ...: print(data.columns)
    ...: data.columns = ["James", "Mohena", "Raghav"]
    ...: print(data.columns)
Index(['A', 'B', 'C'], dtype='object')
Index(['James', 'Mohena', 'Raghav'], dtype='object')
```

2) Checks for null Values, Returns Boolean Array
   Syntax - pd.isnull()



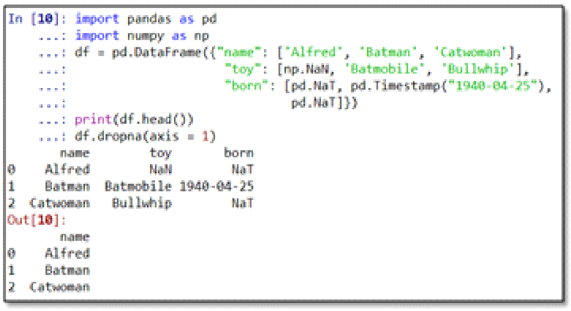
```
In [9]: df = pd.DataFrame([['ant', 'bee', 'cat'], ['dog', None, 'fly']])
   ...: df
   ...: pd.isnull(df)
Out[9]:
       0      1      2
0  False  False  False
1  False   True  False
```

3) Drop all rows that contain null values
   Syntax - df.dropna()



```
In [4]: df = pd.DataFrame({"name": ['Alfred', 'Batman', 'Catwoman'],
   ...:                    "toy": ["NaN", 'Batmobile', 'Bullwhip'],
   ...:                    "born": [pd.NaT, pd.Timestamp("1940-04-25"),
   ...:                             pd.NaT]})
   ...: print(df.head())
   ...: df.dropna()
       name        toy       born
0    Alfred        NaN        NaT
1    Batman  Batmobile 1940-04-25
2  Catwoman   Bullwhip        NaT
Out[4]:
     name        toy       born
1  Batman  Batmobile 1940-04-25
```

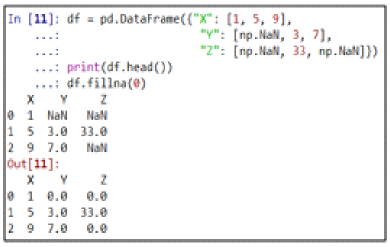
4) Drop all rows that contain null values
   Syntax - df.dropna(axis=1)



```
In [10]: import pandas as pd
    ...: import numpy as np
    ...: df = pd.DataFrame({"name": ['Alfred', 'Batman', 'Catwoman'],
    ...:                    "toy": [np.NaN, 'Batmobile', 'Bullwhip'],
    ...:                    "born": [pd.NaT, pd.Timestamp("1940-04-25"),
    ...:                             pd.NaT]})
    ...: print(df.head())
    ...: df.dropna(axis = 1)
       name        toy       born
0    Alfred        NaN        NaT
1    Batman  Batmobile 1940-04-25
2  Catwoman   Bullwhip        NaT
Out[10]:
       name
0    Alfred
1    Batman
2  Catwoman
```

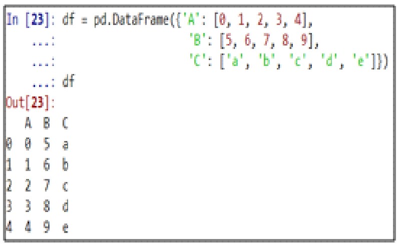
ISBN 978-93-80788-93-7

Replace all null values with x
Syntax - df.fillna(x)

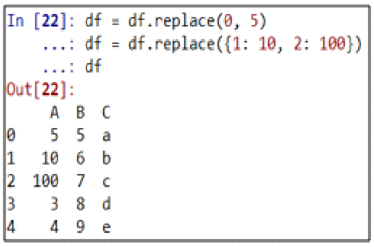


```
In [11]: df = pd.DataFrame({"X": [1, 5, 9],
    ...:                    "Y": [np.NaN, 3, 7],
    ...:                    "Z": [np.NaN, 33, np.NaN]})
    ...: print(df.head())
    ...: df.fillna(0)
   X    Y     Z
0  1  NaN   NaN
1  5  3.0  33.0
2  9  7.0   NaN
Out[11]:
   X    Y     Z
0  1  0.0   0.0
1  5  3.0  33.0
2  9  7.0   0.0
```

Replace all null values with x
Syntax - df.replace(old, new)

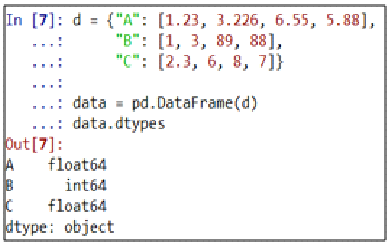


```
In [23]: df = pd.DataFrame({'A': [0, 1, 2, 3, 4],
    ...:                    'B': [5, 6, 7, 8, 9],
    ...:                    'C': ['a', 'b', 'c', 'd', 'e']})
    ...: df
Out[23]:
   A  B  C
0  0  5  a
1  1  6  b
2  2  7  c
3  3  8  d
4  4  9  e
```



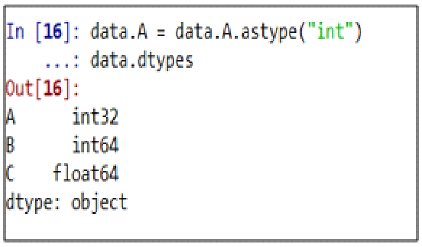
```
In [22]: df = df.replace(0, 5)
    ...: df = df.replace({1: 10, 2: 100})
    ...: df
Out[22]:
     A  B  C
0    5  5  a
1   10  6  b
2  100  7  c
3    3  8  d
4    4  9  e
```

Cast series into specific data type
Syntax - series.astype(data type)

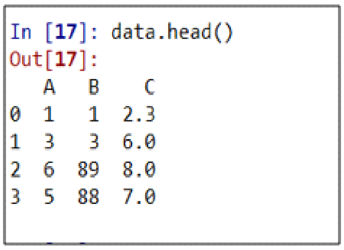


```
In [7]: d = {"A": [1.23, 3.226, 6.55, 5.88],
   ...:      "B": [1, 3, 89, 88],
   ...:      "C": [2.3, 6, 8, 7]}
   ...:
   ...: data = pd.DataFrame(d)
   ...: data.dtypes
Out[7]:
A    float64
B      int64
C    float64
dtype: object
```

To check data type of each variables on dataframe
Syntax - dataframe.dtypes



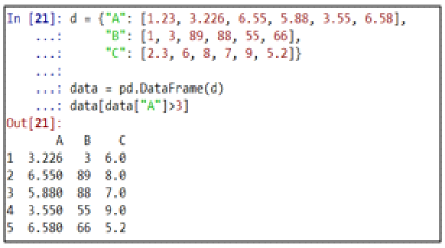
```
In [16]: data.A = data.A.astype("int")
    ...: data.dtypes
Out[16]:
A      int32
B      int64
C    float64
dtype: object
```



```
In [17]: data.head()
Out[17]:
   A   B    C
0  1   1  2.3
1  3   3  6.0
2  6  89  8.0
3  5  88  7.0
```

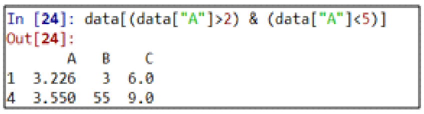
- **Filter and Sort**

Rows where the column col is greater than y
Syntax : df[df[col] > y]

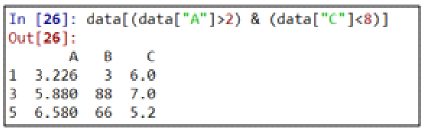


```
In [21]: d = {"A": [1.23, 3.226, 6.55, 5.88, 3.55, 6.58],
    ...:      "B": [1, 3, 89, 88, 55, 66],
    ...:      "C": [2.3, 6, 8, 7, 9, 5.2]}
    ...:
    ...: data = pd.DataFrame(d)
    ...: data[data["A"]>3]
Out[21]:
       A   B    C
1  3.226   3  6.0
2  6.550  89  8.0
3  5.880  88  7.0
4  3.550  55  9.0
5  6.580  66  5.2
```

Rows where x > col > y
Syntax : df[(df[col] > y) & (df[col] < x)]

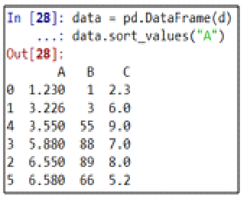


```
In [24]: data[(data["A"]>2) & (data["A"]<5)]
Out[24]:
       A   B    C
1  3.226   3  6.0
4  3.550  55  9.0
```



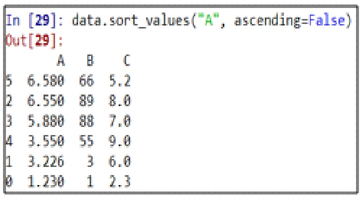
```
In [26]: data[(data["A"]>2) & (data["C"]<8)]
Out[26]:
       A   B    C
1  3.226   3  6.0
3  5.880  88  7.0
5  6.580  66  5.2
```

Sort values by col1 in
Syntax :
df.sort_values(col1)



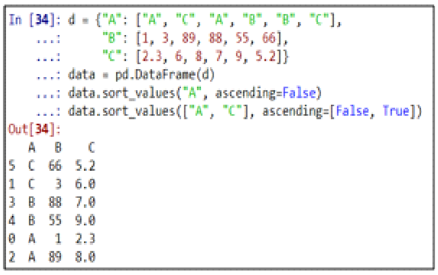
```
In [28]: data = pd.DataFrame(d)
    ...: data.sort_values("A")
Out[28]:
       A   B    C
0  1.230   1  2.3
1  3.226   3  6.0
4  3.550  55  9.0
3  5.880  88  7.0
2  6.550  89  8.0
5  6.580  66  5.2
```

Syntax :
df.sort_values(col1, ascending=False)



```
In [29]: data.sort_values("A", ascending=False)
Out[29]:
       A   B    C
5  6.580  66  5.2
2  6.550  89  8.0
3  5.880  88  7.0
4  3.550  55  9.0
1  3.226   3  6.0
0  1.230   1  2.3
```

Sort col1 in ascending order then col2 in descending order
Syntax:
df.sort_values([col1,col2],ascending=[True,False])



```
In [34]: d = {"A": ["A", "C", "A", "B", "B", "C"],
    ...:      "B": [1, 3, 89, 88, 55, 66],
    ...:      "C": [2.3, 6, 8, 7, 9, 5.2]}
    ...: data = pd.DataFrame(d)
    ...: data.sort_values("A", ascending=False)
    ...: data.sort_values(["A", "C"], ascending=[False, True])
Out[34]:
   A   B    C
5  C  66  5.2
1  C   3  6.0
3  B  88  7.0
4  B  55  9.0
0  A   1  2.3
2  A  89  8.0
```

Join / Concatenation

Pandas provide various facilities for easily combining together Series or DataFrame with various kinds of set logic for the indexes and relational algebra functionality in the case of join / merge-type operations. Here are the different types of the Joins:

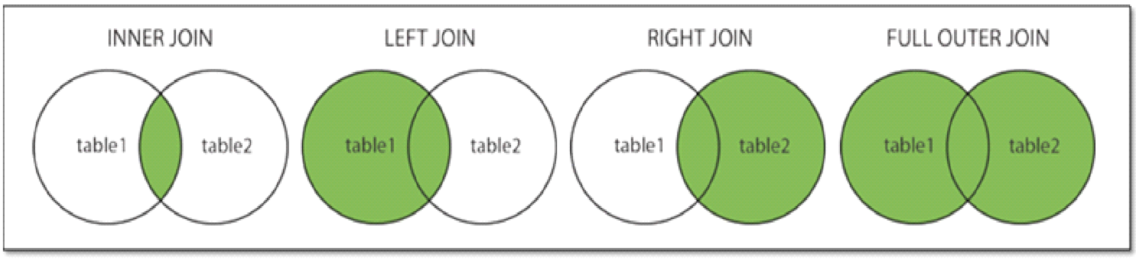
Inner Join: Returns records that have matching values in both tables

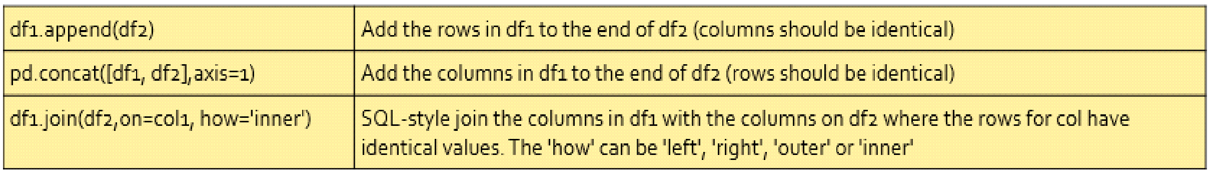
Left Join: Returns all records from the left table, and the matched records from the right table

Right Join: Returns all records from the right table, and the matched records from the left table

ISBN 978-93-80788-93-7

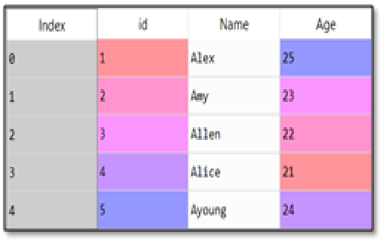
Full Join: Returns all records when there is a match in either left or right table





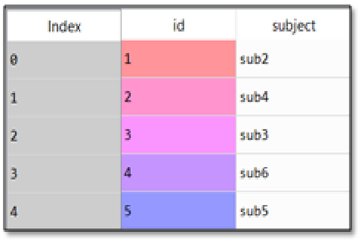
df1.append(df2)	Add the rows in df1 to the end of df2 (columns should be identical)
pd.concat([df1, df2],axis=1)	Add the columns in df1 to the end of df2 (rows should be identical)
df1.join(df2,on=col1, how='inner')	SQL-style join the columns in df1 with the columns on df2 where the rows for col have identical values. The 'how' can be 'left', 'right', 'outer' or 'inner'

Inner Join



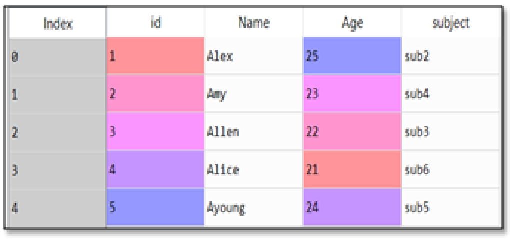
Index	id	Name	Age
0	1	Alex	25
1	2	Amy	23
2	3	Allen	22
3	4	Alice	21
4	5	Ayoung	24

Left Table



Index	id	subject
0	1	sub2
1	2	sub4
2	3	sub3
3	4	sub6
4	5	sub5

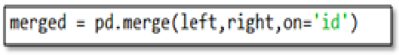
Right Table



Index	id	Name	Age	subject
0	1	Alex	25	sub2
1	2	Amy	23	sub4
2	3	Allen	22	sub3
3	4	Alice	21	sub6
4	5	Ayoung	24	sub5

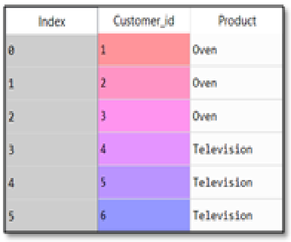
After Merging datasets on **id**

Syntax :



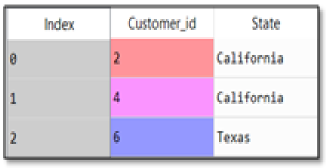
```
merged = pd.merge(left,right,on='id')
```

Left Join



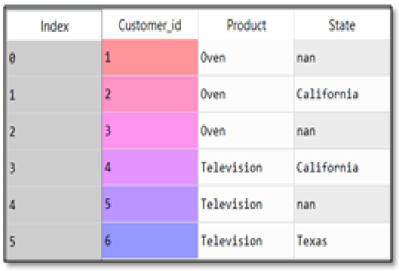
Index	Customer_id	Product
0	1	Oven
1	2	Oven
2	3	Oven
3	4	Television
4	5	Television
5	6	Television

Left Table



Index	Customer_id	State
0	2	California
1	4	California
2	6	Texas

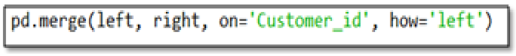
Right Table



Index	Customer_id	Product	State
0	1	Oven	nan
1	2	Oven	California
2	3	Oven	nan
3	4	Television	California
4	5	Television	nan
5	6	Television	Texas

After Left Join datasets on Customer_**id**

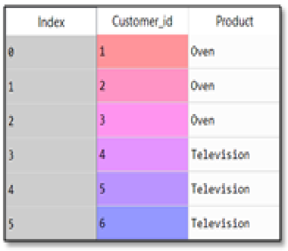
Syntax :



```
pd.merge(left, right, on='Customer_id', how='left')
```

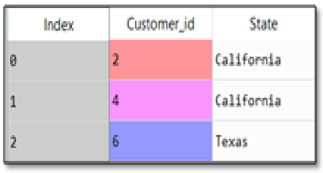
ISBN 978-93-80788-93-7

Right Join



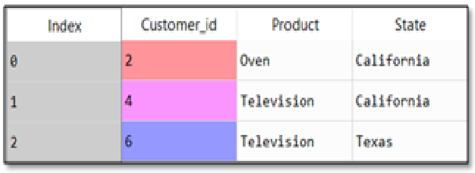
Index	Customer_id	Product
0	1	Oven
1	2	Oven
2	3	Oven
3	4	Television
4	5	Television
5	6	Television

Left Table



Index	Customer_id	State
0	2	California
1	4	California
2	6	Texas

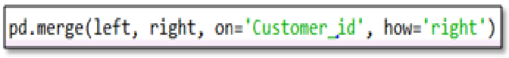
Right Table



Index	Customer_id	Product	State
0	2	Oven	California
1	4	Television	California
2	6	Television	Texas

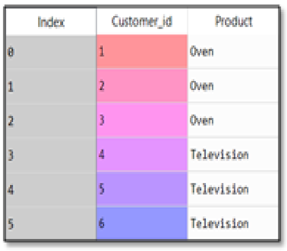
After Right Join datasets on Customer_**id**

Syntax :



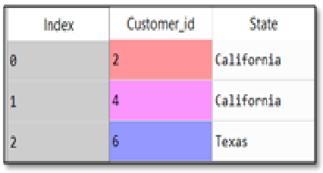
```
pd.merge(left, right, on='Customer_id', how='right')
```

Outer Join



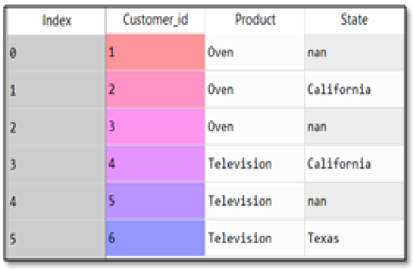
Index	Customer_id	Product
0	1	Oven
1	2	Oven
2	3	Oven
3	4	Television
4	5	Television
5	6	Television

Left Table



Index	Customer_id	State
0	2	California
1	4	California
2	6	Texas

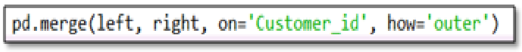
Right Table



Index	Customer_id	Product	State
0	1	Oven	nan
1	2	Oven	California
2	3	Oven	nan
3	4	Television	California
4	5	Television	nan
5	6	Television	Texas

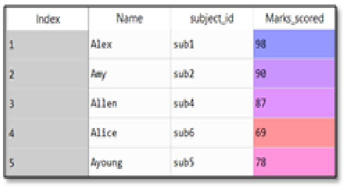
After Outer Join datasets on Customer_**id**

Syntax :



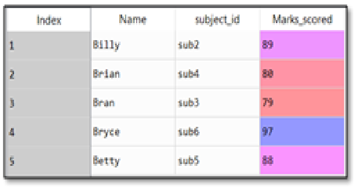
```
pd.merge(left, right, on='Customer_id', how='outer')
```

Concatenation using Pandas



Index	Name	subject_id	Marks_scored
1	Alex	sub1	98
2	Amy	sub2	90
3	Allen	sub4	87
4	Alice	sub6	69
5	Ayoung	sub5	78

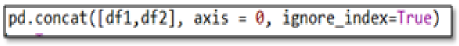
df1



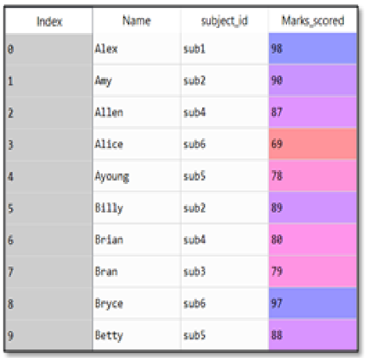
Index	Name	subject_id	Marks_scored
1	Billy	sub2	89
2	Brian	sub4	80
3	Bran	sub3	79
4	Bryce	sub6	97
5	Betty	sub5	88

df2

Syntax :

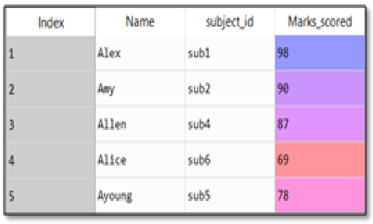


```
pd.concat([df1,df2], axis = 0, ignore_index=True)
```



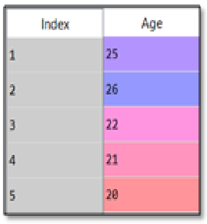
Index	Name	subject_id	Marks_scored
0	Alex	sub1	98
1	Amy	sub2	90
2	Allen	sub4	87
3	Alice	sub6	69
4	Ayoung	sub5	78
5	Billy	sub2	89
6	Brian	sub4	80
7	Bran	sub3	79
8	Bryce	sub6	97
9	Betty	sub5	88

After Concatenating df1 and df2



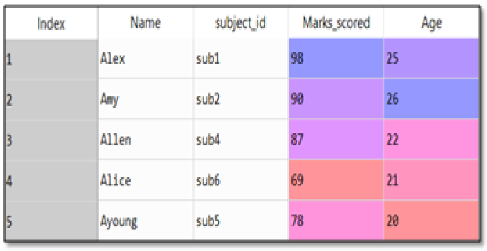
Index	Name	subject_id	Marks_scored
1	Alex	sub1	98
2	Amy	sub2	90
3	Allen	sub4	87
4	Alice	sub6	69
5	Ayoung	sub5	78

df1



Index	Age
1	25
2	26
3	22
4	21
5	20

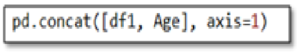
Pandas Series Age



Index	Name	subject_id	Marks_scored	Age
1	Alex	sub1	98	25
2	Amy	sub2	90	26
3	Allen	sub4	87	22
4	Alice	sub6	69	21
5	Ayoung	sub5	78	20

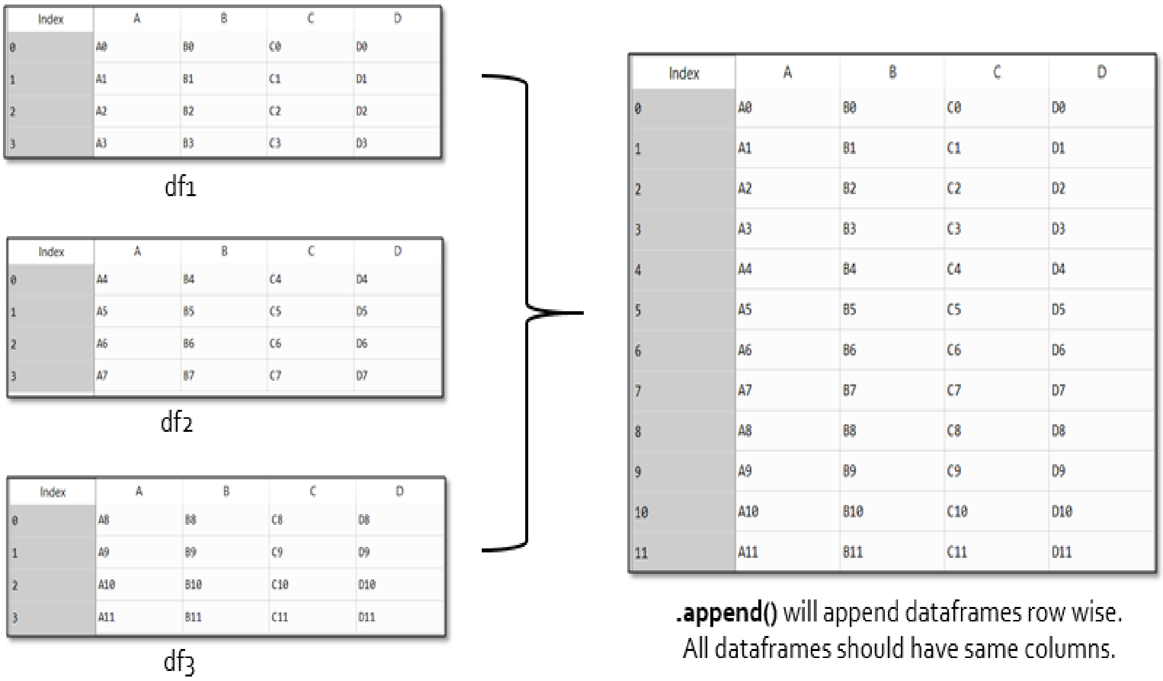
Concatenating Series to Dataframe columnwise

Syntax :



```
pd.concat([df1, Age], axis=1)
```

ISBN 978-93-80788-93-7



Index	A	B	C	D
0	A0	B0	C0	D0
1	A1	B1	C1	D1
2	A2	B2	C2	D2
3	A3	B3	C3	D3

df1

Index	A	B	C	D
0	A4	B4	C4	D4
1	A5	B5	C5	D5
2	A6	B6	C6	D6
3	A7	B7	C7	D7

df2

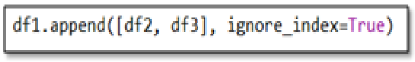
Index	A	B	C	D
0	A8	B8	C8	D8
1	A9	B9	C9	D9
2	A10	B10	C10	D10
3	A11	B11	C11	D11

df3

Index	A	B	C	D
0	A0	B0	C0	D0
1	A1	B1	C1	D1
2	A2	B2	C2	D2
3	A3	B3	C3	D3
4	A4	B4	C4	D4
5	A5	B5	C5	D5
6	A6	B6	C6	D6
7	A7	B7	C7	D7
8	A8	B8	C8	D8
9	A9	B9	C9	D9
10	A10	B10	C10	D10
11	A11	B11	C11	D11

.append() will append dataframes row wise.
All dataframes should have same columns.

Syntax :

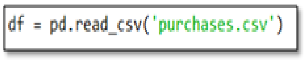


```
df1.append([df2, df3], ignore_index=True)
```

Importing and Exporting Dataframes

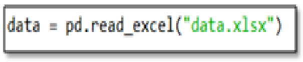
The pandas I/O API is a set of top level reader functions accessed like **pd.read_csv()** that generally return a pandas object. The corresponding writer functions are object methods that are accessed like df.to_csv()

Reading csv files using pandas in python.



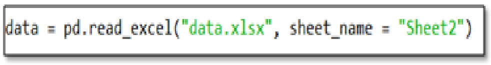
```
df = pd.read_csv('purchases.csv')
```

Reading Excel files using pandas in python



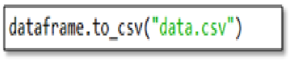
```
data = pd.read_excel("data.xlsx")
```

Reading specific sheet from an excel file



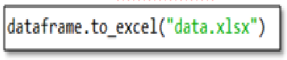
```
data = pd.read_excel("data.xlsx", sheet_name = "Sheet2")
```

Exporting pandas dataframe to csv format



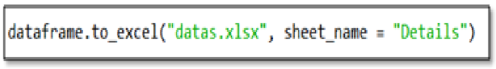
```
dataframe.to_csv("data.csv")
```

Exporting pandas dataframe to excel format



```
dataframe.to_excel("data.xlsx")
```

Exporting pandas dataframe to excel format



```
dataframe.to_excel("datas.xlsx", sheet_name = "Details")
```

2.6 References

1. https://www.programiz.com/python-programming
2. https://www.tutorialspoint.com/python/index.htm
3. https://www.w3resource.com/python/python-tutorial.php

Chapter 3
Optimisation Strategy in Python- I (Linear Programming Problem and Integer Programming Problem using pulp code)

***Dr. Asha Jindal**, Associate Professor and Head, Department of Statistics,*
K. C. College

The objective of pulp is to allow an Operations Research programmer to express Linear Programming (LP), and Integer Programming (IP) models in python in a way similar to the conventional mathematical notation or framework. Pulp will also solve these problems using a variety of free and non-free LP solvers. This Chapter is aimed to provide opportunity to python programmer who may wish to use pulp in their code in the simplified form.

3.1 Introduction

Operations Research (O.R.) is the discipline of applying advanced analytical methods to help to make sound and effective decisions. The particular area of operations research where pulp is useful for the development and modelling of Linear Programming (LP) and Integer Programming (IP) problems. Mathematically, an LP problem is a point in an n-dimensional linearly constrained region that maximises a given linear objective function. Integer Programming is an LP where the solution must contain discrete variables which take an integer value at the solution.

Solve the following Linear Programming Problem:
Maximize $Z = 20x + 30y$
Subject to
$x + 2y \leq 100$
$2x + y \leq 100$
$x, y \geq 0$

Solution
To carry out this Analysis one has to install pulp library and steps are as follows:

1. Go to search in start button and type Anaconda Prompt which will open the window.
2. Type pip install pulp and press Enter key
3. Installation process will begin and it will complete in approximately 2 minutes depends on speed of Internet.

```
In [9]: import pulp
        #Create Lp Maximization Problem
        prob=pulp.LpProblem('maximizeProfit',pulp.LpMaximize)
        #create a variable x>=0
        x=pulp.LpVariable("x",lowBound=0)
        #create another variable y>=0
        y=pulp.LpVarible("y",lowBound=0)
        prob+=20*x+30*y #objective function
        prob+=1*x+2*y<=100 #Constraint 1
        prob+=2*x+1*y<=100 #Constraint 2
        prob

Out [9]: maximizeProfit
        MAXIMIZE
        20*x + 30*y + 0
        SUBJECT TO
        _c1: x + 2y <= 100
        _c2: 2x + y<= 100
        VARIABLES
        x Continuous
        y Continuous

In [18]: prob.solve()
        pulp.value(x),pulp.value(y),pulp.value(prob.objective)
Out [18]:
(33.333333, 33.333333, 1666.6666500000001)
```

Note: Decisions Variables are not integers. So, LPP is solved in the form of IPP by adding category argument in defining variables.

```
In [19]: import pulp
      #Create Lp Maximization Problem
      prob=pulp.LpProblem('maximizeProfit',pulp.LpMaximize)
      #create a variable x>=0
      x=pulp.LpVariable("x",lowBound=0, cat=('Integer'))
      #create another variable y>=0
      y=pulp.LpVarible("y",lowBound=0, cat=('Integer'))
      prob+=20*x+30*y #objective function
      prob+=1*x+2*y<=100 #Constraint 1
      prob+=2*x+1*y<=100 #Constraint 2
      prob

Out [19]:maximizeProfit:
      MAXIMIZE
      20*x + 30*y + 0
      SUBJECT TO
      _c1: x + 2y <= 100
      _c2: 2x + y<= 100
      VARIABLES
      0 <= x Integer
      0 <= y Integer


In [20]: prob.solve()
      pulp.value(x),pulp.value(y),pulp.value(prob.objective)
Out [20]:
(32.0,34.0,1660.0)
```

3.2 References

1. Python Tutohttps://www.py4e.com/lessons
2. https://amzn.to/2VmpDwK
3. https://amzn.to/2GQSV3D

Chapter 4
Optimisation Strategy in Python-II (Dynamic Programming)

***Principal Dr. C. S. Kakade**, Anandibai Raorane Arts, Commerce & Science College, Vaibhavwadi*

4.1 Introduction

Python is one of the most popular and fastest growing languages nowadays. It is probably the best alternative for C++ and Java. It serves you with the requisite speed, leverages easy readability and also ensures shorter codes. Developers from the remotest corners of the world are embracing the presence of Python.

Python is currently the most widely used multi-purpose, high-level programming language. It allows programming in **Object-Oriented** and **Procedural** paradigms. The programs generally are smaller than other programming languages like Java. Programmers have to type relatively less and indentation requirement of the language, makes them readable all the time. Python language is being used by almost all tech-giant companies like – Google, Amazon, Facebook, Instagram, Dropbox, Uber etc.

Some examples of real-world Python use:

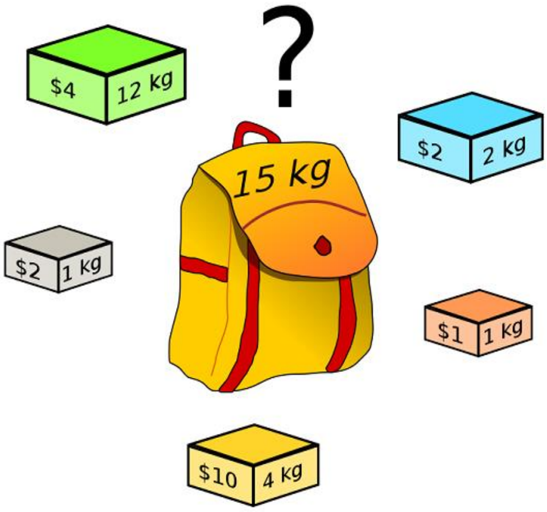
- Quora is mostly Python based.
- Snapchat is almost entirely Python based.
- Instagram heavily uses Python.
- YouTube is served for the most part by a Python app.
- Facebook uses Python in several backend applications.
- Many of the start-up and management scripts on Linux use Python (and the WWW is mainly based on Linux Servers).
- Many **machine-learning** research projects across the world use Python.

4.2 Dynamic Programming

Dynamic programming is a method for solving a complex problem by breaking it down into a collection of simpler **subproblems**, solving each of those subproblems just once, and storing their solutions using a **memory-based data structure** (array, map, etc). Each of the subproblem solutions is indexed in some way, typically based on the values of its input parameters, so as to facilitate its lookup. So, the next time the same subproblem occurs, instead of recomputing its solution, one simply looks up the previously computed solution, thereby saving computation time. We shall be deep diving into python programming with the help of an interesting case study.

4.3 Problem statement

Consider a thief gets into a home to rob and he carries a knapsack. There are fixed number of items in the home, each with its own weight and value. E.g.: Jewellery, with less weight and highest value v/s tables, with less value but a lot heavy. To add fuel to the fire, the thief has an old knapsack which has limited capacity. Obviously, he can't split the table into half or jewellery into 3/4ths. He either takes it or leaves it. Determine the items which the thief needs to steal so that the final contents in the knapsack have maximum value.



Consider the following scenario:
Knapsack Max weight: W = 15 (kg)
Total items: N = 5
Values of items: val = [4, 2, 1, 10, 2]
Weight of items: wt = [12, 2, 1, 4, 1]

The way this is **optimally** solved is using dynamic programming – solving for smaller sets of knapsack problems and then expanding them for the bigger problem.

4.4 Approach

1. The function printknapSack is defined.
2. It takes four arguments: two lists value (val) and weight (wt); total capacity (W) and total items (n).
3. It prints the maximum value of items that doesn't exceed capacity in weight.
4. The function creates a table K where K[n][W] will store the maximum value that can be attained with a maximum capacity of W and using only the first n items.
5. If K[n][W] was already computed before, this value is immediately returned.
6. If i = 0, then 0 is returned.
7. If w = 0, then 0 is returned.
8. If wt[i] > w, then K[i][w] is set to K [i – 1] [w].
9. Otherwise, K[i][w] = (K [i – 1] [w – wt[i]] + val[i]) or K[i][w] = K [i – 1] [w], whichever is larger.
10. The above computations are done in a python for-loop. Once the table has been populated, the final solution can be found at the last row in the last column, which

ISBN 978-93-80788-93-7

represents the maximum value obtainable with all the items and the full capacity of the knapsack.

Below is the source code of a Python program to solve the **0/1 knapsack** problem using dynamic programming with **memorization.**

```
def printknapSack(W, wt, val, n):
    K = [[0 for w in range(W + 1)] for i in range(n + 1)]

    # Build table K[][] in bottom
    # up manner
    for i in range(n + 1):
        for w in range(W + 1):
            if i == 0 or w == 0:
                K[i][w] = 0
            elif wt[i - 1] <= w:
                K[i][w] = max(val[i - 1] + K[i - 1][w - wt[i - 1]],
                K[i-1][w])
            else:
                K[i][w] = K[i - 1][w]

    # stores the result of Knapsack
    res = K[n][W]
    print(res)

    w = W
    for i in range(n, 0, -1):
        if res <= 0:
            break
        # either the result comes from the
        # top (K[i-1][w]) or from (val[i-1]
        # + K[i-1] [w-wt[i-1]]) as in Knapsack
        # table. If it comes from the latter
        # one/ it means the item is included.
        if res == K[i - 1][w]:
            continue
        else:

            # This item is included.
            print(wt[i - 1])

            # Since this weight is included
            # its value is deducted
            res = res - val[i - 1]
            w = w - wt[i - 1]


# Driver code
val = [4, 2, 1, 10, 2]
wt = [12, 2, 1, 4, 1]
W = 15
```

```
n = len(val)

printknapSack(W, wt, val, n)
```

Solution

The maximum value that the knapsack can contain is $15 with item 2, item 3, item 4 and item 5 having weights 2kg, 1kg, 4kg and 1kg respectively. Thus, the thief can steal four items and obtain maximum value of $15. Analyzing the complexity of the solution is pretty straight-forward. We just have a loop for W within a loop of n => O (nW).

4.5 References

1. https://www.hackerearth.com/practice/notes/the-knapsack-problem/
2. https://www.techiedelight.com/introduction-dynamic-programming/
3. https://en.wikipedia.org/wiki/Knapsack_problem
4. https://www.quora.com/Where-is-Python-used-in-the-real-world

Chapter 5

Graphs and Diagrams (2D and 3D using Python)

Mrs. Pratiksha Kadam, *Assistant Professor, Department of Statistics,*
K. C. College

In this chapter we are going to learn how to create simple 2D and 3D plots using Python. For plotting in python we use ***pyplot*** module which is stored in ***Matplotlib*** package. Also we need an array of numbers to plot which we import as *plt* alias. Various array functions are defined in the ***NumPy*** library which is imported with the *np* alias.

5.1 pyplot Functions

matplotlib.pyplot contains command style functions which make Matplotlib work like MATLAB. Each pyplot function makes some change to a figure.

Following table shows the **pyplot function** for specific types of plots:

bar	Bar plot
barh	Horizontal bar plot
boxplot	Box plot/ Whisker plot
hist	Histogram
pie	Pie chart
plot	Lines and markers to the axes
scatter	Scatter plot
stackplot	Stacked area plot
stem	Stem plot
step	Step plot
quiver	2D field of arrows

Following table shows some **figure functions**:

figure	To create a new figure
figtext	To add text to figure
show	To show the figure
savefig	To save the figure

ISBN 978-93-80788-93-7

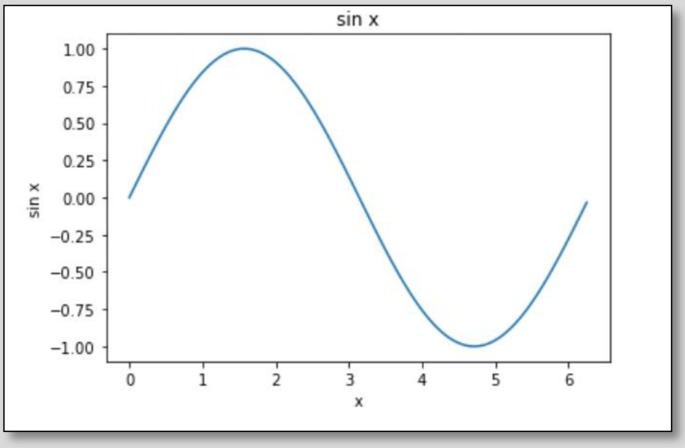
Following table shows **axis functions**:

axes	To add axes to the figure
text	To add text to the axes
xlabel	To set the label for x axis
ylabel	To set the label for y axis
xlim	To define the limit on x axis
ylim	To define the limit on y axis
xscale	To set the scaling on axis
yscale	To set the scaling on y axis
xticks	To set the tick marks on x axis
yticks	To set the tick marks on y axis

Also when we create the different plots, instead of line we can use various symbols and different colours to the plot.

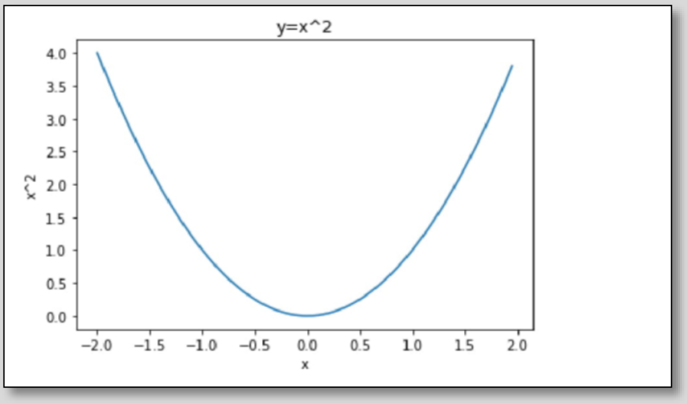
Plot symbols can be: - , –, -., , . , o , ^ , v , < , > , s , + , x , D , d , 1 , 2 , 3 , 4 , h , H , p , | , _
Plot colours can be: b, g, r, c, m, y, k, w (these are the basic colours which can be used). More colours are also available. Here b represents Blue, g represents Green, r represents Red, c represents Cyan, m represents Magenta, y represents Yellow, k represents Black and w represents white. Following example shows sine wave plotting on the domain$[0, 2\pi]$. Here *xlabel, ylabel* and *title* represent the label for x axis, label for y axis and the plot title respectively.

```
from matplotlib import pyplot as plt
import numpy as np
import math
x=np.arange(0, math.pi*2, 0.05)
y=np.sin(x)
plt.plot(x,y)
plt.xlabel("x")
plt.ylabel("sin x")
plt.title(“sin x”)
plt.show()
```



Another example of plot function that displays the graph of the function $y = x^2$ on the domain [-2,2]. Default plot colour is Blue..

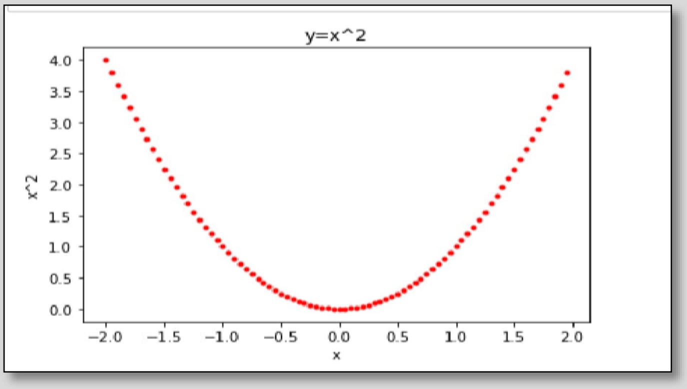
```
from matplotlib import pyplot as plt
import numpy as np
import math
x=np.arange(-2, 2, 0.05)
y=pow(x,2)
plt.plot(x,y)
plt.xlabel("x")
plt.ylabel("x^2")
plt.title(“y=x^2”)
plt.show()
```



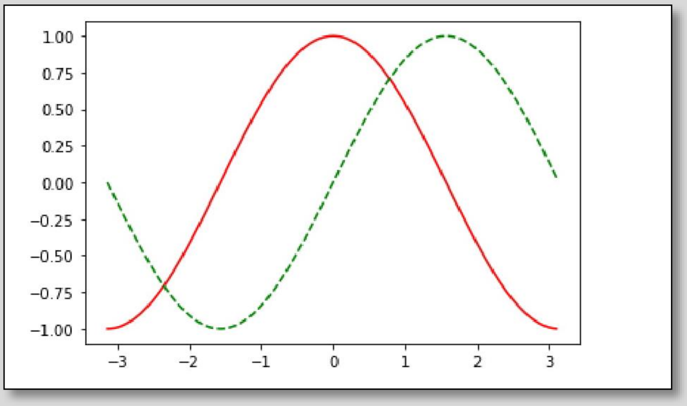
Plot symbols are available in * library which is available in a module called ***pylab*** which is a procedural interface for matplotlib. So in the following illustration, we plot dotted curve with red dots. The format ‘r.’ in the plot statement specified represents red coloured dotted plot.

```
from matplotlib import pyplot as plt
import numpy as np
import math
x=np.arange(-2, 2, 0.05)
y=pow(x,2)
plt.xlabel("x")
plt.ylabel("x^2")
plt.title(“y=x^2”)
plot(x,y,'r.')
```

```
plt.show()
```



```
from matplotlib import pyplot as plt
import numpy as np
from pylab import *
import math
x=np.arange(-math.pi, math.pi, 0.05)
plot(x, cos(x), 'r-')
plot(x, sin(x), 'g--')
show()
```



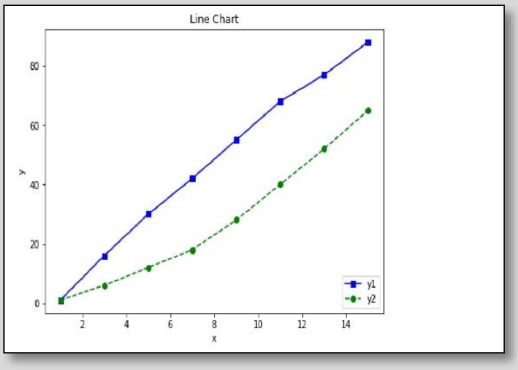
Multiple plot commands can be used in a single program to plot multiple plots together. Following example shows graphs of $\sin x$ and $\cos x$ on the domain $[-\pi, \pi]$. Here 'r-' represents red continuous line and 'g--' represents green dashed line.

5.2 Line Plot

Following example shows the line plots. Line representing y1 is a solid line with blue color and square markers whereas y2 line is a dashed line with green colour and circle marker. The *add_axes()*method is used to add axes to the figure. It requires a list of 4 values corresponding to left, bottom, width and height of the figure. Each value must be between 0 and 1.

ax.legend, ax.set_title,ax.set_xlabel, ax.set_ylabel are used to add plot legends, plot title, x axis label and y axis label respectively.

```
import matplotlib.pyplot as plt
x = [1,3,5,7,9,11,13,15]
y1 = [1, 16, 30, 42,55, 68, 77,88]
y2 = [1,6,12,18,28, 40, 52, 65]
fig=plt.figure()
ax=fig.add_axes([0,0, 1, 1])
l1=ax.plot(x,y1, 'bs-') # -:solid line b:blue colour s:square marker
l2=ax.plot(x,y2, 'go--")# --:dash Line g:green colour o:circle marker
ax.legend(labels=( 'yl', 'y2'), loc= ' lower right") # legend placed at lower
right
ax.set_title("Line Chart")
ax.set_xlabel('x')
ax.set_ylabel('y')
plt.show()
```



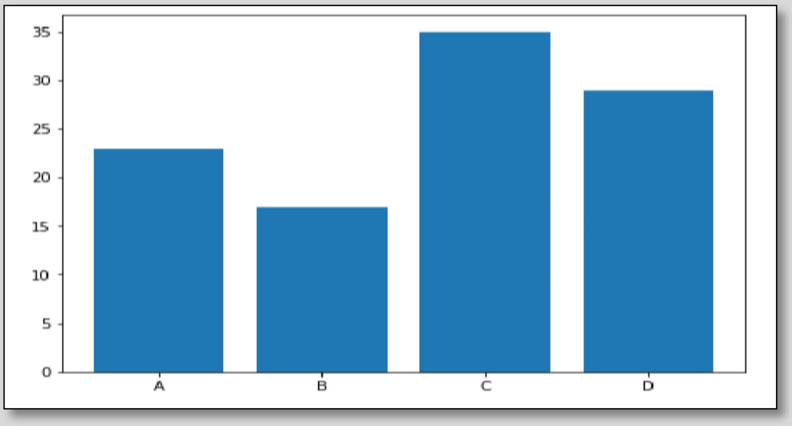
5.3 Bar Plot

A Bar chart is a chart or graph that presents categorical data with rectangular bars with heights proportional to the values that they represent.

*bar(x, height[, width, bottom, align])*function is used to plot bar diagram, where x represents category, height represents height of the bar, other three parameters are optional. width represents width of the bar (default width is 0.8), bottom represents y coordinates of the bar (by default its none), align represents alignment which can be either center or edge (by default its center).

In the following example number of students in the divisions A, B, C and D are presented using bar chart.

```
import matplotlib.pyplot as plt
fig=plt.figure()
ax=fig.add_axes([0,0,1,1])
divs=['A', 'B', 'c', 'D']
students=[23,17,35,29]
ax.bar(divs,students)
plt.show()
```



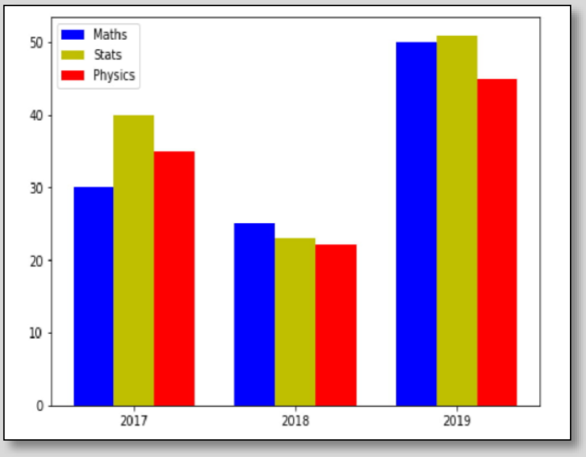
5.4 Multiple Bar Plot

Multiple bar charts can be plotted adjusting the thickness and the positions of the bars. In the following example, the data variable contains three series of three values.

The following script will show three bar charts of three bars. The bars will have a thickness of 0.25 units. Each bar chart will be shifted 0.25 units from the previous one. The data contains number of students passed in three subjects (Maths, Stats and Physics) from year 2017 to 2019.

```
import numpy as np
import matplotlib.pyplot as plt
data=[[30, 25, 50],
[ 40, 23, 51],
[35, 22, 45]]
X = np.arange(3)
fig=plt.figure()
ax=fig.add_axes([0,0,1,1])
ax.bar(X + 0.00, data[0], color= 'b', width 0.25)
ax.bar(X + 0.25, data[1], color='y', width 0.25)
ax.bar(X + 0.50, data[2], color= 'r', width 0.25)
ax.set_xticks([0.25,1.25,2.25])
ax.set_xticklabels([2017,2018,2019])
ax.legend(labels=['Maths', 'Stats', 'Physics'])
plt.show()
```

ISBN 978-93-80788-93-7

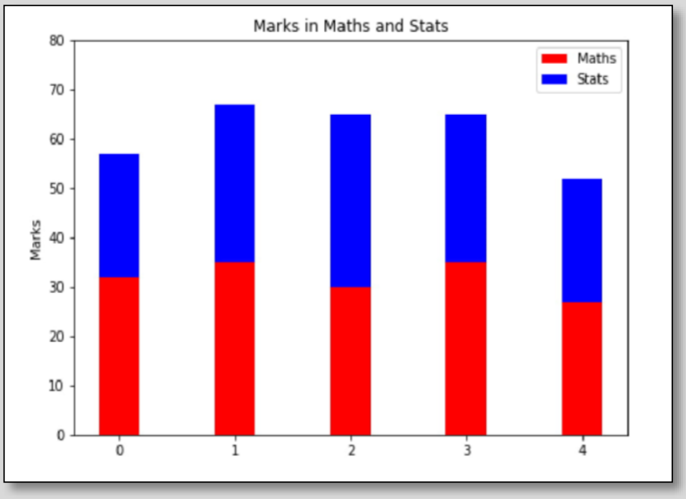


5.5 Stacked Bar Chart

This chart represents different groups on top of each other. The height of the resulting bar shows the combined result of the groups.

The optional bottom parameter of the **pyplot.bar()** function allows us to specify a starting value for a bar. So in the following example, in which marks scored by 5 students in maths and stats are represented by stacked bar plot, where first pyplot.bar() plots the red bars representing maths marks. The second pyplot.bar() plots the blue bars representing stats marks, with the bottom as the top of red bars. Data consists of
`Maths= (32, 35, 30, 35, 27); Stats= (25, 32, 35, 30, 25)`

```
import numpy as np
import matplotlib.pyplot as plt
N = 5
Maths= (32, 35, 30, 35, 27)
Stats= (25, 32, 35, 30, 25)
ind= np.arange(N) # the x Locations far the groups
width= 0.35
fig=plt.figure()
ax=fig.add_axes([0,0,1,1])
ax.bar(ind, Maths, width, color='r')
ax.bar(ind, Stats, width,bottom=Maths, color='b')
ax.set_ylabel('Marks')
ax.set_title('Marks in Maths and Stats')
ax.set_xticks(ind,('A', 'B', 'C', ·o·, 'E'))
ax.set_yticks(np.arange(0, 90, 10))
ax.legend(labels=['Maths', 'Stats'])
plt.show()
```

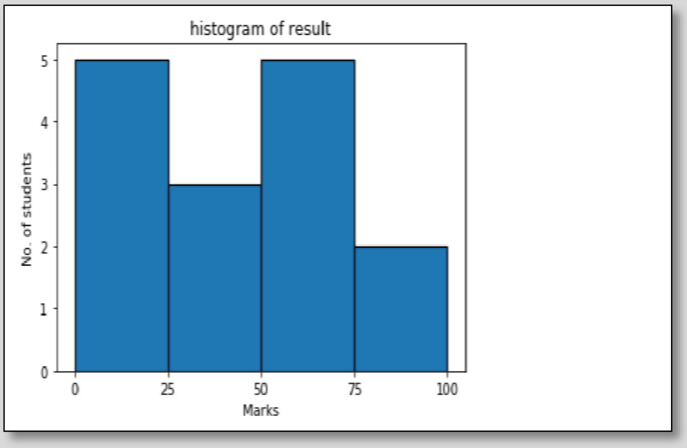


5.6 Histogram

It is a representation of a frequency distribution by means of rectangles whose widths represent class intervals and whose areas are proportional to the corresponding frequencies.

The command *hist()* used in the following program plots the histogram for the data given in array ‘a’ with the class intervals -0-25, 25-50, 50-75 and 75-100. edgecolor and linewidth parameters are optional which set border-colour of histogram as black and width of border as 1.2 points.

```
from matplotlib import pyplot as plt
import numpy as np
fig,ax=plt.subplots(1, 1)
a = np.Array[ (22,87,5,43,56,73,55,54,11,20,51,5,79,31,27])
ax.hist(a, bins = [0,25,50,75,100],edgecolor='black', linewidth=l.2)
ax.set_title("histogram of result")
ax.set_xticks([0, 25, 50, 75, 100])
ax.set_xlabel('Marks')
ax.set_ylabel('No. of students')
plt.show()
```

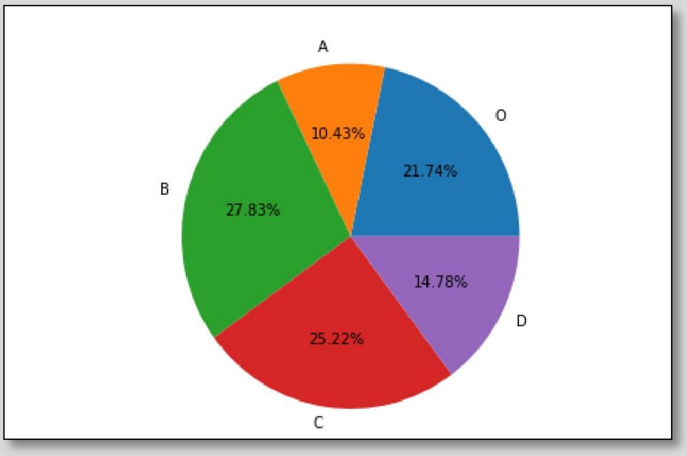


5.7 Pie Chart

It is a type of graph in which a circle is divided into sectors that each represent a proportion of the whole.

To draw pie chart, *pie()* function is used. Colors in the following example are basic colors. One can use an additional argument colors to specify slice colors. One can also set the distance between graph and labels, slice distance by using optional arguments labeldistance and slice distance. So the following code plots a pie chart that displays percentage of number of students who have secured the grades O, A, B, C and D.

```
from matplotlib import pyplot as plt
import numpy as np
fig=plt.figure()
ax=fig.add_axes([0,0,1,1])
ax.axis('equal')
grade=['O','A','B','C','D']
students=[25,12,32,29,17]
ax.pie(students, labels=grade,autopct='%1.2f%%')
plt.show()
```



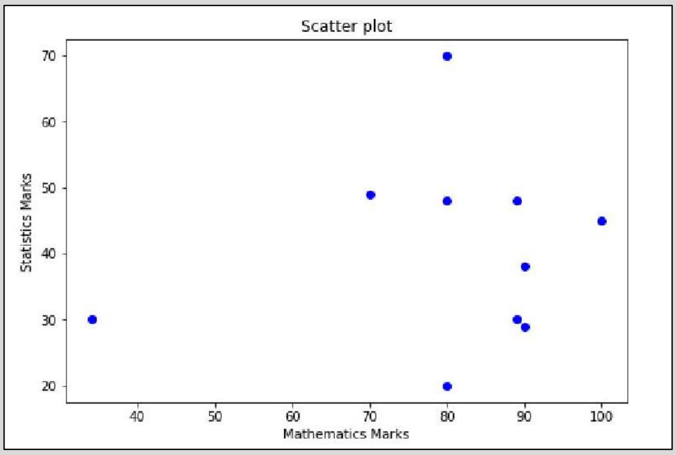
5.8 Scatter Plot

The observations (x,y) that are plotted using Cartesian coordinates is called as a Scatter plot. To draw scatter plot, *scatter*() function used with the arguments as values of x array, y array. An optional argument color is used the the following example to set the point color as blue. Marks of Mathematics are considered as x array values and Marks of Statistics are considered as y array values.

```
import matplotlib.pyplot as plt
Mathematics= [89, 90, 70, 89, 80, 80, 90, 100, 80, 34]
Statistics= [30, 29, 49, 48, 70, 48, 38, 45, 20, 30]
fig=plt.figure()
ax=fig.add_axes([0,0,1,1])
```

ISBN 978-93-80788-93-7

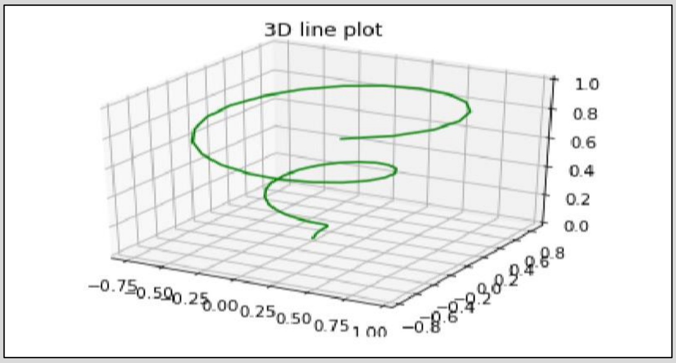
```
ax.scatter(Mathematics, Statistics, color='b')
ax.set_xlabel('Mathematics Marks')
ax.set_ylabel('Statistics Marks')
ax.set_title('Scatter plot')
plt.show()
```



5.9 3D Line Plot

For 3D plotting we import *mplot3d* module from *mplot3d toolkit. axes()* function has an argument as projection='3d' which enables 3 dimensional setup for plotting. *plot3d()* function is used with three parameters as x, y and z coordinates. One can use additional parameter for pattern and color of the line. In the following example, (sin 15z, cos 15z, z) are plotted where z varies from 0 to 10.

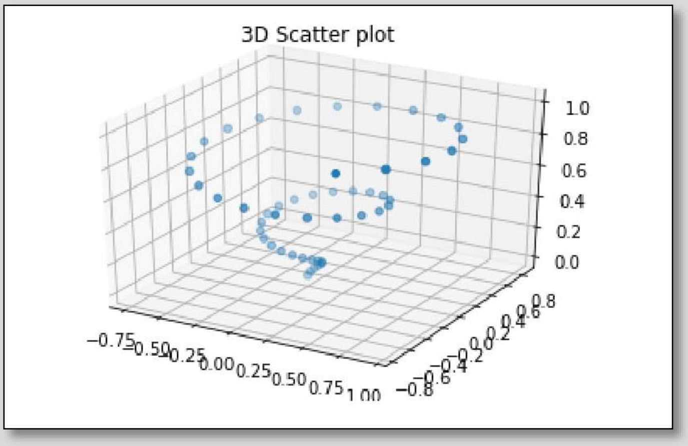
```
from mpl_toolkits import mplot3d
import numpy as np
import matplotlib.pyplot as plt
fig= plt.figure()
ax= plt.axes(projection='3d’)
z= np.linspace(0, 1, 50)
x = z * np.sin(15 * z)
y = z * np.cos(15 * z)
ax.plot3D(x, y, z, 'g')
ax.set_title('3D line plot')
plt.show()
```



5.10 3D Scatter Plot

Function *scatter()* is used to draw the scatter plot. arguments are x, yand z coordinate values. In the following example, points (sin 15z, cos 15z, z) are plotted where z varies from 0 to 10.

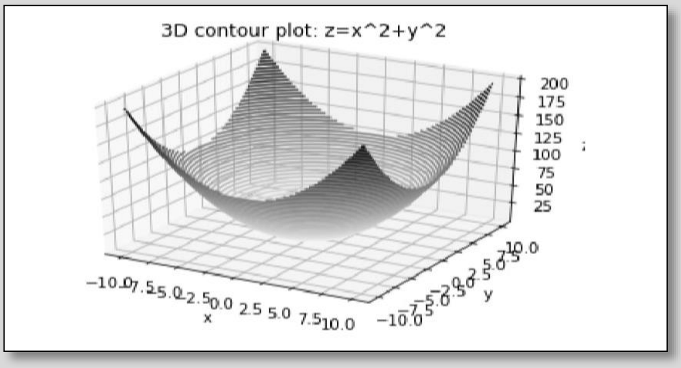
```
from mpl_toolkits import mplot3d
import numpy as np
import matplotlib.pyplot as plt
fig= plt.figure()
ax= plt.axes(projection='3d')
z = np.linspace(0, 1, 50)
x = z * np.sin(15 * z)
y = z * np.cos(15 * z)
ax.scatter(x, y, z, 'g')
ax.set_title('3D Scatter plot')
plt.show()
```



5.11 Contour Plot

To draw contour plot, we use function *contour3d()*. The input required for this function is 2 dimensional grid. So X,Y in the following example defines grid. cmap arguments represents color mapping. Binary stands for black and white color mapping. Contour plot of $z = x^2 + y^2$ is plotted using the following code.

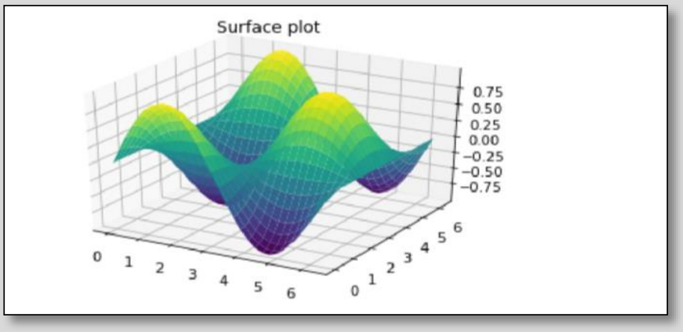
```
from mpl_toolkits import mplot3d
import numpy as np
import matplotlib.pyplot as plt
def f(x, y):
     return x**2+y**2
x = np.linspace(-10, 10, 20)
y = np.linspace(-10, 10, 20)
X, Y = np.meshgrid(x, y)
Z = f(X, Y)
fig= plt.figure()
ax= plt.axes(projection='3d')
ax.contour3D(X, Y, Z, 50, cmap='binary')
ax.set_xlabel('x')
ax.set_ylabel('y')
ax.set_zlabel('z')
ax.set_title('3D contour plot: z=x^2+y^2')
plt.show()
```



5.12 Surface Plot

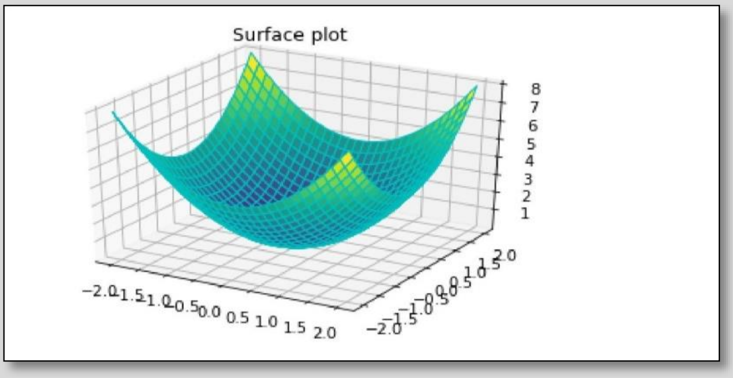
To plot surface plot, we use *plot_surface()* function with arguments : x, y, z, cmap. edgecolor is an optional argument. Following example plots the surface $z = \sin x . \cos y$. Here colour theme viridis is used. There are many other themes like 'BrBG', 'twilight_shifted', 'jet' etc. No edgecolor is set.

```
from mpl_toolkits import mplot3d
import numpy as np
import math
import matplotlib.pyplot as plt
x = np.outer(np.linspace(0, math.pi*2, 30), np.ones(30))
y = x.copy().T    # transpose
z = np.sin(x)*np.cos(y)
fig = plt.figure()
ax= plt.axes(projection='3d')
ax.plot_surface(x, y, z,cmap='viridis', edgecolor='none')
ax.set_title('Surface plot')
plt.show()
```



The following example is a surface plot that represents equation $z = x^2 + y^2$. Here edgecolor is defines as cyan, so the surface grids are drawn with cyan colored lines.

```
from mpl_toolkits import mplot3d
import numpy as np
import math
import matplotlib.pyplot as plt
x = np.outer(np.linspace(-2, 2, 30), np.ones(30))
y = x.copy().T                                              # transpose
z = x**2+y**2
fig= plt.figure()
ax= plt.axes(projection='3d')
ax.plot_surface(x, y, z,cmap='viridis', edgecolor='c')
ax.set_title('Surface plot')
plt.show()
```



5.13 References

1. *Matplotlib.* (2016). Tutorials Point.
2. Zhong, Y. (2019, July 18). *Beyond data scientist: 3d plots in Python with examples*. Retrieved from https://medium.com: https://medium.com/@yzhong.cs/beyond-data-scientist-3d-plots-in-python-with-examples-2a8bd7aa654b

Chapter 6
Factorization of a Polynomial over a Finite Field

Mrs. Mrunal Hardikar*, Assistant Professor, Department of Mathematics,*
K. C. College

6.1 Introduction

Finding roots of polynomials or "solving algebraic equations" was the old problem in mathematics. With the developments in abstract algebra, innovative techniques have developed for finding roots of polynomials over a field. Factorizing large integers is a hard problem. In fact, some techniques in cryptography are based on this fact. However, there is a beautiful algorithm for factorization of a polynomial over a finite field. This Berlekamp's algorithm is based on some basic techniques in linear algebra and field theory. We will start with some necessary results in finite fields.

A finite field $\mathcal{F}$ has characteristic p for some prime p, and hence it is a finite dimensional vector space over $\mathcal{F}_p$(a finite field with p elements, isomorphic to $\mathbb{Z}/p\mathbb{Z}$). If the dimension is n, then $\mathcal{F}$ has precisely p^n elements. We will denote this field by $\mathcal{F}_q$ where $q = p^n$.

Let $f(x)$ be a polynomial of degree n over $\mathcal{F}_q$. Let $f(x) = e_1 e_2 \cdots e_k$ be a factorization of $f(x)$ in to the prime factors, i.e. each e_i is irreducible over $\mathcal{F}_q$ and deg $(e_i) \geq 1$. We may assume that all e_i's are distinct i.e. $f(x)$ is square free. This is not the restriction but if $f(x)$ involves square, we can make it square-free. We will apply the algorithm for a square-free polynomial and find out k and all $e_i's$.

6.2 Square-Free Procedure

A polynomial $f(x)$ over a field $\mathcal{F}$ has a multiple zero in some extension of $\mathcal{F}$ if and only if $f(x)$ and $f'(x)$ have a common factor of positive degree in $\mathcal{F}[x]$ where $f'(x)$ denotes the formal derivative of $f(x)$. We will use this result to make $f(x)$ square-free as follows:

1. Step 1: Compute $g(x) = \gcd\big(f(x), f'(x)\big)$
2. Step 2: If $g(x)$ has degree zero then $f(x)$ is square free and ready for Berlekamp.
3. Step 3: If $f'(x) = 0$, then $f(x)$ is a perfect p^{th} power, where p is the characteristic of $\mathcal{F}_q$.
4. Else if deg $(g(x)) \geq 1$ and $f'(x) \neq 0$ then recursively apply square-free procedure to g and f/g.

ISBN 978-93-80788-93-7

Now given polynomial is square free and ready for Berlekamp algorithm.

6.3 Preliminaries from Field Theory

Let $A_f = \mathcal{F}_q[x]/ < f >$. This is a vector space over $\mathcal{F}_q$ with dimension n.
Let $B_f = \{h \in A_f : h^q \equiv h \ \ (mod\, f)\}$
Define $Q_f : A_f \to A_f$ by $a(x) \mapsto a(x)^q$ for each $a(x)$ in A_f.
Since $\mathcal{F}_q$is a field of characteristic p and $q = p^n$, $(a(x) + b(x))^q = (a(x))^q + (b(x))^q$ and hence Q_f is a linear transformation over A_f.
Claim: $\boldsymbol{B_f = ker\ (Q_f - I)}$
If $a(x) \in B_f$, then $\left(Q_f - I\right)\left(a(x)\right) = Q_f(a(x)) - I\left(a(x)\right) = \left(a(x)\right)^q - a(x) \equiv 0\ (mod\, f)$
Hence $B_f \subseteq \ker(Q_f - I)$.
Conversely if $b(x) \in \ker\ (Q_f - I)$, then $(Q_f - I)b(x) \equiv 0\ (mod\, f)$ and hence
$(b(x))^q \equiv b(x)\ \ (mod\, f)$ i.e. $b(x) \in B_f$. Hence the claim.
By Chinese reminder theorem (CRT),

$$A_f \cong \mathcal{F}_q[x]/ < e_1 > \times\ \mathcal{F}_q[x]/ < e_2 > \times \cdots \times \mathcal{F}_q[x]/ < e_k >$$

6.4 Preliminaries from Linear Algebra

Let $\{1, x, x^2, \cdots, x^{n-1}\}$ be a standard basis for A_f. Let M be a matrix of $Q_f w.r.t.$ this standard basis.
We can compute basis of $B_f = \ker\ (Q_f - I)$ by applying row reduction on $M - I$.
Claim: Nullity of $\boldsymbol{Q_f - I}$ is the number of irreducible factors of $\boldsymbol{f(x)}$.
By hypothesis, $f(x) = e_1 e_2 \cdots e_k$
By CRT, $A_f \cong \mathcal{F}_q[x]/ < e_1 > \times\ \mathcal{F}_q[x]/ < e_2 > \times \cdots \times \mathcal{F}_q[x]/ < e_k >$
Hence for each $h \in B_f$, CRT maps sends h to $(r_1, r_2, \cdots, r_k)$
Since $h \in B_f$, $0 \equiv h^q - h \equiv (r_1^q - r_1, r_2^q - r_2, \cdots, r_k^q - r_k)$
Therefore, $r_i^q = r_i$, for each i. So, $r_i \in \mathcal{F}_q$ (Again by a result in finite field: $\mathcal{F}_q$is a spitting field of a polynomial $x^q - x$).
Hence, $B_f = \ker(Q_f - I) \cong {\mathcal{F}_q}^k$. i.e. $\dim\left(B_f\right) = k$.
Therefore, Nullity of $Q_f - I$ is the number of irreducible factors of $f(x)$.
That means, now we can find number k by row reduction on $M - I$. It remains to find the actual irreducible factors.
For each $h \in B_f$, $f(x) = \prod_{c \in \mathcal{F}_q}[\gcd(f, h - c)]$. (Note that, LHS divides RHS and vice versa)
Hence for each $h \in B_f$ with $\deg h > 0$, there exists $c \in \mathcal{F}_q$ such that $g = \gcd\ (f, h - c) \neq 1$.
Such g is a non-trivial factor of f.

6.5 Worked Example

Let $f(x) = x^5 + x^4 + 1$ be a square-free polynomial over$\mathbb{Z}/2\mathbb{Z}$. We need to find a polynomial $h(x)$ such that $h^q \equiv h \ (mod\ f)$.
We can represent given polynomial by 110001. (Listing the coefficients in order)
Now we can represent x^0, x^2 etc as follows

$$x^0 \equiv 00001\ (mod\ f)$$
$$x^2 \equiv 00100\ (mod\ f)$$
$$x^4 \equiv 10000\ (mod\ f)$$
$$x^6 \equiv 10011\ (mod\ f)$$
$$x^8 \equiv 11111\ (mod\ f)$$

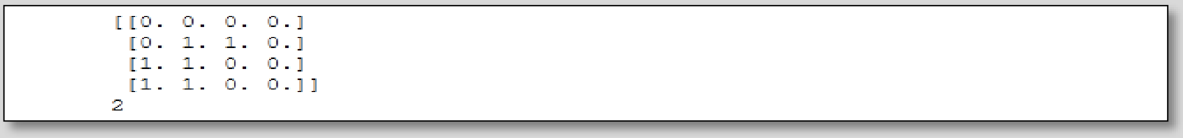
Let $h = a_4x^4 + a_3x^3 + \ a_2x^2 + a_1x + a_0$.
$h^2 = a_4x^8 + a_3x^6 + \ a_2x^4 + a_1x^2 + a_0$, over the field of characteristic 2 .

$$\begin{bmatrix} a_4 \\ a_3 \\ a_2 \\ a_1 \\ a_0 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 \end{bmatrix} \cdot$$

$$\begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 \end{bmatrix} \cdot$$

Now write the matrix M and reduce $(M - I)$ to row echelon form with basis of the kernel (1,1,1,0,0) and (0,0,0,0,1)
Hence h_1 is 11100 and h_2 is 00001
gcd (f, h_1) is 111 and gcd $(f, h_1 + 1)$ is 1011
Hence required factorization is $x^5 + x^4 + 1 = (x^2 + x + 1)(x^3 + x + 1)$ over $\mathbb{Z}/2\mathbb{Z}$.
[Note that $110001 = 111 \cdot 1011$]
Now the Python program to find such factorization will be as follows:

```
from nummpy.polynomial import Polynomial as P
import numpy as np
from numpy.linalg import matrix_rank

import fractions
p=np.array([1,0,1,1,1])

p1=np.array([1])
p2=np.array([1,O,O])
p3=np.array([1,0,0,0,0])
p4=np.array([1,0,0,0,0,0,0])
al=np.abaolute(np.polydiv(p1,p)[1])[::-1].tolist()
a2=np.abaolute(np.polydiv(p2,p)[1])[::-1].tolist()
a3=np.abaolute(np.polydiv(p3.p)[1])[::-1].tolist()
a4=np.abaolute(np.polydiv(p4.p)[1])[::-1].tolist()

for i in range (len(a4)-len(al)):
      al.append(0)
for i in range (len(a4)-len(a2)):
      a2.append(0)
for i in range (len(a4)-len(a3)):
      a3.append(0)
for i in range (len(a4)-len(a4)):
      a4.append(0)
x=np.array([a1,a2,a3,a4])
y=np.absolute(x-np.array([[1,0,0,0],[0,1,0,0],[0,0,1,0],[0,0,0,1]]))

print (y)
r=matrix_rank(y)
print(r)
```



```
[[0. 0. 0. 0.]
 [0. 1. 1. 0.]
 [1. 1. 0. 0.]
 [1. 1. 0. 0.]]
2
```

6.6 References

1. Abstract Algebra (Second Edition) by David S. Dummit and Richard M. Foote.
2. The Berlekamp Algorithm by John Kerl, 2009 Integration Workshop.

ISBN 978-93-80788-93-7

Chapter 7
Descriptive Statistics

***Mr. Shubham Niphadkar**, Assistant Professor, Department of Statistics,*
K. C. College

7.1 Introduction

Descriptive Statistics deals with description and understanding of data and its features with the help of short summaries about the sample and measures of the data. The basic two types of descriptive statistics are:

- Measures of central tendency
- Measures of dispersion

7.2 Measures of Central Tendency

Central representative value of the entire data is referred to as measure of central tendency. The various measures of central tendency are as follows:

1. **Mathematical Averages:**

a) Arithmetic mean
b) Geometric mean
c) Harmonic mean

2. **Positional Averages:**

a) Partition Values
- Median
- Quartiles
- Deciles
- Percentiles

b) Mode

For Raw Data:
x_i is the value of i^th observation, i=1,2, …, n and n is the size of the data.

Mathematical Averages

Arithmetic Mean (A.M.)	**Geometric Mean (G.M.)**	**Harmonic Mean (H.M.)**
$A.M. = \frac{\sum_{i=1}^{n} x_i}{n}$	$G.M. = \left(\prod_{i=1}^{n} x_i\right)^{1/n}$	$H.M. = \frac{n}{\sum_{i=1}^{n} \frac{1}{x_i}}$

Positional Averages

ISBN 978-93-80788-93-7

We will assume that the data is arranged in ascending or descending order.

Median

If size of the data, N, is odd, $Median = Value\ of\ \left(\frac{N+1}{2}\right)^{th}\ observation$

If size of the data, N, is even, $Median = \frac{\left(Value\ of\ \left(\frac{N}{2}\right)^{th} observation + Value\ of\ \left(\frac{N}{2}+1\right)^{th} observation\right)}{2}$

Quartiles	Deciles	Percentiles
$Value\ of\ i\left(\frac{N}{4}\right)^{th}\ observation,$ $i = 1, 2, 3$	$Value\ of\ i\left(\frac{N}{10}\right)^{th}\ observation,$ $i = 1, 2, \ldots, 9$	$Value\ of\ i\left(\frac{N}{100}\right)^{th}\ observation,$ $i = 1, 2, \ldots, 99$

Mode: The value which occur maximum number of times is the mode.

For ungrouped frequency distribution:

x_i is the ith observation and f_i is the frequency of ith observation. $N = \sum_{i=1}^{n} f_i$.

Mathematical Averages

Arithmetic Mean (A.M.)	Geometric Mean (G.M.)	Harmonic Mean (H.M.)
$A.M. = \frac{\sum_{i=1}^{n} f_i x_i}{\sum_{i=1}^{n} f_i}$	$G.M. = \left(\prod_{i=1}^{n} x_i^{f_i}\right)^{1/N}$	$H.M. = \frac{N}{\sum_{i=1}^{n} \frac{f_i}{x_i}}$

Positional Averages

Median

If N is odd, $Median = Value\ of\ \left(\frac{N+1}{2}\right)^{th}\ observation$

If N is even, $Median = \frac{\left(Value\ of\ \left(\frac{N}{2}\right)^{th} observation + Value\ of\ \left(\frac{N}{2}+1\right)^{th} observation\right)}{2}$

Quartiles	Deciles	Percentiles
$Value\ of\ i\left(\frac{N}{4}\right)^{th}\ observation,$ $i = 1, 2, 3$	$Value\ of\ i\left(\frac{N}{10}\right)^{th}\ observation,$ $i = 1, 2, \ldots, 9$	$Value\ of\ i\left(\frac{N}{100}\right)^{th}\ observation,$ $i = 1, 2, \ldots, 99$

Mode: The value with maximum frequency is the mode.

For grouped frequency distribution:

x_i is the class mark of ith class, f_i is the frequency of ith class. l_1 is the lower boundary of the class, l_2 is the upper boundary of the class, f is the frequency of the class and cf is the cumulative frequency less than l_1.

Mathematical Averages

Arithmetic Mean (A.M.)	Geometric Mean (G.M.)	Harmonic Mean (H.M.)

$A.M.=\frac{\sum_{i=1}^{n} f_i x_i}{\sum_{i=1}^{n} f_i}$	$G.M.=\left(\prod_{i=1}^{n} {x_i}^{f_i}\right)^{1/N}$	$H.M.=\frac{N}{\sum_{i=1}^{n}\frac{f_i}{x_i}}$

Positional Averages

For calculating median, quartiles, deciles, percentiles and mode, we will be dealing with, median class, Q_i class, D_i class, P_i class and modal class respectively.

Median: $Median = l_1 + (l_2 - l_1)\left(\frac{N}{2} - cf\right)/f$

Quartiles	Deciles	Percentiles
$l_1 + (l_2 - l_1)\left(\frac{iN}{4} - cf\right)/f,$ $i = 1, 2, 3$	$l_1 + (l_2 - l_1)\left(\frac{iN}{10} - cf\right)/f$ $i = 1, 2, \dots, 9$	$l_1 + (l_2 - l_1)\left(\frac{iN}{100} - cf\right)/f$ $i = 1, 2, \dots, 99$

Mode: $Mode = l_1 + (l_2 - l_1)\,(d_1)/(d_1 + d_2)$

$d_1 = f_m - f_0$, $d_2 = f_m - f_1$, f_m is the frequency of the modal class, f_0 is the frequency of the class preceding to the modal class and f_1 is the frequency of the class succeeding to the modal class.

7.3 Measures of Dispersion

The representative value of the data which is used to represent the spread or scatterness of data is referred to as measure of dispersion. The various measures of dispersion are as follow:

1. Range
2. Quartile Deviation
3. Mean Deviation about 'a'
4. Variance
5. Standard Deviation
6. Skewness and Kurtosis

Range:

The range for a data is defined as the difference between maximum value and minimum value. The corresponding relative measure of dispersion is coefficient of range.

$Range = Max\,Value - Min\,Value$	$Coefficient\ of\ range = \frac{Max\,Value - Min\,Value}{Max\,Value + Min\,Value}$

Quartile Deviation (Q.D.):

The quartile deviation is half of the range of values within the quartiles. The corresponding relative measure of dispersion is coefficient of quartile deviation.

$Q.D.= (Q_3 - Q_1)/2$	$Coefficient\ of\ Q.D.=\frac{Q_3 - Q_1}{Q_3 + Q_1}$

ISBN 978-93-80788-93-7

Mean Absolute Deviation (M.A.D.):

For raw data	For ungrouped and grouped frequency distribution	Relative measure:
$M.A.D. about\ 'a' = \frac{\sum_{i=1}^{n} \lvert x_i - a \rvert}{n}$	$M.A.D. about\ 'a' = \frac{\sum_{i=1}^{n} f_i \lvert x_i - a \rvert}{\sum_{i=1}^{n} f_i}$	$Coefficient\ of\ M.A.D. = \frac{M.A.D. about\ 'a'}{a}$

Variance:

For raw data	For ungrouped and grouped frequency distribution
$\sigma^2 = \frac{\sum_{i=1}^{n}(x_i - \bar{x})^2}{n}$	$\sigma^2 = \frac{\sum_{i=1}^{n} f_i (x_i - \bar{x})^2}{\sum_{i=1}^{n} f_i}$

If the denominator in the above cases is $'n - 1'$ or $'\sum_{i=1}^{n} f_i - 1'$, then it is referred to as sample variance.

Standard Deviation (S.D.):
The square root of variance is called as standard deviation. The corresponding relative measure of dispersion is coefficient of variation. $Coefficient\ of\ Variation = \frac{S.D.}{\bar{x}} \times 100$

Skewness and Kurtosis:
Lack of symmetry is called as skewness. The corresponding relative measure of dispersion is coefficient of skewness.

Karl Pearson's	**Bowley's**	**Based on Moments ($\boldsymbol{\beta_1}$)**
$Mean - Mode$	$Q_3 + Q_1 - 2Q_2$	μ_3^2/μ_2^3

Coefficient of Skewness

Karl Pearson's ($\boldsymbol{SK_P}$)	**Bowley's ($\boldsymbol{SK_B}$)**	**Based on Moments ($\boldsymbol{\gamma_1}$)**
$\frac{(Mean - Mode)}{S.D.}$	$(Q_3 + Q_1 - 2Q_2)/(Q_3 - Q_1)$	$\pm\sqrt{\beta_1}$

If $\gamma_1 > 0$, then the data is positively skewed. If $\gamma_1 < 0$, then it is negatively skewed. If $\gamma_1 = 0$, then the data is symmetric.

Kurtosis gives the idea about the flatness or peakedness. It is given by,

$$\boldsymbol{\beta_2 = \frac{\mu_4}{\mu_2^2}, \gamma_2 = \beta_2 - 3}$$

If $\beta_2 > 3\ or\ \gamma_2 > 0$, then the frequency curve is leptokurtic. If $\beta_2 < 3\ or\ \gamma_2 < 0$, then the frequency curve is platykurtic. If $\beta_2 = 3\ or\ \gamma_2 = 0$, then the frequency curve is mesokurtic.

ISBN 978-93-80788-93-7

7.4 Examples

1. For the following data representing the average rainfall in every month in the year 2017, find arithmetic mean, geometric mean, harmonic mean. Also find, median, 1st quartile, 4th decile, 87th percentile and mode. Also find range, quartile deviation, mean deviation about median, mean deviation about mean, variance, standard deviation, sample variance, sample standard deviation and their coefficients.

Month	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sept	Oct	Nov	Dec
Rainfall (in mm)	10	10	10	10	11	560	640	520	320	70	20	15

```
In [1]:import numpy as np
        import scipy.stats as stats
        import statistics
        rainfall=[10,10,10,11,560,640,520,320,70,20,15]
        #arithmetic mean
        am=np.average (rainfall)
        print ("Arithmetic Mean :", am)

Arithmetic Mean: 183.0
```

```
In [2]:#Geometric Mean
       gm=stats.gmean (rainfall)
       print ("Geometric Mean:",gm)

Geometric Mean: 47.671751500797924
```

```
In [3]:#Harmonic Mean
      hm=stats.hmean (rainfall)
      print("Harmonic Mean:",hm)
Harmonic Mean: 19.039828944815717
```

```
In [4]: #Median
      median=statistics.median (rainfall)
      print ("Median:",median)

Median:17.5
```

```
In[5] :        # First Quartile
               q1=np.quantile(rainfall,0.25)
               print("First Quartile:",q1)
               #Fourth Quartile
               d4=np.quantile (rainfall,0.4)
               print("Fourth Decile:",d4)
               #87 th Percentile
               p87=np.quantile(rainfall,0.87)
               print ("87 Percentile:", p87)
               First Quartile:10.0
               Fourth Decile :12.600000000000001
               87th Percentile:542.8
```

```
In [6] :#Mode
        mode=statistics.mode(rainfall)
        print ("Mode:",mode)
        # For multiple mode use statistics.multimode() in phyton 3.8

        Mode: 10
```

```
In [7] :import pandas as pd
      #Range
      r=max(rainfall)-min(rainfall)
      print("Range:",r)
      #Coefficient of Quartile Deviation
      print ("Coefficient of Range :",crange)

      Range: 630
      Coefficient Of Range : 0.9692307692307692
```

```
In [8] : #Quart:iie Deviation
      q1=np.quantile(rainfall ,0.25)
      q2=np.quantile(rainfall ,0.5)
      q3=np.quantile(rainfall ,0.75)
      qd-(q3-q1)/2
      pr1nt("Quartile Deviation:",  qd)
      #Coefficient of Quartiie Deviation
      cqd=(q3-q1)/(q3+q1)
      print("coefficient of Quartile Deviation:"cqd)

      Quartile Deviation: 180.0
      Coefficient of Quartile Deviation: 0.9473684210526315
```

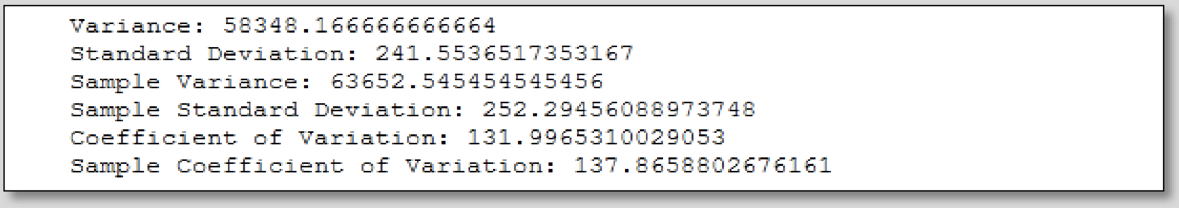
```
In [9]: #Mean Absolute Deviation about Median
      mad_median=np.average(np.absolute(no.substract(rainfall,statistics.m
      edian(rainfall))))
      print ("Mean Absolute Deviation about Median:",mad_median)
      #Coefficient of Mean Absolute Deviation about Median
      cmad_median=mad_median/statistics.median(rainfall)
      print    ("Coefficient    of    Mean    Absolute    Deviation    about
      Median:",cmad_median)

      Mean Absolute Deviation about Median:172.0
      Coefficient of Mean Absolute Deviation about Median:9.82857142857143
```

```
In [10] :   #Mean Absolute Deviation about mean
      series=pd.Series (rainfall)
      print ("Mean Absolute Deviation about Mean:",mad_mean)
      #Coefficient of Mean Absolte Deviation about Mean
      cmad_mean=mad_mean/np.average(rainfall)
      print("Coefficient   of   Mean   Absolute   Deviation   about   Mean:",
      cmad_mean
```

ISBN 978-93-80788-93-7

```
      Mean Absolute Deviation about Mean :      218.0
      Coefficient     of     Mean     Absolute     Deviation     about     Mean:
      1.1912568306010929
```

```
In [11] : #Population variance
      pvar=statistics.pvariance (rainfall,np.average (rainfall))
      print("Variance:" , pvar)
      #Population standard deviation
      psd=statistics.pstdev(rainfall,np.average(rainfall))
      print("Standard Deviation:",psd)
      #Sample Variance.
      svar=statistics.variance(rainfall,np.average(rainfall))
      print("Standard Variance:", svar)
      #Sample standard deviation
      psd=statistics.variance (rainfall,np.average(rainfall))
      print("Sample Standard Deviation:", ssd)
      #Population coefficient of variation
      pcv=psd/np.average(rainfall)*100
      print("coefficient of variation:", pcv)
      #Sample coefficient of variation
      scv=ssd/np.average(rainfall)*100
      print("Sample Coefficient of Variation:,' scv)
```



```
    Variance: 58348.16666666664
    Standard Deviation: 241.5536517353167
    Sample Variance: 63652.545454545456
    Sample Standard Deviation: 252.29456088973748
    Coefficient of Variation: 131.9965310029053
    Sample Coefficient of Variation: 137.8658802676161
```

2. For the following data representing the distribution of weight of 50 students in a class. Find arithmetic mean, geometric mean, harmonic mean. Also find, median, 2nd quartile, 8th decile, 5th percentile and mode. Also find range, quartile deviation, mean deviation about median, mean deviation about mean, variance, standard deviation, sample variance, sample standard deviation and their coefficients.

Weight (in kg)	42	46	50	54	58	62	66
No. of Students	3	7	14	11	8	5	2

```
In [1]:import statistics
      import pandas as pd
      import scipy. stats as stats
      import numpy as np
      data=        {'Weight':        [42,46,50,54,58,62,66],         'Numbar':
      [3,7,14,11,8,5,2]}
      df=pd.DataFrame (data)
      a=list(np.array(df['Weight']).repeat(df['Numbar']))
```

```
In [2]:#Arithmetic Mean
```

```
      am=up,average(a)
      print ( "Arithmetic Mean:" ,am)

      Arithmetic Mean: 41.5
```

```
In [3]: #Geometric Mean
      gm=stats.gmean(a)
      print ( "Geometric Mean:" ,gm)

      Geometric Mean: 52.96
In [4]:#Harmonic Mean
      hm=stats.hmean(a)
      print ( "Harmonic Mean:" ,hm)
      Harmonic Mean: 52.62458543450998
```

```
In [5]: #Median
      median=statistics.median(a)
      print ( "Median:" ,median)

      Median: 54.0
In [6]: # Second Quartile
      q2=np.quantile(a,0.5)
      print(“Second Quantile:”,q2)
      #Eighth Decile
      d8=np.quantile(a,0.8)
      print(“Eight decile:”,d8)
      #5th percentile
      p5=np.quantile(a,0.05)
      print(“5th Percentile:”,p5)
      Second Quantile: 54.0
      Eight Quantile: 58.0
      5th percentile: 43.800000000000000004
```

```
In [7]: #Mode
      mode=statistics.mode(a)
      print("Mode:", mode)
      #For multiple mode use statistics.multimode() in Phython 3.8

      Mode: 50
```

```
In [8]: #Range
      r=max(a) -min(a)
      print("Range:", r)
      #Coefficient of Range
      crange=(max(a)- min(a) ) / (max(a) +min (a) )
      print ("Coefficient of Range:", crange)

      Range: 24
      Coefficient of Range: 0.22222222222222
```

```
In [9] : #Quartile Deviation
      q1=np.quantile (a,0.25)
```

ISBN 978-93-80788-93-7

```
      q2=np.quantile (a,0.5)
      q3=np.quantile(a,0.75)
      print("Quartile Deviation:" ,qd)
      #Coefficient of Quartile Deviation
      cqd=(q3-q1)/(q3+q1)
      print("Coefficient of Quartile Deviation:", cqd)

      Quartile Deviation:4.0
      Coefficient of Quartile Deviation: 0.07407407407407407
```

```
In [10] :   #Mean Absolute Deviation about Median
      mad_median=ap.average(np.absolute(np.subtract(a,median(a)))
      #Coefficient of Mean Absolute Deviation about Median
      print ( "Mean Absolute Deviation about Median: ", mad_ median)
      cmad _ median=mad _ median/statistics.median (a)
      print ("Coefficient of Mean Absolute Deviation about Median: ",
      cmad_median)

      Mean Absolute Deviation about Median: 6.142857142857143
      Coefficient of Mean Absolute Deviation about Median:
      0.09037037037037036
```

```
In [11] :   #Mean Absolute deviation about Mean
      series=pd.Series(a)
      mad_mean=series.mad ( )
      print("Mean Absolute Deviation about mean:", mad_mean
      cmad_mean=mad_mean/np.average (a)
      print("Coefficient of Mean Absolute Deviation about Mean:" ,
      cmad_mean)

      Mean Absolute Deviation about Mean:4.921599999999999
      Coefficient of Mean Absolute Deviation about Mean:
      0.9293051359516614
```

```
In [12] :   #Population variance
      pvar=statistics.pvariance(a,np.average(a) )
      print("Variance:", pvar)
      #Population standard deviation
      psd=statistics.pstdev(a,np.average(a) )
      print ("Standard Deviation:" , psd)
      #Sample Variance
      svar=statistics.variance (a,np.average(a) )
      print ("Sample Variance:", svar)
      #Sample standard deviation
      ssd=statistics.variance(a,np.average (a) )
      print ("Sample Standard Deviation:",ssd )
      #Population coefficient of variation
      pcv=psd/np.average(a) *100
      print ("Sample Coeffcient of Variation:", scv )
```

```
      Variance: 35.7184
      Standard Deviation : 5.97648726598233
      Sample Variance: 36.44734693877551
      Sample Standard Deviation : 6.0371663815797573
      Coefficient of Variation:11.28490797973184145
      Sample Coefficient of Variation: 11.399478504149496
```

3. Consider the following data:

Age (in years)	No. of employees (f)
20-25	1
25-30	3
30-35	6
35-40	10
40-45	14
45-50	9
50-55	5
55-60	2

Find arithmetic mean, geometric mean, harmonic mean. Also find, median, 1st quartile, 6th decile, 25th percentile and mode. Also find range, quartile deviation, mean deviation about median, mean deviation about mean, variance, standard deviation, sample variance, sample standard deviation and their coefficients. Also calculate measure and coefficient of skewness and kurtosis.

```
In [1] :import statistic
      import panda as pd
      import scipy.stats as stats
      import numpy as np
      data={'Age   in   years']:   ['20-25,25-30,30-35,35-40,40-45,45-50,50-
      55,55-60'], 'f':[1,3,6,10,14,9,5,2]}
      df=pd.DataFrame(data)
      #Lover boundaries
      l1=list(range(20,60,5) )
      #Upper Boundaries
      u1=list(range(25,65,5) )
      #Class Marks
      cm=np.divide(np.add(l1,u1),2)
      #Cumulative Frequencies
      series=pd.series(df['f'])
      cmf=series.cumsum()
      a=list(np.array(cm).repeat(df['f']))
```

```
In [2]:#Arithmetic Mean
      am=np.average(a)
      print ( "Arithmetic Mean:" ,am)

      Arithmetic Mean: 41.5
```

```
In [3]:#Geometric Mean
      gm=stats.gmean(a)
      print ( "Geometric Mean:" ,gm)
      Geometric Mean: 40.71854614513001
```

```
In [4]:#Harmonic Mean
      hm=stats.hmean(a)
      print ( "Harmonic Mean:" ,hm)

      Harmonic Mean: 39.88120004143535
```

```
In [5]:#Median
      med=sum(df['f'])/2
      med_class=list(cumf).index(min(cumf[cumf>med]))
      median=ll  [med_class]  +  (ul[med_class])  *  (sum(df['f'])/2-cumf[med-
      class-1])/df['f'] [med_class]
      print ( "Median:" ,median)

      Median: 41.785714285714285
```

```
In [6]: #First Quartile
      q=sum(df['f'])/4
      q_class=list(cumf).index(min(cumf[cumf>q]) )
      q1=ll   [q_class]+(ul[q_class]   -   (ll[q_class])   *   (sum(df['f'])/4-
      cumf[q_-class-1])/df['f'] [q_class]
      print("First Quartile:" ,q1

      First Quartile :36.25
```

```
In [7]: #Sixth Decile
      d=6*sum(df['f'])/10
      d_class=list (cumf).index(min(cumf[cumf>d]) )
      d6=ll[d_class]+(ul[d_class]-ll[d_class])*(6*sum(df['f'])/10-
      cumf[d_class-1])df['f'][d_class]
      print ("Sixth Decile:",d6)

      Sixth Decile: 43.5714285714857
```

```
In [8]:#25 Percentile
      p=25*sum(df['f'])/100
      p=class=list(cumf).index(min(cumf[cumf>p]) )
      p25=ll[p_class]+(ul[p_class]-ll[p_class])*(25*sum(df['f'])/100-
      cumf[p_class-1])df['f'][p_class]
      print ("25th Percentile:",p25)

      25th Percentile : 36.25
```

```
In [9]: #Mode
      m=max  (df['f'])
      m_class=list(cumf).index(min(cumf[cumf>p]) )
      d1=df [' f '] [m_class] - df [ ' f '] [m_class-1]
      d2=df [' f '] [m_class] - df [ ' f '] [m_class+1]
      mode=ll[m_class]+(ul[m_class]-ll[m_class])*(d1/(d1+d2) )
      print ("Mode:", mode)
```

ISBN 978-93-80788-93-7

```
      Mode:42.22222222222222
```

```
In [10]: #Range
      a=list(np.array(cm).repeat (df[ ' f ']) )
      r=max(ul)-min(ll))/(max(ul)+min(ll) )
      print ("Range:" , r)
      #Coefficient of Range
      crange=(max(ul)-min(ll))/(max(ul)+min(ll) )
      print("Coefficient of Range:", crange)

      Range: 40
      Coefficient of Range: 0.5
```

```
In [11]:#Quartile Deviation
      q=sum(df[ 'f' ]) /4
      q_class=list(cumf).index(min(cumf(cumf>q]))
      q1=ll[q_class]+(ul[q_class]-ll(q_class])*(sum(df('f')/4-
      cumf[q_class-1])/df['f'][q_class]
      qa=3*sum(df['f']))/4
      qa_class-list(cumf).index(ll(cumf[cumf>qa]))
      q3=ll[qa_class]+(ul[qa      class]-ll(qa      class])*(3*sum(df['f'])/4-
      cumf(qa_class-1))/df['f'][qa_class]
      qd=(q3-q1)/2
      print("Quartile Deviation:", qd)
      #Coefficient of Quartile Deviation
      cqd=(q3-q1)/(q3+q1)
      print("Coefficient of Quartile Deviation:", cqd)

      Quartile Deviation: 5.347222222222221
      Coefficient of Quartile Deviation: 0.12854757929883137
```

```
In [12]: #Mean Absolute Deviation about Median
      mad_median=np.average(np.absolute(np.subtract(a,median)))
      #Coefficient of Mean Absolute Deviation about Median
      print ( "Mean Absolute Deviation about Median: ", mad_ median)
      cmad _ median=mad _ median/median
      print  ("Coefficient  of  Mean  Absolute  Deviation  about  Median:  ",
      cmad_median)

      Mean Absolute Deviation about Median: 6.14285714285714      3
      Coefficient of Mean Absolute Deviation about Median:  0.147008547008547
```

```
In [13]:#Mean Absolute Deviation about Mean
      series1=pd.Series(a)
      mad_mean=series1.mad()
      print("Mean Absolute Deviation about Mean:",mad_mean)
      #Coefficient of Mean .Absolute Deviation about Mean
      cmad_mean=mad_mean/np.average(a)
      print("Coefficient   of   Mean   Absolute   Deviation   about   Mean:",
      cmad_mean)
      Mean Absolute Deviation about Mean: 6.2
      Coefficient    of    Mean    Absolute    Deviation    about    Mean:
      0.1493975903614458
```

```
In [14]: # Population variance
      pvar=statistics.pvariance(a,np.average(a) )
      print("Variance:", pvar)
      #Population standard deviation
      psd=statistics.pstdev(a,np.average(a) )
      print ("Standard Deviation:" , psd)
      #Sample Variance
      svar=statistics.variance (a,np.average(a) )
      print ("Sample Variance:", svar)
      #Sample standard deviation
      ssd=statistics.variance(a,np.average (a) )
      print ("Sample Standard Deviation:",ssd )
      #Population coefficient of variation
      pcv=psd/np.average(a) *100
      print ("Sample Coeffcient of Variation:", scv )
      Variance: 61.0
      Standard Deviation : 7.810249675967590664
      Sample Variance: 62.244897959183675
      Sample Standard Deviation : 7.889543581905186
      Coefficient of Variation : 18.81987873712447
      Sample Coefficient of Variation: 19.01094839447033
```

```
In [15]:from scipy.stats import skew,kurtosis
      #Karl Pearsons measure of skewness
      karl_skew=(am-mode)
      print("Karl Pearsons measure of skewness:", karl_skew)
      #Bowleys measure of skewness
      bowl_skew=q3+q1-2*median
      print("Bowleys measure of skewness:", bowl_skew )

      Karl Pearsons measure of skewness: -0.7222222222222214
      Bowleys measure of skewness: -0.37698412698412653
```

```
In [16]: #Karl Pearsons of skewness based on moments
      b1=pow(skew (a) ,2)
      print("Karl Pearsons coefficient of skewness:", skp)
      #Bowleys coefficient of skewness
      skb=bowl_skew/(q3-q1)
      print("Coefficient of skewness based on moments:",skb)

      Karl Pearsons coefficient of skewness: -0.09247107995153589
      Coefficient of skewness based on moments: -0.035250463821892355
```

```
In [17]: #Measure of skewness based on moments
      b1=pow(skew(a),2)
      print("Measure of skewness based on moments:", b1)
      #Coefficient of skewness based on moments
      c1=skew(a)
      print("coefficent of skewness based on moments:", c1)

      Measure of skewness based on moments: 0.019191033610742744
      Coefficent of skewness based on moments: -0.13853170615690383
```

```
In [18]: #Measure of kurtosis based on moments
      b2=kurtosis(a)+3
      print("Measure of kurtosis based on moments:", b2)
      #Coefficient of kurtosis based on moments
      c2=kurtosis(a)
      print("Coefficent of kurtosis based on moments:", c2)

      Measure of kurtosis based on moments: 2.705993012631013
      Coefficent of kurtosis based on moments: -0.294006987368987
```

7.5 References

1. https://www.tutorialspoint.com/python_pandas/python_pandas_descriptive_statistics.htm
2. https://www.investopedia.com/terms/d/descriptive_statistics.asp
3. https://en.wikipedia.org/wiki/Descriptive_statistics
4. https://statistics.laerd.com/statistical-guides/descriptive-inferential-statistics.php

ISBN 978-93-80788-93-7

Chapter 8
Correlation, Regression and Curve Fitting

Mr. Pritesh Patil, *Assistant Professor, Department of Statistics, Kirti College*

8.1 Correlation and Regression

When bivariate data are typically organized in a graph can be called as scatter plot. A scatter plot has two dimensions, a horizontal dimension (called the x-axis) and a vertical dimension (called the y-axis).

The pattern and direction of the relationship between X and Y can be seen from the scatter plot. The strength of the relationship between two numerical variables depends on how closely the data resemble a certain pattern. Correlation coefficient is used to measure the strength and direction of the linear relationship between two numerical variables X and Y.

Once we find a linear pattern in the scatter plot, and the correlation between the two numerical variables is moderate to strong, we can create an equation that allows you to predict one variable using the other. This equation is called as simple linear regression line. Generally Y is called as the depended variable and X is called as the independent variable.

The simple linear regression is given by, $Y_i = \beta_0 + \beta_1 X_i + e_i$, where, Y is dependent variable, X is independent variable, β_0 is intercept, β_1 is slope and e is error term.

Ordinary Least Squares (OLS) is the most common estimation method for linear models. Ordinary least squares (OLS) regression is a statistical method of analysis that estimates the relationship between one or more independent variables and a dependent variable; the method estimates the relationship by minimizing the sum of the squares in the difference between the observed and predicted values of the dependent variable.

Examples:

1. Plot the Scatter diagram from the following pairs of values. Find the correlation coefficient and covariance between the sales (Rs. Lakhs) and expenses (Rs. Lakhs) from the data given below :

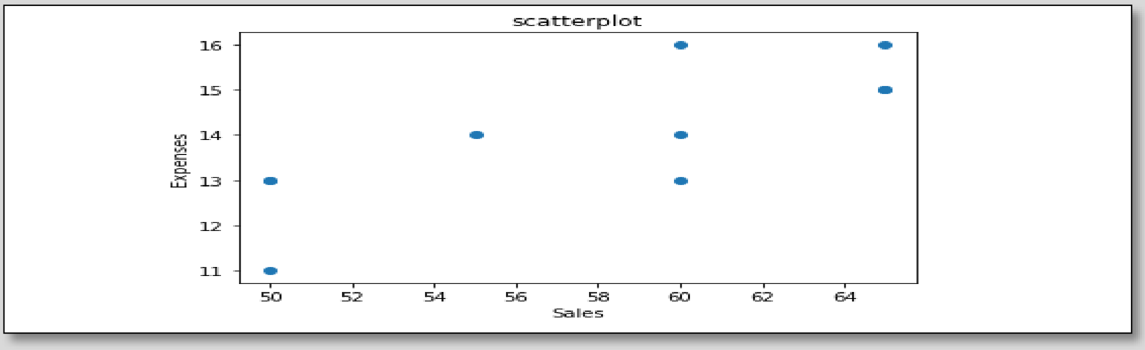
Sales	50	50	55	60	65	65	65	60	60	50
Expenses	11	13	14	16	16	15	15	14	13	13

Also, calculate the regression equation of Sales on expenses.

Solution

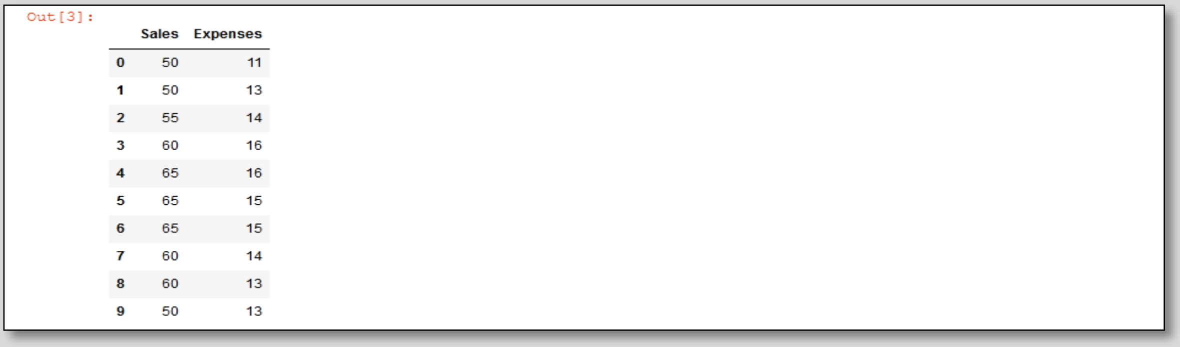
```
In [1]:import numpy as np
```

```
      import pandas as pd
      import matplotlib.pyplot as plt
      x=np.array( [50,50,55,60,65,65,60,60,50] )
      y=np.array( [11,13,14,16,16,15,15,14,13,13] )
      plt.scatter(x,  y)
      plt.title ( 'scatterplot' )
      plt.xlabel (' Sales ')
      plt.ylabel ( ' Expenses ')
      plt.show ()
```



```
In [2]: np.cov(x,y)                                          #covariance
Out [2]: array ([[40.,        7.77777778],
                   [7.77777778,   2.44444444] ] )
```

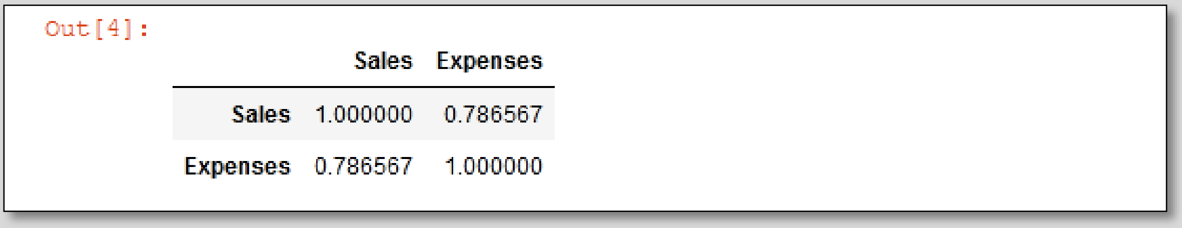
```
In   [3]:data=('Sales':[50,50,55,60,65,65,60,60,50],   'Expenses'   :[11,13,14,
       16,16,15,15,14,13,13])
       df = pd.DataFrame(data)
       df
```



Out[3]:

	Sales	Expenses
0	50	11
1	50	13
2	55	14
3	60	16
4	65	16
5	65	15
6	65	15
7	60	14
8	60	13
9	50	13

```
In [4] : df.corr(method='pearson')
```

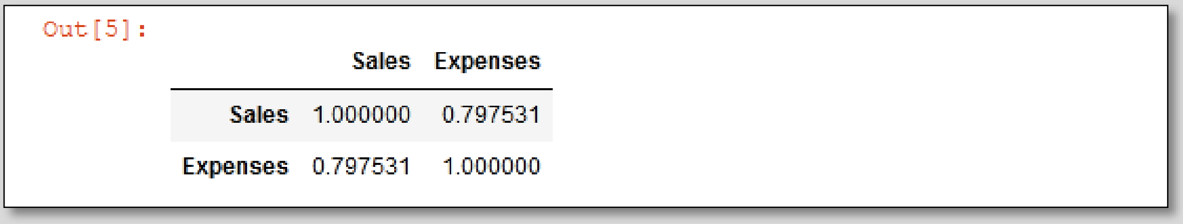


Out[4]:

	Sales	Expenses
Sales	1.000000	0.786567
Expenses	0.786567	1.000000

```
In [5] : df.corr(method='spearman')
```

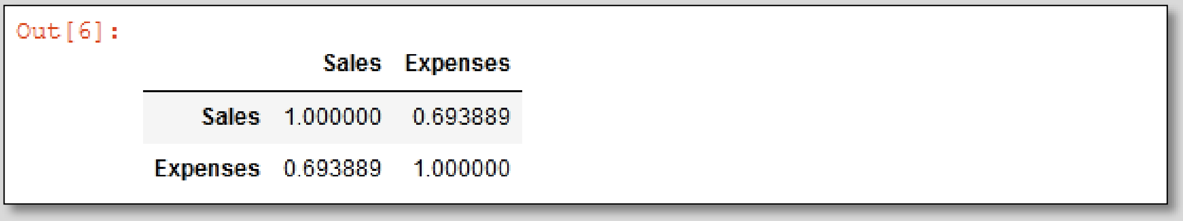
ISBN 978-93-80788-93-7



Out[5]:

	Sales	Expenses
Sales	1.000000	0.797531
Expenses	0.797531	1.000000

```
In [6] : df.corr(method='kendall')
```



Out[6]:

	Sales	Expenses
Sales	1.000000	0.693889
Expenses	0.693889	1.000000

```
In [7]: #To test the signifance of correlation coefficient
      import numpy as np
      import scipy.stats
      x=np.array([50,50,55,60,65,65,65,60,50])
      y=np.array([11,13,14,16,16,15,15,15,14,13,13])
      scipy.stats.pearsonr(x,   y)
Out [7]: (0.7865665062071158,  0.006954016570582233)
```

```
In [8]: scipy.stats.spearmanr(x,y)
Out[8]: SpearmanrResult (coorelation = 0.7975307304350421, pvalue=
        0.005712188346423168
```

```
In[9]: scipy.stats.kendalltau(x,y)
Out[9]:KendalltauResult (coorelation = 0.7975307304350421, pvalue=
       0.012577800357482053)
```

Conclusion

H_0: Sales(x) and Expenses(y) are not significantly correlated

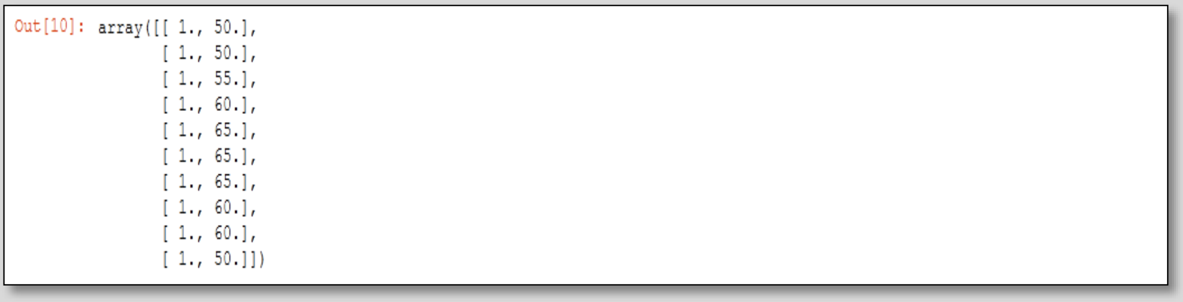
H_1: Sales(x) and Expenses(y) are significantly correlated

As, all the p-value in the test for significance i.e. 0.006954016570582246, 0.005712188346423168 and 0.012577800357482053 are less than 0.05 we can reject H_0 at 5% level of significance and say Sales(x) and Expenses(y) are significantly correlated.

#Regression

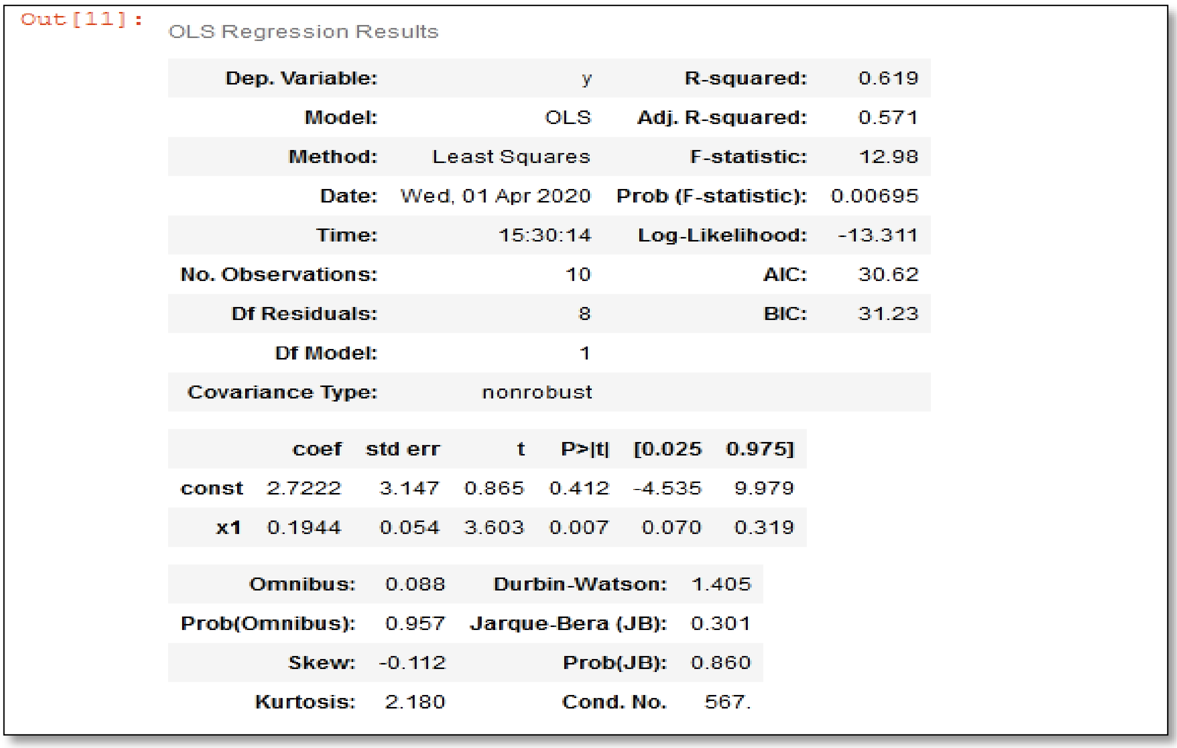
```
In [10]: import statsmodels.api as sm
      x= sm.add_constant(x)
      #to add the coloumn of 1's to the input if we want statsmodels to
      calculate the intercept b0
      x
```

ISBN 978-93-80788-93-7



```
Out[10]: array([[ 1., 50.],
                [ 1., 50.],
                [ 1., 55.],
                [ 1., 60.],
                [ 1., 65.],
                [ 1., 65.],
                [ 1., 65.],
                [ 1., 60.],
                [ 1., 60.],
                [ 1., 50.]])
```

```
In [11]: model = sm.OLS (y,  x)
         results = model.fit()
         results.summary()

         C:\Users\Admin\anaconda3\lib\site-packages\scipy-
         packages\stats.py:1535: UserWarning: kurtosistest only valid for  n>=20
... continuing anyway, n=10
            “anyway, n=%i” %int(n))
```



```
Out[11]:   OLS Regression Results

              Dep. Variable:                  y          R-squared:     0.619
                     Model:                OLS      Adj. R-squared:     0.571
                    Method:      Least Squares           F-statistic:   12.98
                      Date:   Wed, 01 Apr 2020  Prob (F-statistic):   0.00695
                      Time:           15:30:14     Log-Likelihood:   -13.311
          No. Observations:                 10                 AIC:     30.62
              Df Residuals:                  8                 BIC:     31.23
                  Df Model:                  1
           Covariance Type:          nonrobust

                   coef   std err      t     P>|t|   [0.025   0.975]
          const  2.7222    3.147    0.865   0.412   -4.535    9.979
             x1  0.1944    0.054    3.603   0.007    0.070    0.319

                Omnibus:   0.088      Durbin-Watson:   1.405
          Prob(Omnibus):   0.957   Jarque-Bera (JB):   0.301
                   Skew:  -0.112            Prob(JB):   0.860
               Kurtosis:   2.180           Cond. No.     567.
```

```
Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

Conclusion

From above table we obtain fitted equation as y=2.7222+0.1944*x

Since the value of Adj. R-squared is 0.571=> 57.1% variation in sales is explained by independent variable expenses.

Test for significance of β_0 and β_1:

- H_0: $\beta_0=0$ against H_1: $\beta_0 \neq 0$

Since, the p-value is 0.412(from above table) which is greater than 0.05 we do not reject H_0 at 5% level of significance i.e., β_0 is non-significant.

- H_0: β_1=0 against H_1: $\beta_1 \neq 0$

Since, the p-value is 0.007(from above table) which is less than 0.05 we reject H_0 at 5% level of significance i.e., β_1 is significant.

```
In [12]:#Regression (Alternate Solution)
      import numpy as np
      form sklearn.linear_model import LinearRegression
      x = np.array ([ 50,50,55,60,65,65,65,6560,60,60,50]).reshape((-1, 1))
      y= np.array ([11,13,14,16,16,15,15,14,13,13])
      x
Out [12]: array([[50),
      [50],
      [55],
      [60],
      [65],
      [65],
      [65],
      [60],
      [60],
      [50]])
```

```
In [13]: y
Out [13]: array([11, 13, 14, 16, 16, 15, 15, 14, 13, 13] )
```

```
In [14]:model = LinearRegression()
        model.fit(x, y)
Out[14]: LinearRegression(copy_x=True,fit_intercept=True,n_jobs=None,
         normalize=False)
```

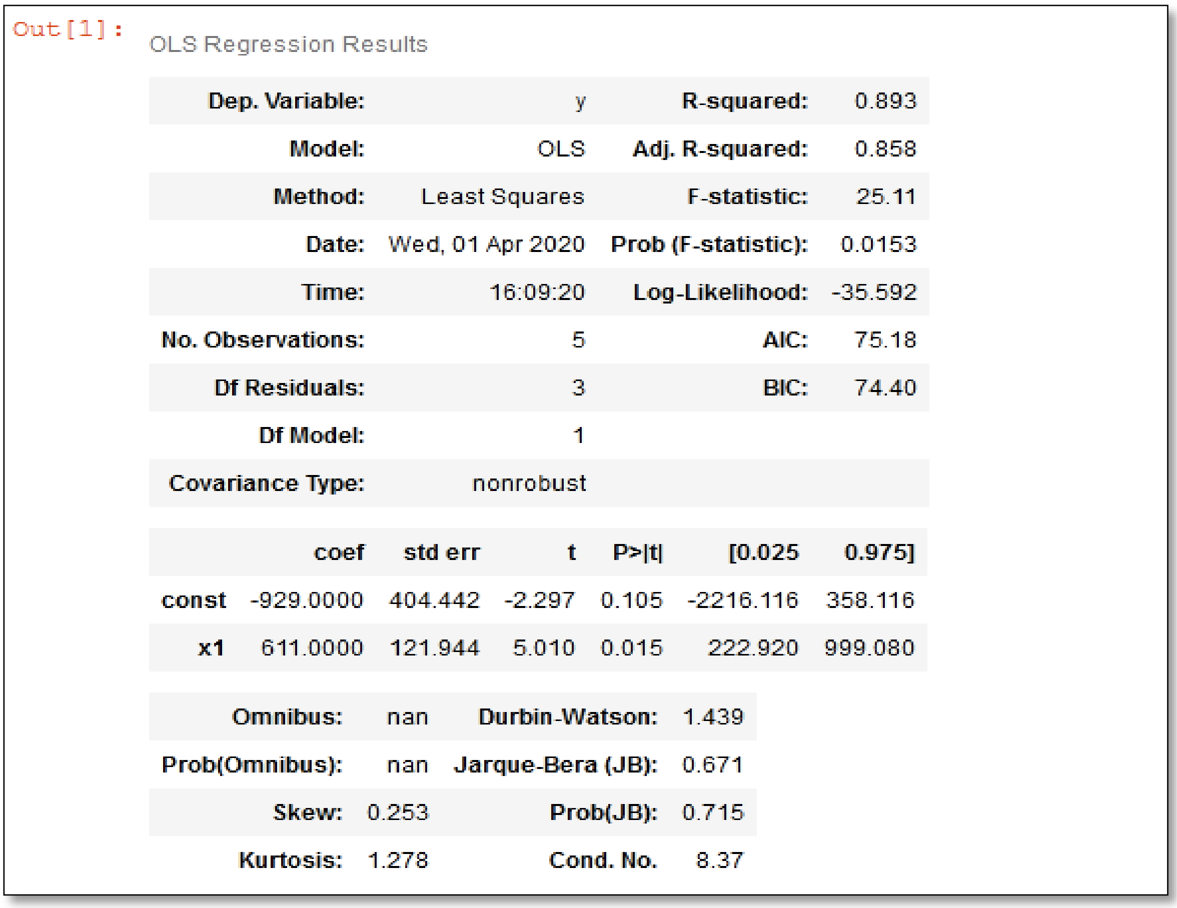
```
In [15] : model - LinearRegression().fit(x, y)
      r_sq = model.score(x, y)
      print('coefficient of determination:', r_sq)
      print ( 'intercept:' , model.intercept_)
      print ( 'slope: ' , model.coef_)

      coefficient of determenation:  0.6186868686868685
      intercept: 2.722222222222223
      slope: (0.19444444)
```

2. Fit a regression equation to the following data

x	1	2	3	4	5
y	20	150	550	1300	2500

Solution

```
In [1] :    import numpy  as np
      import statsmodels.api as sm
      x = np.array( [1,2,3,4,5] )
      y = np.array ( [20,150,550,1300,2500] )
      x=sm.add_constant (x)
      model=sm.OLS(y,x)
      results=model.fit()
      results.summary()

C:\Users\Admin\anaconda3\lib\site-packages\statsmodels\stats\stattools.py:71:
ValueWarning: omni_normtest is not valid with less than 8 observations; 5
samples were given.
"Samples were given." & int(n), ValueWarning)
```



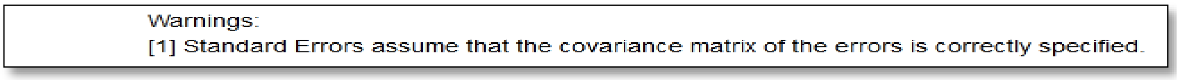
Out[1]:

OLS Regression Results

Dep. Variable:	y	R-squared:	0.893
Model:	OLS	Adj. R-squared:	0.858
Method:	Least Squares	F-statistic:	25.11
Date:	Wed, 01 Apr 2020	Prob (F-statistic):	0.0153
Time:	16:09:20	Log-Likelihood:	-35.592
No. Observations:	5	AIC:	75.18
Df Residuals:	3	BIC:	74.40
Df Model:	1		
Covariance Type:	nonrobust		

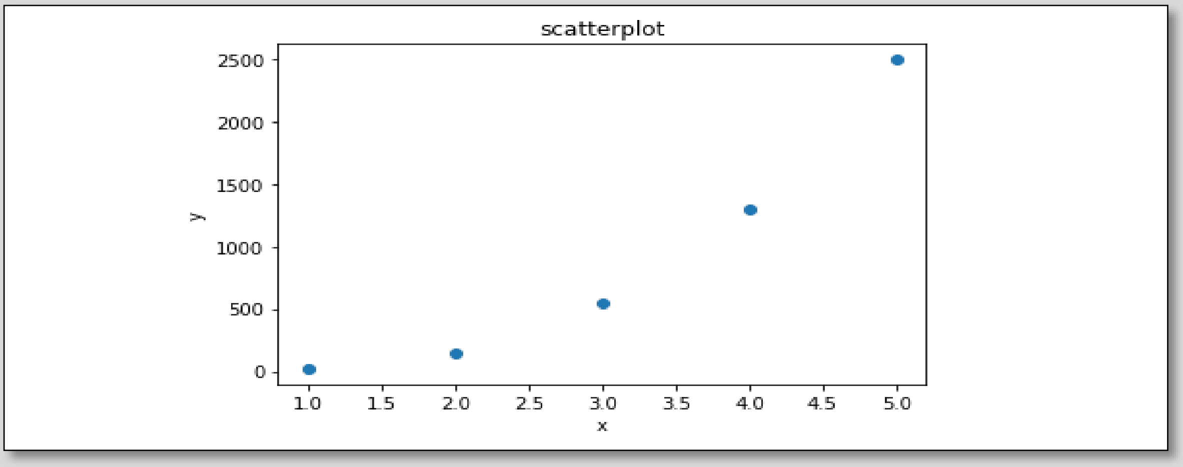
	coef	std err	t	P>|t|	[0.025	0.975]
const	-929.0000	404.442	-2.297	0.105	-2216.116	358.116
x1	611.0000	121.944	5.010	0.015	222.920	999.080

Omnibus:	nan	Durbin-Watson:	1.439
Prob(Omnibus):	nan	Jarque-Bera (JB):	0.671
Skew:	0.253	Prob(JB):	0.715
Kurtosis:	1.278	Cond. No.	8.37

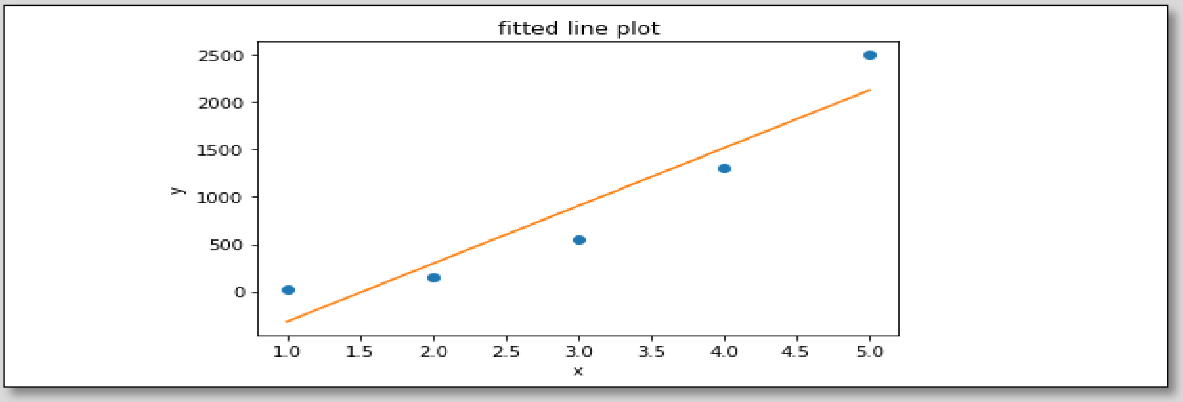


Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

```
In [2]: #From above table we obtain fitted equation as y=-929+611*x
      a=-929
      b=611
      import matplotlib.pyplot as plt
      x = np.array ( [ 1,2,3,4,5] )
      y = np.array ( [ 20,150,550,1300,2500] )
      plt.plot (x,y, 'o' )
      plt.xlabel ( 'x' )
      plt.ylabel ( 'y')
      plt.title ( 'scatterplot')
      plt.show( )
```



```
In[3] :plt.plot (x,y, 'o' )
      plt.plot (x, a+b*x)
      plt.title ( 'fitted line plot')
      plt.xlabel ( 'x' )
      plt.ylabel ( 'y' )
      plt.show ( )
```



8.2 Curve Fitting

The relationship between the two variables may not always be linear. Hence if scatter plot indicates curvilinear relationship we try to fit curvilinear model instead of linear model. There are many curvilinear models in this chapter we cover following models.

- **Quadratic curve:**

Quadratic curve is given by y=a+bx+cx^2.It is second degree polynomial equation.

- **Power curve:**

Power curve is given by y=ax^b. For simplification, we can write it as log(y)=log(a)+b*log(x).

- **Exponential curve:**

Exponential curve is given by y=ab^x. For simplification, we can write it as log(y)=log(a)+x*log(b)

- **Logarithmic curve:**

Logarithmic curve is given by y=a+b*log(x).

Note: log is Log to the base e. y is dependent variable x is independent variable

ISBN 978-93-80788-93-7

Examples:

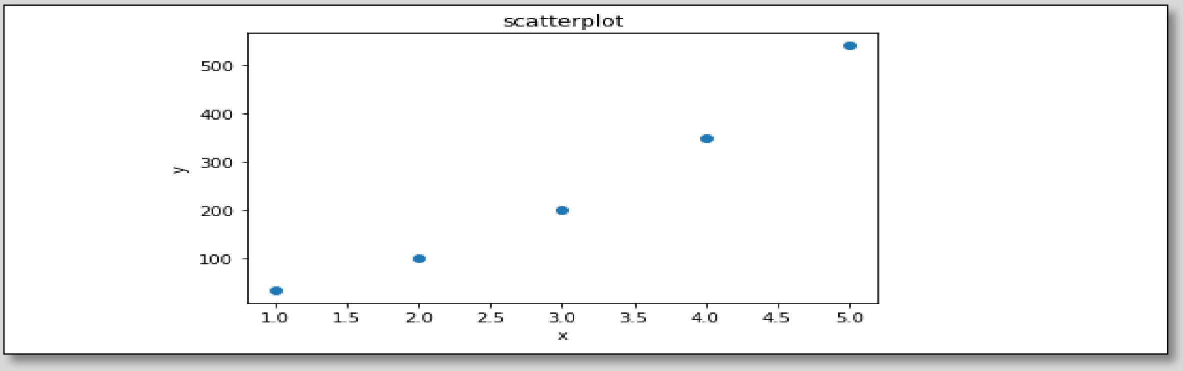
1. Fit a quadratic curve to the following data and estimate y when x=5.

x	1	2	3	4	5
y	35	100	200	30	540

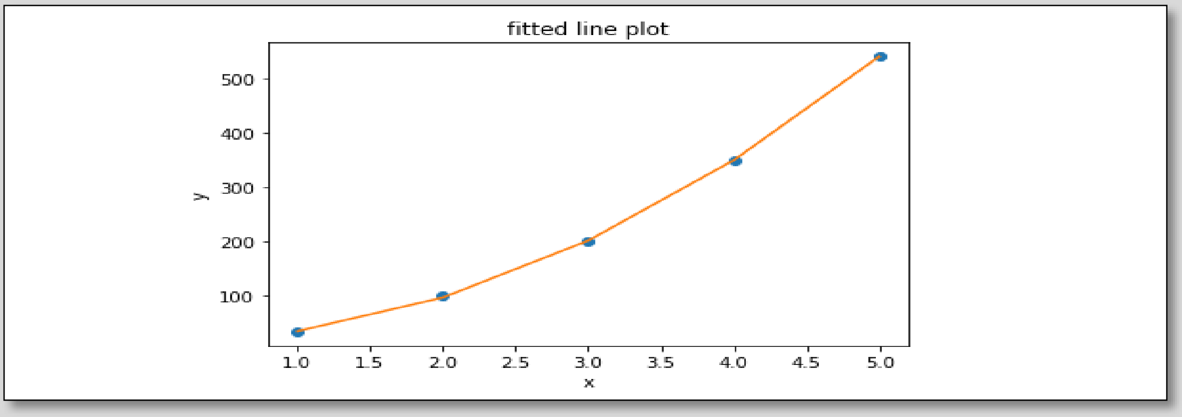
Solution

```
In [1]: import numpy as np
      from sklearn.preprocessing import PolynomialFeatures
      x = np.array ([1,2,3,4,5] )
      y = np.array ([35,100,200,350,540])
      np.polyfit (x,y,2) #2 indicates degree of the polynomial

Out [1]:    array ([21.42857143, -2.57142857, 17.])
```

```
In [2]: #array ([‘coeeficient2’, ‘coefficient1’, 'intercept '])
      #here b0=17 b1= -2.57142857 and b2=21.4257143
      a=17
      b=-2.57142857
      c=21.42857143
      import matplotlib.pyplot as plt
      plt.plot (x, y, ' o ‘)
      plt.title (‘scatterplot ')
      plt.xlabel (‘x ' )
      plt.ylabel (‘y ' )
      plt.show ( )
```



```
In [3]:plt.plot (x,y, ' o ' )
      plt.plot (x, a+b*x+c*x**2)
      plt.title( ' fitted line plot ' )
      plt.xlabel (' x ')
      plt.ylabel (' y’)
      plt.show ( )
```



```
In [4]:#when x=5 then y
      y_=a+b*5+c*5**2
      y_

Out [4]: 539.8571429000001
```

2. Fit a power curve to the following data and estimate y when x=6.

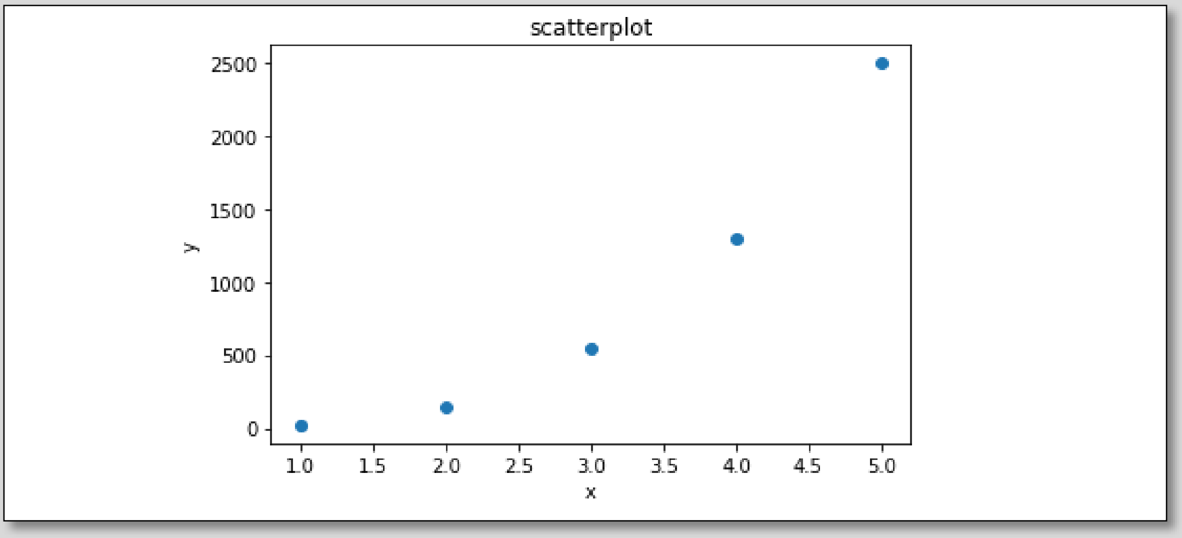
x	1	2	3	4	5
y	20	150	550	1300	2500

Solution

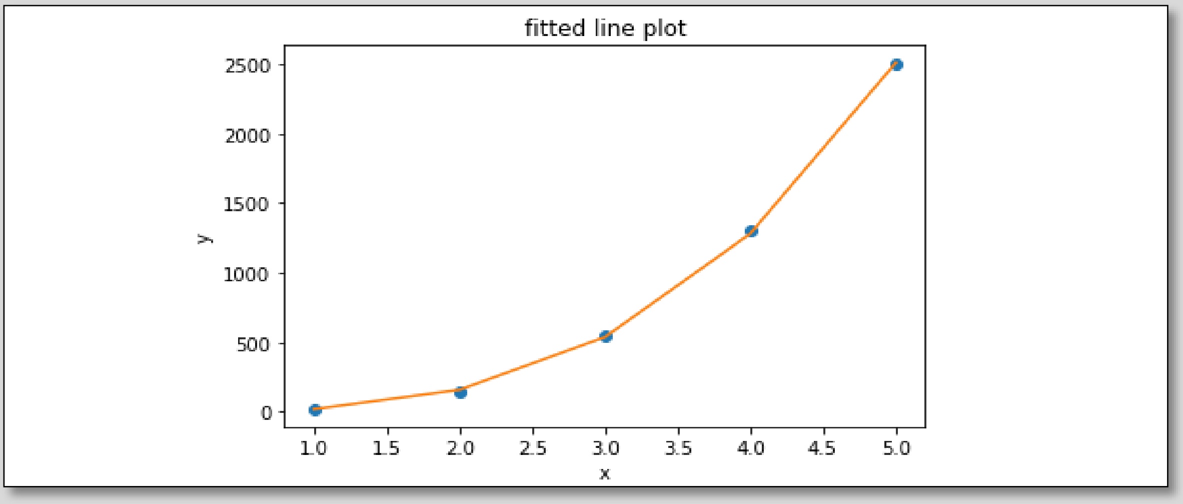
```
In [1] :    import numpy as np
      from sklearn.preprocessing import PolynomialFeatures
      x = np.array ( [1,2,3,4,5] )
      y = np.array ( [20,150,550,1300,2500] )
      np.polyfit (np.log (x) ,np.log (y),1)

Out [1] :   array ( [3.01627884,     2.97400827] )
```

```
In [2] :    m=2. 97400827
      n=3. 01627884
      a=np.exp(m)
      a

Out [2] : 19.57020526822196
```

```
In [4] :    b=n
      import matplotlib.pyplot as plt
      plt.plot (x,y, ' o ')
      plt.title ( ' scatterplot ' )
      plt.xlabel (' x ')
      plt.ylabel (' y' )
      plt.show ( )
```



```
In [5]:plt.plot (x,y, 'o' )
      plt.plot (x,a*x**b)
      plt.title (' fitted line plot' )
      plt.xlabel (' x ')
      plt.ylabel (' y' )
      plt.show ( )
```



```
In [6]: # when x=6 then y
      y_=a*6**b
      y_
Out[6]: 4352.27703479539
```

3. Fit a exponential curve to the following data:

x	1	3	5	7
y	5	20	100	400

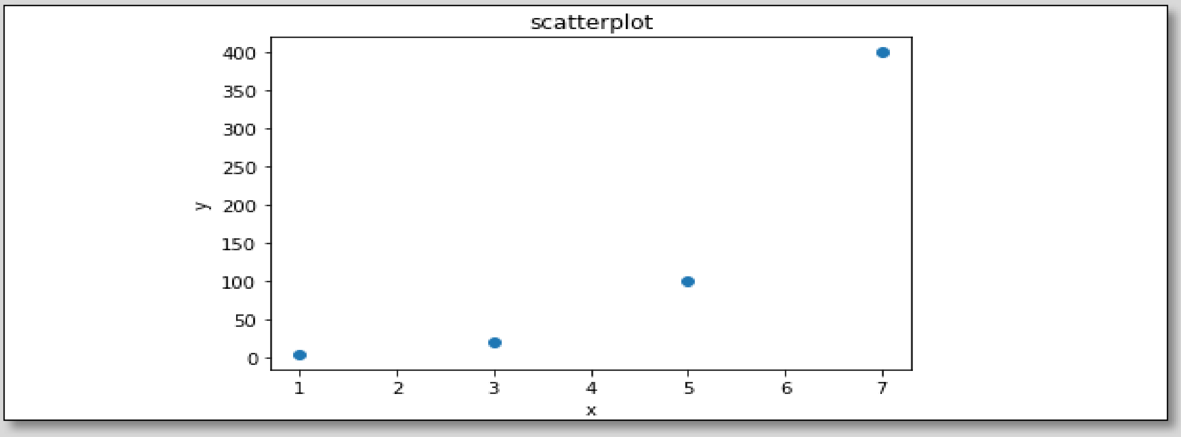
Solution

```
In [1] :    import numpy as np
      from sklearn.preprocessing import PolynomialFeatures
      x = np.array ( [1,3,5,7] )
      y = np.array ( [5,20,100,400] )
      np.polyfit(x,np.log(y),1)

Out [1] :   array ( [0.73777589,     0.84934767] )
```

```
In [2]:slope=0.73777589
      intercept=0.84934767
      a=np.exp (intercept)
      b=np.exp(slope)
      a

Out [2]:    2.3381211277869918
```

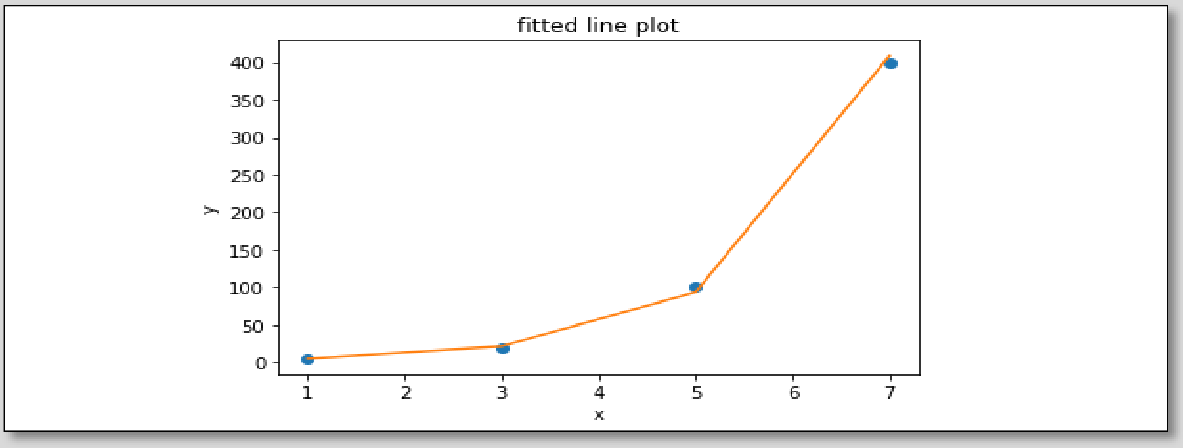
```
In [3]:     b
Out [3]:    2.0912791034618685
```

```
In [4]: import matplotlib.pyplot as plt
      plt.plot (x, y, ' o ')
      plt.title (‘scatterplot ' )
      plt.xlabel (' x ')
      plt.ylabel ('y’)
      plt.show ( )
```



```
In [5]:plt.plot (x, y, ‘o’)
      plt.plot (x, a*b**x)
      plt.title (' fitted line plot’)
      plt.xlabel (‘x’)
      plt.ylabel ('y’)
      plt.show ( )
```

ISBN 978-93-80788-93-7



4. Fit a logarithmic curve to the following data

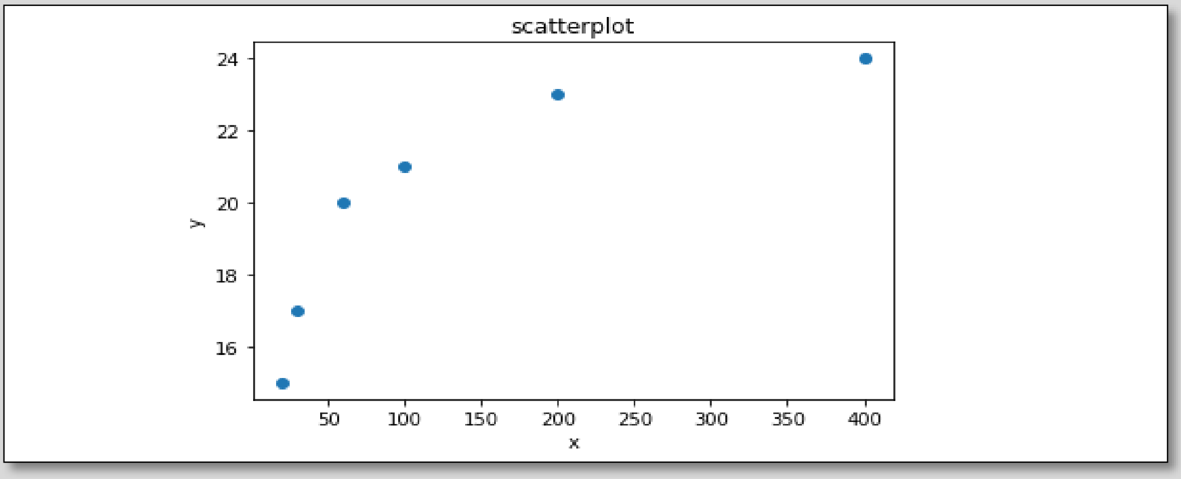
x	20	30	60	100	200	400
y	15	17	20	21	23	24

Solution

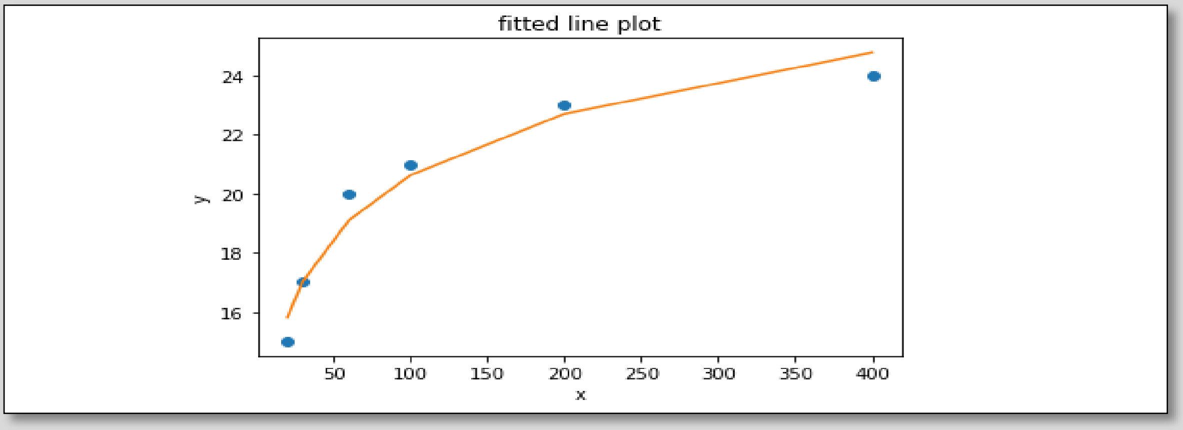
```
In [1]:     import numpy as np
      from sklearn.preprocessing import PolynomialFeatures
      x = np.array ([20,30,60,100,200,400] )
      y = np.array ( [15,17,20,21,23,24] )
      np.polyfit (np.log(x),y,1)

Out [1]:    array ([2.99213903, 6.84145708])
```

```
In [2]:     a=6.84145708
            b=2.99213903
```

```
In [3]: import matplotlib.pyplot as plt
      plt.plot (x,y, ' o ')
      plt.title ( ' scatterplot ' )
      plt.xlabel (' x ')
      plt.ylabel (' y' )
      plt.show ( )
```



```
In [4]:plt.plot (x,y, 'o' )
      plt.plot (x,a+b*np.log(x) )
      plt.title (' fitted line plot’)
      plt.xlabel (' x ')
      plt.ylabel (' y’)
      plt.show ( )
```



```
In [5]: #when x=5 then y
      y_=a+b*np.log(5)
      y_
Out [5]:  11.6571190745794
```

8.3 References

1. https://realpython.com/numpy-scipy-pandas-correlation-python/
2. https://realpython.com/linear-regression-in-python/
3. Dr Asha jindal (2017), Correlation,Regression and Curve fitting, Dr Asha jindal(Ed.),Analyzing and visualizing Data with R software-A practical manual, Shailja Prakashan and K.C. College, Page No:38-48

Chapter 9
Probability Distribution

Mr. Sachin Shamrao Bhaskar*, Assistant Professor, Department of Statistics, Anandibai Raorane Arts, Commerce & Science College, Vaibhavwadi*

9.1 Plotting using matplotlib

The Python core does not include any tools to generate plots. This functionality is added by other packages. By far the most common package for plotting is matplotlib. matplotlib is intended to mimic the style of Matlab. As such, users can either generate plots in the Matlab style. matplotlib contains different modules and features.

matplotlib.pyplot is the module that is commonly used to generate plots. It provides the interface to the plotting library in matplotlib, and is by convention imported in Python functions and modules with import matplotlib.pyplot as plt.

9.2 Probability Distribution

Distribution	**Command for Distribution**	**p.m.f./p.d.f.**	**c.d.f.**
Binomial(n, p)	bd=stats.binom(n,p)	prob=bd.pmf(x)	cdf=bd.cdf(x)
poisson(lm)	pd=stats.poisson(lm)	prob=pd.pmf(x)	cdf=pd.cdf(x)
hypergeometric	hgd=stats.hypergeom(N,m,n)	prob=hgd.pmf(x)	cdf=hgd.cdf(x)
Negative Binomial(r, p)	nbd=stats.nbinom(r,p)	prob=nbd.pmf(x)	cdf=nbd.cdf(x)
Uniform(a, b)	ud=stats.uniform(a,b)	p=ud.pdf(x)	cdf=ud.cdf(x)
Exponential(θ)	ex=stats.expon(θ)	p=ex.pdf(x)	cdf =ex.cdf(x)
Normal(μ, σ^2)	nd=stats.norm(μ, σ^2)	p=nd.pdf(x)	cdf=nd.cdf(x)
Lognormal	lg=stats.lognorm(m,s)	p=lg.pdf(x)	cdf=lg.cdf(x,m,s)
Gamma	gd=stats.gamma(n,th)	p=gd.pdf(x)	cdf=gd.cdf(x)
Beta	bd=stats.beta(n,th)	p=bd.pdf(x)	cdf=bd.cdf(x)
Chi-square	chi = stats.chi(n)	pdf=chi.pdf(x,n)	cdf=chi.cdf(x,n)

For Binomial(n, p)

```
In [1]: import scipy.stats as stats         # it import package scipy.stats
      n=5                            # it assigns the value of shape parameter
      p=0.5                # it assigns the value of probability of success.
      x=3                                           # it assign the value of x.
      bd=stats.binom(n,p)                          # it assigns the value of X
```

ISBN 978-93-80788-93-7

```
      prob=bd.pmf (x)      # it calculates P(X=3) of binomial distribution.
      cdf=bd.cdf(x)           #    it    calculates    c.d.f    of    Binomial
      distribution
      print (prob)
      print (cdf)

      0.3125
      0.8125
```

For Poisson(λ)

```
In [2]:import scipy.stats as stats         # it import package scipy.stats.
      lm=3                              #it assigns the vakue of parameterλ.
      x=3                                       # it assigns the value of X.
      pd=stats.poisson(lm)             #it assigns the Poisson distribution.
      prob=pd.pmf (x)       # it calculates P(X=3) of Poissondistribution.
      print(prob)                                        # it prints P(X=3).

      0.22404180765538775
```

Negative Binomial(r, p)

```
In [3]:import scipy.stats as stats         # it import package scipy.stats.
      r=4                              #it assigns the value of parameter λ.
      p=0.5
      x=3                               # it assigns the Poisson distribution
      nbd=stats.nbinom (r,p)         #it assigns  the Poisson Distribution.
      prob=nbd.pmf(x)         #it calculates P(X=3) of Poisson distribution
      print(prob)                                         # it rints P(X=3).

      0.15625000000000017
```

Lognormal

```
In [4]: import scipy.stats as stats
      m=3
      s=2                              # it assign the value of parameter λ.
      x=3                                         #it assign the value of X.
      lg=stats.lognorm(m,)        # it assigns the log-Normal distribution.
      p=lg.pdf(x)   #it  calculates  probabilities  of  log-Normal  distribution.

      print (p)
      0.0414521374995849
```

9.3 Random Sample Generation

In python, we can generate random sample of size n from different distributions by using the following commands. We need to import numpy package after that use the following commands:

```
In [5]: import numpy as np
```

```
      r1=np.random.binomial (8, 0, 4, 10)
                                  # Sample of size 10 from binomial (8, 0, 4)
      print ("Random numbers are:", r1)
      Random numbers are: [3 3 4 1 4 4 2 2 5 1]

In [6]:r2=np.random.poisson(3.2,10)   #sample of size 10 from poisson (3.2)
      print ("Random numbers are:", r2)
      Random numbers are: [4 3 6 4 0 9 2 3 1]
```

```
In [7]:r3=np.random.poisson(0,1,10)    #sample of size 10 from Normal (0,1)
      print ("Random numbers are:", r3)
      Randoms numbers are: [-0.14398527   -0.00565769       -0.00565769 -
      0.08706065  0.79306237  -0.42853533       -0.01740361
      0.89059493  0.43773545  -0.62410716 1.19725444]

In [8]:r4=np.random.poisson(4,10)   #sample of size 10 from exponetial (4)
      print ("Random numbers are:", r4)
      Random  Numbers  are:  [2.84389684  1.4688718  1.904448382  149540771
      2.50115998 3.91341569 0.9126324 0.37868265 0.03613664 3.44497827]
```

```
In [9]:r5=np.random.poisson(0,5,10)
                                    #sample of size 10 from uniform (0,5)
      print ("Random numbers are:", r5)
      Random  numbbers  are:  [3.79425599  4.56937883  3.06672609  2.41885944
      3.16853992 2.79737245 4.7398593 1.3271012 1.57616696 1.07407285]

In [10]:r6=np.random.lognormal(0,1,10)
                                    #sample of size 10 from logNormal (0,1)
      print ("Random numbers are:", r6)
      Random  numbers  are:  [4.11328901  1.8174152  0.35794728  1.82482682
      1.18830096 0.32319486 2.59994078 0.4198525 1.00776184]
```

```
In [11]:r7=np.random.standard_cauchy(10)
                                    #sample of size 10 from standard_cauchy
      print ("Random numbers are:", r7)
      Random numbers are:[-0.20393381 -0.57215044 -4.74513244 -2.45038412 -
      5.9634032 04568767 70.7991319 -1.20012489 -1.17320447 -1.46330294]
In [12]:r8=np.random.gamma(4,3,10)
                       #sample of size 10 from gamma (with shape 4& scale3)
      print ("Random numbers are:", r8)
      Random numbers are : [3.7795373 16.00762641 3.1506161619 17.59847979
      6.40907394    6.29579046    12.04232884    11.35496534    10.43878574
      15.69201329]
```

```
In [13]:r9=np,random.beta(4,3,10)
                                            # sample of 10 from beta(4, 3)
      print ( "Random numbers are:", r9)
      Random  numbers  are:  [0.41952739  0.71660687  0.61529927  0.65816305
      0.58354979 0.46716569 0.47045574 0.62504264 0.74208358 0.8324287]
```

```
In (14): r10=np.random.standard_t (5, 10)
                        # Sample of size 10 froam standard_t with 5 d.f.
      print("Random numbers are:", r10)
      Random  numbers  are:  [-0.44883174  1.26886067  -1.1837764  -2.54345885
      0.1525956 -0.11678527 -1.09737212 2.67747062 0.25900981 -1.63502092]

In (15):r11=np.random.chiasquare(5,10)
                            # Sample of size 10 froa chisquare with 5 d.f.
      print("Random numbers are:", r11)
      Random   number   are:   [5.8204119   2.02954166   7.92492356   5.37698655
      3.60778101 3.32521995 8.57612513 2.96091453 12.61467725 4.84896307]
```

9.4 Examples

1. If $X\sim$ (10, 0.6). Find **a)** $P(X = 0)$ **b)** $P(X = 2)$ **c)** $P(X \leq 3)$ **d)** $P(X > 5)$

```
In [16]:import numpy as np
      import scipy.stats as stats
      n=10
      p=0.6
      a=np.arrange(n+1)
      bd=stats.binom(n,p)
      prob=bd.pmf (x)
      cdf=bd.cdf(z)
      print ('pmf of x')
      print (prob)
      print (' \n')
      print ('cdf of x')
      print(cdf)
      print ('\n')
      print('P(X=0)=',prob[0] )
      print('P(X=2)=',prob[2] )
      print('P(X<=3)=',cdf[3] )
      print('P(X>5)=',1-cdf[5] )

      pmf of x
      [1.04857600e-04 1.57286400e-03 1.06168320e-02 4.24673280e-02
      1.11476736e-01 2.00658125e-01 2.50822656e-01 2.14990848e-01
      1.20932352e-01 4.03107840e-02 6.04661760e-03]

      cdf of x
      [1.04857600e-04 1.67772160e-03 1.22945536e-02 5.47618816e-02
      1.66238618e-01 3.66896742e-01 6.17719398e-01 8.32710246e-01
      9.53642598e-01 9.93953382e-01 1.00000000e+00]

      P(x=<l)= 0.000104857600000000014
      P(x=2)= 0.010616832
      P(X<=3)= 0.05476188160000002
      P(X>5)= 0.6331032576
```

2. If $X \sim \text{Poisson}(3.2)$. Find **a)** $P(X = 0)$ **b)** $P(X = 3)$ **c)** $P(X = 10)$ **d)** $P(X \leq 1)$ **e)** $P(X > 3)$ **f)** $(X \geq 5)$

```
In [17]:import numpy as np
      import scipy.stats as stats
      L=3.2
      x=np.arrange(11)
      pd=stats.poisson(L)
      prob=bd.pmf (x)
      cdf=pd.cdf(z)
      print ( 'pmf of x')
      print (prob)
      print ( ' \n' )
      print (' cdf of x')
      print(cdf)
      print ( ' \n' )
      print('P(X=0)=',prob[0] )
      print('P(X=2)=',prob[3] )
      print('P(X=10)=',prob[10] )
      print('P(X<=1)=',prob[10] )
      print('P(X>3)='1,cdf[3] )
      print('P(X>=5)=',1-cdf[4] )

      pd of x
      [0.0407622 0.13043905 0.20870248 0.22261598 0.17809279 0.11397938
      0.060789 0.02778926 0.0111157 0.00395225 0.00126472]

      cdf of x
      [0.0407622 0.17120126 0.37990374 0.60251972 0.78061251 0.89459189
      0.9553809 0.98317016 0.99428586 0.99823811 0.99950283]

      P(x=0)= 0.04076220397836621
      P(x=3)= 0.22261598332718394
      P(x=10)= 0.0012647200634353655
      P(X<=1)= 0.17120125670913808
      P(X>3)= 0.3974802755944429
      P(X>=5)= 0.21938748893269577
```

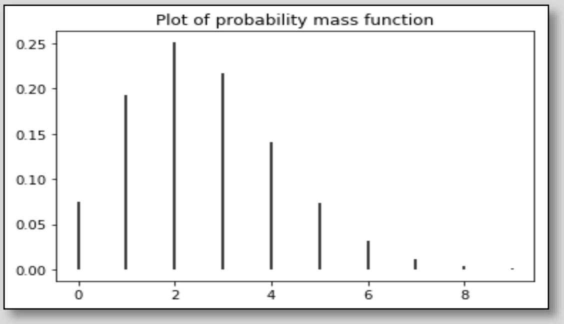
3. If $X \sim HyperGeo(N = 25, M = 5, n = 3)$. Find **a)** $P(X = 0)$ **b)** $P(X = 2)$ **c)** $P(X = 5)$ **d)** $(X \leq 1)$ **e)** $(X > 3)$ **f)** $(X \geq 2)$

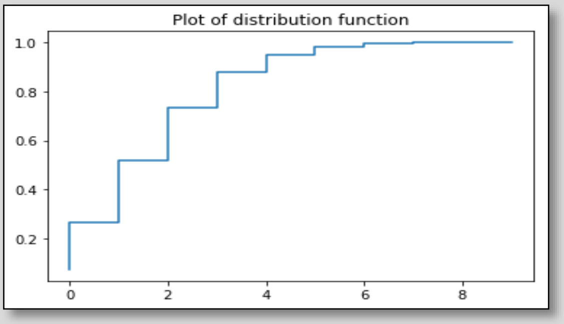
```
In [18] :   import numpy as np
      import scipy.stats as stats
      N=25
      m=5
      n=3
      x=np. arange (n+1)
      hgd.stats.hypergeom(N,m,n)
      prob=hgd.pmf(x)
      cdf=hgd. cdf (x)
      print ( 'pmf of x')
      print (prob)
      print ( ' \n' )
```

```
         print ('cdf of x')
         print(cdf)
         print ( ' \n ' )
         print('P(X=0)=',prob[0] )
         print('P(X=2)=',prob[2] )
         print('P(X=5)=',0)
         print('P(X<=1)=',cdf[1] )
         print('P(X>3)=',1-cdf[3] )
         print('P(X>=2)=',1-cdf[1] )

         pmf of x
         (0.49565217 0.41304348 0.08695652 0.00434783]

         cdf of x
         [0.49565217 0.90869565 0.99565217 1.]

         P(x=0)= 0.4956521739130437
         P(x=2)= 0.08695652173913056
         P(x=5)= 0
         P(X<=1)= 0.908695652173914
         P(X>3)= 0.0
         P(X>=2)= 0.09130434782608599
```

4. Plot probability mass function (pmf) and distribution function for the random variables: **a)** $X \sim \text{Poisson}(2.6)$ **b)** $X \sim Bino(8{,}0.65)$ **c)** $X \sim HyperGeo(N = 50, M = 10, n = 7)$

```
In [19]: import numpy as np
         import scipy.stats as stats
         import matplotlib.pyplot as p1t
         l=2.6
         x=np.arange(10)
         pd=stats.poisson(1)
         prob=pd.pmf(x)
         cdf=pd. cdf (x)
         plt.vlines(x,0,prob)
         plt.title('Plot of probability mass function')
         plt.show()
         plt.step(x,cdf)
         plt.ti.tle('Plot of distribution function')
         plt.show()
```

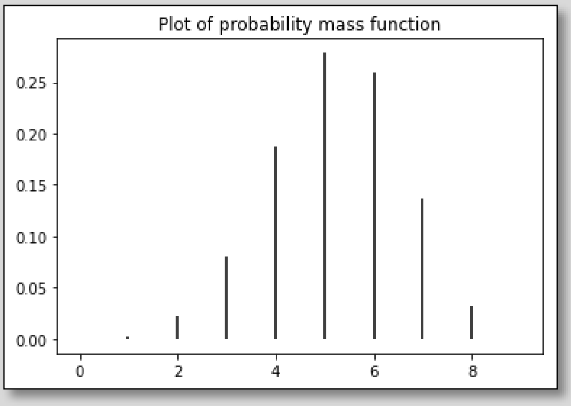


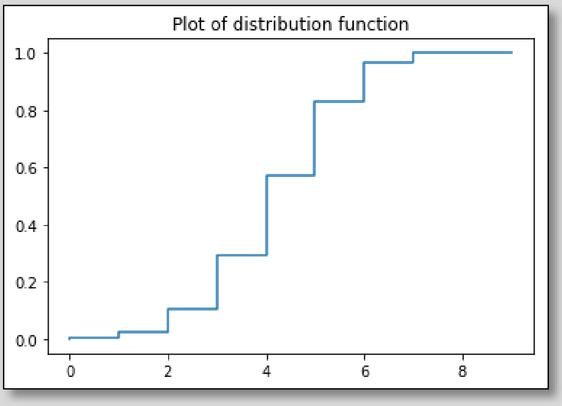


ISBN 978-93-80788-93-7

a)

```
In [20]: import numpy as np
      import scipy.stats as stats
      import matplotlib.pyplot as plt
      n=8
      p=0.65
      x=np.arange(10)
      bd-stats.binom(n,p)
      prob=bd.pmf(x)
      cdf-bd. cdf (x)
      plt.vl.nes(x,0,prob)
      plt.title('Plot of probability mass function')
      plt. show()
      plt.step(x,cdf)
      plt.title('Plot of distribution function')
```

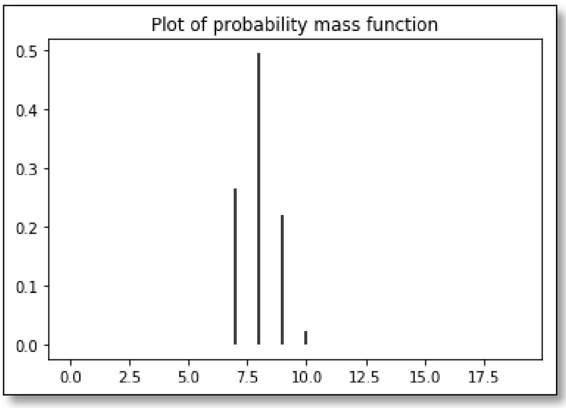


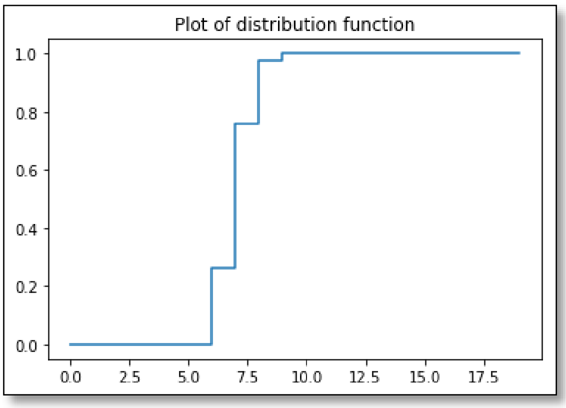


b)

```
In [21]: import numpy as np
      import scipy.stats as stats
      import matplotlib.pyplot as plt
      N=15
      m=12
      n=10
      x=np.arange (20)
      hgd=stats.hypergeom (N,m,n)
      prob=hgd.pmf(x)
      cdf=hgd.cdf (x)
      plt.vlines(x,0,prob)
      plt.title('Plot of probability mass function')
      plt.show()
      plt.step(x,cdf)
      plt.ti.tle('Plot of distribution function')
      plt.show()
```

ISBN 978-93-80788-93-7



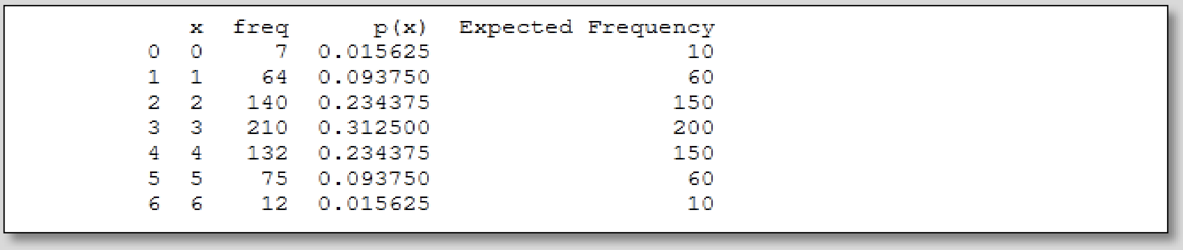


5. A set of similar fair coins are tossed 640 times with the following results:

No. of heads	0	1	2	3	4	5	6
Frequency	7	64	140	210	132	5	12

Fit the binomial distribution to the data.

```
In [22]:import numpy as np
      import scipy.stats as stats
      import pandas as pd
      N=640
      x=np.array([0,1,2,3,4,5,6))
      f=np.array([7,64,140,210,132,75,12J)
      n=6
      p=0.5
      bd=stats.binom(n,p)
      prob-bd. pmf (x)
      E1=np.round (N*prob)
      data={  'x'  :x,  'freq'  :f,  'p(x)'  :prob,  'Expected  Frequency'  :
      E1.astype(int)}
      df=pd.DataFrame (data)
      print(df)
```



```
   x  freq      p(x)  Expected Frequency
0  0     7  0.015625                  10
1  1    64  0.093750                  60
2  2   140  0.234375                 150
3  3   210  0.312500                 200
4  4   132  0.234375                 150
5  5    75  0.093750                  60
6  6    12  0.015625                  10
```

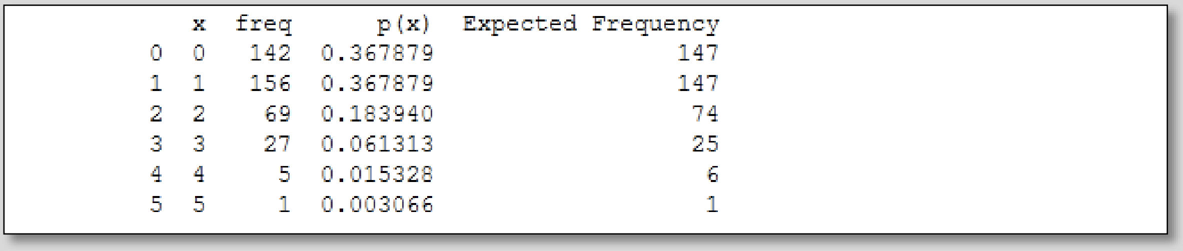
6. Fit a Poisson distribution to the following data with respect to the number of red blood corpuscles (x) per cell:

$\boldsymbol{x}$	0	1	2	3	4	5
Number of cells	142	156	69	27	5	1

```
In [23]: import numpy as np
      import scipy.stats as stats
      import pandas as pd
```

ISBN 978-93-80788-93-7

```
      x=np.array([0,1,2,3,4,5))
      f=np.array([142,156,69,27,5,1))
      N=sum(f)
      l=(x.dot(f))/N
      pdis=stats.poisson(l)
      prob=pdis.pmf(x)
      E=np.round (N*prob)
      data=(  'x'  :x,  'freq'  :f,  'p(x)'  :prob,  'Expected  Frequency'
      :Ei.astype(int))
      df=pd.DataFrame(data)
      print(df)
```

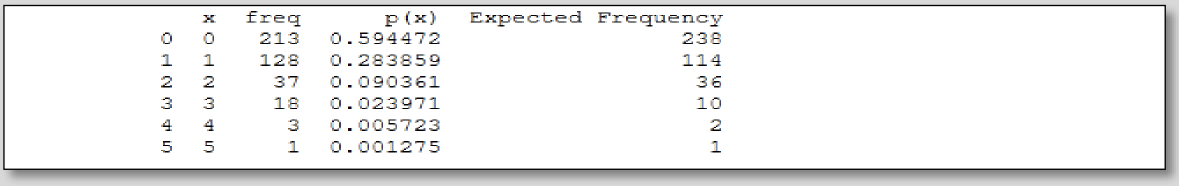


```
   x  freq      p(x)  Expected Frequency
0  0   142  0.367879                 147
1  1   156  0.367879                 147
2  2    69  0.183940                  74
3  3    27  0.061313                  25
4  4     5  0.015328                   6
5  5     1  0.003066                   1
```

7. Fit the negative binomial distribution to the following distribution

X	0	1	2	3	4	5
f	213	128	3	18	3	1

```
In [24]:import numpy as np
      import scipy.stats as stats
      import pandas as pd
      x=np.array([0,1,2,3,4,5))
      f=np.array([213,128,37,18,3,1])
      N=sum(f)
      m=(x.dot(f))/N
      x2=(x*x) .dot(f)
      v=(x2/N)-m**2
      p=m/v
      q=1-p
      r=int(m*p/q)
      nbd=stats.nbinom(r,p)
      prob=pdis.pmf(x)
      E=np.round (N*prob)
      data=(  'x'  :x,  'freq'  :f,  'p(x)'  :prob,  'Expected  Frequency'
      :Ei.astype(int))
      df=pd.DataFrame(data)
      print(df)
```



```
   x  freq      p(x)  Expected Frequency
0  0   213  0.594472                 238
1  1   128  0.283859                 114
2  2    37  0.090361                  36
3  3    18  0.023971                  10
4  4     3  0.005723                   2
5  5     1  0.001275                   1
```

8. If **i)** $X{\sim}N(2{,}1.5)$ **ii)** $X{\sim}Exp(\theta = 1.5)$ **iii)** $X{\sim}U(0{,}5)$
Find **a)** $P(X = 1.5)$ **b)** $P(X \le 1.5)$ **c)** $\mathrm{P}(X > 3.5)$ **d)** $P(0 < X < 2)$ **e)** $P(-2 < X < 2)$

i)

```
In [25]: import numpy as np
      import scipy.stats as stats
      print ('X  follows N (2,1.5)')
      m=2;  v=1.5
      nd=stats.norm(m,v)
      p=nd.cdf(1.5)
      p1=nd.cdf(1.5)
      p3=nd.cdf(3.5)
      p0=nd.cdf(0)
      p2=nd.cdf(2)
      p_2=nd.cdf(-2)
      print('P(x=1.5)=',p)
      print('P(x<=1,5)=',p1)
      print('P(x>3,5)=',1-p3)
      print('P(0<X<2)=',p2-p0)
      print('P(-2<X<2)=",p2-p_2)

      X follows N(2,1.5)
      P(x=1.5)= 0.2515888184619955
      P(x<=1.5)= 0.36944134018176367
      P(x>3.5)= 0.15865525393145707
      P(0<X<2)= 0.4087887802741321
      P(-2<X<2)= 0.4961696194324103
```

ii)

```
In [26]: import numpy as np
      import scipy.stats as stats
      print ('X  follows Exp ( 1.5)')
      m=2;  v=1.5
      nd=stats.norm(m,v)
      p=nd.cdf(1.5)
      p1=nd.cdf(1.5)
      p3=nd.cdf(3.5)
      p0=nd.cdf(0)
      p2=nd.cdf(2)
      p_2=nd.cdf(-2)
      print('P(x=1.5)=',p)
      print('P(x<=1,5)=',p1)
      print('P(x>3,5)=',1-p3)
      print('P(O<X<2)=',p2-p0)
      print('P(-2<X<2)=",p2-p_2)

      X follows Exp(l.5)
      P(x=1.5)= 1.0
      P(x<=1.5)= 0.0
      P(x>3.5)= 0.1353352832366127
      P(0<x<2)= 0.3934693402873666
      P(-2<x<2)= 0.3934693402873666
```

iii)

```
In [27]: import numpy as np
      import scipy.stats as stats
      print ('X  follows Uniform ( 0,5)' )
      a=0;  b=5
      ud=stats.uniform(a,b)
      p=ud.cdf(1.5)
      p1=ud.cdf(1.5)
      p3=ud.cdf(3.5)
      p0=ud.cdf(0)
      p2=ud.cdf(2)
      p_2=ud.cdf(-2)
      print('P(x=1.5)=',p)
      print('P(x<=1,5)=',p1)
      print('P(x>3,5)=',1-p3)
      print('P(O<X<2)=',p2-p0)
      print('P(-2<X<2)=",p2-p_2)

      X follows Uniform(0,5)
      P(x=1.5)= 0.2
      P(x<=1.5)= 0.3
      P(x>-3.5)= 0.30000000000000004
      P(O<X<2)= 0.4
      P(-2<X<2)= 0.4
```

9. Find the value of a, b, c and d if $P(X < a) = 0.8, P(X > b) = 0.9,$ $P(X > c) = 0.2, P(X < d) = 0.3$ for **i)**$X \sim N(2,1.5)$ **ii)** $X \sim Exp(\theta = 1.5)$ **iii)** $X \sim U(0,5)$

i)

```
In [28] : import numpy as np
      import scipy.stats as stats
      print("X follows N(2,1.5)")
      m=2
      v=1.5
      nd=stats.norm(m,v)
      a=nd.ppf(0.8)
      b=nd.ppf(1-0.9)
      c=nd.ppf(1-0.2)
      d=nd.ppf(0.3)
      print( ' a=' , a)
      print( 'b=' ,b)
      print( 'c= ' , c)
      print( "d=" ,d)

      X follows N(2,1.5)
      a= 3.2624318503593717
      b= 0.07767265168309945
      c= 3.2624318503593717
      d= 1.2133992309379387
```

ii)

```
In [29] :   import numpy as np
      import scipy.stats as stats
      print('X follows Exp(0,1)')
      th=1.5
      ex=stats.expon(th)
      a=ex.ppf(0.8)
      b=ea.ppf(1-0.9)
      c=ea.ppf(1-0.2)
      d=ea.ppf (0. 3)
      print(' a=', a)
      print( 'b=' ,b)
      print( 'c=' ,c)
      print( ' d=', d)

      X follows Exp(0,1)
      a= 3.1094379124341005
      b= 1.6053605156578263
      c= 3.1094379124341005
      d= 1.8566749439387324
```

iii)

```
In [30]:import numpy as np
      import scipy.stats as stats
      print('X follows Uniform (a,b)
      a=ud.ppf(0.8)
      b=ud.ppf(1-0.9)
      c=ud.ppf(1-0.2)
      d=ud.ppf (0. 3)
      print(' a=', a)
      print( 'b=' ,b)
      print( 'c=' ,c)
      print( ' d=', d)

      X follows Uniform
      a= 4.0
      b= 0.4999999999999999
      c= 4.0
      d= 1.5
```

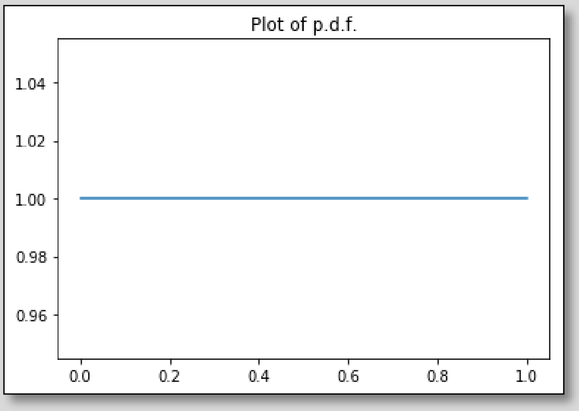
10. Plot probability density function (pdf) and distribution function for the following random variables:

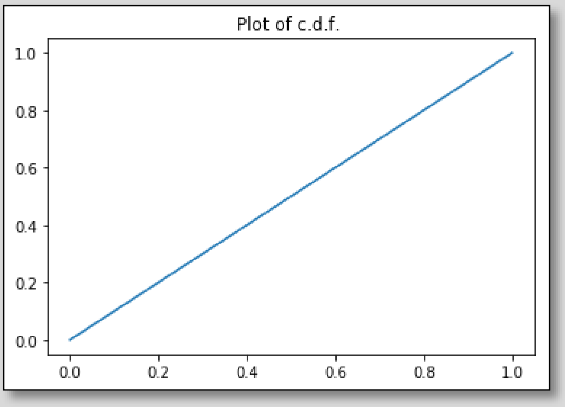
i) $\boldsymbol{X \sim U(0,1)}$

```
In [31]:import numpy as np
      import scipy.stats as stats
      import matplotlib.pyplot as plt
      print('X follows Uniform (0,1)')
      a=0
      b=1
      x=np.linspace(a,b,100)
      pdf=stats.uniform.pdf(x,a,b)
      cdf=stats.uniform.cdf(x,a,b)
```

ISBN 978-93-80788-93-7

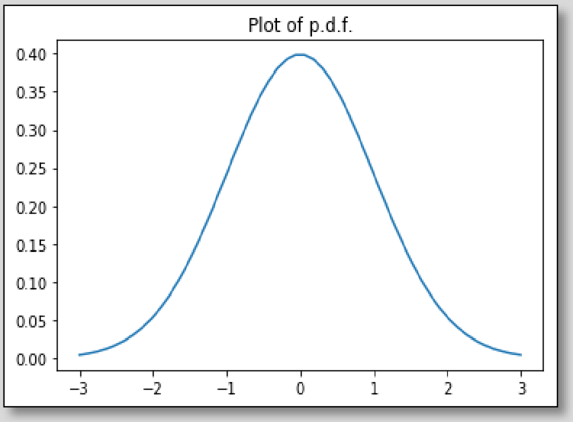
```
      plt.plot(x,pdf)
      plt.title('plot of p.d.f')
      plt.show()
      plt.plot(x,cdf)
      plt.title('Plot of c.d.f.')
      plt show ()

      X follows Uniform (0,1)
```

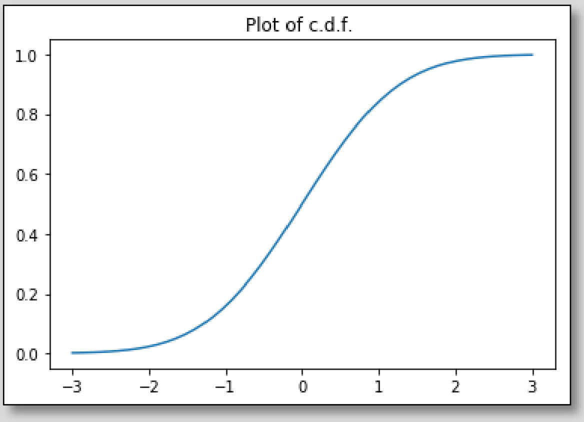




ii) $X \sim N(0,1)$

```
In [32]:import numpy as np
      import scipy.stats as stats
      import matplotlib.pyplot as plt
      import math
      m=0
      v=1
      sd=math.sqrt(v)
      x=np.linspace (m-3*sd,m+3*sd)
      pdf=stats.uniform.pdf(x,m,sd)
      cdf=stats.uniform.cdf(x,m,sd)
      plt.plot(x,pdf)
      plt.title('plot of p.d.f')
      plt.show()
      plt.plot(x,cdf)
      plt.title('Plot of c.d.f.')
      plt show ()
```

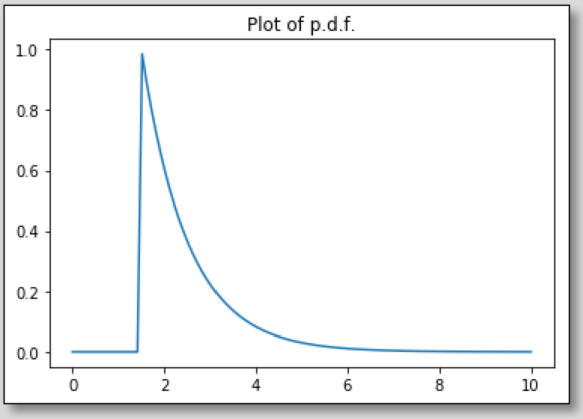


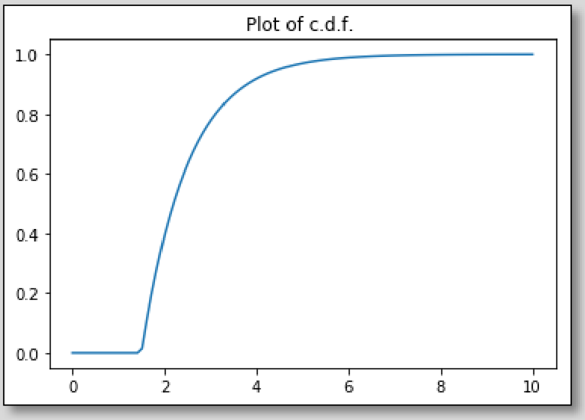


ISBN 978-93-80788-93-7

iii) $X \sim Exp(\theta = 1.5)$

```
In [33] :   import numpy as np
      import scipy.stats as stats
      import matplotlib.pyplot as plt
      th=1.5
      x=np.linspace(0,10,100)
      pdf=stats.expon.pdf (x,th)
      cdf=stats.expon.cdf (x,th)
      plt.plot(x,cdf)
      plt.title('Plot of c.d.f.')
      plt show ()
```

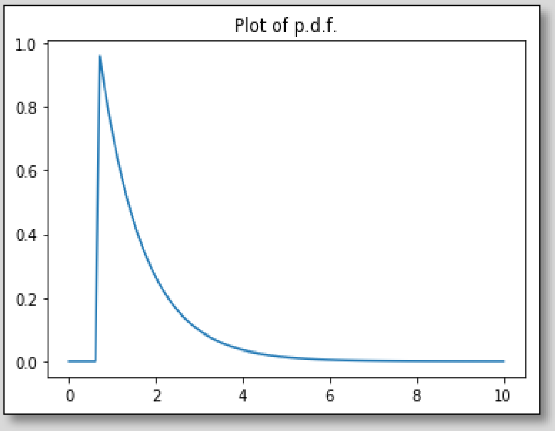


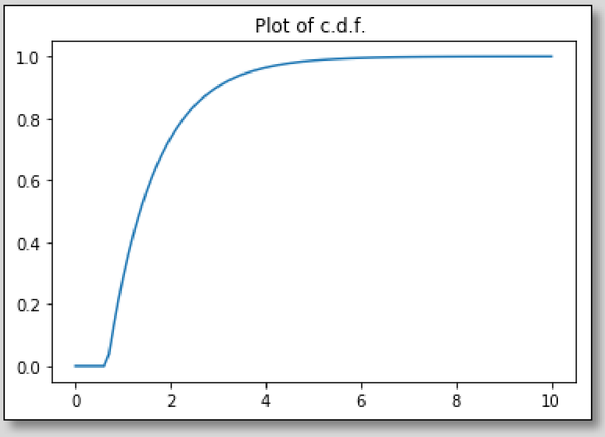


iv) $X \sim Exp(Mean = 1.5)$

```
In [34]:import numpy as np
      import scipy.stats as stats
      import matplotlib.pyplot as plt
      m=1.5
      th=1/m
      x=np.linspace(0,10,100)
      pdf=stats.expon.pdf (x,th)
      cdf=stats.expon.cdf (x,th)
      plt.plot(x,pdf)
      plt.title('plot of p.d.f')
      plt.plot(x,cdf)
      plt.title('Plot of c.d.f.')
      plt show ()
```

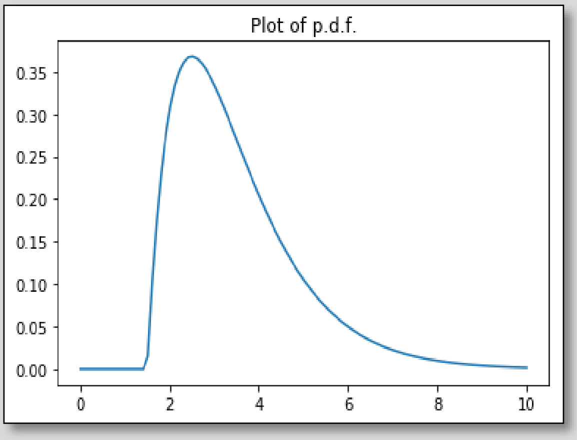
ISBN 978-93-80788-93-7

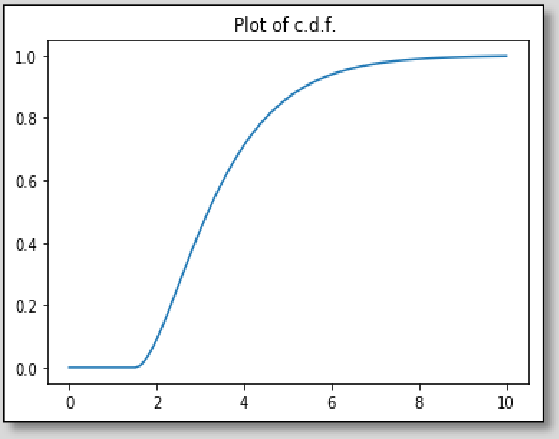




v) $X \sim \text{Gamma}(2, 1.5)$

```
In [35] :   import numpy as np
        import scipy.stats as stats
        import matplotlib.pyplot as plt
        n=2
        th=1.5
        x=np.linspace(0,10,100)
        pdf=stats.expon.pdf (x,th)
        cdf=stats.expon.cdf (x,th)
        plt.plot(x,pdf)
        plt.title('plot of p.d.f')
        plt show ()
        plt.plot(x,cdf)
        plt.title('Plot of c.d.f.')
        plt show ()
```

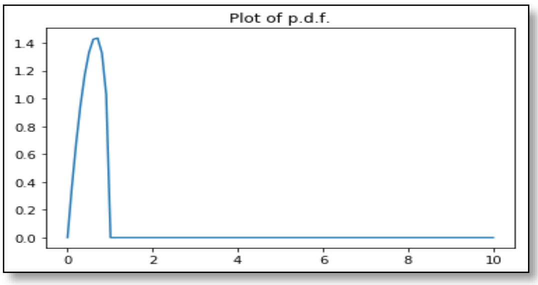


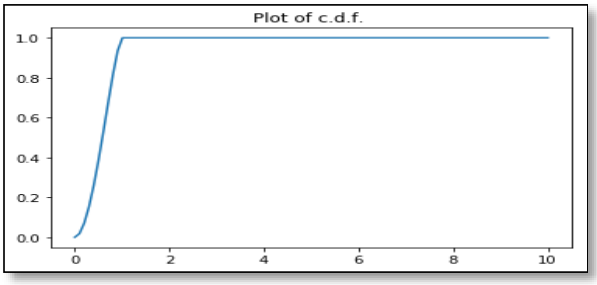


vi) $X \sim \text{beta}(2, 1.5)$

```
In [36] :   import numpy as np
        import scipy.stats as stats
        import matplotlib.pyplot as plt
        n=2
        th=1.5
        x=np.linspace(0,10,100)
        pdf=stats.expon.pdf (x,n,th)
        cdf=stats.expon.cdf (x,n,th)
```

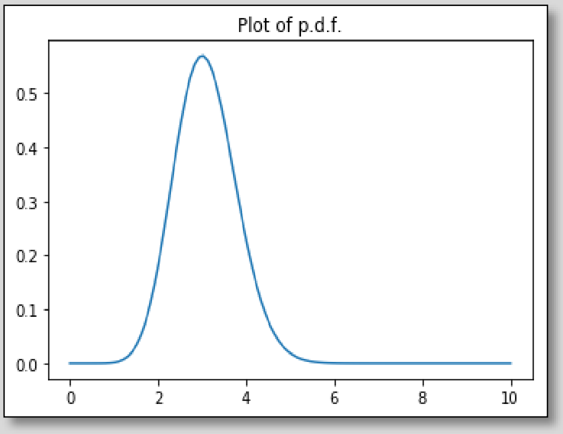
```
      plt.plot(x,pdf)
      plt.title('plot of p.d.f')
      plt show ()
      plt.plot(x,cdf)
      plt.title('Plot of c.d.f.')
      plt show ()
```

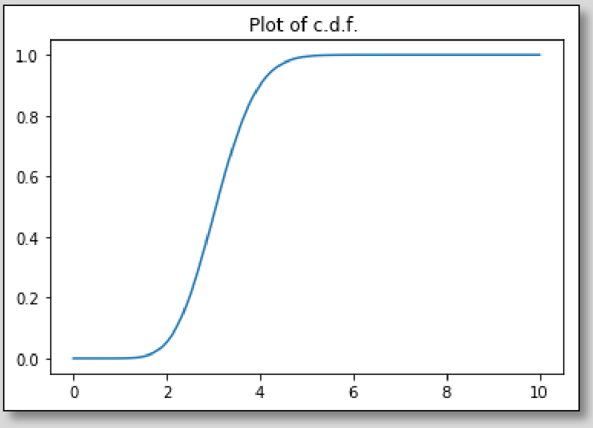




vii) X~Chi – Square(10)

```
In [37] :import numpy as np
      import scipy.stats as stats
      import matplotlib.pyplot as plt
      n=10
      x=np.linspace(0,10,100)
      pdf=stats.expon.pdf (x,n)
      cdf=stats.expon.cdf (x,n)
      plt.plot(x,pdf)
      plt.title('plot of p.d.f')
      plt show ()
      plt.plot(x,cdf)
      plt.title('Plot of c.d.f.')
      plt show ()
```





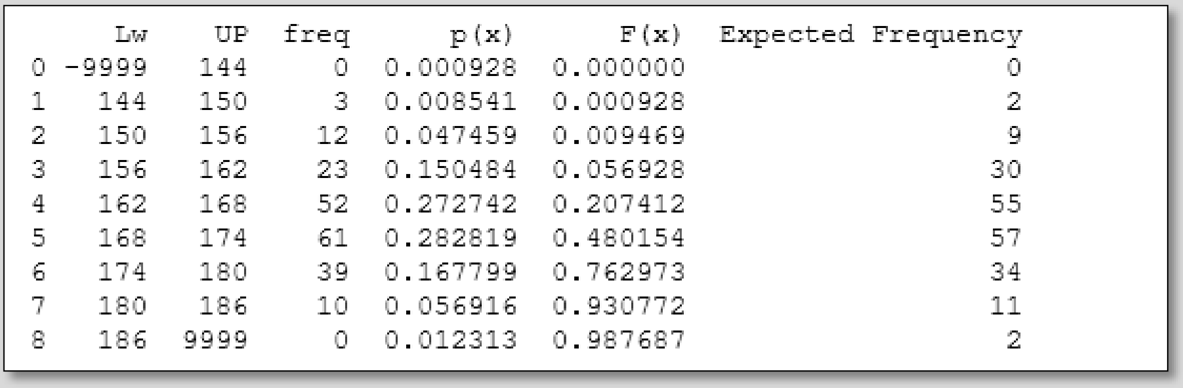
11. Fit a normal distribution to following data:

Marks	144-150	150-156	156-162	162-168	168-174	174-180	180-186
No. of Students	03	12	23	52	61	39	10

```
In [38]:import numpy as np
```

ISBN 978-93-80788-93-7

```
import pandas as pd
import scipy.stats as stats
import matplotlib.pyplot as plt
ll=np.array([144,150,156,162,168,174,180])
ul=np.array([150,156,162,168,174,180,186])
f=np.array([3,12,23,52,61,39,10])
x=(ll+ul)/2
k=len(f)
N=sum(f)
m=(x.dot(f))/N
x2=(x*x).dot(f)
v=(x2/N)-m**2
sd=np.sqrt (v)
L1=np.array ([-9999,144,150,156,162,168,174,180,186])
U1=np.array ([144,155,156,162,168,174,180,186,9999])
F=np.concatenate (([0] ,f,[0]))
nd=stats.norm(m,ad)
cp=nd.cdf(L1)
p=np.diff(cp)
p=np.append(p,1-cp[k+1])
ef=np.round(N*p)
data={'Lw':L1,  'UP'  :U1  'freq':F,  'p(x)':p,  'F(x)':cp,  'Expected
Frequency':ef.astype (int)}
df=pd.dataframe(data)
print (df)
```



```
     Lw    UP  freq      p(x)      F(x)  Expected Frequency
0 -9999   144     0  0.000928  0.000000                   0
1   144   150     3  0.008541  0.000928                   2
2   150   156    12  0.047459  0.009469                   9
3   156   162    23  0.150484  0.056928                  30
4   162   168    52  0.272742  0.207412                  55
5   168   174    61  0.282819  0.480154                  57
6   174   180    39  0.167799  0.762973                  34
7   180   186    10  0.056916  0.930772                  11
8   186  9999     0  0.012313  0.987687                   2
```

References

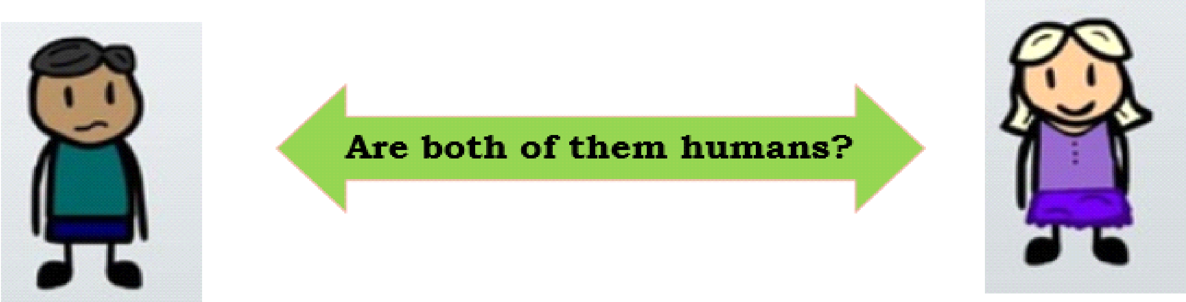
1. https://www.investopedia.com/terms/d/descriptive_statistics.asp
2. https://pythonfordatascience.org/descriptive-statistics-python/

Chapter 10
Statistical Tests

Ms. Divya Poojari*, Tata Consultacy Services, Statistical Programmer*

10.1 t-test

It is used to compare two samples to determine if they came from the same population.



What is t-test?
The t-test is a basic test that is limited to two groups. It is also called Student's T Test compares two averages (means) and tells you if they are different from each other. The t test also tells you how significant_the differences are; In other words it lets you know if those differences could have happened by chance.

For Example: Let's say you have a cold and you try a naturopathic remedy. Your cold lasts a couple of days. The next time you have a cold, you buy an over-the-counter pharmaceutical and the cold lasts a week. You survey your friends and they all tell you that their colds were of a shorter duration (an average of 3 days) when they took the homeopathic remedy. What you really want to know is, are these results repeatable? A t test can tell you by comparing the means of the two groups and letting you know the probability of those results happening by chance. The basic principle is to test the null hypothesis that the means of the two groups are equal.

Assumptions of t-test:
- ➢ The population from which the sample has been drawn should be normally distributed.
- ➢ Underlying variances are equal.

Use of t-test:
It is used when there is random assignment and only two sets of measurement to compare.

Types of t-tests:

There are three types of t-tests we can perform based on the data at hand:

- One sample t-test
- Two sample t-test
- Paired sample t-test

T-test in Python using SciPy:

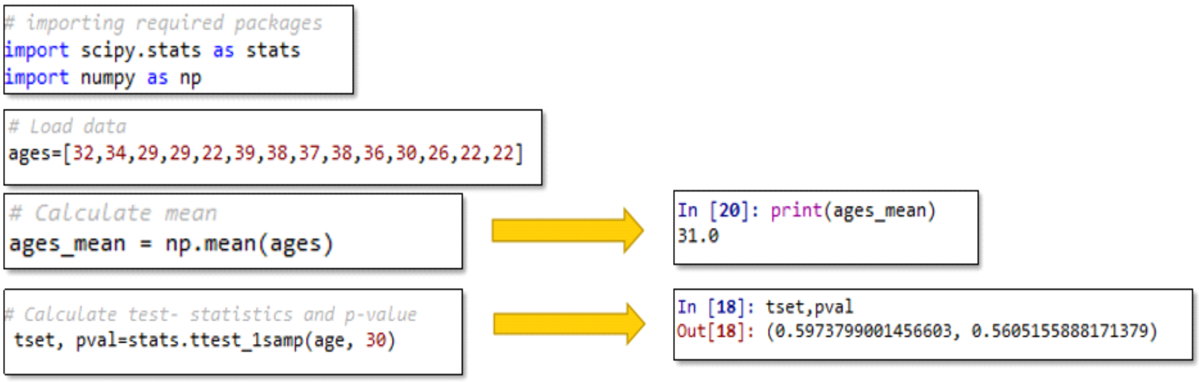
1. **One sample t-test:** The One-sample t test is used to compare a sample mean to a specific value. The One Sample t-test is a parametric test.

Example:-You have 10 ages and you are checking whether average age is 30 or not.

Null hypothesis: μ=30

Alterntaivehypothesis: μ≠30

Scipy implements this as **ttest_1samp**



```
# importing required packages
import scipy.stats as stats
import numpy as np
```

```
# Load data
ages=[32,34,29,29,22,39,38,37,38,36,30,26,22,22]
```

```
# Calculate mean
ages_mean = np.mean(ages)
```

```
In [20]: print(ages_mean)
31.0
```

```
# Calculate test- statistics and p-value
 tset, pval=stats.ttest_1samp(age, 30)
```

```
In [18]: tset,pval
Out[18]: (0.5973799001456603, 0.5605155888171379)
```

Interpretation:-Our t-statistic value is 0.59, and along with our degrees of freedom (n-1; 9) this can be used to calculate a pvalue. The p-value in this case is 0.56, which is greater than the standard thresholds of 0.05 , so we accept the null hypothesis and we can say there is a no statistically significant difference between the mean of sample and population mean.

2. **Two-sample T-test:-**A two sample T-test is used to compare the means of two separate samples.

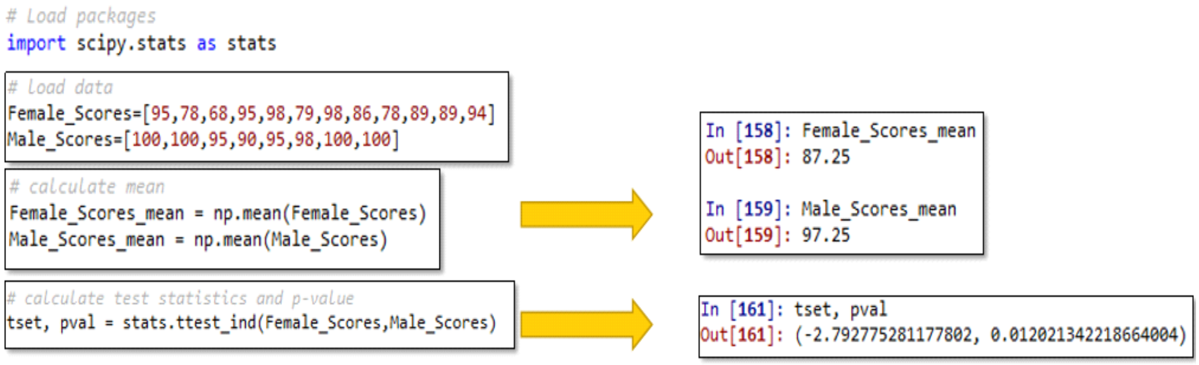
Example: In the population, what is the difference between the female's average score and male's average score on the test.

Null hypothesis: There is no statistically significant difference in the mean score of male and female on the test

Alternative hypothesis: There is a statistically significant difference in the mean score of male and female on the test

Scipy implements this as **ttest_ind()**

ISBN 978-93-80788-93-7



```
# Load packages
import scipy.stats as stats
```

```
# load data
Female_Scores=[95,78,68,95,98,79,98,86,78,89,89,94]
Male_Scores=[100,100,95,90,95,98,100,100]
```

```
# calculate mean
Female_Scores_mean = np.mean(Female_Scores)
Male_Scores_mean = np.mean(Male_Scores)
```

```
In [158]: Female_Scores_mean
Out[158]: 87.25

In [159]: Male_Scores_mean
Out[159]: 97.25
```

```
# calculate test statistics and p-value
tset, pval = stats.ttest_ind(Female_Scores,Male_Scores)
```

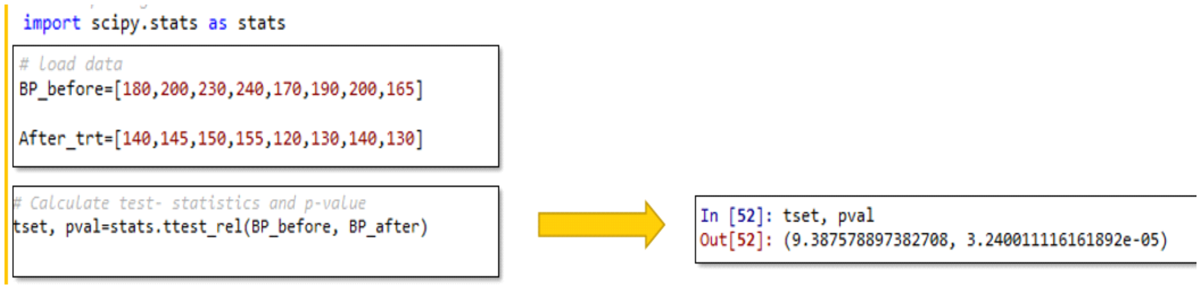
```
In [161]: tset, pval
Out[161]: (-2.792775281177802, 0.012021342218664004)
```

Interpretation: Our t-statistic value is -2.79 i.e. 2.79, and along with our degrees of freedom (18) this can be used to calculate a p-value. The p-value in this case is 0.012, which is less than the standard thresholds of 0.05, so we reject the null hypothesis and we can say there is a statistically significant difference between the means of female score on the test and male score on the test

3. **Paired sampled t-test:** The paired sample t-test is also called dependent sample t-test. It's a univariate test that tests for a significant difference between 2 related variables.

Example: To evaluate the blood pressure of 8 patients before and after treatment.

Null hypothesis:-Mean difference between two samples is 0
Alternative hypothesis:-Mean difference between two sample is not 0
Scipy implements the paired t-test as **ttest_rel()**



```
import scipy.stats as stats
```

```
# load data
BP_before=[180,200,230,240,170,190,200,165]

After_trt=[140,145,150,155,120,130,140,130]
```

```
# Calculate test- statistics and p-value
tset, pval=stats.ttest_rel(BP_before, BP_after)
```

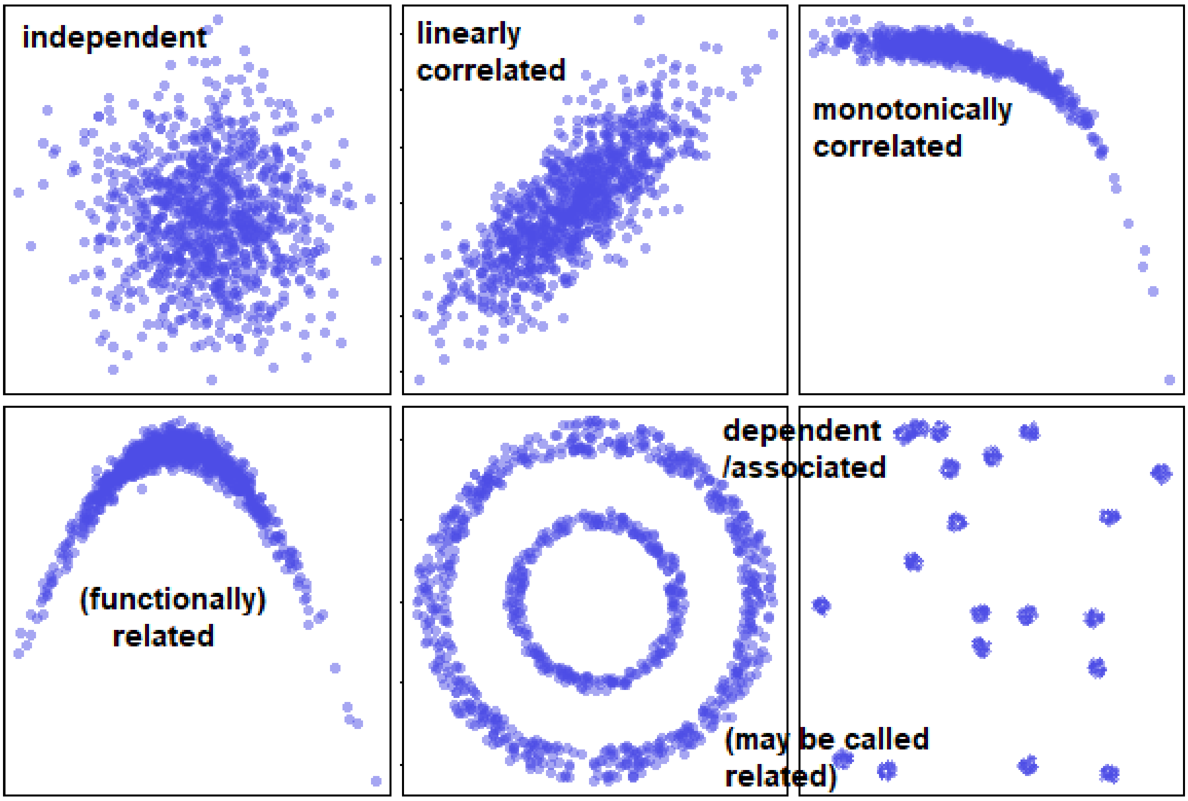
```
In [52]: tset, pval
Out[52]: (9.387578897382708, 3.240011116161892e-05)
```

Interpretation: Our t-statistic value is 9.39, and along with our degrees of freedom (7) this can be used to calculate a p-value.

The p-value is 0.0000324, which is far less than below standard thresholds of 0.05, so we reject the null hypothesis and we can say there is a statistically significant difference in blood pressure readings before and after treatment.

ISBN 978-93-80788-93-7

10.2 Correlation Test

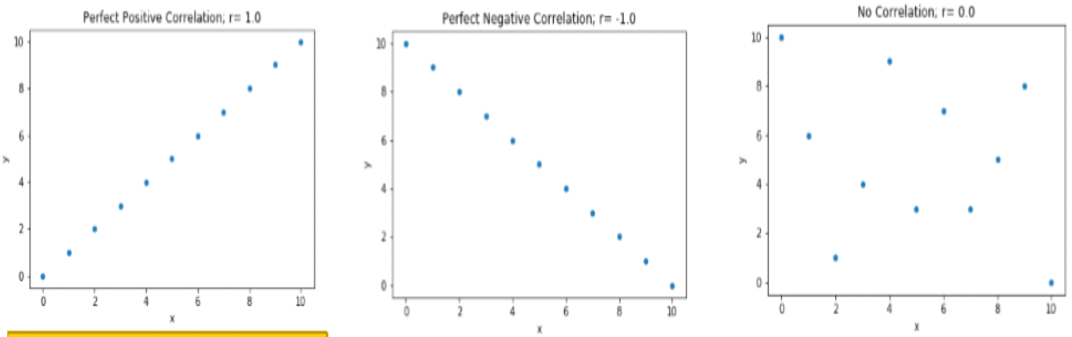


What is correlation test?

Correlation test is used to evaluate the association between two or more variables and that is measured on a -1 to 1 scale.

The closer the correlation value is to -1 or 1 the stronger the relationship, the closer to 0, the weaker the relationship. It measures how change in one variable is associated with change in another variable.

There are a few types of correlation:


Perfect Positive Correlation; r= 1.0

Perfect Negative Correlation; r= -1.0

No Correlation; r= 0.0

Positive correlation: as one variable increases so does the other

Negative correlation: as one variable increases the other variable decreases

No correlation : there is no association between the changes in the two variable

The strength of the correlation matters. The closer the absolute value is to -1 or 1, the stronger the correlation.

Absolute r value	**Strength**
0.0 – 0.3	Weak correlation
0.3 – 0.7	Moderate correlation
0.7 – 1.0	Strong correlation

Correlation test:

There are a few common types of tests to measure correlation, they are:

1. Pearson Correlation (r)

2. Spearman Rank Correlation

Each have their own assumptions about the data that needs to be meet in order for the test to be able to accurately measure the level of correlation. Each type of correlation test is testing the following hypothesis:

Null hypothesis: There is no relationship between variable 1 and variable 2

Alternative hypothesis: There is a relationship between variable 1 and variable 2

Interpretation: If obtained p-value is less than what it is being tested at, then one can state that there is a significant relationship between the variables with an alpha level of 0.05.

1. Assumptions for Pearson Correlation test

Pearson correlation test is a parametric test that makes assumption about the data. In order for the results of a Pearson correlation test to be valid, the data must meet these assumptions:

- The sample is independently and randomly drawn.
- A linear relationship between the two variables is present.
- When plotted, the points form a line and are not curved.
- There is homogeneity of variance

2. Assumptions for Spearman Rank Correlation test

The assumptions for Spearman Rank Correlation test are same. The test is a non-parametric test and the only assumption is there should be a monotonic relationship between the variables being tested.

Correlation Examples:

The data used for correlation test in this example is:

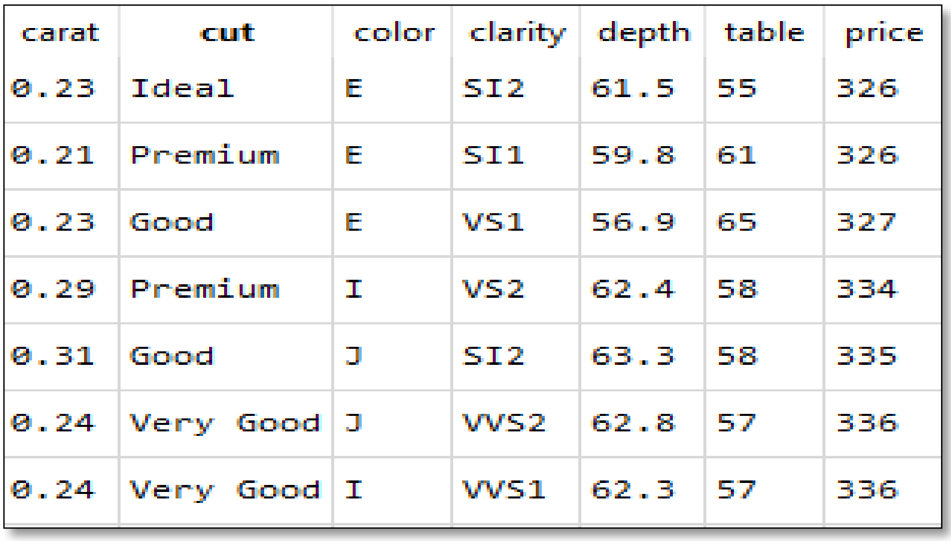
If there is a significant relationship between the carat and price of diamonds.

Null hypothesis: There is no relationship between carat and diamonds

Alternative hypothesis: There is a relationship between carat and diamonds

```
#importing required packages
import pandas as pd
import scipy.stats as stats
```

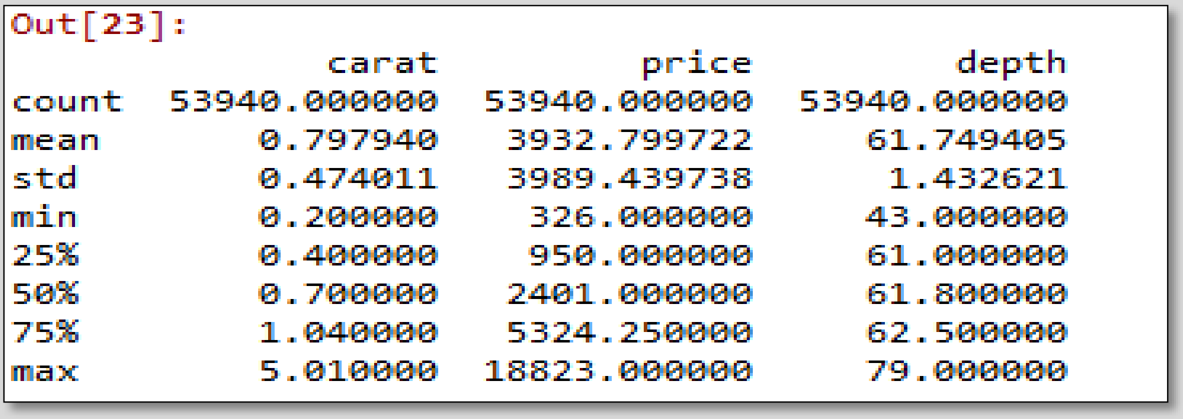
```
# Load data
df=pd.read_csv("D:correlation.csv")
```



carat	cut	color	clarity	depth	table	price
0.23	Ideal	E	SI2	61.5	55	326
0.21	Premium	E	SI1	59.8	61	326
0.23	Good	E	VS1	56.9	65	327
0.29	Premium	I	VS2	62.4	58	334
0.31	Good	J	SI2	63.3	58	335
0.24	Very Good	J	VVS2	62.8	57	336
0.24	Very Good	I	VVS1	62.3	57	336

```
#Descriptive Statistics
df[["carat", "price", 'depth']].describe()
```

```
In [23]: df[["carat", "price", 'depth']].describe()
```



```
Out[23]:
              carat         price         depth
count  53940.000000  53940.000000  53940.000000
mean       0.797940   3932.799722     61.749405
std        0.474011   3989.439738      1.432621
min        0.200000    326.000000     43.000000
25%        0.400000    950.000000     61.000000
50%        0.700000   2401.000000     61.800000
75%        1.040000   5324.250000     62.500000
max        5.010000  18823.000000     79.000000
```

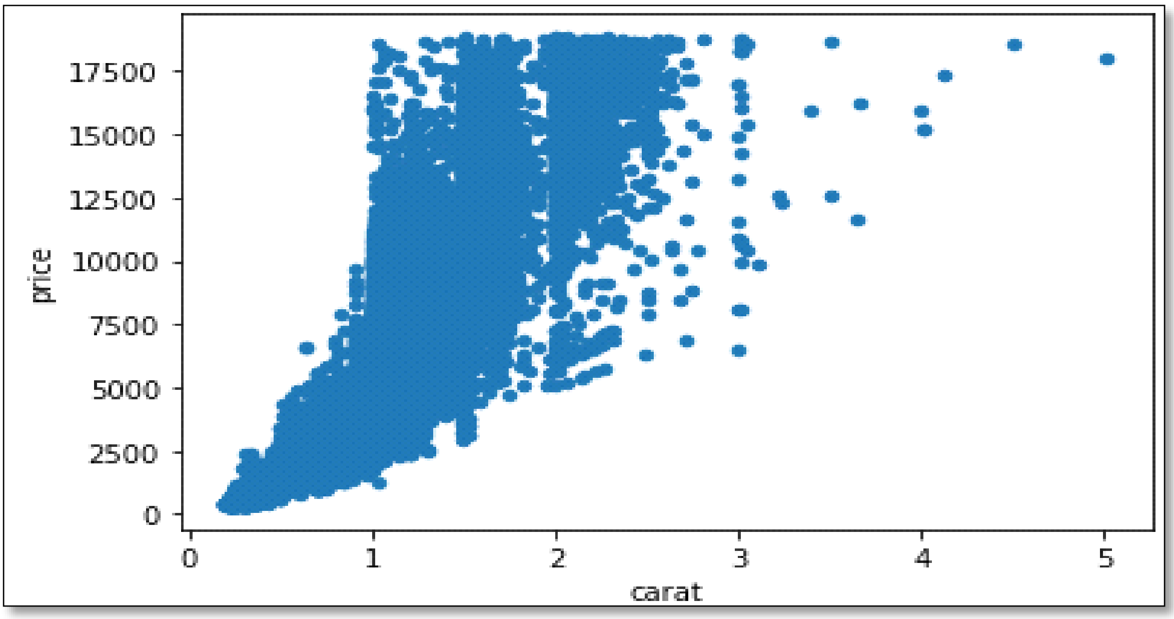
Checking the Assumptions:

➢ **Linear Relationship**

Using built-in method from pandas to plot a scatter plot to look for a linear relationship.

```
#Linearity check
df.plot.scatter("carat","price")
```

ISBN 978-93-80788-93-7



It appears that there is a linear relationship present- as the carat increases so does the price. It is a indication of violating the assumption of homoscedasticity between the variables.

➢ Homogeneity of variances

To formally test homogeneity of variances, use the Levene’s test of homogeneity of variances which is the stats.levene() method from scipy.stats.

```
#homogeneity of variance check
Stats.levene(df[‘carat’],df[‘price’])
Out [28]: LeveneResult(statistic=40965.26627380512, pvalue=0.0)
```

Since p-value is less than alpha level 0.05, the Levene’s test for equal variances is significant, meaning we violate the assumption of homoscedasticity. Given that, the appropriate correlation test to use would be a non-parametric test such as the Spearman rank correlation.

Different methods of calculating the level of correlation between the variables:

1. Pearson correlation method using scipy.stats.pearsonr()

To conduct the Pearson correlation test using scipy.stats, use the .pearsonr() method.

```
#Method-1 using Pearson correlation of scipy function
stats.pearsonr(df['carat'],df['price'])
```

```
#Method-2 using Pearson correlation of scipy function
df[‘carat’].corr(df[‘price’])
```

```
In [35]: stats.pearsonr(df[‘carat’],df[‘price’])
Out[35]: (0.9215913011934768,0.0)
```

The Pearson correlation indicates there is a statistically significant strong relationship between the price and carat of a diamond. This method of measuring correlation is not the measure to use since the data violated the assumption of homoscedasticity of variance.

2. Spearman rank correlation method using scipy.stats.spearmanr()

Now to conduct non-parametric measure of correlation, which is, better to the relationship between the carat and price of a diamond. To do this using scipy.stats, use the .spearmanr() method.

```
#Method-1 using spearman correlation of scipy function
Stats.spearmanr(df['carat'],df['price'])
```

```
#Method-2 using spearman correlation of scipy function
df['carat'].corr(df['price'],method= 'spearman')
```

```
In [33]: stats.spearmanr(df['carat'], df['price'])
Out [33]: SpearmanrResult(correlation=0.962882798813001, pvalue=0.0)
```

The Spearman rank correlation method indicates that the correlation is strong and significant between the size of the carat and the price of the diamond.

3. Kendall Tau correlation method using scipy.stats.kendalltau()

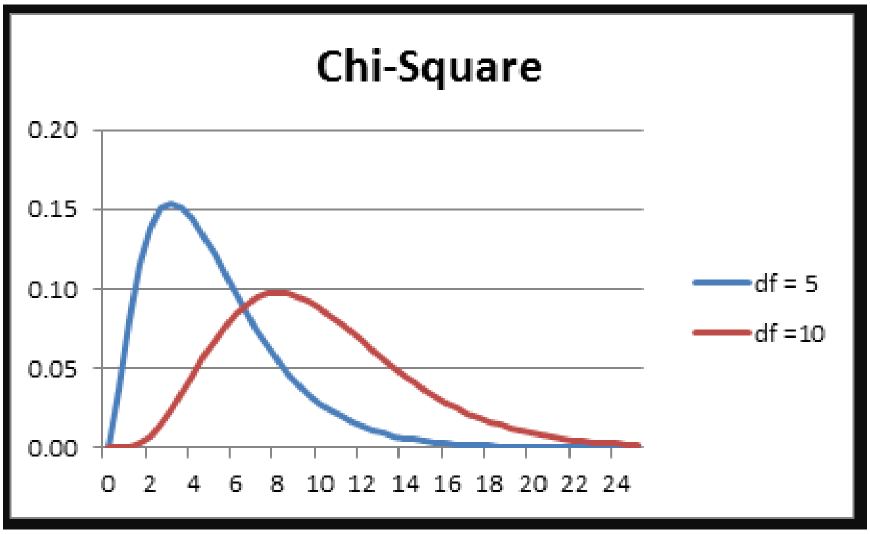
To conduct the Kendall Tau correlation measure using scipy.stats, use the .kendalltau() method.

```
#using Kendall Tau correlation of scipy function
stats.kendallutau(df['carat'],df['price'])
```

```
In [34]: stats.kendallutau(df['carat'],df['price'])
Out [34]: KendalltauResult(correlation=0.8341049107108127, pvalue=0.0)
```

Using this method, the test indicates that there is a strong, significant correlation between the size of the carat and the price of the diamond.

10.3 Chi-Square (χ2) Test



The Chi-Square test of independence is a statistical test to determine if there is a significant relationship between 2 categorical variables. In simple words, the Chi-Square statistic will test whether there is a significant difference in the observed vs the expected frequencies of both variables. It's typically used with categorical data such as educational attainment, colors, or gender.

Rules to use the Chi-Square Test:

1. Variables are Categorical
2. Frequency is at least 5
3. Variables are sampled independently (The groups being tested must be independent)

The 5 steps in Chi-Square Test:

1. **State the hypothesis**

Null Hypothesis: There is no relationship between variable one and variable two.
Alternative Hypothesis: There is a relationship between variable one and variable

2. **Check the conditions**

All the expected counts should >1. Atleast 80% expected counts should >5.

3. **Calculate Test statistic and p-value.**

$$Expected = \frac{Row\ Total\ \times Column\ Total}{Overall\ \ Total}$$

$$\chi^2 = \sum \frac{(Observed - Expected)^2}{Expected}$$

$$DF = (Rows - 1) \times (Columns - 1)$$

4. **Decide whether or not the results are statistically significant.**

The results are statistically significant if the p-value is less than alpha, where alpha is the Significance level (usually α =0.05).

5. **Interpret results.**

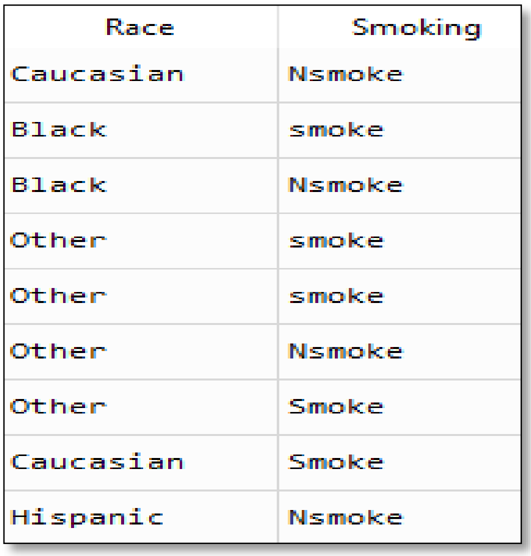
P-value less than 0.05, Reject Null Hypothesis & Accept Alternative Hypothesis.
P-value greater than 0.05, Fail to Reject Null Hypothesis & Reject Alternative Hypothesis.

Example of Chi-square using Python:

To check Is there is any relationship between Race and Smoking?
The data that resulted from the survey is summarized in the following table:

```
#Load Packages
import pandas as pd
from scipy import stats
```

```
#Load data file
df_chi =pd.read_csv("D:chi-sq.csv")
```



Race	Smoking
Caucasian	Nsmoke
Black	smoke
Black	Nsmoke
Other	smoke
Other	smoke
Other	Nsmoke
Other	Smoke
Caucasian	Smoke
Hispanic	Nsmoke

Solution
Null hypothesis: There is no relationship between Race and Smoking.
Alternative hypothesis: There is a relationship between Race and Smoking

Race has more than 2 values.

Chi-square Test of Independence using crosstab() theSciPy function:

After importing data next step is to format the data into a frequency count table. This is called a **Contingency Table**, we can accomplish this by using the pd.crosstab() function in pandas. To perform this you should have already imported Scipy.stats package. The *chi2_contingency()* method conducts the Chi-square test on a contingency table (crosstab)

```
#contingency table
contingency_table=pd.crosstab(df_chi[“Race”],df_chi[“Smoking”])
```

```
In [20]: print(‘contingency_table :-\n’contingency_table)
Contingency_table:-
Smoking      Nsmoke     Smoke
Race
 Black      248         41
Caucasian   628         75
hispanic    138         29
other       198         38
```

```
# Observed Values
Observed_Values = contingency_table.values
```

```
In [23]: print(“observed Values :-\n”, observed_values)
Observed Values :-
[[240 41]
[620 75]
[130 29]
[190 38]]
```

Chi-square Test of Independence using chi2_contingency() theSciPy function:

Chi-square test of independence of variables in a contingency table. This function computes the chi-square statistic and p-value for the hypothesis test of independence of the observed frequencies in the contingency table observed. The expected frequencies are computed based on the marginal sums under the assumption of independence.

```
#Method-1 Calculate Chi-Square Statistics
chisq_output=stats.chi2_contingency(contingency_table)
```

```
In [29]: print(“chi_Squre_output :-\n”,chisq_output)
Chi_square_output:-
(9.787819942992163,8.821225883679331798,3,array([[243.27219369,37.72788631],
      [601.68745415, 93.31254585],
      [137.65223771, 21.34776229],
      [197.38811445, 30.61188555]]))
```

```
# Expected Values
ExpectedValues = chisq_output[3]
print (“chi square output :-\n”chisq_output)
```

Expected values is calculated for chi-square statistics value in method-2

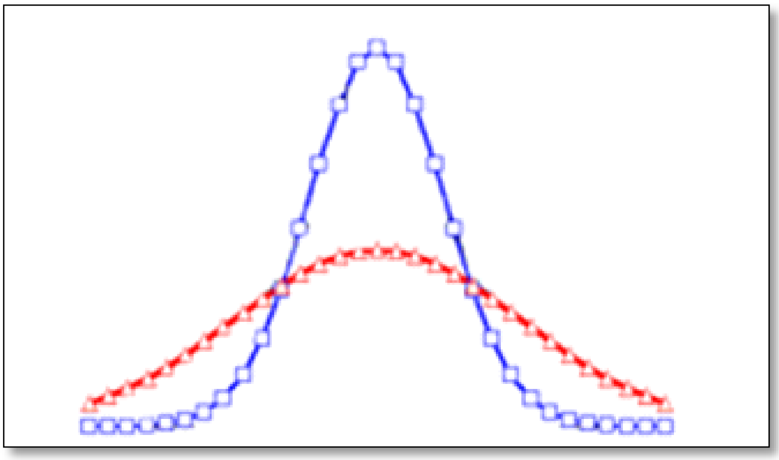
```
In [38]: print(‘chi_square_statistics :-\n’,chi_square_stat)
chi_squre_statistics :-
9.707019942092163
```

```
Method-2 Calculate Chi-Square Statistics
chi_squared_stat = (((Observed_Values-Expected_Values)**2)/Expected_Values)
                  .sum().sum()
```

Interpretation: The first value (9.707) is the Chi-square value, followed by the p-value (0.021), then comes the degrees of freedom (3), and lastly it outputs the expected frequencies as an array. Since all of the expected frequencies are greater than 5, the chi^2 test results can be trusted. We can reject the null hypothesis as the p-value is less than 0.05. Thus, the results indicate that there is a association between different group of Race and Smoking.

ISBN 978-93-80788-93-7

10.4 Normality Tests



Testing for Normality — Applications with Python:

A normality test is a statistical process used to determine if a sample or any group of data fits a standard normal distribution. A normality test can be performed mathematically or graphically.

Assumption for normality:

If Data Is Gaussian:

Use Parametric Statistical Methods

Else:

Use Nonparametric Statistical Methods

So you have a dataset and you're about to run some test on it but first, you need to check for normality.

For Example:

Parametric Test	Non-Parametric Test
Paired t-test	Wilcoxon Rank sum Test
One way Analysis of variance Anova	Krushkal Wallis Test

There are a range of techniques that you can use to check if your data sample deviates from a Gaussian distribution (Normal distribution). Below are the three tests you might want to consider to check the normality on test data:

- The Shapiro-Wilk test;
- The Anderson-Darling test, and;
- The Kolmogorov-Smirnov test.

As well, there are some visual measures to be implemented:

- Histogram
- QQ Plots

ISBN 978-93-80788-93-7

Shapiro-Wilk Test:
The Shapiro-Wilk test evaluates a data sample and quantifies if a random sample came from a normal distribution. The Shapiro-Wilk test is believed to be a reliable test of normality, although there is some suggestion that the test may be suitable for smaller samples of data, e.g. thousands of observations or fewer.

Null hypothesis: The data is normally distributed.
Alternative hypothesis: The data is not normally distributed.

Checking normality of data using Shapiro-Wilk test in python

The shapiro() SciPy function will calculate the Shapiro-Wilk on a given dataset. The function returns both the statistic value calculated by the test and the p-value.

```
# import requires libraries
import pandas as pd
from scipy.stats import shapiro
```

```
#sample data set
Data= { 'salary': [100,200,500,300,600,100],
'age': [20,21,26,22,23,24],
'rating': [2.3,4.3,5.0,2.3,4.5,3.5]}
```

```
# load the data to data frame
df=pd.Dataframe(data)
```

```
In [36]: print(df)
      Salary      age         rating
0     100         20          2.3
1     200         21          4.3
2     500         26          5.0
3     300         22          2.3
4     600         23          4.5
5     100         24          3.5
```

```
# Assigning x Variable
x=(df[‘salary’])
```

```
In [22]: print(x)
0     100
1     200
2     500
3     300
4     600
5     100
```

```
# normality test                      In [19]: print(stat)
stat, p=shapiro(x)                    0.8904141783714294
In [20]: print(p)
```

ISBN 978-93-80788-93-7

```
0.320363849401474
```

The example first calculates the test on the data sample, then prints the statistic and calculated p-value.

```
#interpret
alpha= 0.05
if p > alpha:
      print('Sample looks Gaussian (fail to reject H0)')
else:
      print('sample does not look Gaussian (reject H0)')
```

```
Sample looks Gaussian(fail to reject H0)
```

Interpretation: The p-value returned is greater than 0.05 and finds that the data is likely drawn from a Gaussian distribution (Normal distribution).

Anderson-Darling Test:

The test is a modified version of a more sophisticated nonparametric goodness-of-fit statistical test called the **Kolmogorov-Smirnov test**. A feature of the Anderson-Darling test is that it returns a list of critical values rather than a single p-value. This can provide the basis for a more thorough interpretation of the result.

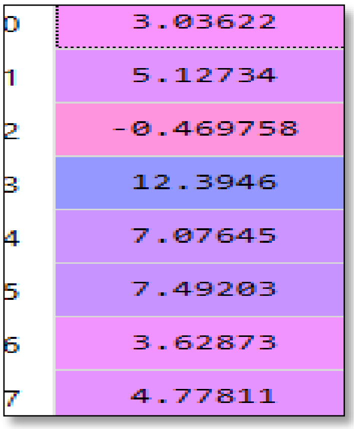
Null hypothesis: The data is normally distributed.
Alternative hypothesis: The data is not normally distributed.

Checking normality of data using Anderson Darling test for a variable in python

The **anderson() SciPy function** implements the Anderson-Darling test. It takes as parameters the data sample and the name of the distribution to test it against. By default, the test will check against the Gaussian distribution (dist='norm').

```
# Load require package
import scipy.stats as stats
```

```
# Create the random varibles with mean 5, and sd 3
x_50 = stsats.norm.rvs(loc=5, scale=3, size=50)
```



0	3.03622
1	5.12734
2	-0.469758
3	12.3946
4	7.07645
5	7.49203
6	3.62873
7	4.77811

This example calculates the statistic on the test data set and prints the critical values.

```
# Perform the AD test against a normal distribution with
# mean=5 and std=5
anderson_results_50= stats.anderson(x_50, dist=’norm’)
```

```
In [49]: anderson_results_50
Out[49]:AndersonResult(statistic=0.2166625962964872,critical_values   =   array
([0.538, 0.613, 0.736,0.858, 1.021]),significance_level=array([15.  , 10.  , 5.
, 2.5 , 1. ]))
```

Critical values in a statistical test are a range of pre-defined significance boundaries at which the H0 can be failed to be rejected if the calculated statistic is less than the critical value. Rather than just a single p-value, the results give us the p-values for various significance levels [15. , 10. , 5. , 2.5, 1.] so if you’re working with boundaries outside of .05 you can see those results as well.

Here our p-value for .05 is outside the rejection region of 0.736, meaning we cannot reject the null hypothesis our data comes from a normal distribution.

Interpretation: We can interpret the results by failing to reject the null hypothesis that the data is normal since our calculated test statistic is less than the critical value at a chosen significance level. We can see that at each significance level, the test has found that the data follows a normal distribution

Visualization of data using Histogram Plot:

A simple and commonly used plot to quickly check the distribution of a sample of data is the histogram. The plot shows the bins across the x-axis maintaining their ordinal relationship, and the count in each bin on the y-axis. A histogram can be created using the **hist() matplotlib function**. By default, the number of bins is automatically estimated from the data sample.

The below example demonstrate the histogram plot on the test problem:

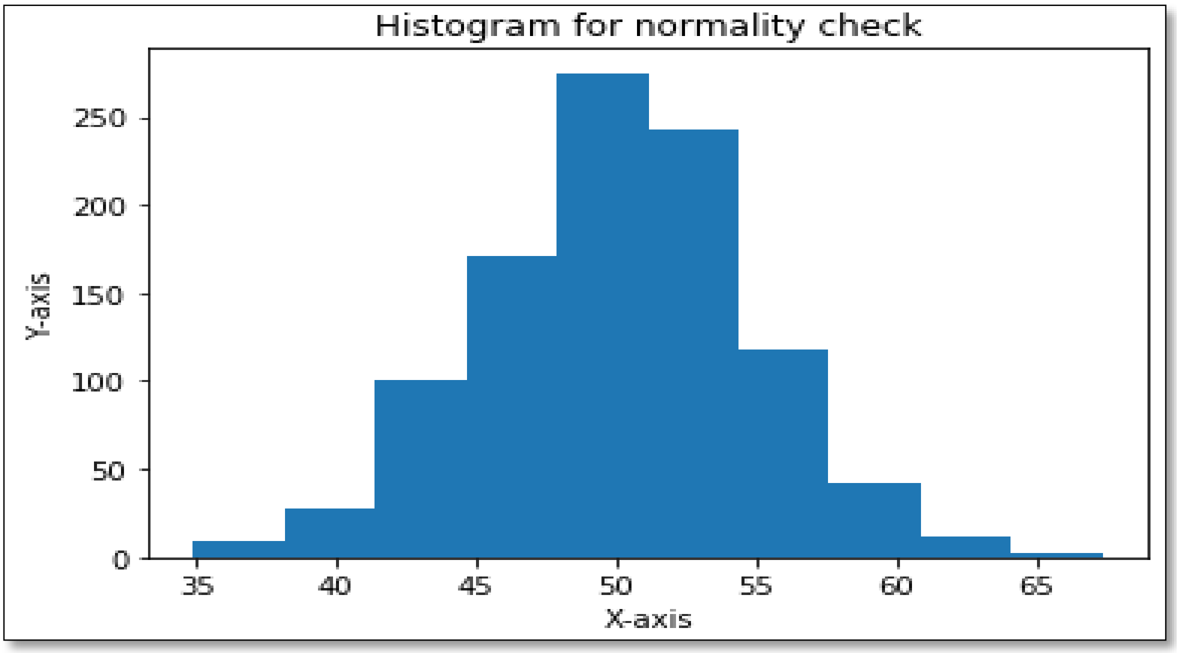
```
#import required packages for plot
from numpy.random imort randn
from matplolib import pyplot
```

```
# generate univariate observations
Date=5*randn(1000)+50
```

```
#givig a title to plot
pyplot.title(“history for normality check”)

# naming x-axis
pyplot.xlabel(‘x-axis’)

# naming x-axis
pyplot.ylabel(‘y-axis’)
```

```
#history plot
pyplot.hist(data)
```

```
# to view the plot
pyplot.show()
```



This is the histogram plot showing the number of observations in each bin. We can see a Gaussian-like shape to the data, that although is not strongly the familiar bell-shape, is a rough approximation.

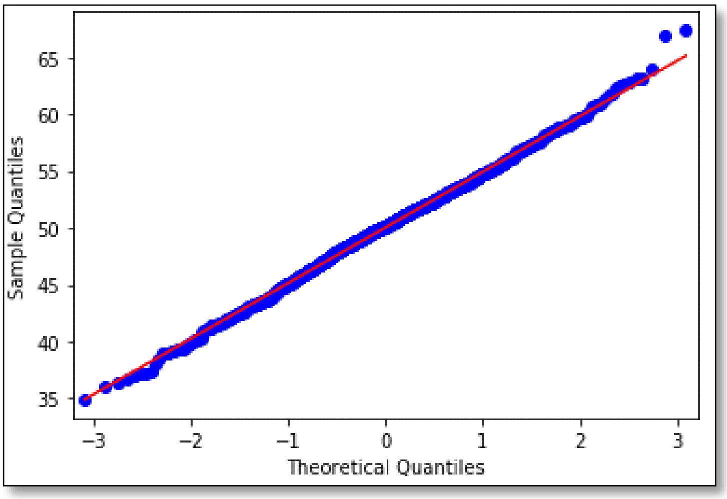
Visualization of data using QQ plots (Quantile-Quantile Plot):

Another popular plot for checking the distribution of a data sample is the quantile-quantile plot, Q-Q plot, or QQ plot for short. We can develop a QQ plot in Python using the qqplot() statsmodels function. The function takes the data sample and by default assumes we are comparing it to a Gaussian distribution. We can draw the standardized line by setting the ‘line‘ argument to ‘s‘.

Below is the example of plotting the test dataset as a QQ plot :

```
# import required package for plot
from numpy.random import randn
from matplotlib import pyplot
from statsmodels.graphics.gofplots import qqlot
```

```
# generate univariate observation      # Q-Q plot
Data = 5 * randn(100) + 50             qqplot(data, line=’s’)
                                       pyplot.show()
```



Running the example creates the QQ plot showing the scatter plot of points in a diagonal line, closely fitting the expected diagonal pattern for a sample from a Gaussian distribution. There are a few small deviations, especially at the bottom of the plot, which is to be expected given the small data sample.

10.5 References

1. https://www.youtube.com/watch?v=dPXBN8ms-cU
2. https://towardsdatascience.com/inferential-statistics-series-t-test-using-numpy-2718f8f9bf2f
3. https://data-flair.training/blogs/python-statistics/

Chapter 11
ANOVA Procedure: One Way and Two Way

***Dr. Asha Jindal**, Associate Professor and Head, Department of Statistics,*
K. C. College

11.1 Introduction

Analysis of Variance (ANOVA) is used when there are more than two populations to compare. ANOVA provides a means of comparing the variation within each subset or treatment of data to the variation between the different subsets of data. The between subset variation is a reflection of the possible differences between the subset averages. The within subset variation, for each subset, is a reflection of the inherent variation observed when sampling from the subset repeatedly. ANOVA is the statistical model that you use to predict a continuous outcome on the basis of one or more categorical predictor variables.

Assumptions:
Analysis of Variance assumes that

1. The observations on Y are independent.
2. Populations being sampled are normal.
3. Populations being sampled have equal variances.

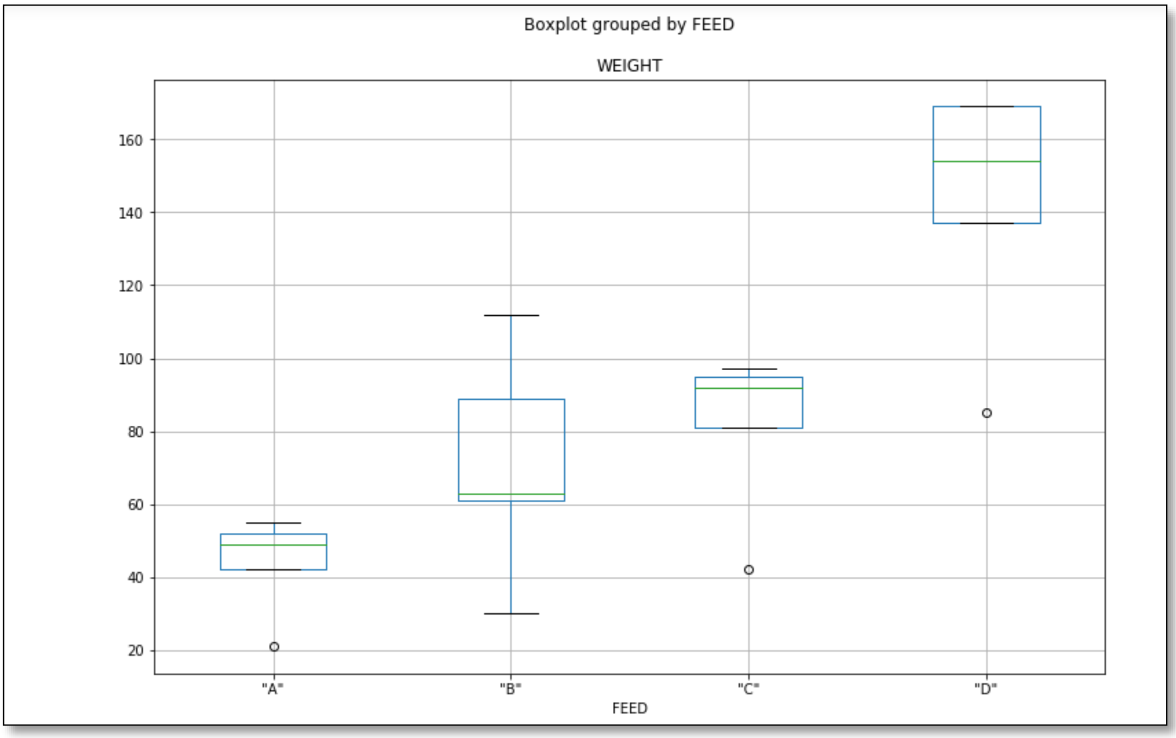
11.2 One Way ANOVA using Python

One Way ANOVA or One-Factor ANOVA as a Linear Model. An equivalent way to express the one-factor model is to say that treatment j came from a population with a common mean (μ) plus a treatment effect (α_j) plus random error (e_{ij}):
$y_{ij} = \mu + \alpha_j + e_{ij}$, j = 1, 2, …, c and i = 1, 2, …, n
Random error is assumed to be normally distributed with zero mean and the same variance for all treatments.

I have used **babychicks.csv** which has data on feed and weight. Weight is dependent variable and a type of Feed is factor.

```
In [1]: import pandas as pd
      data=pd.read_csv("C:/User/Admin/Desktop/babychicks.csv")
      data.head(5)      ## to check first 5 rows of data along with heading
Out [1]:    WEIGHT      FEED
      0     55          'A'
      1     49          'A'
      2     42          'A'
      3     21           'A'
      4     52           'A'
```

```
In [2]: data.boxplot(‘WEIGHT’, BY=’FEED’, figsize=(12,8)).show()
                                #To draw Boxplot to check normality
```



```
In [3]: import statsmodels.api as sm
      from statsmodels.formula.api import ols
      mod = ols('WEIGHT~FEED', data=data).fit()
      aov_tables = sm.stats.anova_lm(mod, typ=2)
      print(aov_table)
                  sum_sq            df          F           PR(>F)
FEED              26234.95          3.0         12.10504    0.000218
Residual          11558.80          16.0        NaN         NaN
```

```
In [4]: ### type choice adds the mean square column in ANOVA
      aov_table = sm.stats.anova_lm(mod,typ=1)
      print(aov_table)
            df    sum_sq      mean_sq           F           PR(>F)
FEED        3.0   26234.95    8744.983333       12.10504    0.000218
Residual    16.0  11558.80    722.425000        NaN         NaN
```

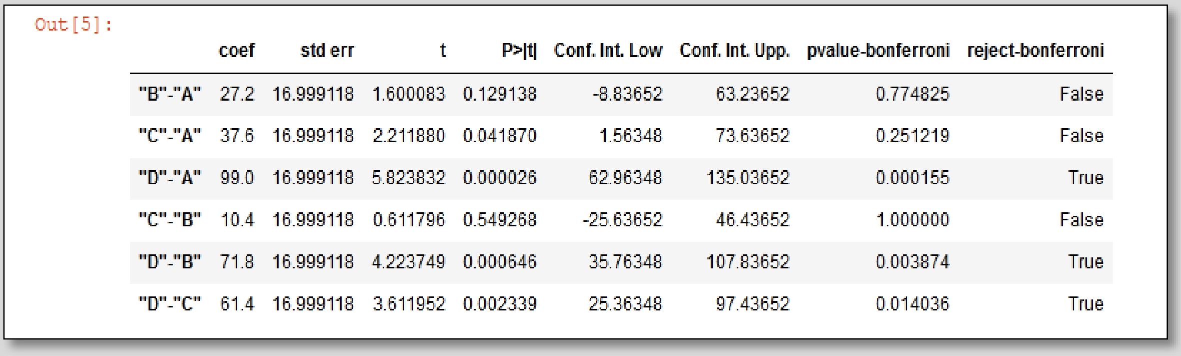
p-value= 0.000218< 0.05 indicates significant difference between four types of feed. Post hoc analysis Bonferroni is used to answer which feed differs from others.

# General coding for pairwise analysis syntax

Method 1: Bonferroni:

```
In[5]:  pair_t = mod.t_test_pairwise('FEED',method='bonferroni')
```

```
        pair_t.result_frame
```



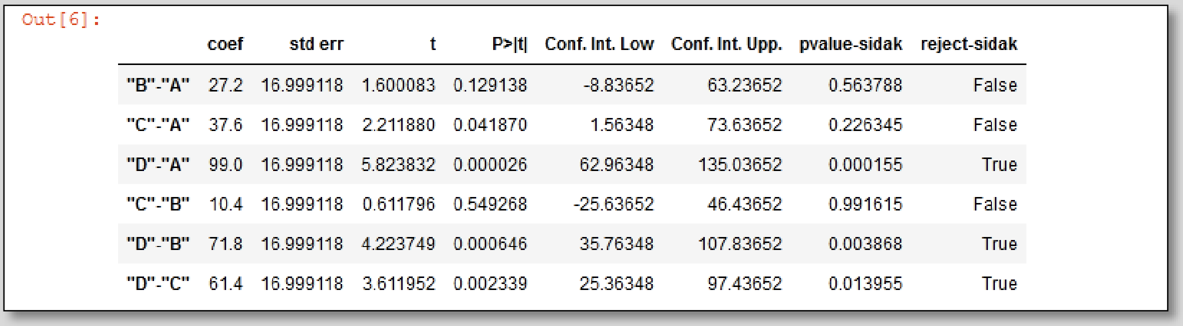
Out[5]:

	coef	std err	t	P>|t|	Conf. Int. Low	Conf. Int. Upp.	pvalue-bonferroni	reject-bonferroni
"B"-"A"	27.2	16.999118	1.600083	0.129138	-8.83652	63.23652	0.774825	False
"C"-"A"	37.6	16.999118	2.211880	0.041870	1.56348	73.63652	0.251219	False
"D"-"A"	99.0	16.999118	5.823832	0.000026	62.96348	135.03652	0.000155	True
"C"-"B"	10.4	16.999118	0.611796	0.549268	-25.63652	46.43652	1.000000	False
"D"-"B"	71.8	16.999118	4.223749	0.000646	35.76348	107.83652	0.003874	True
"D"-"C"	61.4	16.999118	3.611952	0.002339	25.36348	97.43652	0.014036	True

Groups D and A, D and B as well as D and C differ significantly.
Same results are demonstrated using Sidak method.

Method 2: Sidak:

```
In[6]:  pair_t = mod.t_test_pairwise('FEED',method='sidak')
        pair_t.result_frame
```



Out[6]:

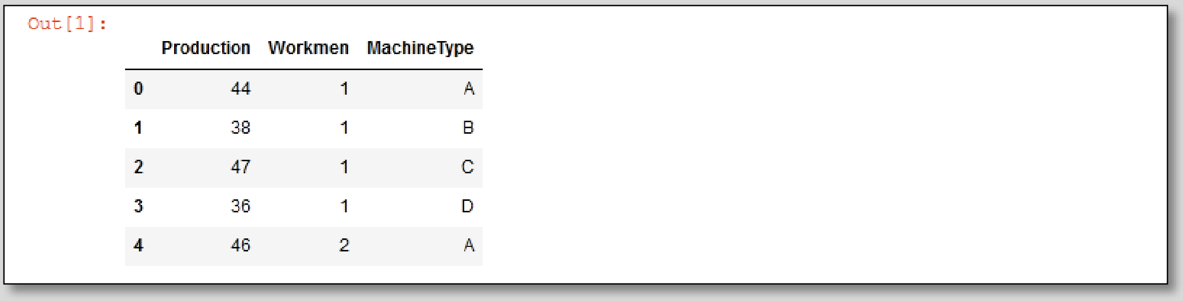
	coef	std err	t	P>|t|	Conf. Int. Low	Conf. Int. Upp.	pvalue-sidak	reject-sidak
"B"-"A"	27.2	16.999118	1.600083	0.129138	-8.83652	63.23652	0.563788	False
"C"-"A"	37.6	16.999118	2.211880	0.041870	1.56348	73.63652	0.226345	False
"D"-"A"	99.0	16.999118	5.823832	0.000026	62.96348	135.03652	0.000155	True
"C"-"B"	10.4	16.999118	0.611796	0.549268	-25.63652	46.43652	0.991615	False
"D"-"B"	71.8	16.999118	4.223749	0.000646	35.76348	107.83652	0.003868	True
"D"-"C"	61.4	16.999118	3.611952	0.002339	25.36348	97.43652	0.013955	True

11.3 Two Way ANOVA using Python

A **two-way ANOVA** test is a statistical test used to determine the effect of **two** nominal predictor (2 Factors) variables on a continuous outcome(dependent) variable. **ANOVA** stands for analysis of variance and tests for differences in the effects of independent variables on a dependent variable.

I have used **Production.csv** which has data on Machine Type, Workmen and Production. Production is dependent Variable and Machine Type and type of Workmen are two factors.

```
In [1]: import pandas as pd
      data1=pd.read_csv(“C:/User/Admin/Desktop/Production.csv”)##File Path
            data.head(5)                               ## First 5 rows
```



Out[1]:

	Production	Workmen	MachineType
0	44	1	A
1	38	1	B
2	47	1	C
3	36	1	D
4	46	2	A

ISBN 978-93-80788-93-7

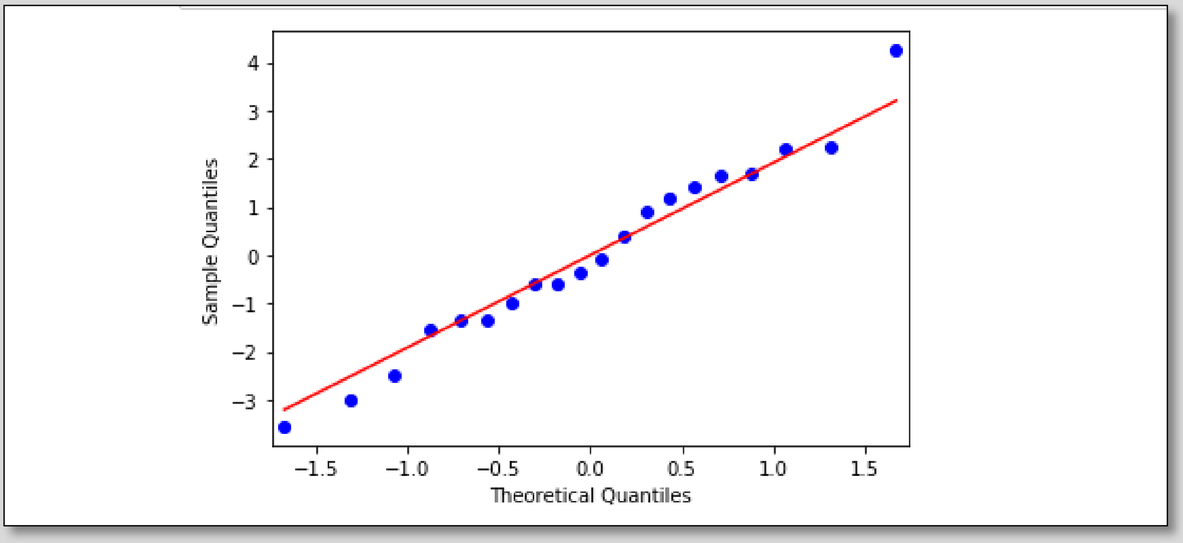
```
In [2]: import statsmodels.api as sm
      from statsmodels.formula.api import ols
      formula = 'Production~C(Workmen)+C(MachineType)'
      model = ols(formula,data1).fit()
      aov_table = sm.stats.anova_lm(model,typ=2)
      print(aov_table)
                  sum_sq      df          F           PR(>F)
C(Workmen)        201.5       4.0         8.202171    0.001990
C(MachineType)    353.8       3.0         19.202171   0.000071
Residual          73.7        12.0        NaN         NaN
```

```
In [3]: aov_table=sm.stats.anova_lm(model,typ=1)
        print(aov_table)
                        df     sum_sq      mean_sq     F           PR(>F)
C(Workmen)              4.0    201.5       50.375000   8.202171    0.001990
C(MachineType)          3.0    353.8       117.933333  19.202171   0.000071
Residual                12.0   73.7        6.141667    NaN         NaN
```

Both the P value < 0.05 indicates significant difference between five types of workmen and four types of machine.

We can also do some diagnostics. It is to show the linear model fitted with the OLS method and get a Quantile-Quantile (QQplot).

```
In [4]: import matplolib.pyplot as plt
      res=model.resid
      fig=sm.qqplot(res, line='s')
      plt.show()
```



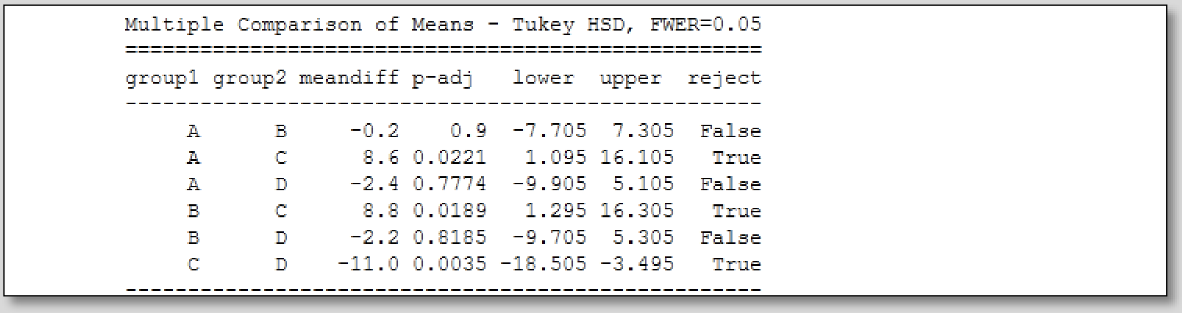
Above Q-Q plot shows normality.

Post-hoc Testing:

There are a few different methods of post-hoc testing to find a difference between groups of factors. I will show how to use Tukey's HSD. We have to test for difference for each factor separately. To Carry out Post Hoc Analysis one has to install statsmodel library and steps are as follows:

1. Go to search in start button and type Anaconda Prompt which will open the window.
2. Type pip install statsmodels and press Enter key
3. Installation process will begin and it will complete in approximately 2 minutes.

```
In[5]:mc=statsmodels.stats.multicomp.MultiComparison(data1[‘Producion’],
data1[‘MachineType’])
mc_results = mc.tukeyshsd()
print(mc_results)
```

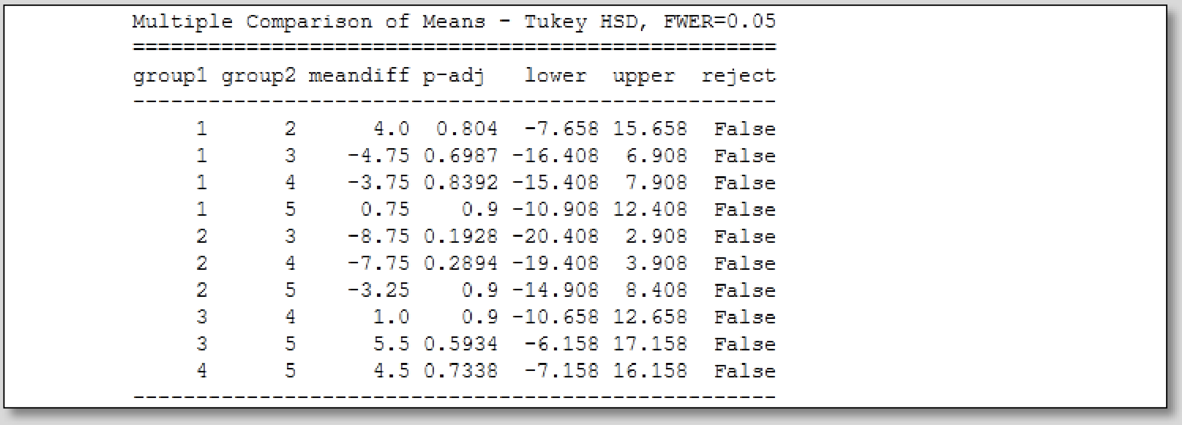


```
Multiple Comparison of Means - Tukey HSD, FWER=0.05
===================================================
group1 group2 meandiff p-adj   lower  upper  reject
---------------------------------------------------
     A      B     -0.2    0.9  -7.705  7.305  False
     A      C      8.6 0.0221   1.095 16.105   True
     A      D     -2.4 0.7774  -9.905  5.105  False
     B      C      8.8 0.0189   1.295 16.305   True
     B      D     -2.2 0.8185  -9.705  5.305  False
     C      D    -11.0 0.0035 -18.505 -3.495   True
---------------------------------------------------
```

Above table shows Machine Types Aand C, B and C as well as C and D differ significantly.

```
In [6]: ## Converting workmen from float to string
      data1[‘Workmen’]=data1[‘Workmen’].astype(str)
```

```
In [7]:mc=statsmodels.stats.multicomp.MultiComparison(data1[‘Production’]
      ,data1[‘Workmen’])
      mc_results=mc.tukeyhsd()
      print(mc_results)
```



```
Multiple Comparison of Means - Tukey HSD, FWER=0.05
===================================================
group1 group2 meandiff p-adj   lower   upper  reject
---------------------------------------------------
     1      2      4.0  0.804  -7.658 15.658  False
     1      3    -4.75 0.6987 -16.408  6.908  False
     1      4    -3.75 0.8392 -15.408  7.908  False
     1      5     0.75    0.9 -10.908 12.408  False
     2      3    -8.75 0.1928 -20.408  2.908  False
     2      4    -7.75 0.2894 -19.408  3.908  False
     2      5    -3.25    0.9 -14.908  8.408  False
     3      4      1.0    0.9 -10.658 12.658  False
     3      5      5.5 0.5934  -6.158 17.158  False
     4      5      4.5 0.7338  -7.158 16.158  False
---------------------------------------------------
```

Above table shows all workmen group does not differ significantly.

11.4 References

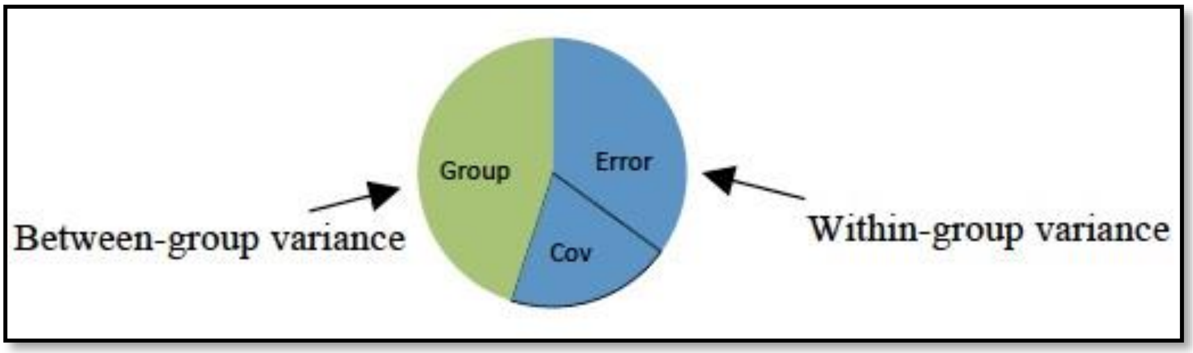
1. https://pythonfordatascience.org/anova-2-way-n-way/#comparison
2. https://www.marsja.se/four-ways-to-conduct-one-way-anovas-using-python/
3. Python Tutorial: https://www.py4e.com/lessons
4. https://chrisalbon.com

Chapter 12
ANCOVA Procedure

***Mr. Shubham Niphadkar**, Assistant Professor, Department of Statistics,*
K. C. College

12.1 What is Analysis of Covariance?

Analysis of covariance is a mixture of Analysis of variance and Regression. For performing ANCOVA, data must consist of a dependent variable, categorical independent variable and continuous independent variables. The categorical independent variable is called as treatment and continuous independent variables are called as covariates. Generally, dependent variable is denoted by **'DV',** treatment or categorical independent variable by **'IV'** and covariates by **'CV'.** The main aim of ANCOVA is to test whether the means of a dependent variable are same across all levels or categories of treatment, while statistically controlling the effects of all covariates which are not of interest, at least primarily. In ANCOVA, the variance in the dependent variable (DV) is split into the variance explained by treatment or independent variable (IV), variance explained by covariates (CV) and residual variance.



The ANCOVA assumes linear relationship between the dependent variable (DV) and covariate (CV).

12.2 Assumptions of ANCOVA

Analysis of Covariance has several underlying assumptions. The assumptions of standard linear regression also hold. The assumptions are given as follows:

1. Linearity of regression
2. Independence of error terms
3. Homogeneity of regression slopes
4. Normality of error terms
5. Homogeneity of error variances

ISBN 978-93-80788-93-7

Linearity of regression:
The relationship between the dependent variable and the associated variables must be linear.
Independence of error terms:
The error terms must not be correlated. The variance-covariance matrix of the error term must be diagonal.
Homogeneity of regression slopes:
The slopes of regression lines must be equal across all the levels of treatment. The regression lines must be parallel among all levels.
Normality of error terms:
The distribution of error terms must be normal.
Homogeneity of error variances:
The variances of the error term must be same for different categories or levels of treatments. We can perform Analysis of Covariance (ANCOVA) and from the p-value we can conclude whether the means of a dependent variable are same across all levels or categories of treatment, while statistically controlling the effects of all covariates. We will use 'pingouin' package. We need to install this package using pip command ***pip install pingouin***

12.3 Examples

1. Consider a dataset **"LoanData"**. This data consist of 3 variables, viz. Education, ApplicantIncome and LoanAmount. The observations were recorded for 432 people on each variable. The variable Education represents whether or not the applicant has graduated high school.ApplicantIncome is the income of the applicant. LoanAmount is the amount of loan which applicant wants to borrow.

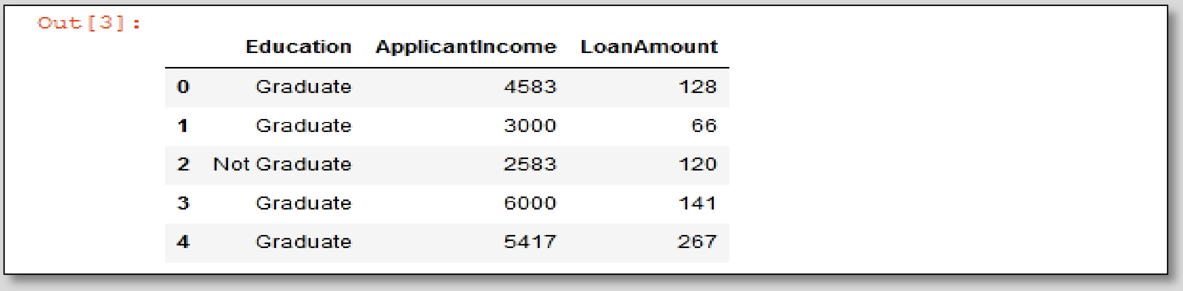
Objective: Our aim is to examine how applicant's loan amount varies with their education and income.
Analysis:

```
In [1] : import numpy as np
      import pandas as pd
      import scipy.stats as stats
      import matplotlih.pyplot as plt
      import statsmodels
      import statimodels.api as sm
      from statsmodels.formula.api
      import ols
      from statsmodels.grapbics.gofplots import qqplot
      from pingouin import ancova
```

```
In [2]: #Importing dataset
      df=pd.read_csv("D:/ANCOVA Pyhon/LoanData.csv")
      print(list(df))
      ['Education', 'ApplicatIncome','LoanAmount']
```

We can see that there are 3 columns in dataset, viz., Education, ApplicantIncome and LoanAmount.

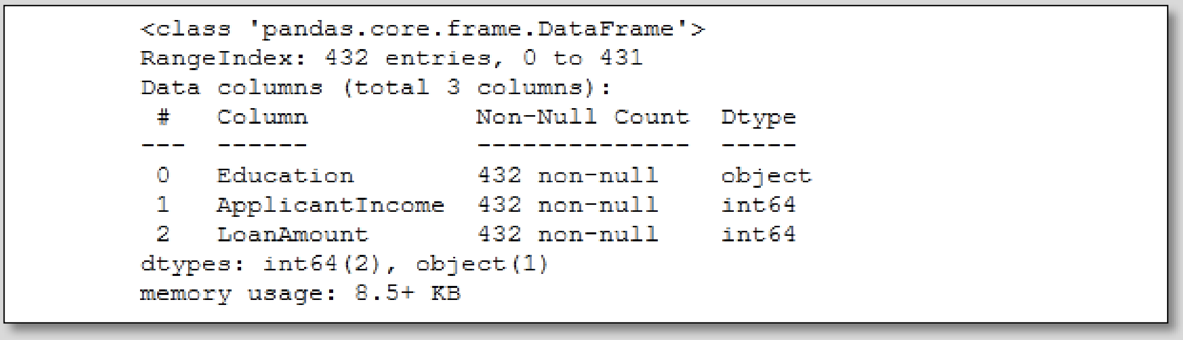
```
In [3]: # to view forst 5 observations in our dataset
      df.head()
```



Out[3]:

	Education	ApplicantIncome	LoanAmount
0	Graduate	4583	128
1	Graduate	3000	66
2	Not Graduate	2583	120
3	Graduate	6000	141
4	Graduate	5417	267

```
In [4]: # To get information about variables in dataset
      df.info()
```



```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 432 entries, 0 to 431
Data columns (total 3 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   Education        432 non-null    object
 1   ApplicantIncome  432 non-null    int64
 2   LoanAmount       432 non-null    int64
dtypes: int64(2), object(1)
memory usage: 8.5+ KB
```

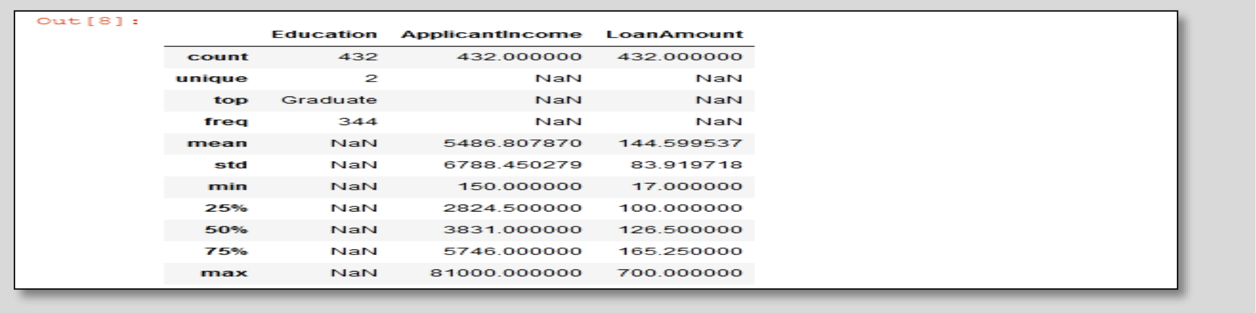
Education is a categorical variable. ApplicantIncome and LoanAmount are continuous variables. There are 432 observations on each variable, and there is no missing observation.

We are interested to examine how applicants loan amount varies with their education and income. So, LoanAmount will be dependent variable (DV). Since, Education is a categorical variable, it will be considered as independent variable (IV) for ANCOVA. Since, ApplicantIncome is continuous variable, it will be considered as covariate (CV).

```
In [5]: #to obtain and description of categories or levels in variable 'Gender'
      df['Education'].nunique()
Out [5]: 2
In [6]: df['Education'].unique()
Out [6]: array(['Graduate','Not Graduate'], dtype=object)
In [7]: df['Education'].value_counts()
Out [7]: Gradute  344
      Not Gradute 88
      Name:Education, dtype: int64
```

There are 2 categories in variable ‘Education’, viz., ‘Graduate’ and ‘Not Graduate’. Out of 432 applicants for whom the observations were recorded, 344 are graduated and 88 are not graduated.

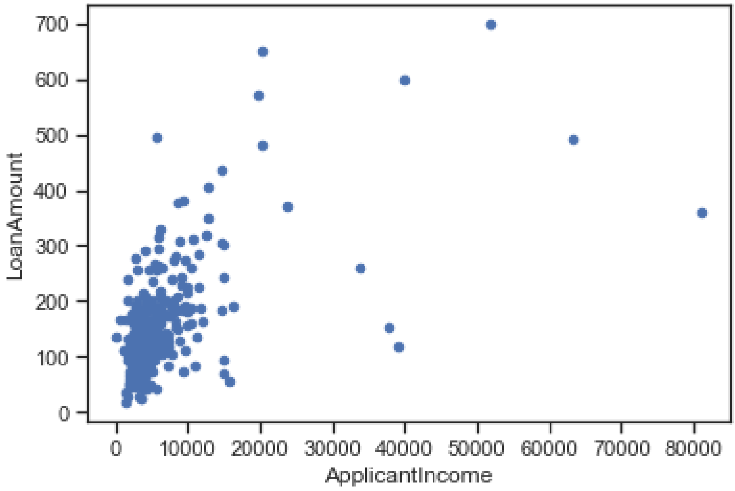
```
In [8]: # to get description about values in data
df.describe(include=’all’)
```



```
Out[8]:
         Education  ApplicantIncome  LoanAmount
count          432       432.000000  432.000000
unique           2              NaN         NaN
top       Graduate              NaN         NaN
freq           344              NaN         NaN
mean           NaN      5486.807870  144.599537
std            NaN      6788.450279   83.919718
min            NaN       150.000000   17.000000
25%            NaN      2824.500000  100.000000
50%            NaN      3831.000000  126.500000
75%            NaN      5746.000000  165.250000
max            NaN     81000.000000  700.000000
```

We want to check whether there is any relationship between ApplicantIncome and LoanAmount and whether that relationship is linear. This can be done easily by using scatter plot.

```
In [10]: #Scatter Plot
      df.plot.scatter(“ApplicatIncome”,”LoanAmount”)
      plt.show()
```

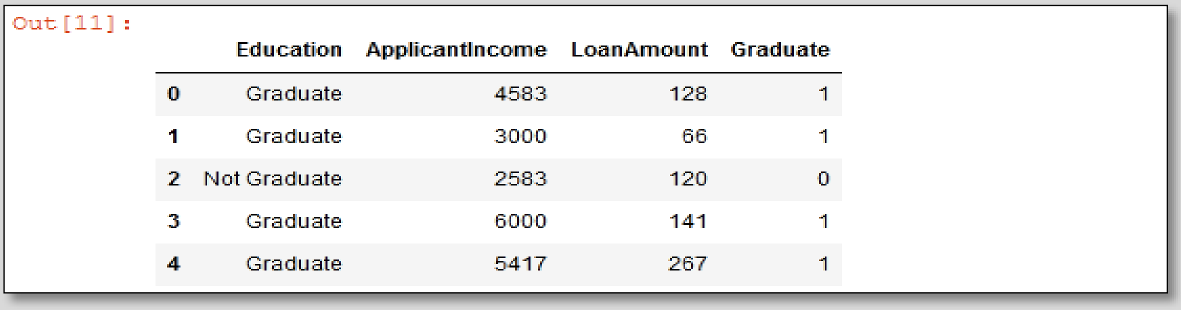


From the scatter plot we can assume the linear relationship between ApplicantIncome and LoanAmount.

Since the observations are collected on different applicants, we can assume that the error terms will be independent of each other. Now, we have to check whether the assumption about homogeneity of slopes of regression lines. For this, we will run ANCOVA by including covariate (CV) as well as interaction between independent variable (IV) and CV. To obtain a variable which will represent interaction, we will create dummy variable for ‘Education’, such that 0 will represent ‘Not Graduate’ and 1 will represent ‘Graduate’.

```
In [11]: #creating dummies
      dummy=pd.get_dummies(df[‘Education’])
      df=pd.concat([df,dummy[‘graduate’]],axis=1)
      df.head()
```

ISBN 978-93-80788-93-7



```
Out[11]:
          Education  ApplicantIncome  LoanAmount  Graduate
   0       Graduate             4583         128         1
   1       Graduate             3000          66         1
   2   Not Graduate             2583         120         0
   3       Graduate             6000         141         1
   4       Graduate             5417         267         1
```

Now we have included dummy variable in our dataset. We need to calculate variable for representing interaction term. It can be done multiplying IV by dummy variable.

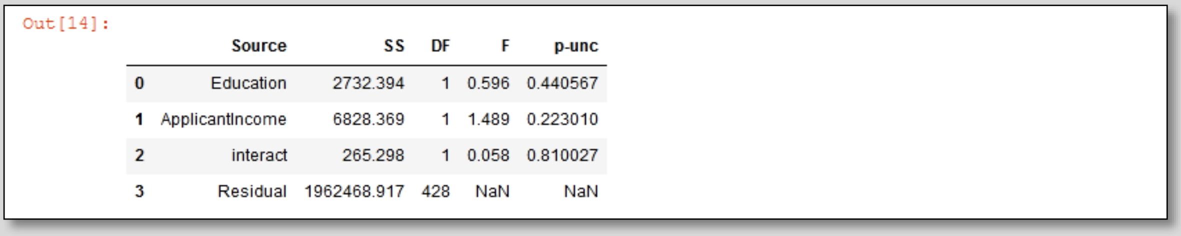
```
In [12]: #creating interaction variable
      Interact = df ['Graduate']*df['ApplicantIcome']
      df=pd.concat([df,interact], axis=1)
      print(list(df))
      ['Education','ApplicantIncome','LoanAmount','Graduate',0]
```

We have included the variable representing interaction in our dataset. But we need to change the variable name.

```
In [13]: df= df.rename(columns={0:'interacts'})
          prints(list(df))
      ['Education','ApplicantIncome','LoanAmount','Graduate','interacts']
```

Now we can run the required ANCOVA for checking the homogeneity of regression slopes.

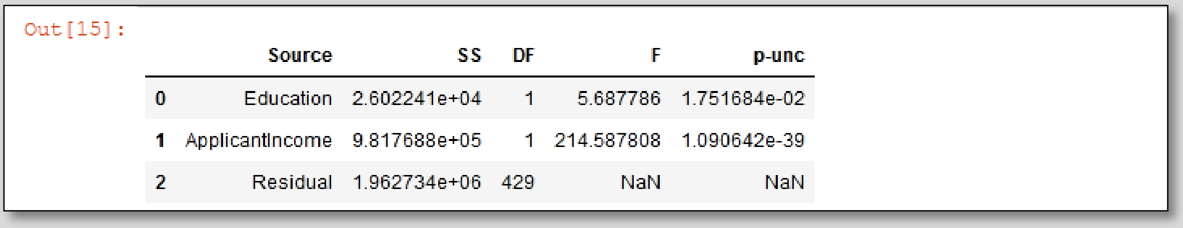
```
In[14]: #ANCOVA
      Ancova(data=df,dv='LoanAmount',covar=['ApplicantIncome','interact'],
      between ='Education')
```



```
Out[14]:
              Source           SS   DF      F     p-unc
   0       Education     2732.394    1  0.596  0.440567
   1  ApplicantIncome    6828.369    1  1.489  0.223010
   2         interact     265.298    1  0.058  0.810027
   3         Residual  1962468.917  428    NaN       NaN
```

We can see that the p-value for interact is 0.810027> 0.05. So we can conclude that the interaction effect of 'Education' and 'ApplicantIncome' is not significant at 5% level of significance. So, the slope of regression lines is same across all categories of Education. Thus we can proceed to perform ANCOVA.

```
In[15]:ancova(data=df,dv='LoanAmout',
covar='ApplicantIncome',between='Education')
```

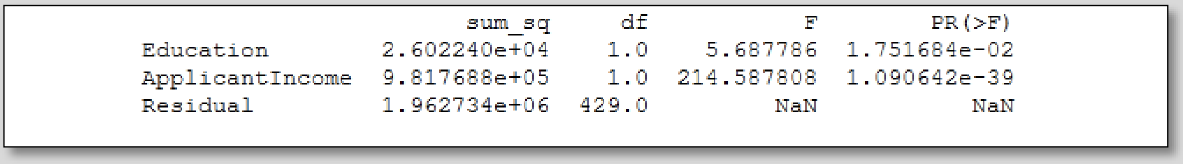


Out[15]:

	Source	SS	DF	F	p-unc
0	Education	2.602241e+04	1	5.687786	1.751684e-02
1	ApplicantIncome	9.817688e+05	1	214.587808	1.090642e-39
2	Residual	1.962734e+06	429	NaN	NaN

We can see that the p-value for Applicant Income is 1.090642e-39< 0.05. So we can conclude that Applicant Income is having significant effect on Loan Amount at 5% level of significance. But, the p-value for is Education1.751684e-02 < 0.05. So, Education is having significant effect on LoanAmount at 5% level of significance. Thus, means of Loan Amount are not same across all levels of Education, while statistically controlling the effect of Applicant Income.

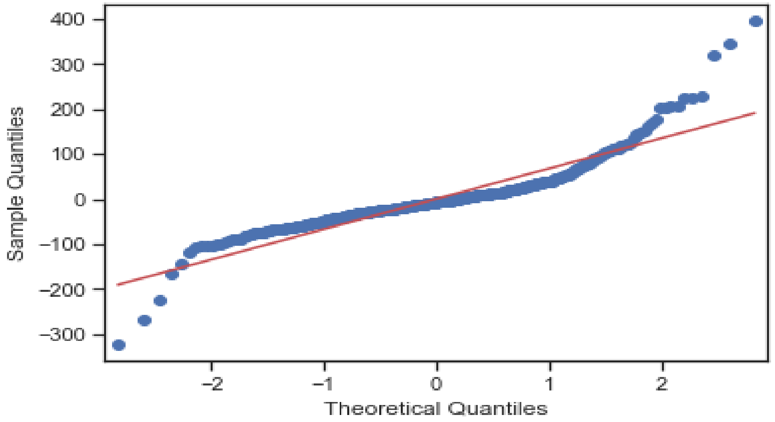
```
In [16]: #Regression and ANCOVA
    model_ancova=ols(“LoanAmount~education+ApplicantIncome”,data=df).fit()
    aov=sm.stats.anova_lm(model_ancova,typ=2)
    print(aov)
```



```
                        sum_sq     df           F        PR(>F)
Education         2.602240e+04    1.0    5.687786  1.751684e-02
ApplicantIncome   9.817688e+05    1.0  214.587808  1.090642e-39
Residual          1.962734e+06  429.0         NaN           NaN
```

So, it is verified that ANCOVA is a mixture of ANOVA and regression analysis. We will also check for normality of error term with the help of Q-Q plot.

```
In [17]: #Normality of Residuals
      qqplot(model_ancova.read,line=’s’)
      plt.show()
```



From the Q-Q plot we can see that many points are lying almost around the line. Only few points out of 432 observations are far away from the line. So we can assume normality of error term.

ISBN 978-93-80788-93-7

Now we will also check whether variance of the error term is same for female and male. This can be done by using Levene's test.

```
In [18]: #Levene's test
      Stats.levene(df['LoanAmount'][df['Education']=='Graduate'],df['Loanamou
nt'][df['Education']=='Not Graduate'])
Out[18]:LeveneResult(statistic=10.175983971254613,pvalue=0.0015269752598705
      57)
```

We can see that the p-value is 0.0015269752598705557< 0.05. So, we can conclude that variance of the dependent variable and hence that of the error term is not same for graduate and not graduate. So, the assumption about homogeneity of variance is not satisfied. Thus, it is not appropriate to perform ANCOVA for this dataset.

2. Consider a dataset **"Gender_Height_Weight"**. This data consist of 3 variables, viz. Gender, Height and Weight. The observations were recorded for 237 children on each variable. Height is measured in inches (1 inch = 2.54 cm), and Weight is measured in pounds (1 pound = 0.45 kg).

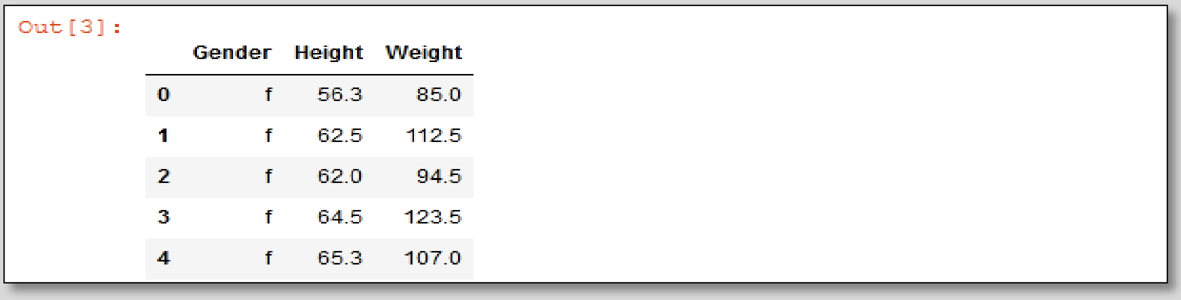
Objective: Our aim is to examine how children weight varies with their gender and height.

Analysis:

```
import numpy as np
import pandas as pd
import scipy.stats as stats
import matplotlib.pyplot as plt
import statsmodels
import statsmodels.api as sm
from statsmodels.formula.api import ols
from statsmmodels.graphics.gofplots import qqlot
from pingouin import ancova
In [2]: #Import dataset
      df= pd.read_csv("D:/Gender__Height_Weight.csv")
      print(list(df))
      ['Gender','Height','Weight']
```

We can see that there are 3 columns in dataset, viz., Gender, Height and Weight.

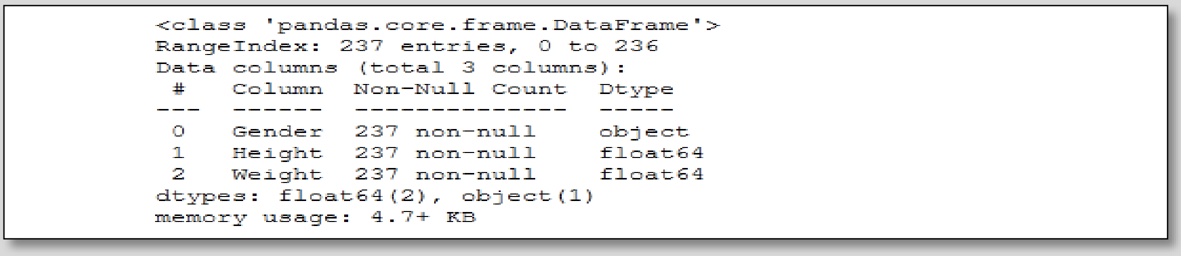
```
In [3]: # to view first 5 observation in our dataset
      df.head()
```



Out[3]:

	Gender	Height	Weight
0	f	56.3	85.0
1	f	62.5	112.5
2	f	62.0	94.5
3	f	64.5	123.5
4	f	65.3	107.0

```
In [4]: # to get information about variables in dataset
      df.info()
```



```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 237 entries, 0 to 236
Data columns (total 3 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   Gender  237 non-null    object
 1   Height  237 non-null    float64
 2   Weight  237 non-null    float64
dtypes: float64(2), object(1)
memory usage: 4.7+ KB
```

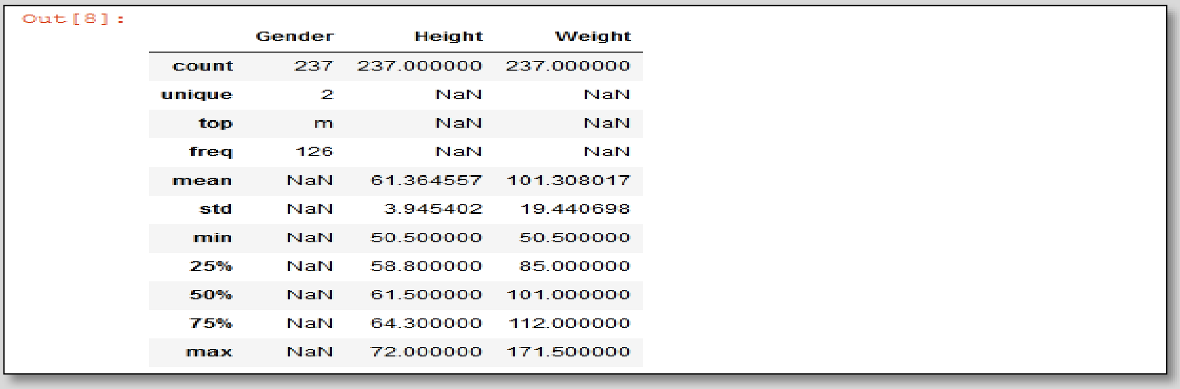
Gender is a categorical variable. Height and Weight are continuous variables. There are 237 observations on each variable, and there is no missing observation.

We are interested to examine how children weight varies with their gender and height. So, Weight will be dependent variable (DV). Since, Gender is a categorical variable, it will be considered as independent variable (IV) for ANCOVA. Since, Height is continuous variable, it will be considered as covariate (CV).

```
In [5]: # to obtain number and description of categories or level in variable
‘Gender’
      df[‘Gender’].nunique
Out [5]: 2
In [6]: df[‘Gender’].unique()
Out [6]: array([‘f’,’m’],dtype=object)
In [7]:df[‘Gender’].value_counts()
Out [7]:  m 126
         f  111
        name: Gender, dtype: int64
```

There are 2 categories in variable ‘Gender’, viz., ‘m’ and ‘f’. Here ‘m’ indicates male and ‘f’ indicates female. Out of 237 children for whom the observations were recorded, 126 are male and 111 are female.

```
In [8]: #to get description about values in data
      df.describe(include=’all’)
```

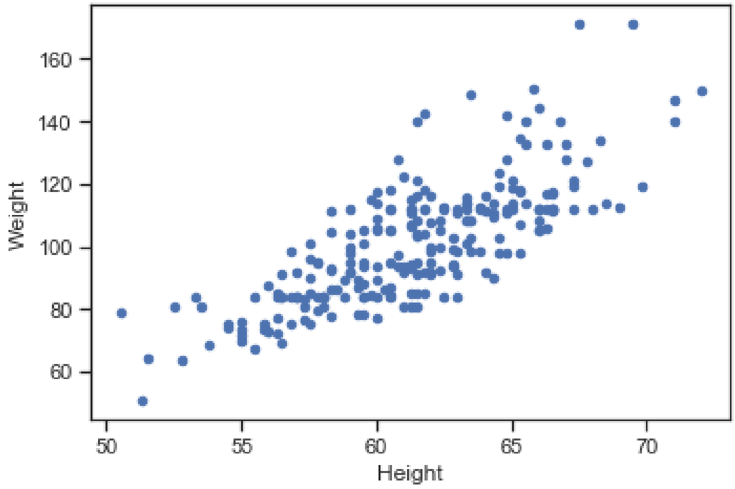


Out[8]:

	Gender	Height	Weight
count	237	237.000000	237.000000
unique	2	NaN	NaN
top	m	NaN	NaN
freq	126	NaN	NaN
mean	NaN	61.364557	101.308017
std	NaN	3.945402	19.440698
min	NaN	50.500000	50.500000
25%	NaN	58.800000	85.000000
50%	NaN	61.500000	101.000000
75%	NaN	64.300000	112.000000
max	NaN	72.000000	171.500000

We want to check whether there is any relationship between Height and Weight and whether that relationship is linear. This can be done easily by using scatter plot.

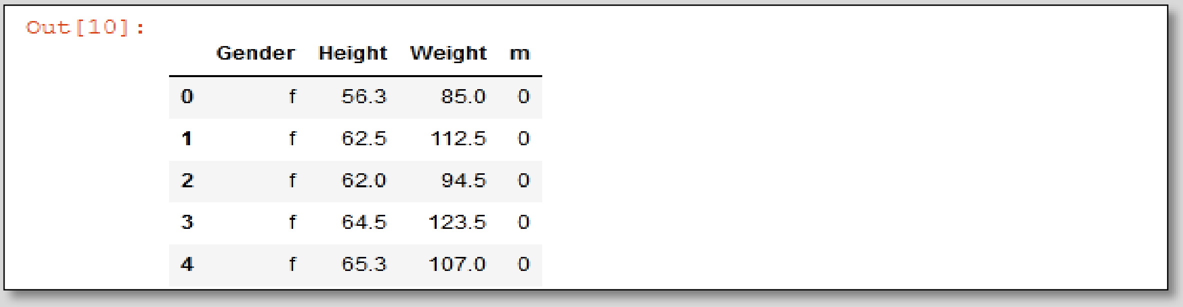
```
In [9]: #Scatter Plot
        df.plot.scatter(“Height”, “Weight”)
        plt.show()
```



From the scatter plot we can assume the linear relationship between Height and Weight.

Since the observations are collected on different children, we can assume that the error terms will be independent of each other. Now, we have to check whether the assumption about homogeneity of slopes of regression lines. For this, we will run ANCOVA by including covariate (CV) as well as interaction between independent variable (IV) and CV. To obtain a variable which will represent interaction, we will create dummy variable for ‘Gender’, such that 0 will represent ‘f’ and 1 will represent ‘m’.

```
In [10]: #creating dummies
         dummy=pd.get_dummies (df[‘Gender’])
         df=pd.concat([df,dummy[‘m’]],axis=1)
         df.head()
```



```
Out[10]:
            Gender  Height  Weight  m
         0       f    56.3    85.0  0
         1       f    62.5   112.5  0
         2       f    62.0    94.5  0
         3       f    64.5   123.5  0
         4       f    65.3   107.0  0
```

Now we have included dummy variable in our dataset. We need to calculate variable for representing interaction term. It can be done multiplying IV by dummy variable.

```
In [11]: #Creating interaction variable
         interact=df[‘m’]*df[‘Height’]
         df=pd.concat([df,interact],axis=1)
         print(list(df))
         [‘Gemder’,’Height’,’Weight’,’m’,0]
```

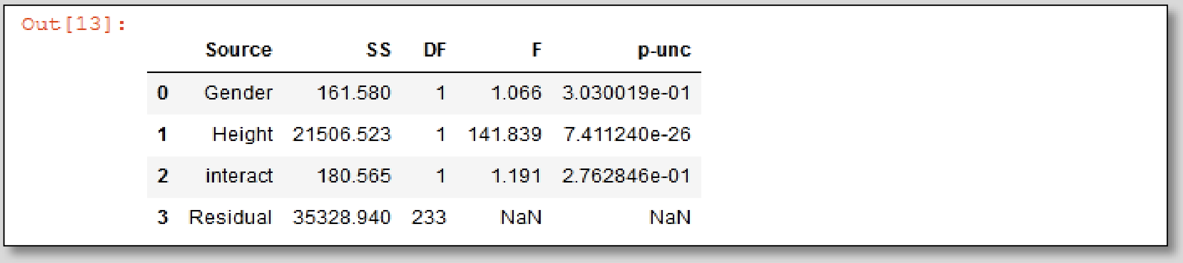
ISBN 978-93-80788-93-7

We have included the variable representing interaction in our dataset. But we need to change the variable name.

```
In [12]: df =df.rename(colums=(0:’interact’))
      print(list(df))
      [‘Gender’,’Height’,’Weight’,’m’,’interact’]
```

Now we can run the required ANCOVA for checking the homogeneity of regression slopes.

```
In [13]: #ANCOVA
   ancova(data=df,dv=’Weight’,cover=[‘Height’,’interact’],between=’Geder’)
```

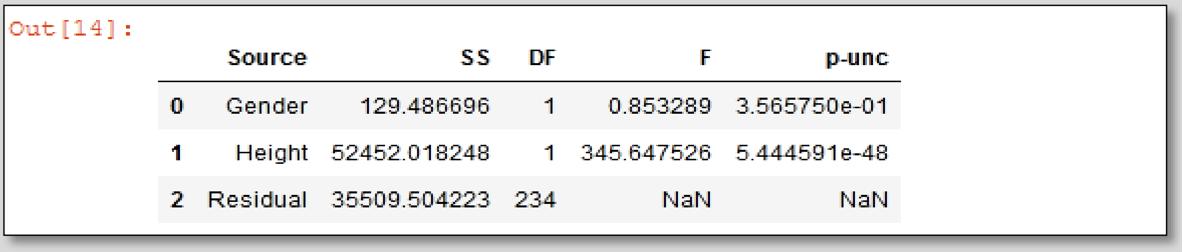


Out[13]:

	Source	SS	DF	F	p-unc
0	Gender	161.580	1	1.066	3.030019e-01
1	Height	21506.523	1	141.839	7.411240e-26
2	interact	180.565	1	1.191	2.762846e-01
3	Residual	35328.940	233	NaN	NaN

We can see that the p-value for interact is 2.762846e-01> 0.05. So we can conclude that the interaction effect of ‘Height’ and ‘Gender’ is not significant at 5% level of significance.So, the slope of regression lines is same across all categories of Gender. Thus we can proceed to perform ANCOVA.

```
In [14]: ancova(data=df, dv=’Weight’,covar=’Height’,between=’Gender’)
```

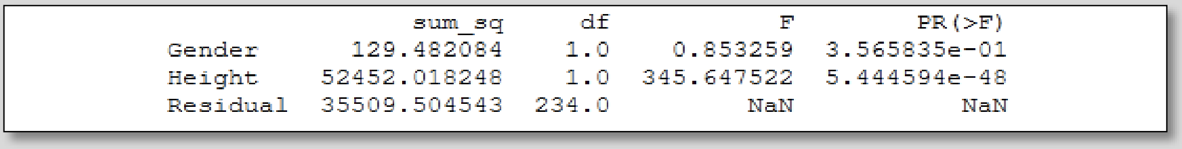


Out[14]:

	Source	SS	DF	F	p-unc
0	Gender	129.486696	1	0.853289	3.565750e-01
1	Height	52452.018248	1	345.647526	5.444591e-48
2	Residual	35509.504223	234	NaN	NaN

We can see that the p-value for Height is 5.444591e-48 < 0.05. So we can conclude that Height is having significant effect on Weight at 5% level of significance. But, the p-value for Gender is 3.565750e-01> 0.05. So, Gender is not having significant effect on Weight at 5% level of significance. Thus, means of weight aresame across all levels of Gender, while statistically controlling the effect of Height.

```
In [15]: #Regression and ANCOVA
      model_ancova=ols(“weight~Gender+height”,data=df).fit()
      print(aov)
```

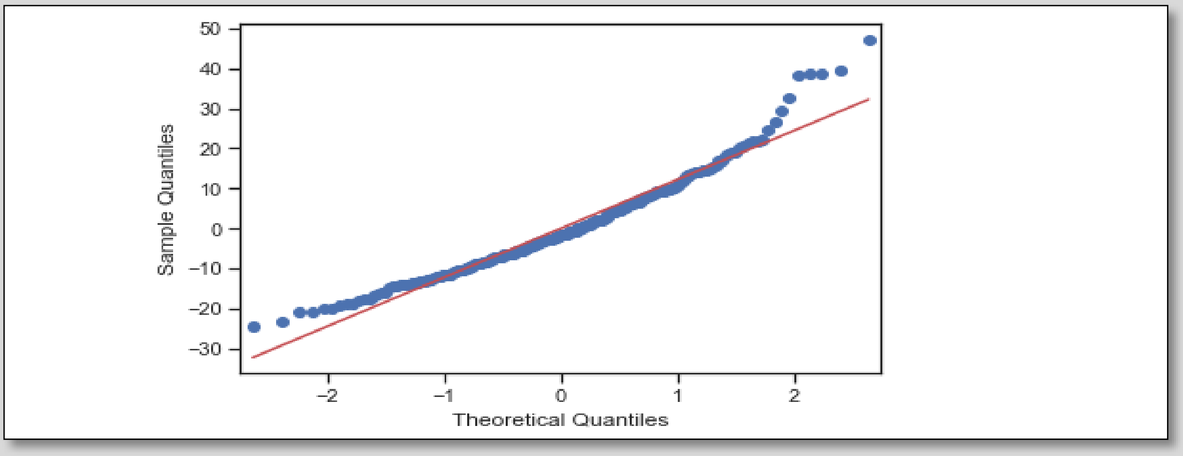


```
                sum_sq     df           F        PR(>F)
Gender      129.482084    1.0    0.853259  3.565835e-01
Height    52452.018248    1.0  345.647522  5.444594e-48
Residual  35509.504543  234.0         NaN           NaN
```

So, it is verified that ANCOVA is a mixture of ANOVA and regression analysis. We will also check for normality of error term with the help of Q-Q plot.

ISBN 978-93-80788-93-7

```
In [16]: #Normality of Residuals
      qqplot(model_ancova.resid,line='s')
      plt.show()
```



From the Q-Q plot we can see that many points are lying almost around the line. So we can assume normality of error term.

Now we will also check whether variance of the error term is same for female and male. This can be done by using Levene’s test.

```
In [17]: #Levene’s Test
      stats.levene(df[‘Weight’][df[‘Gender’]==’f’],df[‘Weight’][df[‘Gender
      ’]==’m’])
Out[17]: LeveneResult(statistic=0.6272547811263836,value=0.429167936219046)
```

We can see that the p-value is 0.42916279362190046 > 0.05. So, we can conclude that variance of the dependent variable and hence that of the error term is same for female and male.

References

1. https://en.wikipedia.org/wiki/Analysis_of_covariance
2. https://www.statisticshowto.com/ancova/
3. https://pingouin-stats.org/generated/pingouin.ancova.html

Chapter 13
Predictive Analysis of Medical Cost (Study of Regression Analysis to Random Forest Regression on Medical Cost)

Mr. Sourav S. Tiwari, *Postgraduate student, Department of Statistics, NMIMS*

To make profit, insurance companies should collect higher premium than the amount paid to the insured person. Due to this, insurance companies invest a lot of time, effort, and money in creating models that accurately predicts health care costs. I have explore what features are important predictors for how much a person will be charged. In this case study I have tried to build the most accurate model as possible by using multiple linear regression and random forest regression algorithms.

13.1 Data Description

Data Name: Medical Cost Personal dataset, Source: Kaggle

13.2 Data Analysis

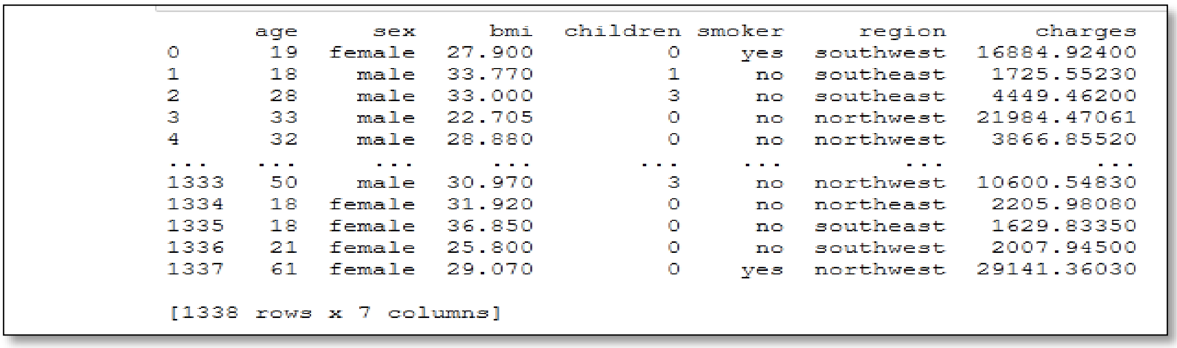
Data file is renamed as **"insurance.csv".**

Data used for Analysis:

```
In   [1] : # # # Importing libraries
            import pandas as pd
            import numpy as np
            from sklearn import datasets, linear_model
            from sklearn.model_selection import train_test_split
            from sklearn import preprocessing
            from matplotlib import pyplot as plt
            import seaborn as sns
            from sklearn . preprocessing import StandardScaIer
            import statsmodels . api as sm
            import statsmodels . stats . api as smf
            import statsmodels . formula.api  as smf
            import statsmodels. tools. tools as ct
            from statsmodels . stats . outliers_influence import
            variance_inflation_factor
            import os
            from sklearn.metrics import accuracy_score
            from  sklearn.metrics import mean_squared_error
            from sklearn. ensemble import RandomForestRegressor
```

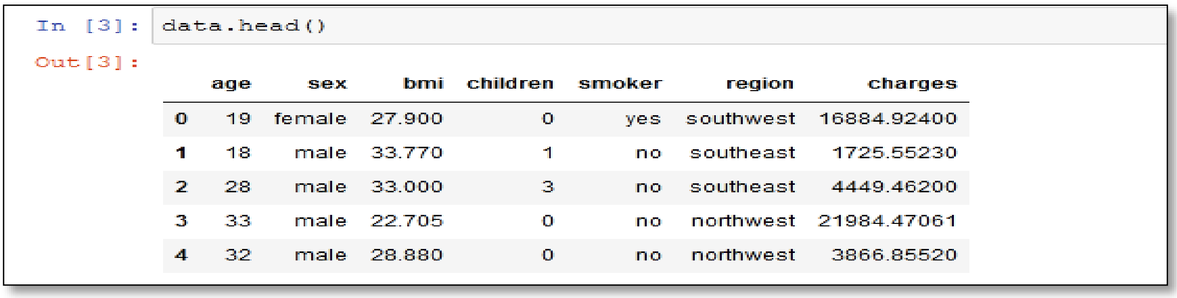
```
                                           #Required to build random forest
           from statsmodels.compat import lzip
```

```
In [2] : ## Importing dataset
           data = pd.read_csv ("C:/Users/Admin/Desktop/insurance.csv")
           print(data)
```



```
      age     sex     bmi  children smoker     region      charges
0      19  female  27.900         0    yes  southwest  16884.92400
1      18    male  33.770         1     no  southeast   1725.55230
2      28    male  33.000         3     no  southeast   4449.46200
3      33    male  22.705         0     no  northwest  21984.47061
4      32    male  28.880         0     no  northwest   3866.85520
...   ...     ...     ...       ...    ...        ...          ...
1333   50    male  30.970         3     no  northwest  10600.54830
1334   18  female  31.920         0     no  northeast   2205.98080
1335   18  female  36.850         0     no  southeast   1629.83350
1336   21  female  25.800         0     no  southwest   2007.94500
1337   61  female  29.070         0    yes  northwest  29141.36030

[1338 rows x 7 columns]
```

```
In [3]: data.head()
```

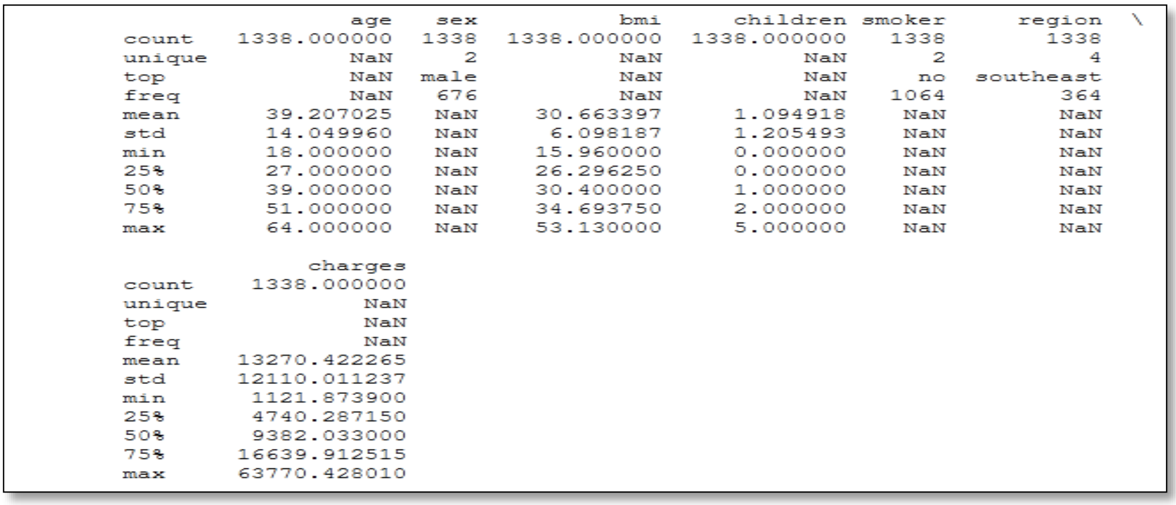


```
In [3]: data.head()
Out[3]:
```

	age	sex	bmi	children	smoker	region	charges
0	19	female	27.900	0	yes	southwest	16884.92400
1	18	male	33.770	1	no	southeast	1725.55230
2	28	male	33.000	3	no	southeast	4449.46200
3	33	male	22.705	0	no	northwest	21984.47061
4	32	male	28.880	0	no	northwest	3866.85520

Summary of the data:

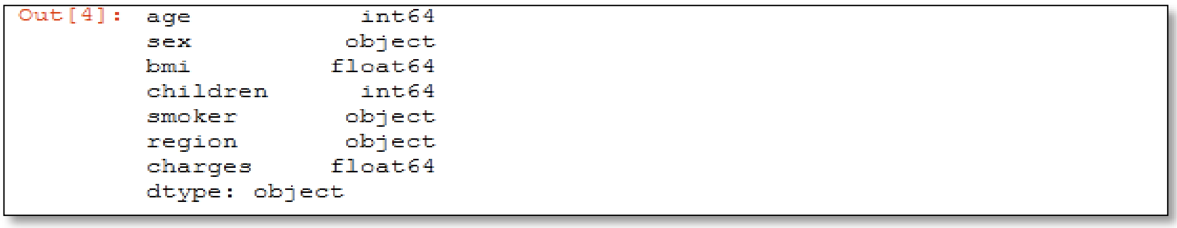
```
In [6]: data_summm=data.describe(include='all')    ##Summary statisics
      print(data_summ)
```



```
                age   sex          bmi     children smoker     region  \
count   1338.000000  1338  1338.000000  1338.000000   1338       1338
unique          NaN     2          NaN          NaN      2          4
top             NaN  male          NaN          NaN     no  southeast
freq            NaN   676          NaN          NaN   1064        364
mean      39.207025   NaN    30.663397     1.094918    NaN        NaN
std       14.049960   NaN     6.098187     1.205493    NaN        NaN
min       18.000000   NaN    15.960000     0.000000    NaN        NaN
25%       27.000000   NaN    26.296250     0.000000    NaN        NaN
50%       39.000000   NaN    30.400000     1.000000    NaN        NaN
75%       51.000000   NaN    34.693750     2.000000    NaN        NaN
max       64.000000   NaN    53.130000     5.000000    NaN        NaN

             charges
count    1338.000000
unique           NaN
top              NaN
freq             NaN
mean    13270.422265
std     12110.011237
min      1121.873900
25%      4740.287150
50%      9382.033000
75%     16639.912515
max     63770.428010
```

Structure of the data

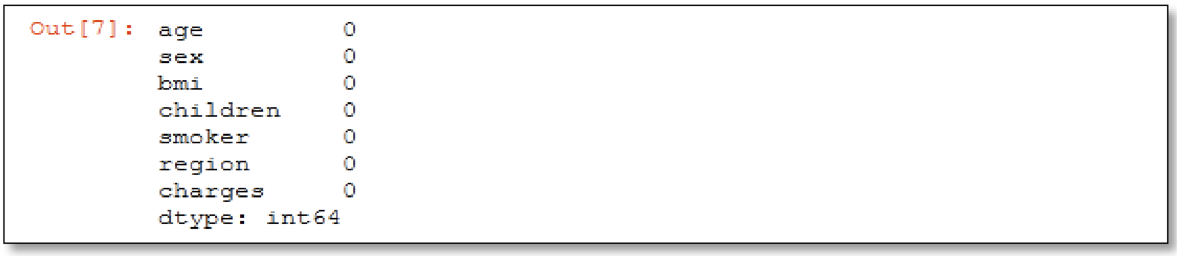
```
In [4]: data.dtypes   ##data types
```



```
Out[4]: age            int64
        sex           object
        bmi          float64
        children       int64
        smoker        object
        region        object
        charges      float64
        dtype: object
```

Missing Values

```
In [7]: ###Checking for missing values
          data.isnull() .sum()
```

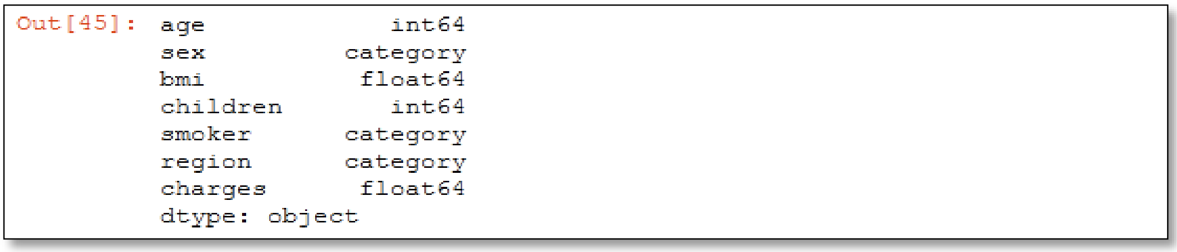


```
Out[7]: age         0
        sex         0
        bmi         0
        children    0
        smoker      0
        region      0
        charges     0
        dtype: int64
```

Conversion of the variables:

Conversion of the variables named as sex, smoker, and region of type object to type category can be done in the following way.

```
In [45]:  data[“sex”] = data[“sex”].astype('category')
          data[“smoker”] =data[“smoker”].astype('category')
          data[“region “] = data["region"].astype('category')
          data.dtypes
```



```
Out[45]: age            int64
         sex         category
         bmi          float64
         children       int64
         smoker      category
         region      category
         charges      float64
         dtype: object
```

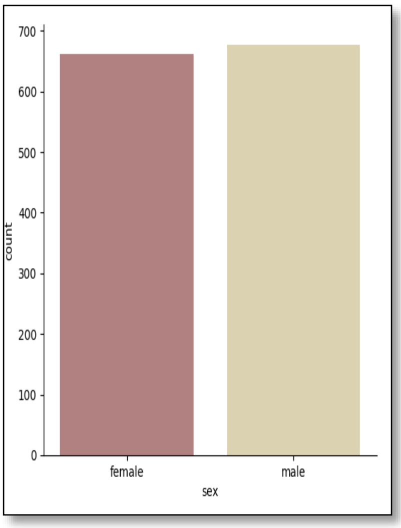
Visualization:

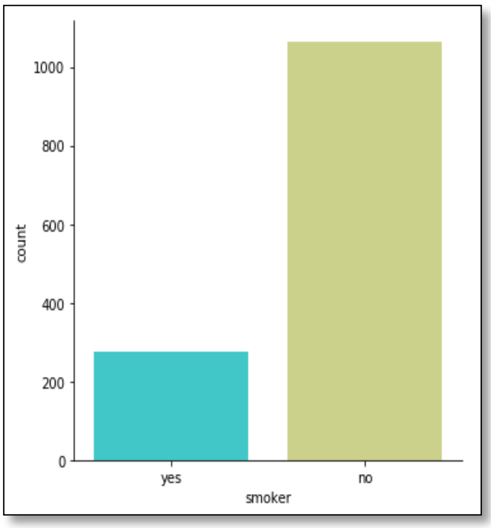
Bar Plots

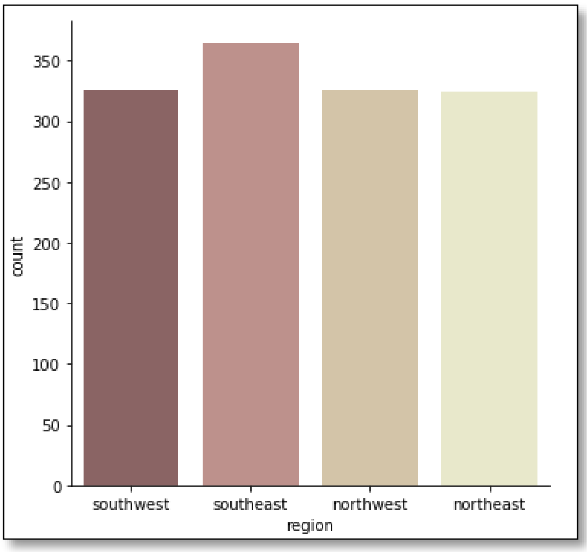
```
In [20]: ###catplot or double barplot
          ##to change the colour  -  palette=”pink" inside the catplot ()
          sns.catplot(x="sex”,    kind="count”,  palette-"pink",  data=data  ,
          size=5)
```

```
In [21]: sns.catplot(x="smoker”,  kind="count”, palette-"rainbow",
         data=data,size=5)
```

```
In [22]: sns.catplot(x="region”,  kind="count”, palette-"pink",
          data=data,size=5)
```







From the above bar graphs we can state that the female count is less than the male count where the female count is around 660 and male count is around 670. The number of non-smokers is more than number of smokers i.e. the non-smokers are more than 1000 while the number of smokers are less than 300. The number of patients is more in south-east region while in the other regions the numbers of patients are almost equal.

Swarm Plots

```
In [24]: sns.swarmplot(x= ‘sex' ,  y = 'charges' ,  data = data ,  hue =
         'sex')
```

```
In [25]: sns.swarmplot(x= ‘smoker' ,  y = 'charges' ,  data = data ,  hue=
         'smoker')
```

```
In [23]: sns.swarmplot(x= ‘region' ,  y = 'charges' ,  data = data ,  hue=
         'region')
```

ISBN 978-93-80788-93-7

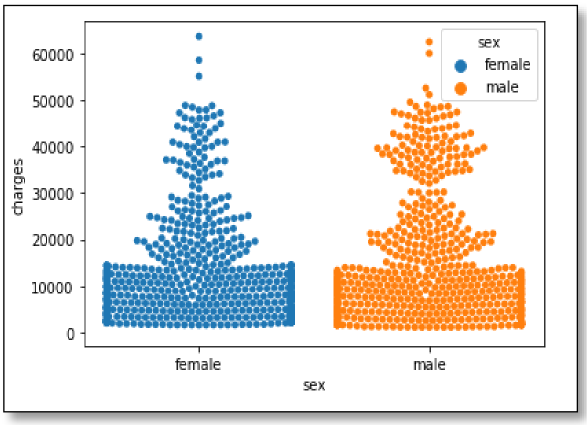


Fig (1)

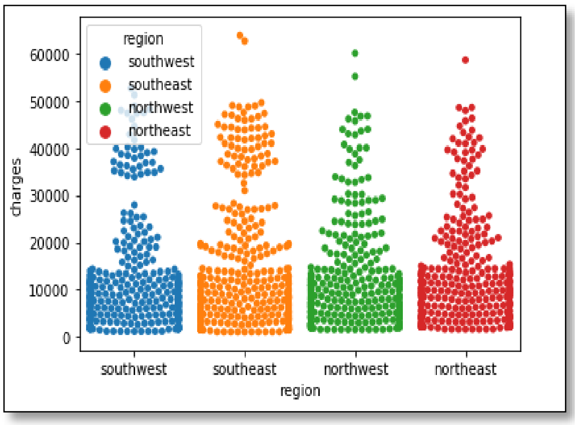


fig (2)

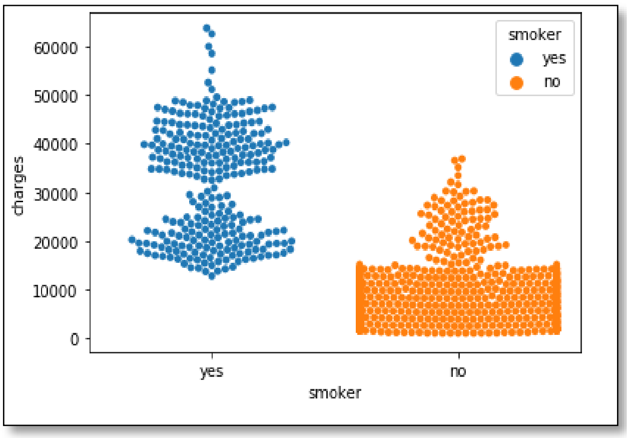
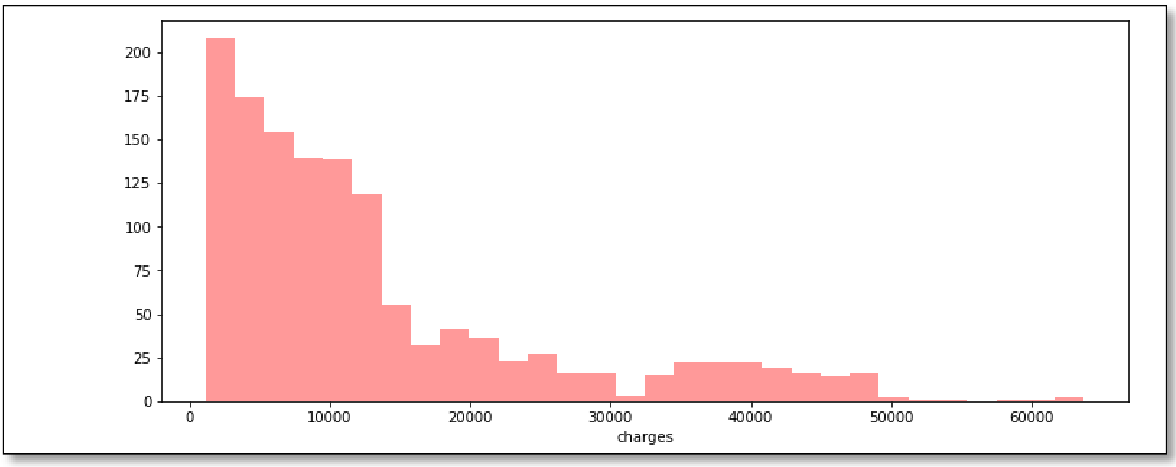


fig (3)

From fig (1) shows that most of the people pay charges between Rs.0 to Rs.15,000 approximately. Some of them are paying charges between Rs.15,000 to Rs.47,000. From fig (2), one can say that in all four regions majority of the people pay charges upto Rs.15,000 (approximately). Also south-east region has more number of insurers than any other regions. From fig (3), the smokers pay more charges than the non-smokers. Most of the non-smokers pay charges less than Rs.20,000. While the insurers in smoker category pay more than Rs.10,000 and many smokers pay the more than Rs.40,000

Histogram of Charges

```
In [16]: f= plt.figure (figsize=(12, 5))
         sns.distplot(data['charges'],kde=0,color='red')
Out [16] : <matplotlib.axes._subplots.AxesSubplot at 0x9ed23d0>
```

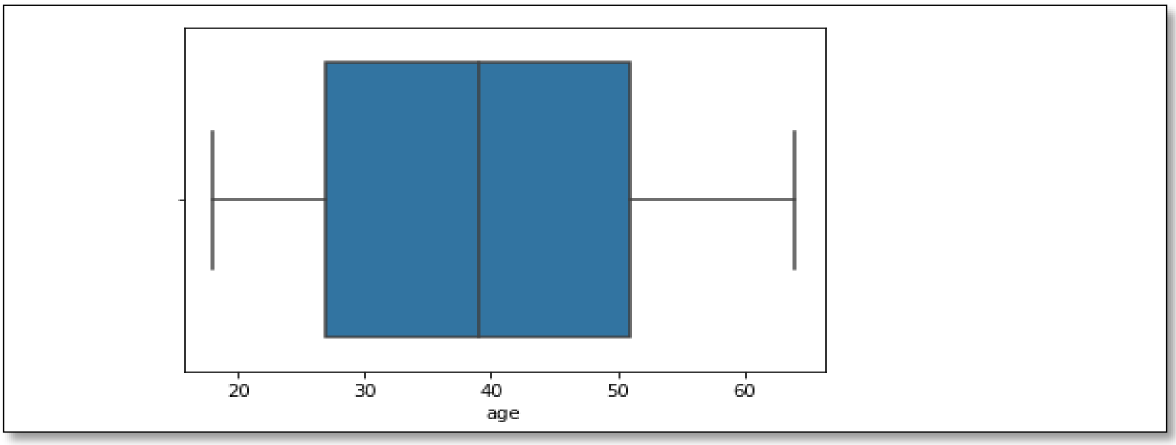


Charges variables positively skewed, i.e., Most of the insurer pay charges less than Rs. 20,000.

Detecting Outliers by using Box plot:

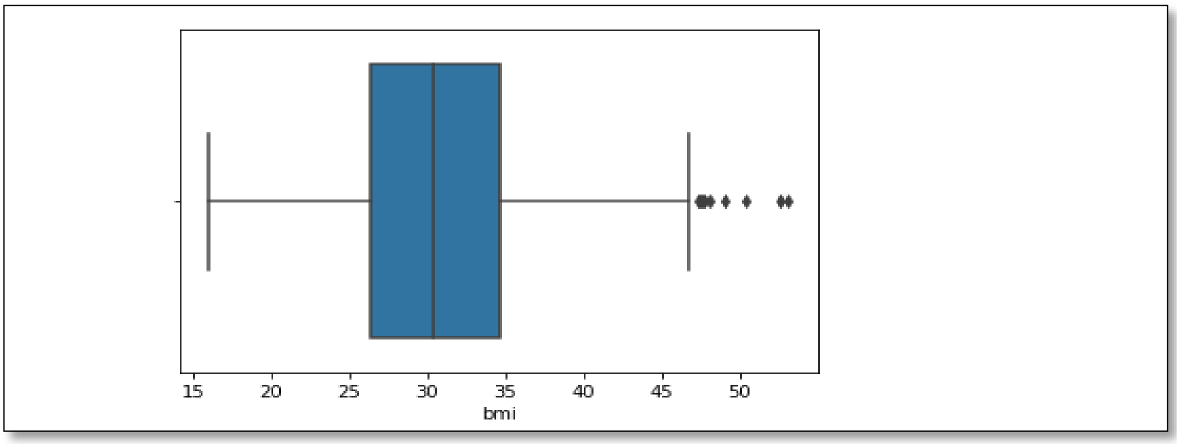
```
In [26]: data.columns
Out [26]: Index(['age', 'sex', 'bmi', 'children', 'smoker', 'region',
        'charges'], dtype='object')
```

```
In [27]: sns.boxplot(data['age'])
Out [27]: <matplotlib.axes._subplots.AxesSubplot at 0xb5d8630>
```



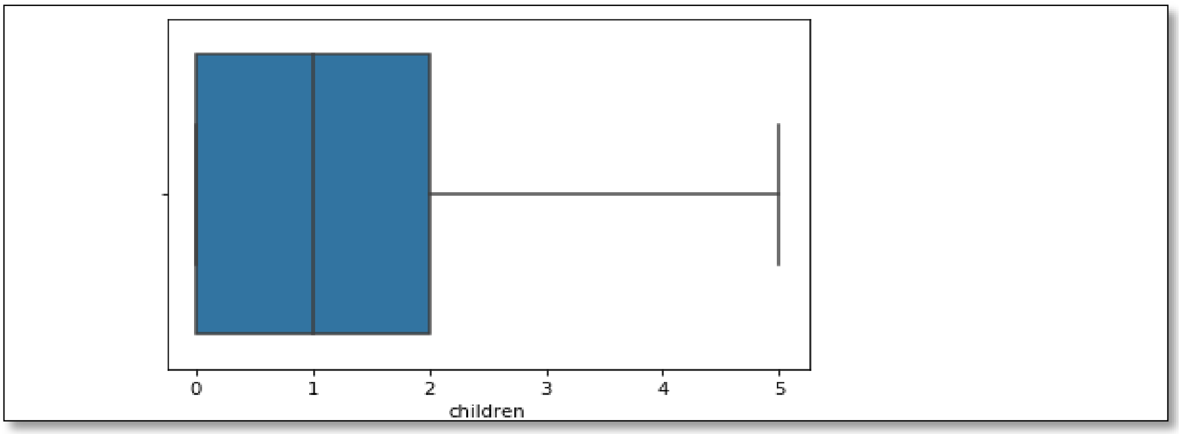
```
In [28]: sns.boxplot(data['bmi']) ###outliers
Out [28]: <matplotlib. axes. _subplots.Axes Subplot at 0xb5f2250>
```

ISBN 978-93-80788-93-7

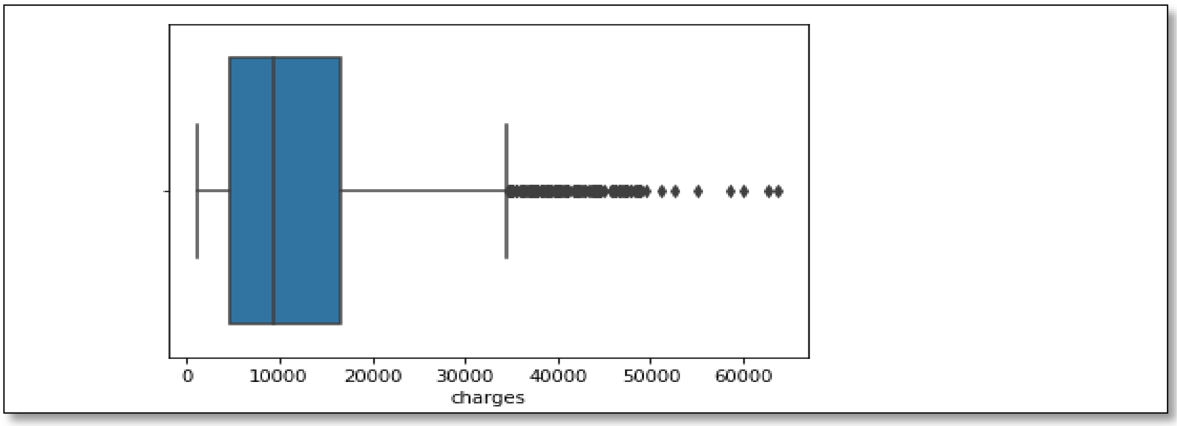


Here box plots are used to detect the outliers. These are the boxplots of variables Age and BMI respectively. So, from these box plots we can say that there is no outliers in the variable Age whereas there are outliers in the variable BMI.

```
In [30]: sns.boxplot(data['children'])
Out [30]: <matplotlib.axes._subplots.AxesSubplot at 0xb681510>
```



```
In [29]: sns.boxplot(data['charges'])  ###outliers
Out [29]: <matplotlib.axes._subplots.AxesSubplot at 0xb654d30>
```



These are the box plots for variables Children and Charges depicting the presence of outliers in variable Charges. Since outliers are present only in two variables i.e. in BMI and Charges and therefore calculating lower whisker and upper whisker only for these two variables so that outliers can be treated.

Inter Quartile Range Score (IQR Score) for the variables BMI and Charges:

For BMI

```
In [31]: Q1=data[‘bmi’].qaantile(0.25)
         Q3=data[‘bmi’].qaantile(0.75)
         IQR=Q3-Q1
         print(IQR)
         8.3975

In [32]:  Lower_whisker=Q1-1.5*IQR
          Upper_whisker=Q3+1.5*IQR
          print(Lower_whisker, Upper_whisker)
          13.7 47.290000000000006
```

For Charges

```
In [37]: ## Outliers for charges
          Q1=data['charges'].quantile(0.25)
          Q3=data['charges’].quantile(0.75)
          IQR=Q3-Q1
          print(IQR)
          11899.625365
```

```
In [38]: Lower_whisker=Q1-1.5*IQR
         Upper_whisker=Q3+1.5*IQR
         print(Lower_whisker,Upper_whisker)
         -13109.1508975 34489.350562499996
```

Interpretation

The values which are less than lower whisker and greater than upper whisker are treated as outliers.

Another way of detecting the outliers:

Calculate the Z-Score

In this procedure we calculate the score for each observation. Any z-score greater than 3 or less than -3 is considered to be an outlier. This rule of thumb is based on the empirical. From this rule we see that almost all of the data (99.7%) should be within three standard deviations from the mean. By calculating the z-score we are standardizing the observation, meaning the standard deviation is now 1. Thus from the empirical rule we expect 99.7% of the z-scores to be within -3 and 3. $Z = \frac{X-\mu}{\sigma}$

ISBN 978-93-80788-93-7

Note: There are also several methods of detecting outliers such as scatter plot (for continuous variables), **DBSCAN (Density Based Spatial Clustering of Applications with Noise), Cook's distance etc.**

Remedies for Outliers

- **Z-Score method**

While calculating the Z-score we re-scale and center the data and look for data points which are too far from zero. In most of the cases a threshold of 3 or -3 is used i.e if the Z-score value is greater than or less than 3 or -3 respectively, that data point will be identified as outliers. So in this method, the data is refined by deleting the observations which are having outliers. **Note:** In this method the rows are directly removed from the data which are having outliers but the observations cannot be removed as such because until and unless the outliers are not absurd values we cannot remove them, and hence removing the outliers is an inappropriate way of dealing with them.

- **Inter Quartile Range Score (IQR Score)**

This technique uses the IQR scores calculated earlier to remove outliers. The rule of thumb is that anything not in the range of Lower whisker = *(Q1 - 1.5*__IQR) and upper whisker = (Q3 + 1.5__ *IQR)* is an outlier, and can be removed. So in this method, the outliers which are lesser than lower whisker value are replaced by the lower whisker value and the outliers which are greater than upper whisker value are replaced by the upper whisker value in the data.

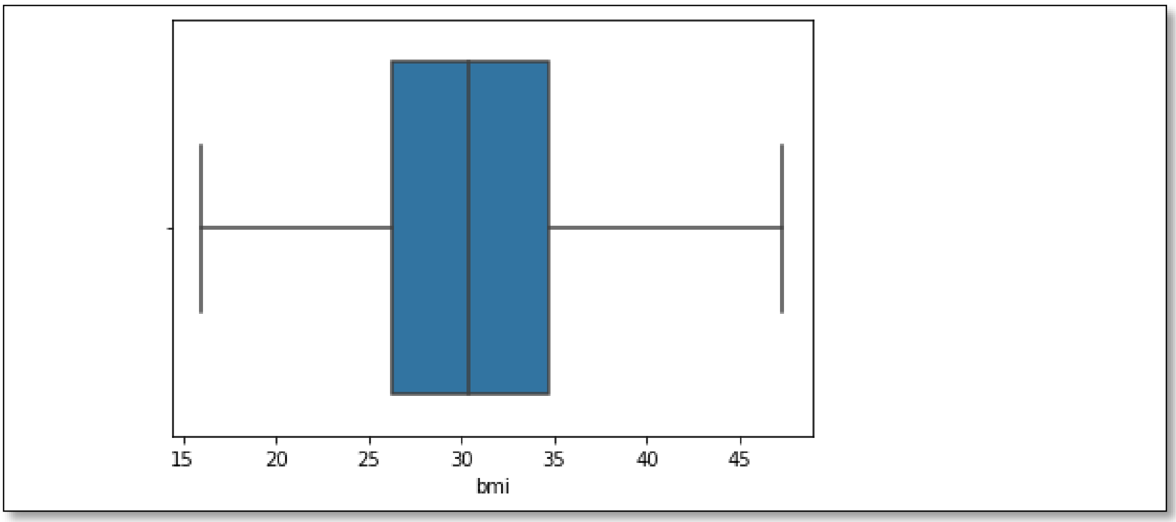
In this analysis, the variable BMI has outliers and they are treated by IQR Score in the given following way:

```
In [39]: print( (data['charges’]>Upper_whisker) .sum())
         #we can see 1282 observation lying above the upper_whisker here
         which seems as outliers
         139
```

```
In [40]: print((data['charges' ]<Lower_whisker) .sum())
                                     # no oservation below the lower_whisker
         0
```

```
In [41]: #replacing all values which are greater than upper_whisker by
         upper_whisker
         data['bmi'] =np.where (data[ ‘bmi']>
         Upper_whisker,Upper_whisker,data ['bmi'])
```

```
In [42]: sns.boxplot(data('bmi'])      #outliers removed
Out [42]: <matplotlib.axe5._subplots.AxesSubplot at 0xc7e2630>
```



Hence the outliers are removed from the variable BMI.

For variable Charges, the outliers can't be treated by using this method because the outliers are much larger than the upper whisker value and hence the outliers can't be a good imputation for the outliers and thereby outliers can't be replaced by upper whisker value.

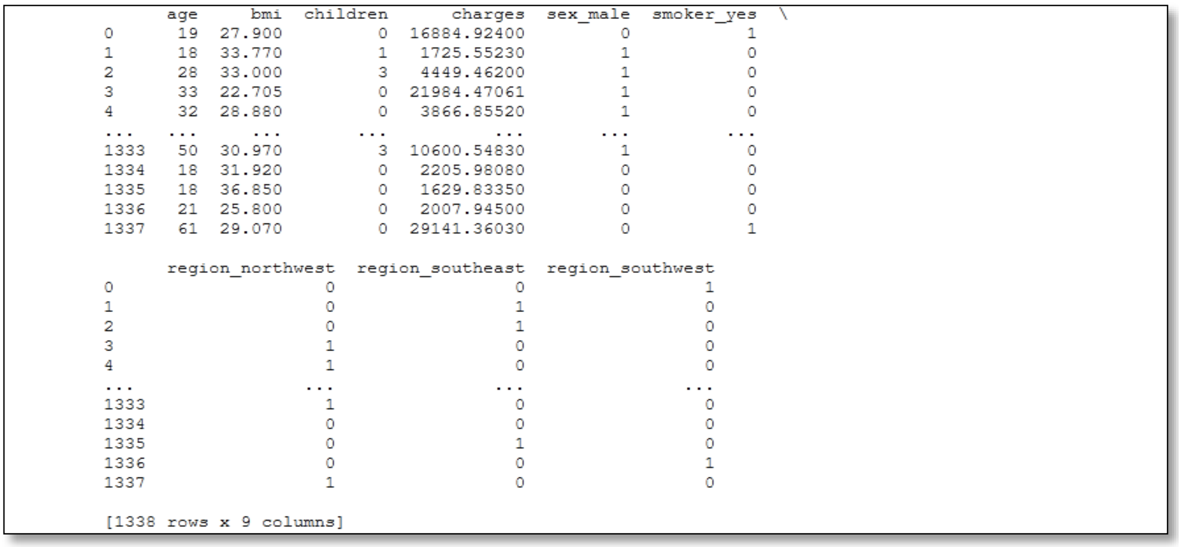
Limitation: So in this analysis without treating outliers for variable charges the analysis has proceeded and fitted the Multiple Linear Regression Model

Note: The Multiple Linear Regression Model has fitted here but in actual scenario if any of the assumptions of multiple linear regression does not satisfy then we can't proceed with this algorithm. Instead of this algorithm we have to go for non parametric test. Here we have used Random Forest Regression (non parametric algorithm) for predicting the insurance charges.

Creating Dummies for Categorical Variables:

While fitting the multiple linear regression models the response variables have to be in a numeric type. Categorical variable has levels and it has to be converted into indicator variable so they can be used in the model.

```
In  [46]:  ins_data=  pd.get_dummies  (data,columns=['sex','smoker','region']
        ,drop_first=True)   ##Creating dummies
         print(ins_data)
```



```
      age     bmi  children      charges  sex_male  smoker_yes  \
0      19  27.900         0  16884.92400         0           1
1      18  33.770         1   1725.55230         1           0
2      28  33.000         3   4449.46200         1           0
3      33  22.705         0  21984.47061         1           0
4      32  28.880         0   3866.85520         1           0
...   ...     ...       ...          ...       ...         ...
1333   50  30.970         3  10600.54830         1           0
1334   18  31.920         0   2205.98080         0           0
1335   18  36.850         0   1629.83350         0           0
1336   21  25.800         0   2007.94500         0           0
1337   61  29.070         0  29141.36030         0           1

      region_northwest  region_southeast  region_southwest
0                    0                 0                 1
1                    0                 1                 0
2                    0                 1                 0
3                    1                 0                 0
4                    1                 0                 0
...                ...               ...               ...
1333                 1                 0                 0
1334                 0                 0                 0
1335                 0                 1                 0
1336                 0                 0                 1
1337                 1                 0                 0

[1338 rows x 9 columns]
```

```
In [47] : ins_data.columns
Out [47]: Index ( [ 'age' , 'bmi' , 'children' , 'charges' , 'sex_male' ,
          'smoker _yes' , 'region_northwest','region_southeast',
          'region_southwest'], dtype= ‘object’)
```

After creating the dummy variables we have following variables:
Age, BMI, Children, Charges, Sex_male, Smoker_yes, Region_northwest, Region_southwest and Region_southeast where variables Sex_male, Smoker_yes, Region_northwest, Region_southwest and Region_southeast are dummy variables of Sex, Smoker and Region respectively.

Splitting the data into Train data and Test data:

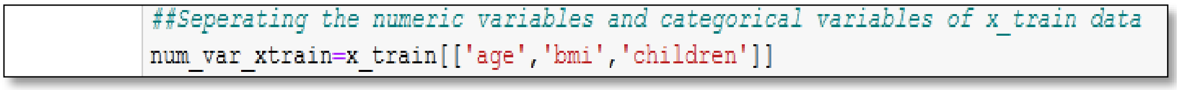
Once the dummy variables are created for the categorical variables the refined data is splitted into train data and test data. Train data consists of maximum part of the refined data and remaining part goes under the test data. In this analysis, the train data consists of 70% and the test data consists of remaining 30%.

```
In [48] : ###Spliting the insurance dataset into train data and test data
              y =ins_data.charges
              x=ins _data. drop ('charges', axis=1)
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.3,random_state
=0)
```

The MLR model is fitted on the train data and has used for the prediction of insurance charges on the test data.

Standardization of Variables:

We standardize the independent and dependent variables, except the dummy variables, so that they all get measured on the same scale.

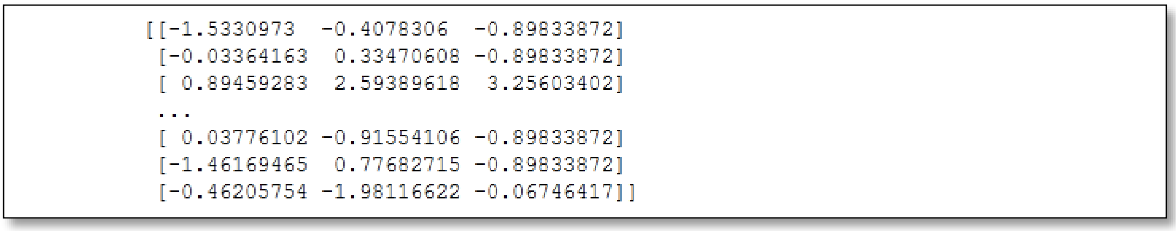


```
##Seperating the numeric variables and categorical variables of x_train data
num_var_xtrain=x_train[['age','bmi','children']]
```

```
##Seperating the numeric variable and categorical variable of x_train data
num_var_xtrain=x_train [[‘age’,’bmi’,’children’]]
```

```
In   [58]:cat_var_xtrain=x_train[[‘sex_male’,smoker_yes’   ,‘region_northwest’
       ,’region_southest’]]
```

```
In [59]: ##Standardizing the num_var_xtrain data
         scaler = StandardScaler ()
         a=scaler.fit_transform(num_var_xtrain)
         print(a)
      ### These codes will give the standardized values but in array form
```

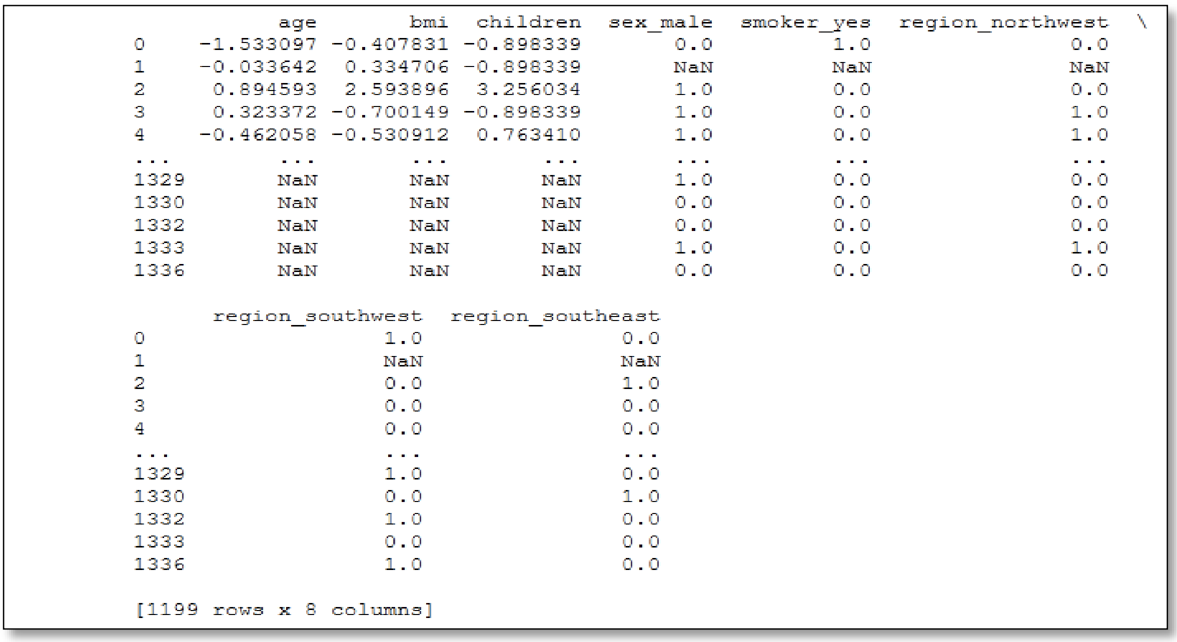


```
[[-1.5330973  -0.4078306  -0.89833872]
 [-0.03364163  0.33470608 -0.89833872]
 [ 0.89459283  2.59389618  3.25603402]
 ...
 [ 0.03776102 -0.91554106 -0.89833872]
 [-1.46169465  0.77682715 -0.89833872]
 [-0.46205754 -1.98116622 -0.06746417]]
```

Converting array into data frame

```
In [60]: ##Converting array into dataframe
      std_num_var_xtrain=pd.DataFrame(a,columns=[‘age’,’bmi’,’children’])
```

```
In [61]: std_xtrain=pd.concat([std_num_var_xtrain,cat_var_xtrain],axis=1)
         print(std_xtrain)
```

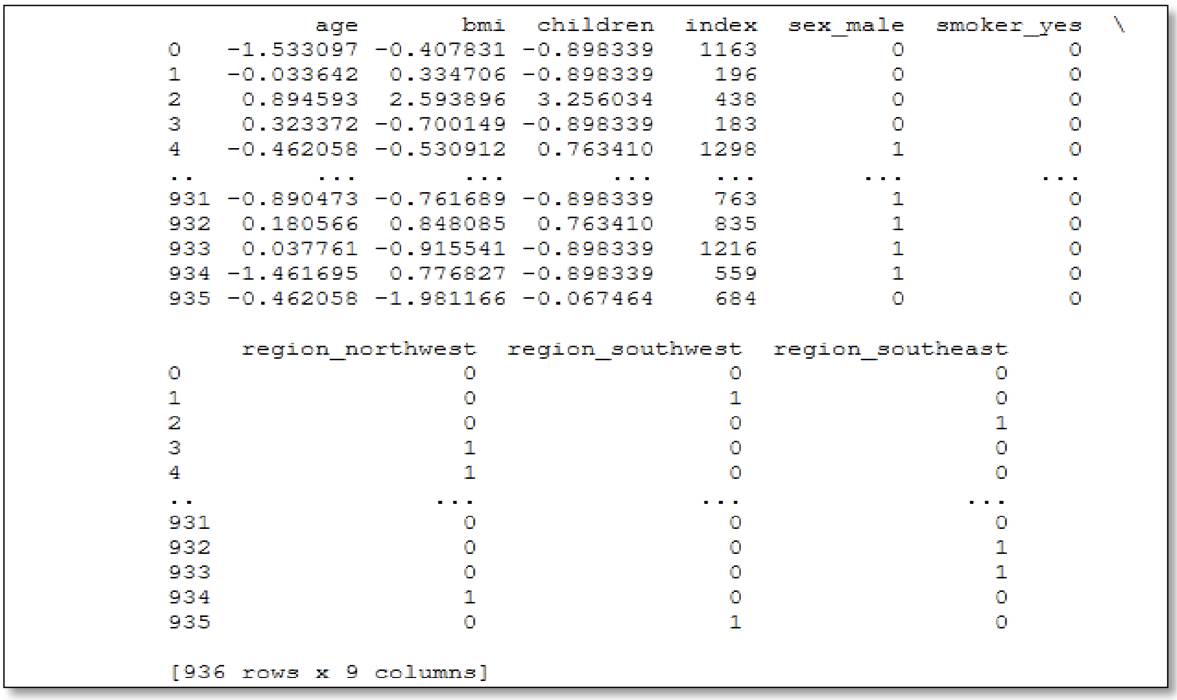


```
           age       bmi  children  sex_male  smoker_yes  region_northwest  \
0    -1.533097 -0.407831 -0.898339       0.0         1.0               0.0
1    -0.033642  0.334706 -0.898339       NaN         NaN               NaN
2     0.894593  2.593896  3.256034       1.0         0.0               0.0
3     0.323372 -0.700149 -0.898339       1.0         0.0               1.0
4    -0.462058 -0.530912  0.763410       1.0         0.0               1.0
...        ...       ...       ...       ...         ...               ...
1329       NaN       NaN       NaN       1.0         0.0               0.0
1330       NaN       NaN       NaN       0.0         0.0               0.0
1332       NaN       NaN       NaN       0.0         0.0               0.0
1333       NaN       NaN       NaN       1.0         0.0               1.0
1336       NaN       NaN       NaN       0.0         0.0               0.0

      region_southwest  region_southeast
0                  1.0               0.0
1                  NaN               NaN
2                  0.0               1.0
3                  0.0               0.0
4                  0.0               0.0
...                ...               ...
1329               1.0               0.0
1330               0.0               1.0
1332               1.0               0.0
1333               0.0               0.0
1336               1.0               0.0

[1199 rows x 8 columns]
```

Note: Since std_xtrain has “NaN” values and it is because std_num_var_xtrain and cat_var_xtrain both has different indexes therefore changing the index of cat_var_xtrain

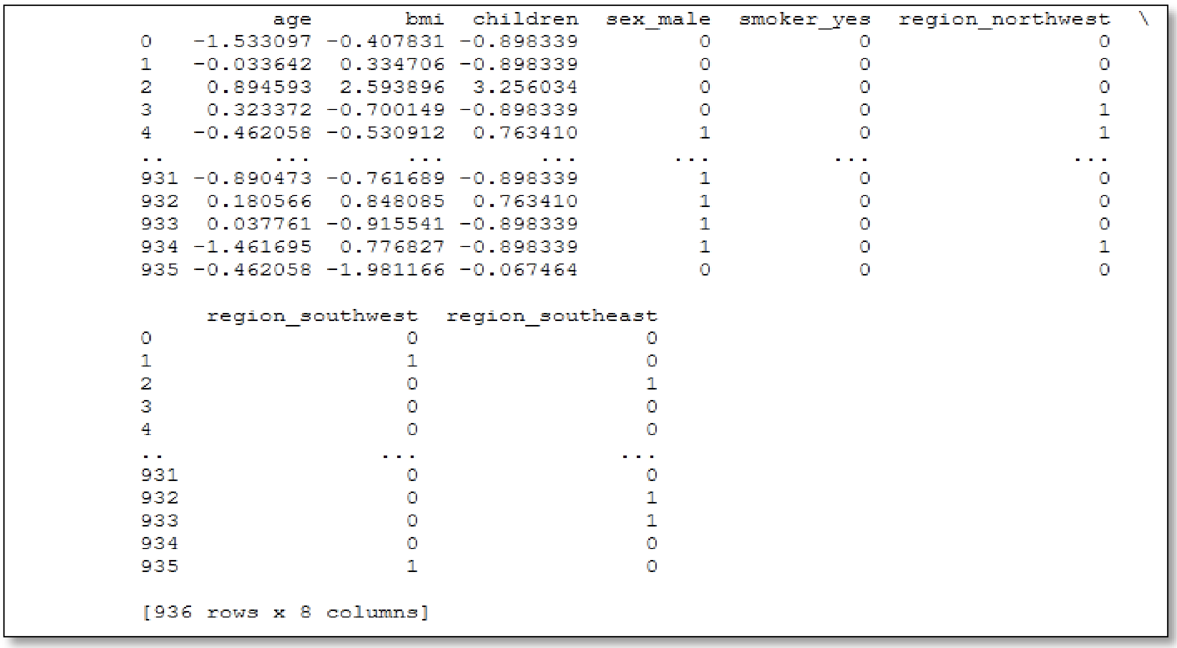
```
In [62]: cat_var_xtrain = cat_var_xtrain.reset_index()
         Std_xtrain=pd.concat([std_num_var_xtrain,cat_var_xtrain],axis=1)
         print(std_xtrain)
```

ISBN 978-93-80788-93-7



```
          age       bmi  children  index  sex_male  smoker_yes  \
0   -1.533097 -0.407831 -0.898339   1163         0           0
1   -0.033642  0.334706 -0.898339    196         0           0
2    0.894593  2.593896  3.256034    438         0           0
3    0.323372 -0.700149 -0.898339    183         0           0
4   -0.462058 -0.530912  0.763410   1298         1           0
..        ...       ...       ...    ...       ...         ...
931 -0.890473 -0.761689 -0.898339    763         1           0
932  0.180566  0.848085  0.763410    835         1           0
933  0.037761 -0.915541 -0.898339   1216         1           0
934 -1.461695  0.776827 -0.898339    559         1           0
935 -0.462058 -1.981166 -0.067464    684         0           0

     region_northwest  region_southwest  region_southeast
0                   0                 0                 0
1                   0                 1                 0
2                   0                 0                 1
3                   1                 0                 0
4                   1                 0                 0
..                ...               ...               ...
931                 0                 0                 0
932                 0                 0                 1
933                 0                 0                 1
934                 1                 0                 0
935                 0                 1                 0

[936 rows x 9 columns]
```

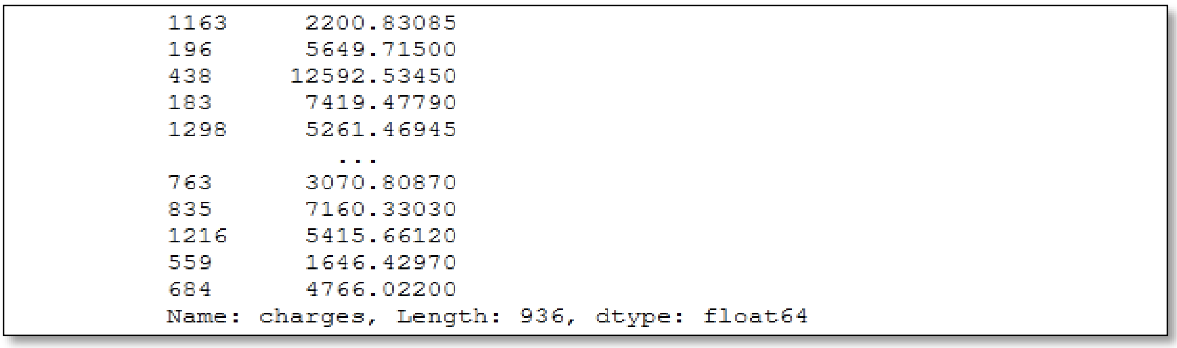
```
In [63]: std_xtrain=std_xtrain.drop(‘index’,axis=1)
         print(std_xtrain)
```



```
          age       bmi  children  sex_male  smoker_yes  region_northwest  \
0   -1.533097 -0.407831 -0.898339         0           0                 0
1   -0.033642  0.334706 -0.898339         0           0                 0
2    0.894593  2.593896  3.256034         0           0                 0
3    0.323372 -0.700149 -0.898339         0           0                 1
4   -0.462058 -0.530912  0.763410         1           0                 1
..        ...       ...       ...       ...         ...               ...
931 -0.890473 -0.761689 -0.898339         1           0                 0
932  0.180566  0.848085  0.763410         1           0                 0
933  0.037761 -0.915541 -0.898339         1           0                 0
934 -1.461695  0.776827 -0.898339         1           0                 1
935 -0.462058 -1.981166 -0.067464         0           0                 0

     region_southwest  region_southeast
0                   0                 0
1                   1                 0
2                   0                 1
3                   0                 0
4                   0                 0
..                ...               ...
931                 0                 0
932                 0                 1
933                 0                 1
934                 0                 0
935                 1                 0

[936 rows x 8 columns]
```

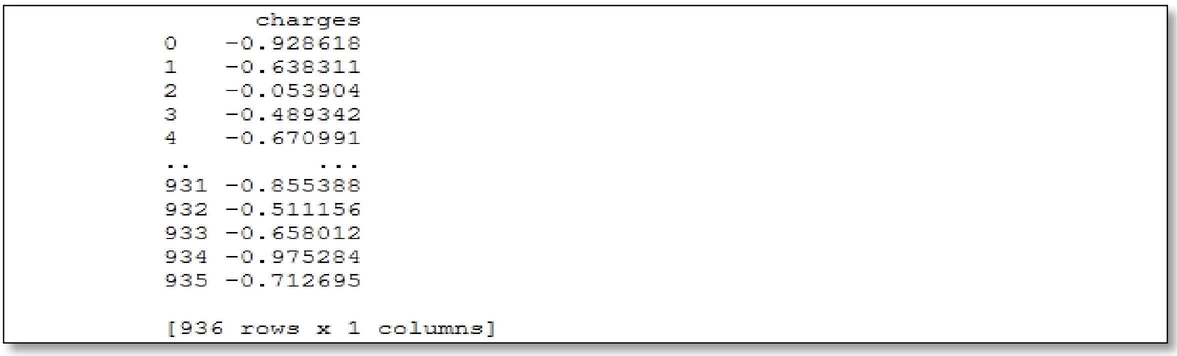
Standardizing y_train and y_test data

```
In [64]:##Standardizing y_train data
        ins_data.charges
        print(y_train)
```



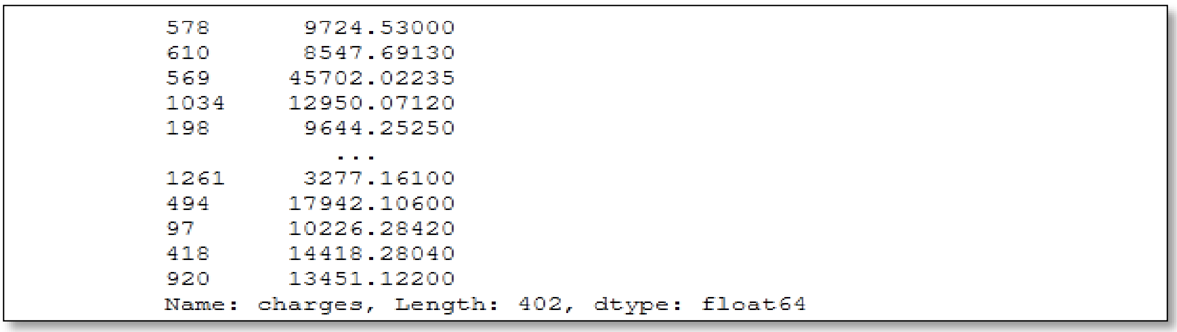
```
1163     2200.83085
196      5649.71500
438     12592.53450
183      7419.47790
1298     5261.46945
            ...
763      3070.80870
835      7160.33030
1216     5415.66120
559      1646.42970
684      4766.02200
Name: charges, Length: 936, dtype: float64
```

```
In [65]: mean_ytrain=y_train.mean()
         std_ytrain=y_train.std()
         std_ytrain=(y_train-mean_ytrain)/(std_ytrain)
         std_ytrain=std_ytrain.reset_index()
         std_ytrain=std_ytrain.drop(‘index , axis=1)
         print(std_ytrain)
```



```
      charges
0   -0.928618
1   -0.638311
2   -0.053904
3   -0.489342
4   -0.670991
..        ...
931 -0.855388
932 -0.511156
933 -0.658012
934 -0.975284
935 -0.712695

[936 rows x 1 columns]
```

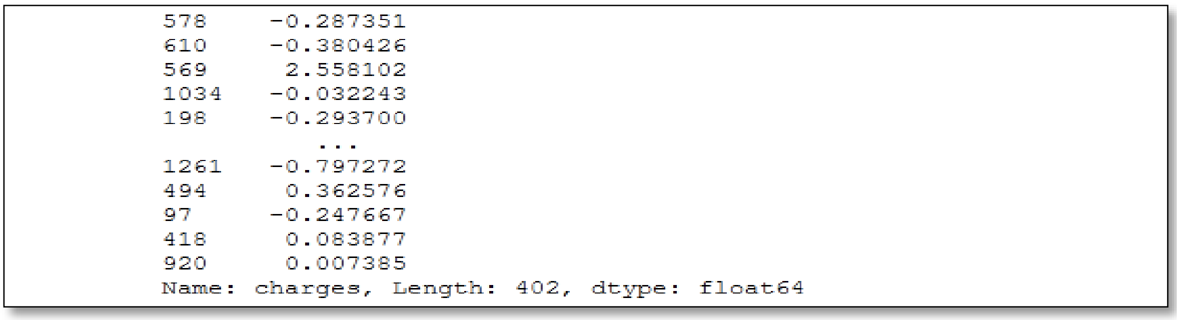
```
In [66]: ##Standardizing y_test data
         ins_data.charges
         print(y_test)
```



```
578      9724.53000
610      8547.69130
569     45702.02235
1034    12950.07120
198      9644.25250
            ...
1261     3277.16100
494     17942.10600
97      10226.28420
418     14418.28040
920     13451.12200
Name: charges, Length: 402, dtype: float64
```

```
In [67]: mean_ ytest=y_test.mean()
         std_ ytest=y_test.std()
         std_ ytest=(y_test-mean_ ytest)/(std_ ytest)
         print(std_ytest)
```

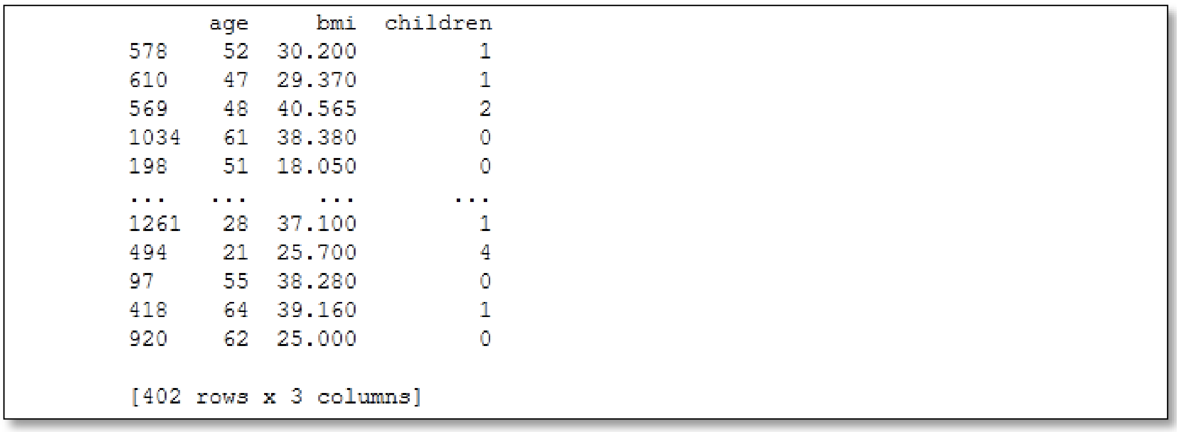
ISBN 978-93-80788-93-7



```
578     -0.287351
610     -0.380426
569      2.558102
1034    -0.032243
198     -0.293700
           ...
1261    -0.797272
494      0.362576
97      -0.247667
418      0.083877
920      0.007385
Name: charges, Length: 402, dtype: float64
```

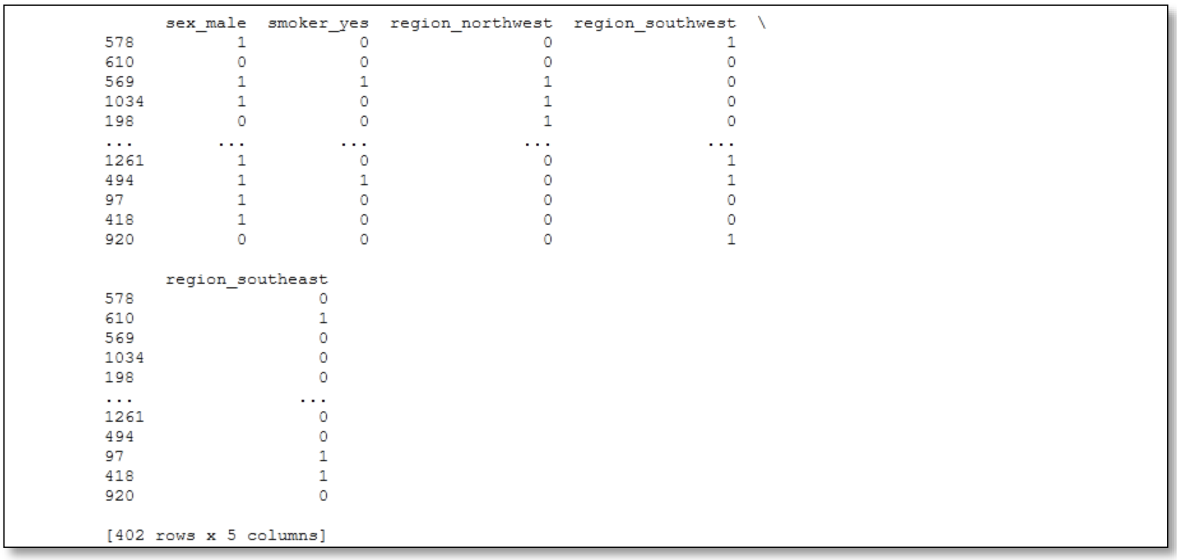
Separating the numeric variables and categorical variables of x_test data

```
In  [68]: ##Separating the numeric variables and categorical variables of
       #x_test data
       num_ var_xtest=x_test[[ 'age', 'bmi', 'children' ]]
       print (num_ var_xtest)
```



```
        age     bmi  children
578      52  30.200         1
610      47  29.370         1
569      48  40.565         2
1034     61  38.380         0
198      51  18.050         0
...     ...     ...       ...
1261     28  37.100         1
494      21  25.700         4
97       55  38.280         0
418      64  39.160         1
920      62  25.000         0

[402 rows x 3 columns]
```

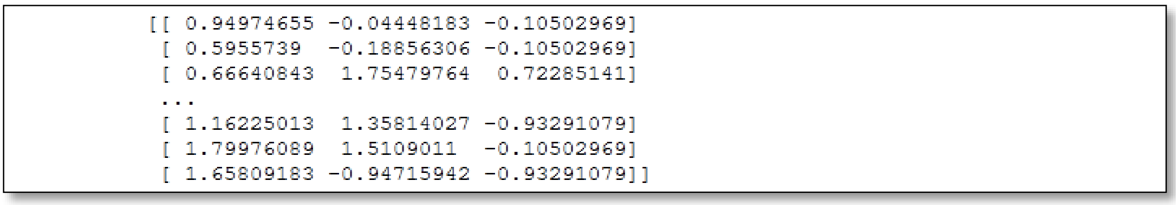
```
In[69]:      cat_var_xtest=x_test[['sex_male','smoker_yes','region_northwest',
       'region_southwest','region_southest']]
        print(cat_var_xtest)
```



```
      sex_male  smoker_yes  region_northwest  region_southwest  \
578          1           0                 0                 1
610          0           0                 0                 0
569          1           1                 1                 0
1034         1           0                 1                 0
198          0           0                 1                 0
...        ...         ...               ...               ...
1261         1           0                 0                 1
494          1           1                 0                 1
97           1           0                 0                 0
418          1           0                 0                 0
920          0           0                 0                 1

      region_southeast
578                  0
610                  1
569                  0
1034                 0
198                  0
...                ...
1261                 0
494                  0
97                   1
418                  1
920                  0

[402 rows x 5 columns]
```

Standardizing the num_var_xtrain data

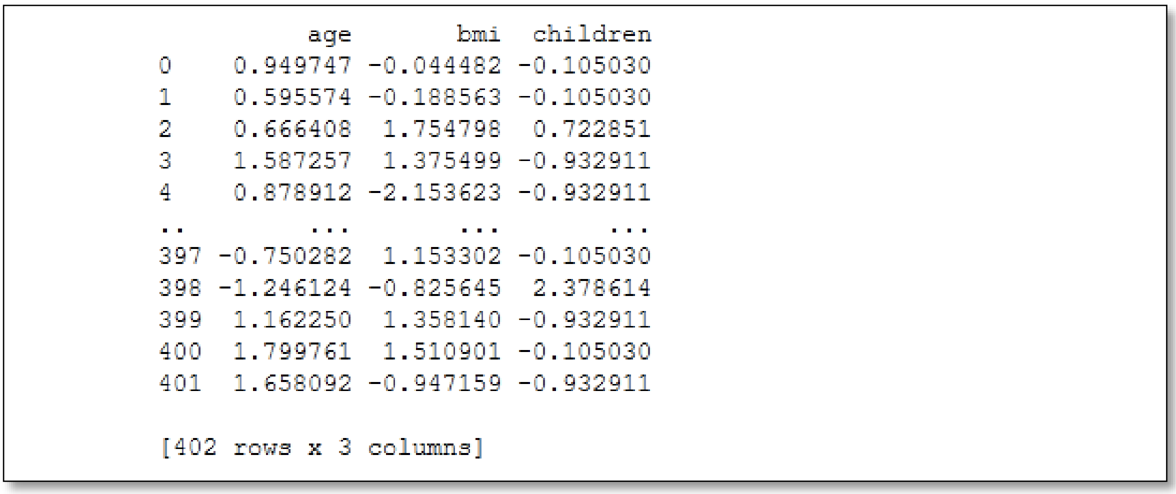
```
In [70]: ##Standardizing the num_var_xtrain data
         Scaler=StandardScaler()
         a=scaler.fit_transform(num_var_xtest)
         print(a)
  ###These codes will give the standardized values but in array form
```



```
[[ 0.94974655 -0.04448183 -0.10502969]
 [ 0.5955739  -0.18856306 -0.10502969]
 [ 0.66640843  1.75479764  0.72285141]
 ...
 [ 1.16225013  1.35814027 -0.93291079]
 [ 1.79976089  1.5109011  -0.10502969]
 [ 1.65809183 -0.94715942 -0.93291079]]
```

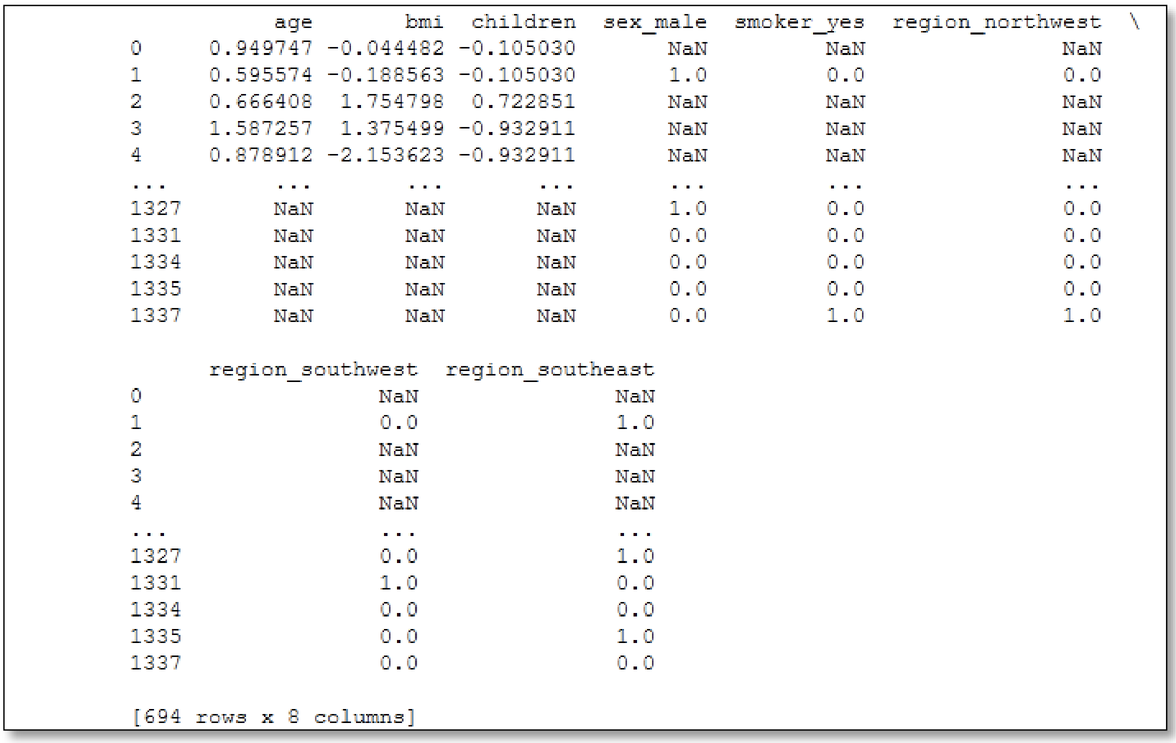
Converting array into data frame

```
In [71]: ##Converting array into data.frame
        std_num_var_xtest=pd.DataFrame(a,columns=['age','bmi','children'])
        print(std_num_var_xtest)
```



```
          age       bmi  children
0    0.949747 -0.044482 -0.105030
1    0.595574 -0.188563 -0.105030
2    0.666408  1.754798  0.722851
3    1.587257  1.375499 -0.932911
4    0.878912 -2.153623 -0.932911
..        ...       ...       ...
397 -0.750282  1.153302 -0.105030
398 -1.246124 -0.825645  2.378614
399  1.162250  1.358140 -0.932911
400  1.799761  1.510901 -0.105030
401  1.658092 -0.947159 -0.932911

[402 rows x 3 columns]
```

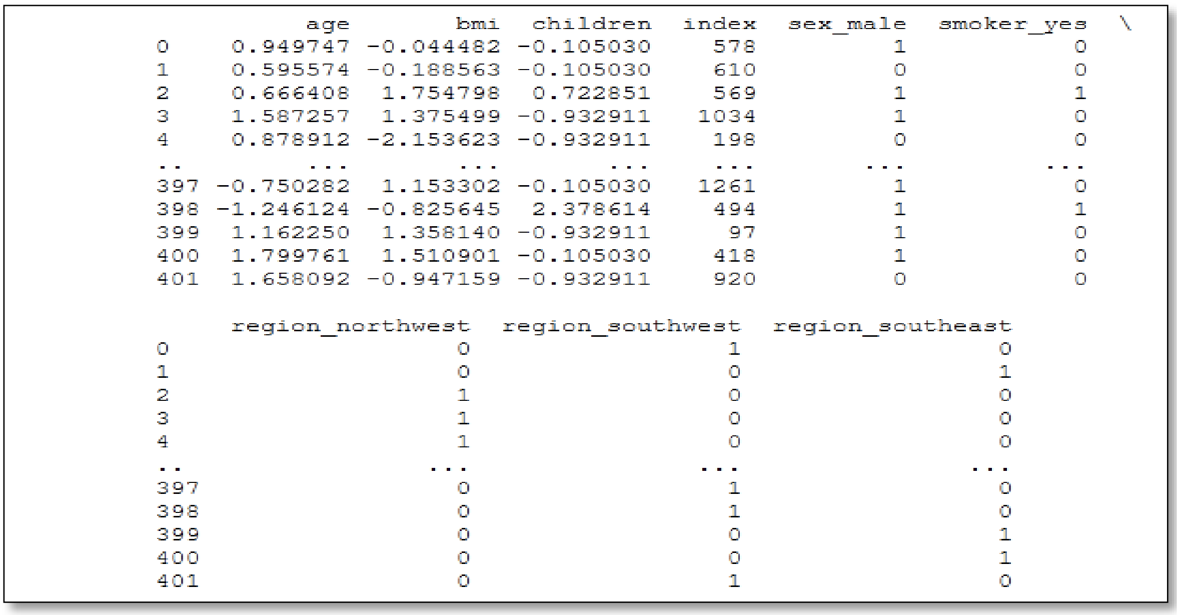
```
In [72]: std_xtest=pd.concat([std_num_var_xtest,cat_var_xtest],axis=1)
         print(std_xtest)
```



```
             age       bmi  children  sex_male  smoker_yes  region_northwest  \
0     0.949747 -0.044482 -0.105030       NaN         NaN               NaN
1     0.595574 -0.188563 -0.105030       1.0         0.0               0.0
2     0.666408  1.754798  0.722851       NaN         NaN               NaN
3     1.587257  1.375499 -0.932911       NaN         NaN               NaN
4     0.878912 -2.153623 -0.932911       NaN         NaN               NaN
...        ...       ...       ...       ...         ...               ...
1327       NaN       NaN       NaN       1.0         0.0               0.0
1331       NaN       NaN       NaN       0.0         0.0               0.0
1334       NaN       NaN       NaN       0.0         0.0               0.0
1335       NaN       NaN       NaN       0.0         0.0               0.0
1337       NaN       NaN       NaN       0.0         1.0               1.0

      region_southwest  region_southeast
0                  NaN               NaN
1                  0.0               1.0
2                  NaN               NaN
3                  NaN               NaN
4                  NaN               NaN
...                ...               ...
1327               0.0               1.0
1331               1.0               0.0
1334               0.0               0.0
1335               0.0               1.0
1337               0.0               0.0

[694 rows x 8 columns]
```

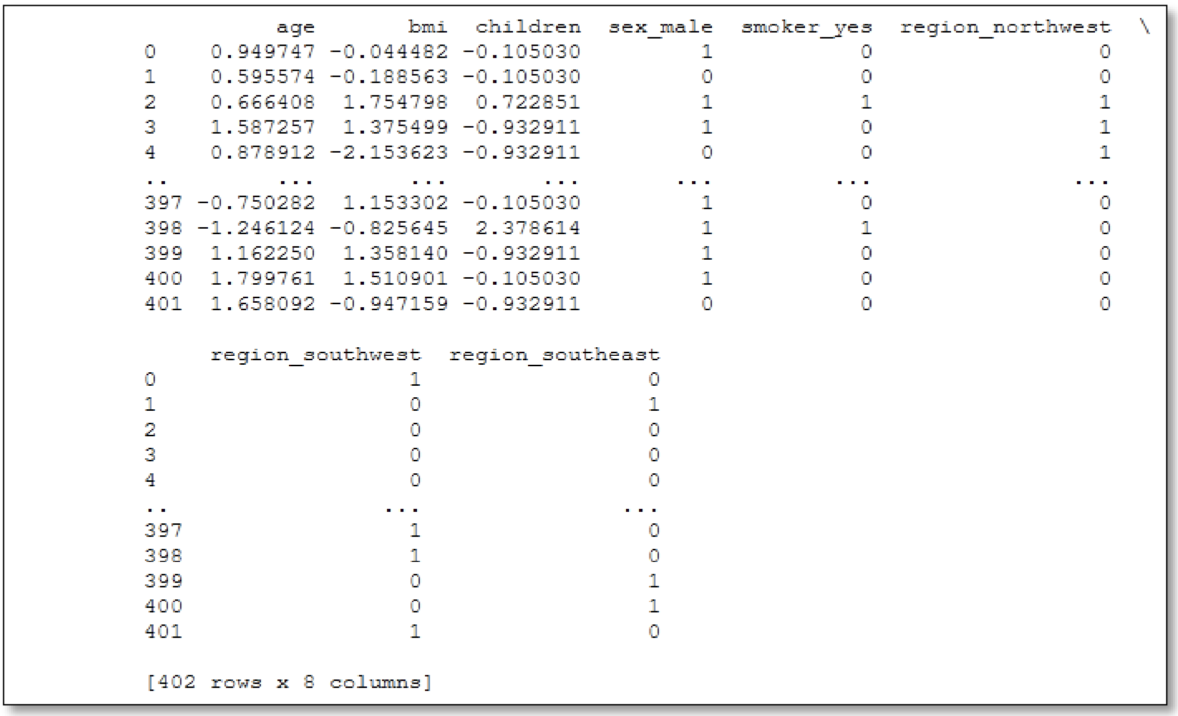
Note: Since std_xtest has NaN values and it is because std_num_var_xtest and cat_var_xtest both has different indexes therefore changing the index of cat_var_xtest

```
In [73]: cat_var_xtest = cat_var_xtest.reset_index()
         std_xtest=pd. concat ( [std_num_var_xtest, cat_var_xtest],axis=1)
         print(std_xtest)
```



```
          age       bmi  children  index  sex_male  smoker_yes  \
0    0.949747 -0.044482 -0.105030    578         1           0
1    0.595574 -0.188563 -0.105030    610         0           0
2    0.666408  1.754798  0.722851    569         1           1
3    1.587257  1.375499 -0.932911   1034         1           0
4    0.878912 -2.153623 -0.932911    198         0           0
..        ...       ...       ...    ...       ...         ...
397 -0.750282  1.153302 -0.105030   1261         1           0
398 -1.246124 -0.825645  2.378614    494         1           1
399  1.162250  1.358140 -0.932911     97         1           0
400  1.799761  1.510901 -0.105030    418         1           0
401  1.658092 -0.947159 -0.932911    920         0           0

     region_northwest  region_southwest  region_southeast
0                   0                 1                 0
1                   0                 0                 1
2                   1                 0                 0
3                   1                 0                 0
4                   1                 0                 0
..                ...               ...               ...
397                 0                 1                 0
398                 0                 1                 0
399                 0                 0                 1
400                 0                 0                 1
401                 0                 1                 0
```

```
In [74]: std_xtest.columns
         std_xtest=std_xtest.drop('index',axis=1)
         print(std_xtest)
```



```
          age       bmi  children  sex_male  smoker_yes  region_northwest  \
0    0.949747 -0.044482 -0.105030         1           0                 0
1    0.595574 -0.188563 -0.105030         0           0                 0
2    0.666408  1.754798  0.722851         1           1                 1
3    1.587257  1.375499 -0.932911         1           0                 1
4    0.878912 -2.153623 -0.932911         0           0                 1
..        ...       ...       ...       ...         ...               ...
397 -0.750282  1.153302 -0.105030         1           0                 0
398 -1.246124 -0.825645  2.378614         1           1                 0
399  1.162250  1.358140 -0.932911         1           0                 0
400  1.799761  1.510901 -0.105030         1           0                 0
401  1.658092 -0.947159 -0.932911         0           0                 0

     region_southwest  region_southeast
0                   1                 0
1                   0                 1
2                   0                 0
3                   0                 0
4                   0                 0
..                ...               ...
397                 1                 0
398                 1                 0
399                 0                 1
400                 0                 1
401                 1                 0

[402 rows x 8 columns]
```

Predictive Modeling:

Multiple Linear Regression

It is a statistical tool that allows you to examine how multiple independent variables are related to a dependent variable. Once we have identified how these multiple variables relate to your dependent variable, we can take information about all of the independent variables and use it to make much more powerful and accurate predictions about why things are the way they are. This latter process is called "Multiple Regression".

A population model for a multiple linear regression model that relates a y-variable to p -1 x-variables is written as $Yi =\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} +\ldots+ \beta_{(p-1)} x_{i\,(p-1)} + \epsilon_i$.

Dependent and Independent variable for modeling

In our given data Charges i.e. Individual medical costs billed by health insurance i.e. Y = Charges. Independent variables are Age, BMI, Children, Sex, Smoker and Region.

Hypothesis Testing

H_0: $\beta_1= \beta_2= \beta_3=\ldots\ldots= \beta_p=0$ i.e. No variable is significant, against

H_1: Not H_0, i=1, 2, 3…p, i.e. Atleast one variable is significant

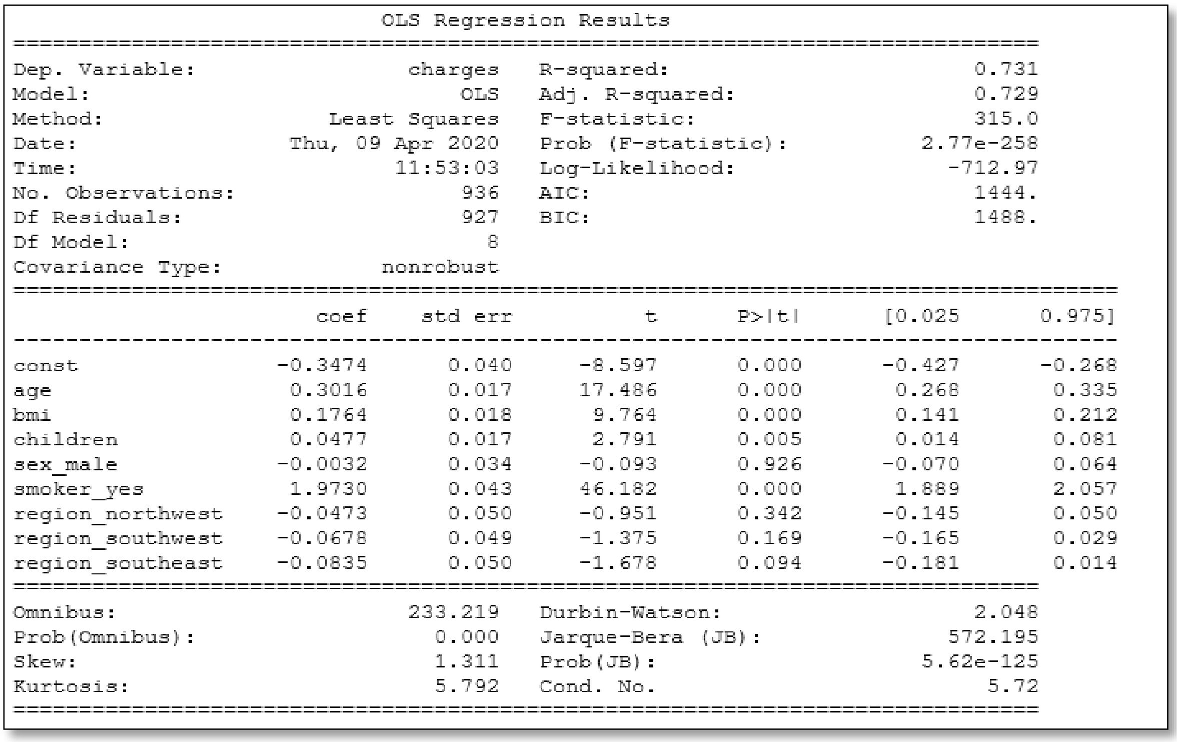
Model Building

Multiple linear regression model between our dependent and independent variables is

Y (charges) = β_0+ β_1*(age) + β_2*(BMI) + β_3*(children) + β_4*(sex_male) + β_5*(smoker_yes) + β_6*(region_northwest) + β_7*(region_southeast) + β_8*(region_southwest)

```
In [78]: ###Fitting of regression model by statsmodel.api library
```

```
               X= sm.add_ constant(std_xtrain) #adding a constant
               std_ytrain = pd.DataFrame(std_ytrain)
               model = sm.OLS(std_ytrain,X).fit()    ###Regression model
               y_train_pred =model.predict(X)
               print_model =model.summary()
               print(print_model)
```



```
                            OLS Regression Results
==============================================================================
Dep. Variable:                charges   R-squared:                       0.731
Model:                            OLS   Adj. R-squared:                  0.729
Method:                 Least Squares   F-statistic:                     315.0
Date:                Thu, 09 Apr 2020   Prob (F-statistic):          2.77e-258
Time:                        11:53:03   Log-Likelihood:                -712.97
No. Observations:                 936   AIC:                             1444.
Df Residuals:                     927   BIC:                             1488.
Df Model:                           8
Covariance Type:            nonrobust
====================================================================================
                       coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------------
const               -0.3474      0.040     -8.597      0.000      -0.427      -0.268
age                  0.3016      0.017     17.486      0.000       0.268       0.335
bmi                  0.1764      0.018      9.764      0.000       0.141       0.212
children             0.0477      0.017      2.791      0.005       0.014       0.081
sex_male            -0.0032      0.034     -0.093      0.926      -0.070       0.064
smoker_yes           1.9730      0.043     46.182      0.000       1.889       2.057
region_northwest    -0.0473      0.050     -0.951      0.342      -0.145       0.050
region_southwest    -0.0678      0.049     -1.375      0.169      -0.165       0.029
region_southeast    -0.0835      0.050     -1.678      0.094      -0.181       0.014
==============================================================================
Omnibus:                      233.219   Durbin-Watson:                   2.048
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              572.195
Skew:                           1.311   Prob(JB):                    5.62e-125
Kurtosis:                       5.792   Cond. No.                         5.72
==============================================================================
```

To obtain Residuals

```
Residuals= model.resid              ##Residuals
###Testing the model on test dataset
X = sm.add_constant(std_xtest)      #adding a constant
y_test_pred = model.predict(X)
```

Assumptions Validation:

1. Multicollinearity

Multicollinearity exists when two or more of the predictors (independent variables) in a regression model are moderately or highly correlated. Unfortunately, when it exists, it can wreak havoc on our analysis and thereby limit the research conclusions we can draw.

Detection

Multicollinearity may be checked multiple ways:

i) Correlation matrix –When computing a matrix of Pearson's bivariate correlations among all independent variables, the magnitude of the correlation coefficients should be less than 0.80.

ii) Variance Inflation Factor (VIF) – The VIFs of the linear regression indicate the degree that the variances in the regression estimates are increased due to multicollinearity. VIF values higher than 5 indicate that multicollinearity is a problem.

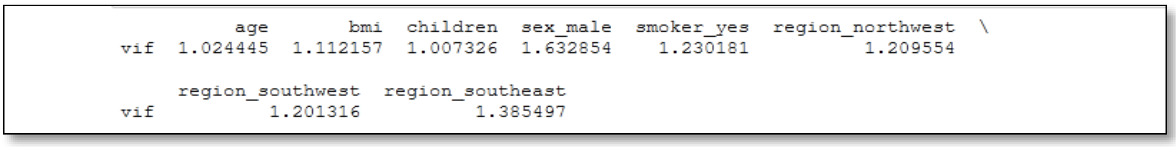
Remedies

- Drop one of the independent variable which is explained by others
- Use Principal Component Regression in case of severe multicollinearity
- Use Ridge Regression

In our data we checked the multicollinearity by using VIF. Here we considered VIF threshold value equals to 5. Consider the model,

Y (charges) = β_0+ β_1*(age) + β_2*(BMI) + β_3*(children) + β_4*(sex_male) + β_5*(smoker_yes) + β_6*(region_northwest) + β_7*(region_southeast) + β_8*(region_southwest)

```
In [83]: ##By VIF
         vif  =    [variance_inflation_factor(std_xtrain.values,i)for  I  in
         range(std_xtrain.shape[1])]
         A=pd.DataFrame((‘vif’ :vif[0:]), index=std_xtrain.columns.T
         print(a)
```



```
          age       bmi  children  sex_male  smoker_yes  region_northwest  \
vif  1.024445  1.112157  1.007326  1.632854    1.230181          1.209554

     region_southwest  region_southeast
vif          1.201316          1.385497
```

Here as we can see none of the values of VIF is greater than or equal to 5. Therefore we can say that there is no multicollinearity in the data.

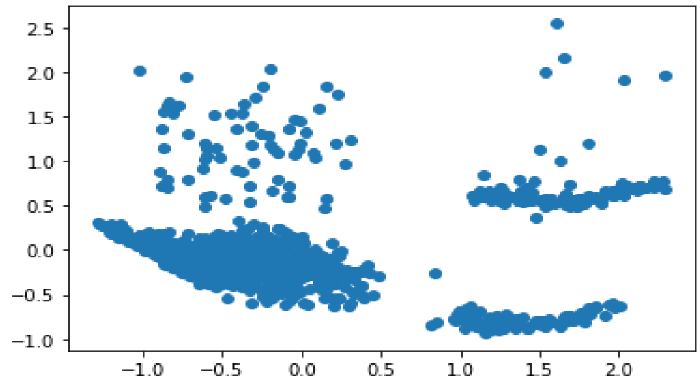
2. Linearity

In multiple linear regression we assume that there is linear relationship between the response variable and predictors.

Detection

One can detect linearity form scatter plot. Another way of testing linearity between the variables is by using the test called Harvey-Collier multiplier test. The Harvey-Collier test performs a t-test on the recursive residuals. If the true relationship is not linear but convex or concave the mean of the recursive residuals should differ from 0 significantly. This means that a significant result means that you can *reject* the null hypothesis that the true model is linear.

```
In [79]: ###Testing for linearity of dependemt and independent variables
        plt.scatter(y_train_pred,residuals)
        plt.show()
```



The scatter plot is slightly convex downward which represents non linearity between the dependent and independent variables.

```
In [80]: ###Another way of testing Linearity assumption
        '''Harvey-Collier  multiplier  test  for  Null  hypothesis  that  the
           linear specification is correct'''
         name = ['t value', 'p value']
         test= sms.linear_harvey_collier(model)
         lzip(name,test)
```

Note: Here the code sms.linear_harvey_collier() will throw an error of singular matrix and that is because of absence of multicollinearity in the data. If the variables would have highly correlated it wouldn't have thrown the error.

Remedy

If this assumption is not satisfied then use some appropriate transformation. As in our case we have applied Log transformation on our response variable.

For this data I have tried to implement different transformations but still the data was not satisfying the linearity assumption. Hence I have proceeded forward without satisfying this assumption.

3. Homoscedasticity

The assumption of homoscedasticity (meaning "same variance") is central to linear regression models. Homoscedasticity describes a situation in which the error term (that is, the "noise" or random disturbance in the relationship between the independent variables and the dependent variable) is the same across all values of the independent variable.

Detection

Homoscedasticity can be tested statistically by **Breush-Pagan test.** Consider the hypothesis,

H_0: The variance of the residuals is constant, against

H_1: The variance of the residuals is not constant.

Breush Pagan Test

```
In [77]: ###Homoscedasticity
        import statsmodels.stats.api as sms
        import statsmodels.formula.api as smf
        #Breusch-Pagan Test for Heteroscedasticity
        bptest=sms.diagnostic.het_breuschpagan(residuals,model.model.exog)
        print(bptest)
      (86.48898360885777, 2.3947340156043142e-15, 11.797270173435852,
                 4.3049684277300757e-16)
```

Here the p-value is lesser than significance level 0.05 hence we reject H_0 and conclude that the variance of the residuals is not constant.

Remedy

The problem of Heteroscedasticity can be resolved by using some appropriate transformation on dependent variables. The Box-Cox transformation is the technique to find out the appropriate transformation for the response variable. Here we tried for both the remedies but couldn’t work for this data. Hence, I have proceeded for the further assumptions without solving the problem of heteroscedasticity.

4. Normality of the residuals

The residual terms are assumed to be normally distributed with mean 0 and variance σ^2.

Detection

For univariate case this assumption can be checked by using Shapiro Wilk’s test. For multivariate this assumption can be checked by Jarque-Bera test.

```
In [78]: ###Test for Normality of residuals
         name=[‘Jarque-Bera’, ‘Chi^2 two-tail prob.’, ‘skew’, Kurtosis’]
         test = sms . jarque_bera(residuals)
         lzip(name,test)
Out  [78]:  ((‘Jarque-Bera’,  572.1949328827559),  (‘Chi^2  two-tail  prob.’,
5.61628370588809e-125),(‘Skew’,       1.311358707239599),       (‘Kurtosis’,
5.791597805827201)]
```

Here the p-value is lesser than significance level 0.05 hence we reject H_0 and conclude that the residuals do not follow normal distribution.

Remedies

- A usual remedy is to use a transformation of the variables to make them closer to normally distributed.
- Generalized linear mixed model

We tried for the transformation of the variables but couldn't attain the normality of the residuals. Further, we didn't try for the generalized linear mixed model because already above mentioned assumptions are violated and the remedy didn't work. So even if this remedy could have worked to attain the normality of residuals still we couldn't fit the MLR model.

5. Autocorrelation

The assumption of uncorrelated and independent error terms for regression models using time series data is not always appropriate. Usually the errors in the time series data exhibit serial correlation. i.e. E(ε_i) and E(ε_j) is not zero. Such error terms are said to be autocorrelated.

Detection

i) Plot of residuals versus time

ii) Durbin Watson test

I have used Durbin Watson test for detecting the autocorrelation in the data.

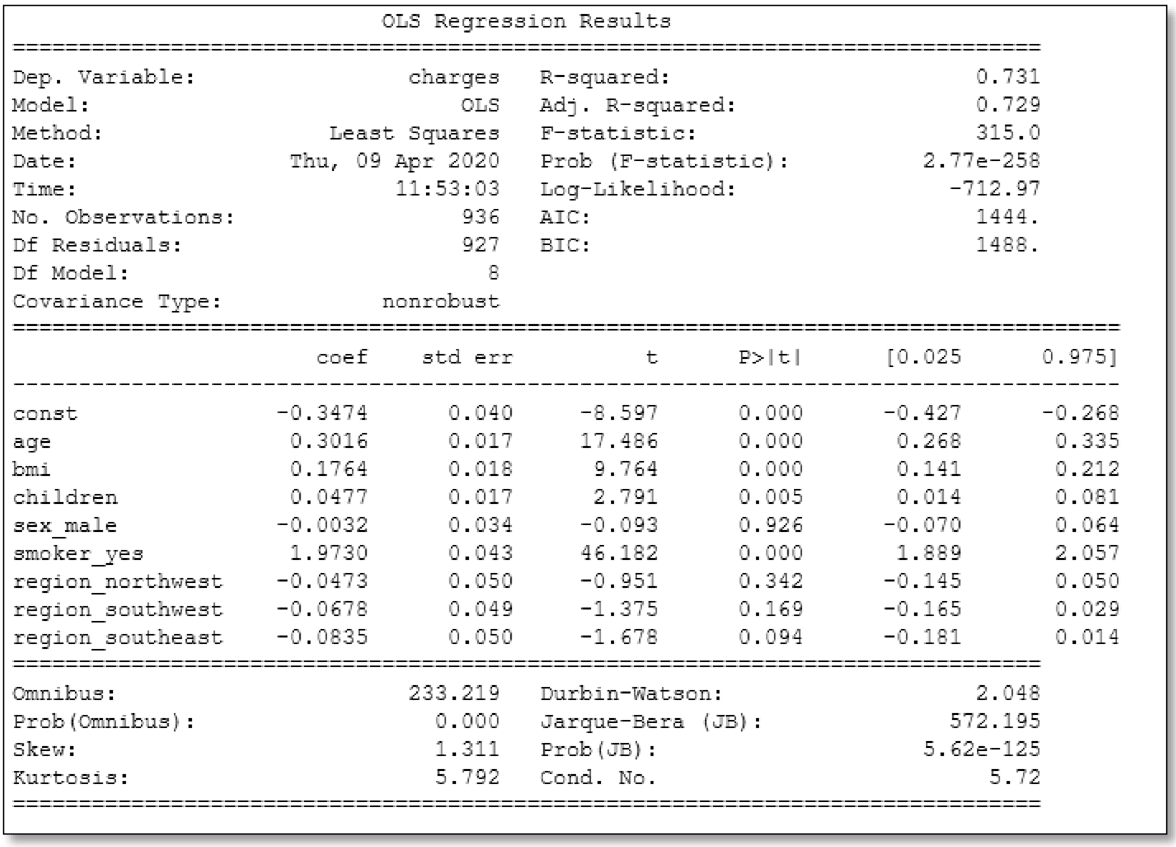
H_0: $\rho = 0$ i.e. there is no autocorrelation present in the data

H_1: $\rho \neq 0$ i.e. there is autocorrelation present in the data

where, ρ is autocorrelation factor

```
In [78]: ###Fitting of regression model by statsmodel.api library
         X= sm.add_ constant(std_xtrain) #adding a constant
         std_ytrain = pd.DataFrame(std_ytrain)
         model = sm.OLS(std_ytrain,X).fit()             ###Regression model
         y_train_pred =model.predict(X)
         print_model =model.summary()
         print(print_model)
```

ISBN 978-93-80788-93-7



```
                            OLS Regression Results
==============================================================================
Dep. Variable:                charges   R-squared:                       0.731
Model:                            OLS   Adj. R-squared:                  0.729
Method:                 Least Squares   F-statistic:                     315.0
Date:                Thu, 09 Apr 2020   Prob (F-statistic):          2.77e-258
Time:                        11:53:03   Log-Likelihood:                -712.97
No. Observations:                 936   AIC:                             1444.
Df Residuals:                     927   BIC:                             1488.
Df Model:                           8
Covariance Type:            nonrobust
====================================================================================
                       coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------------
const               -0.3474      0.040     -8.597      0.000      -0.427      -0.268
age                  0.3016      0.017     17.486      0.000       0.268       0.335
bmi                  0.1764      0.018      9.764      0.000       0.141       0.212
children             0.0477      0.017      2.791      0.005       0.014       0.081
sex_male            -0.0032      0.034     -0.093      0.926      -0.070       0.064
smoker_yes           1.9730      0.043     46.182      0.000       1.889       2.057
region_northwest    -0.0473      0.050     -0.951      0.342      -0.145       0.050
region_southwest    -0.0678      0.049     -1.375      0.169      -0.165       0.029
region_southeast    -0.0835      0.050     -1.678      0.094      -0.181       0.014
==============================================================================
Omnibus:                      233.219   Durbin-Watson:                   2.048
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              572.195
Skew:                           1.311   Prob(JB):                    5.62e-125
Kurtosis:                       5.792   Cond. No.                         5.72
==============================================================================
```

These are the codes for model building and model summary by using statsmodels library of python. Model summary includes Durbin Watson value and therefore there is no need to impute it by using different codes.

In present study, Durbin Watson value is 2.048 which is approximately equal to 2. Hence from the interpretation of Durbin Watson test we can say that there is no autocorrelation in the data.

From the above summary we can see that variables age, BMI, children and region_northwest are the significant variables. Hence these variables should be included in the model and all other variables should be excluded.

The adjusted-R^{2}value suggests the proportion of the variance explained by the model. 72.9% of the total variance is explained by our model.

Interpretation of βi's:
1-unit increase in X multiplies the expected value of Y by corresponding β value.

Since the assumptions were not satisfied I couldn't proceed for the further analysis by using multiple linear regression algorithms and hence I had selected another technique called Random Forest Regression for the predictive analysis.

Random Forest Regression:

Random forest is a Supervised Learning algorithm which uses ensemble learning method for classification and regression**.** Random forest is a bagging technique. Bootstrap Aggregation (Bagging). Bootstrap refers to random sampling with replacement. Bootstrap allows us to better understand the bias and the variance with the dataset. Bootstrap involves random sampling of small subset of data from the dataset.) The trees in random forests are run in parallel. There is no interaction between these trees while building the trees. It operates by constructing a multitude of decision trees at training time and outputting the class that is the mean prediction (regression) of the individual trees.

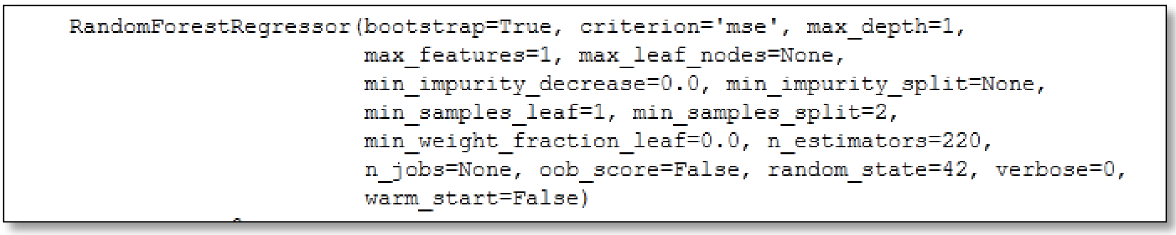
```
import os
from sklearn.metrics import accuracy_score
from  sklearn.metrics import mean_squared_error
from sklearn.model_selection import train_test_split
from sklearn. ensemble import RandomForestRegressor
                                        #Required to build random forest
```

For this algorithm, first created dummies for the categorical variables and splitted the data in train data and test data. After splitting the data into two different data we standardized the every variable.

Note: The codes for creating dummy variables, splitting the data into train and test and standardization of the variables are already given above.

Building Random Forest Regression Model

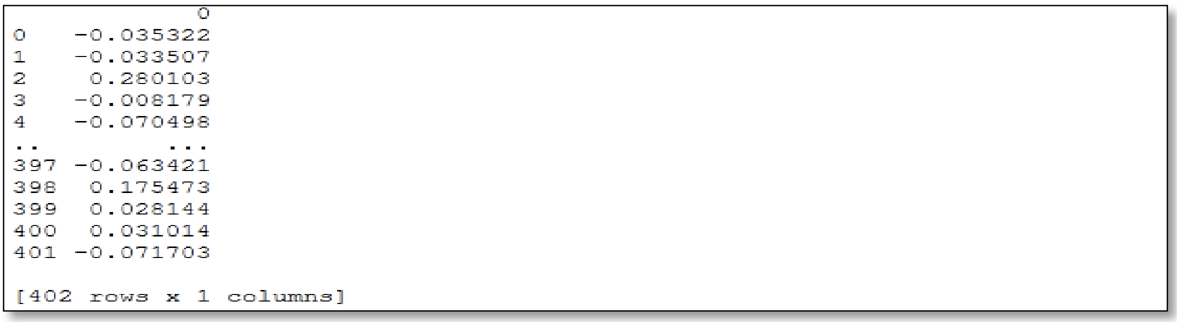
```
In [80]: ######Random Forest Regression ########
##Create_Separate_dataframe_consisting_of_only_dependent_variable
        y=ins_data.charges
        x= ins_data.drop('charges',axis=1)

##Create_Separate_dateframe_consisting_of_only_independent_variable
        x= data.drop(columns=('charges'), inplace =False,axis=1)

##Split the data into train end test and then standardize the both the data by
using the above codes
      #####    Fitting    the    regression    model    to    the    dataset#####

regressor=RandomForestRegressor(n_estimators=220,


                                     max_features=1,
max_depth=1,random_state = 42)
         model-regressor.fit(std_train, std_ytrain)
print(model)
```



```
    RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=1,
                          max_features=1, max_leaf_nodes=None,
                          min_impurity_decrease=0.0, min_impurity_split=None,
                          min_samples_leaf=1, min_samples_split=2,
                          min_weight_fraction_leaf=0.0, n_estimators=220,
                          n_jobs=None, oob_score=False, random_state=42, verbose=0,
                          warm_start=False)
```

Prediction on the test data

```
y_pred = regressor.predict(std_xtest)
test_values =pd.DataFrame(y_pred)
tv _head=test_ values .head ()
print(test_values)
```



```
            0
0   -0.035322
1   -0.033507
2    0.280103
3   -0.008179
4   -0.070498
..        ...
397 -0.063421
398  0.175473
399  0.028144
400  0.031014
401 -0.071703

[402 rows x 1 columns]
```

Checking for the overfit of the model

```
In [94]: ##Mean Square Error and Root mean square error
         mse=mean_squared_error(std_ytest,test_values)
         rmse-np. sqrt (mse)
         print("Mean Square Error:”, mse)
         print("Root Mean Square Error:”,rmse)
         Mean Square Error: 0.8088461734961657
         Root Mean Square Error: 0.8993587568352052
```

MSE Interpretation

A larger MSE means that the data values are dispersed widely around its central moment (mean), and a smaller MSE means otherwise and it is definitely the preferred and/or desired choice as it shows that data values are dispersed closely to its central moment (mean); which is usually great.

RMSE Interpretation

The RMSE is the square root of the variance of the residuals. It indicates the absolute fit of the model to the data–how close the observed data points are to the model's predicted values. Whereas R-squared is a relative measure of fit, RMSE is an absolute measure of fit. As the square root of a variance, RMSE can be interpreted as the standard deviation of the unexplained variance, and has the useful property of being in the same units as the response variable. Lower values of RMSE indicate better fit. RMSE is a good measure of how accurately the model predicts the response.

Here the MSE and RMSE are 0.808846 and 0.899 which is very small and therefore we can say that the model is a good fit.

13.3 Conclusion

The patients which are older in age, having high BMI with smoking habit are found to have high medical charges. The region southeast has maximum number of patients than any other region because the number of smokers having high BMI are more in that region. The number of patients covered by the medical health insurance is the aged patients. So eventually the people who have got the medical insurance are the older in age, have smoking habit, have high BMI and they mostly belongs to southeast region.

13.4 References

1. https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html
2. https://www.geeksforgeeks.org/random-forest-regression-in-python/
3. https://towardsdatascience.com/random-forest-and-its-implementation-71824ced454f

Chapter 14
Factor Analysis

***Mr. Prathamesh Thite**, Data Analyst, Dinero Software Pvt. Ltd.*

14.1 Introduction

Factor analysis is a technique that is used to reduce a large number of variables into fewer numbers of factors. This technique extracts maximum common variance from all variables and puts them into a common score. As an index of all variables, we can use this score for further analysis.

Factor analysis is part of general linear model (GLM) and this method also assumes several assumptions:

1. **No outlier:** Assume that there are no outliers in data.
2. **Adequate sample size:** The case must be greater than the factor.
3. **No perfect multicollinearity:** Factor analysis is an interdependency technique. There should not be perfect multicollinearity between the variables.
4. **Homoscedasticity:** Since factor analysis is a linear function of measured variables, it does not require homoscedasticity between the variables.
5. **Linearity:** Factor analysis is also based on linearity assumption. Non-linear variables can also be used. After transfer, however, it changes into linear variable.

The factor analysis model can be written algebraically as follows. If you have p variables $X_1, X_2, \dots, X_p$ measured on a sample of n subjects, then variable X_i can be written as a linear combination of m factors F1, F2, . . . , Fm where, as explained above m < p. Thus,

$$X_i = a_{i1}F_1 + a_{i2}F_2 + \cdots + a_{im}F_m + e_i$$

where, 'a' is are the factor loadings (or scores) for variable X_i and e_i is the part of variable X_i that cannot be 'explained' by the factors.

Why it is necessary to reduce dimensions of data?

In terms of performance, having data of high dimensionality is problematic because (a) it can mean high computational cost to perform learning and inference and (b) it often leads to overfitting when learning a model, which means that the model will perform well on the training data but poorly on test data. Dimensionality reduction addresses both of these problems, while (hopefully) preserving most of the relevant information in the data needed to learn accurate, predictive models.

Also, we can easily interpret 2D plots and it becomes difficult when we have more dimensions. So, dimensionality reduction technique plays very important role by selecting relevant features without losing much information of the data.

14.2 Abstract

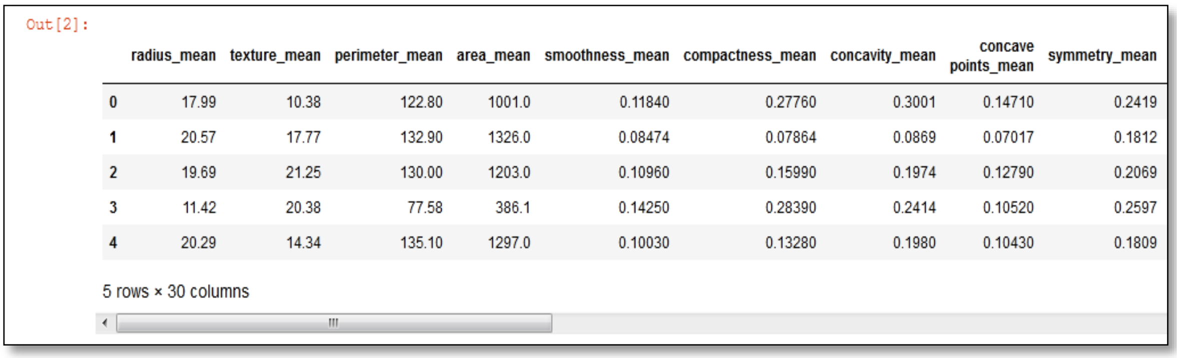
This study aims for learning the technique of Factor Analysis using Python and its application on identifying the factors to find breast cancer. This case study is based on data **"path.csv"** of 569 patients and to find important variables out of 30 variables namely form of radius_mean, texture_mean, perimeter_mean, area_mean, smoothness_mean, compactness_mean, concavity_mean, concave points_mean, symmetry_mean, fractal_dimenion_mean,radius_se, texture_se, perimeter_se, area_se, smoothness_se, compactness_se, concavity_se, concave points_se, symmetry_se, fractal_dimenion_se, radius_worst, texture_worst, perimeter_worst, area_worst, smoothness_worst, compactness_worst, concavity_worst, concave point_worst, symmetry_worst, fractal_dimenion_worst. Factor Analysis is used to understand the correlation structure of collected data and identifying the most important factors for identifying breast cancer.

14.3 Analysis

#Importing required Packages and Data in Python

```
In [1]: import pandas as pd
          from factor_analyzer import FactorAnalyzer
          import matplotlib import style
          from matlplotlib import style
          df=pd.read_csv("C:/Users/Admin/Desktop/path.csv")
          df.shape            ##to check dim of data
Out [1]: (569, 30)
```

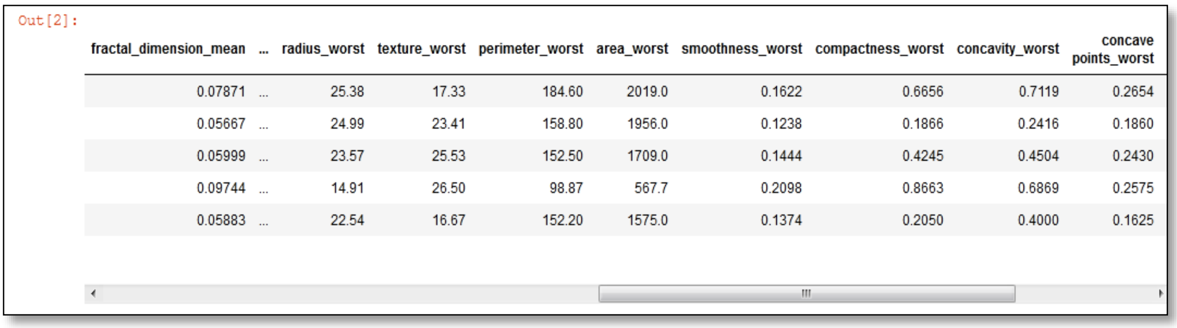
```
In [2]: df.head ()                          ##to check first 5 rows of data
```



Out[2]:

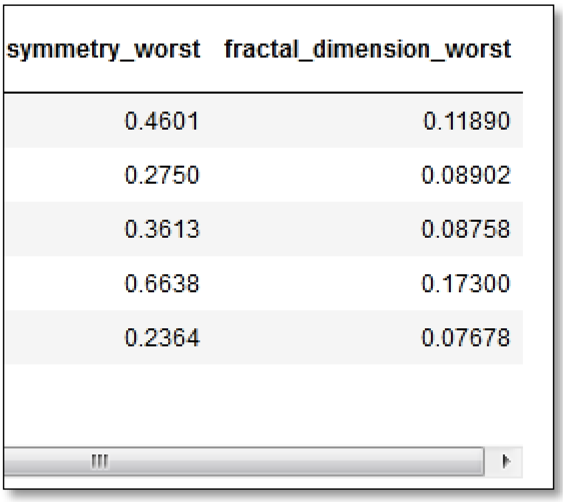
	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	symmetry_mean
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	0.2419
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	0.1812
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	0.2069
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	0.2597
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	0.1809

5 rows × 30 columns



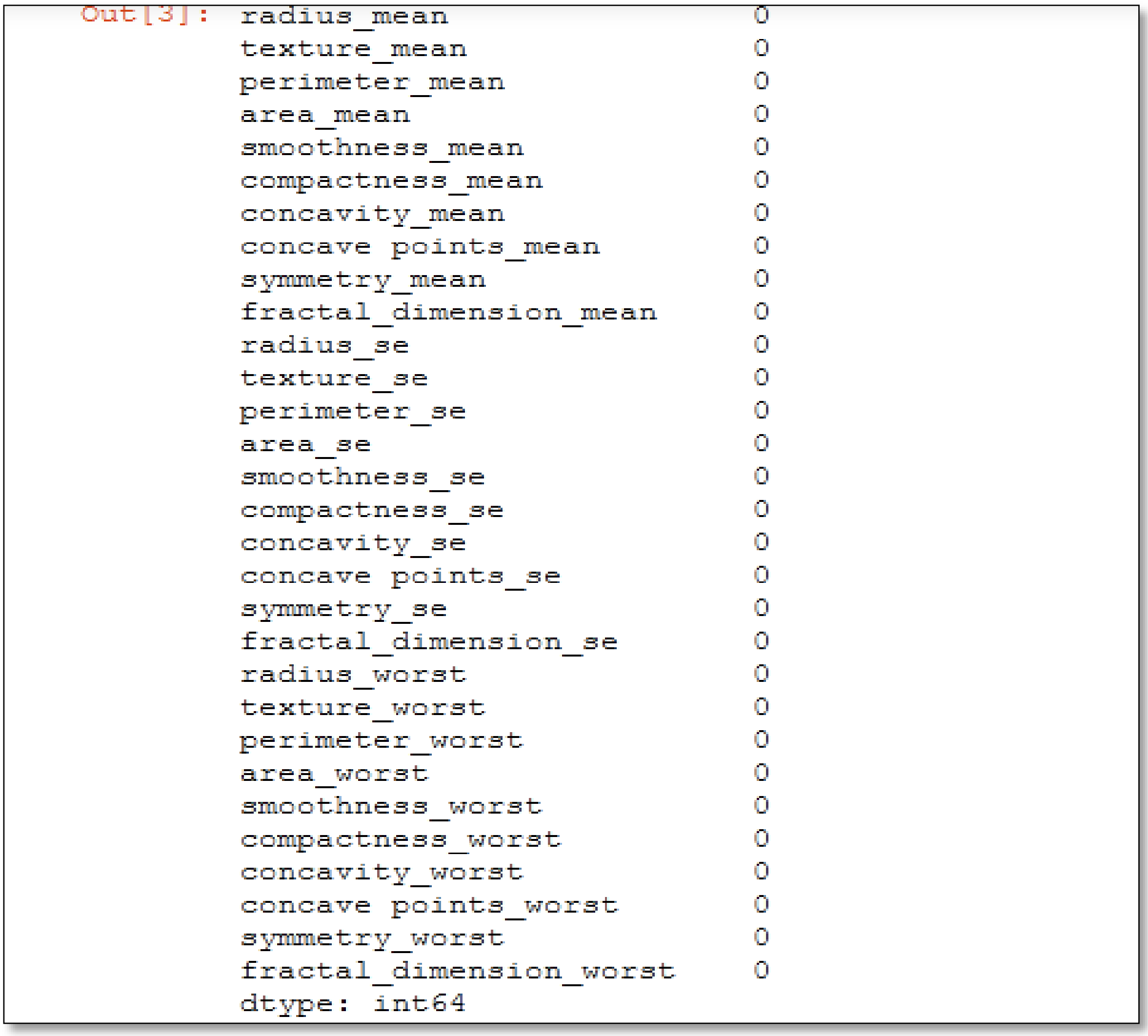
Out[2]:

fractal_dimension_mean	...	radius_worst	texture_worst	perimeter_worst	area_worst	smoothness_worst	compactness_worst	concavity_worst	concave points_worst
0.07871	...	25.38	17.33	184.60	2019.0	0.1622	0.6656	0.7119	0.2654
0.05667	...	24.99	23.41	158.80	1956.0	0.1238	0.1866	0.2416	0.1860
0.05999	...	23.57	25.53	152.50	1709.0	0.1444	0.4245	0.4504	0.2430
0.09744	...	14.91	26.50	98.87	567.7	0.2098	0.8663	0.6869	0.2575
0.05883	...	22.54	16.67	152.20	1575.0	0.1374	0.2050	0.4000	0.1625



symmetry_worst	fractal_dimension_worst
0.4601	0.11890
0.2750	0.08902
0.3613	0.08758
0.6638	0.17300
0.2364	0.07678

```
In [3]: df.isna ().sum ()          ##to check variable vise whether our data contains any null values
```

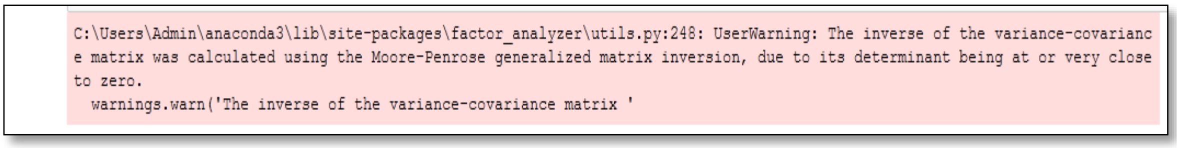


```
Out[3]: radius_mean                0
        texture_mean               0
        perimeter_mean             0
        area_mean                  0
        smoothness_mean            0
        compactness_mean           0
        concavity_mean             0
        concave points_mean        0
        symmetry_mean              0
        fractal_dimension_mean     0
        radius_se                  0
        texture_se                 0
        perimeter_se               0
        area_se                    0
        smoothness_se              0
        compactness_se             0
        concavity_se               0
        concave points_se          0
        symmetry_se                0
        fractal_dimension_se       0
        radius_worst               0
        texture_worst              0
        perimeter_worst            0
        area_worst                 0
        smoothness_worst           0
        compactness_worst          0
        concavity_worst            0
        concave points_worst       0
        symmetry_worst             0
        fractal_dimension_worst    0
        dtype: int64
```

Let's get start with statistical tests

1. Kaiser-Meyer-Olkin measure of sampling adequacy:

```
In [4]: from factor_analyzer.factor_analyzer import calcalate_kmo
        kmo_all,kmo_model=calcalate_kmo(df)
        kmo model
```



```
C:\Users\Admin\anaconda3\lib\site-packages\factor_analyzer\utils.py:248: UserWarning: The inverse of the variance-covarianc
e matrix was calculated using the Moore-Penrose generalized matrix inversion, due to its determinant being at or very close
to zero.
  warnings.warn('The inverse of the variance-covariance matrix '
```

```
Out [4] : 0.8317335254098296
```

Kaiser-Meyer-Olkin measure of sampling adequacy=0.83(greater than 0.5) which indicates data is appropriate for Factor Analysis., i.e., variables (30) and sample size (569) are enough to proceed for factor analysis. Any value less than 0.5 indicate that the correlation between pairs of variable cannot be explained by other variables and that factor analysis may not be appropriate.

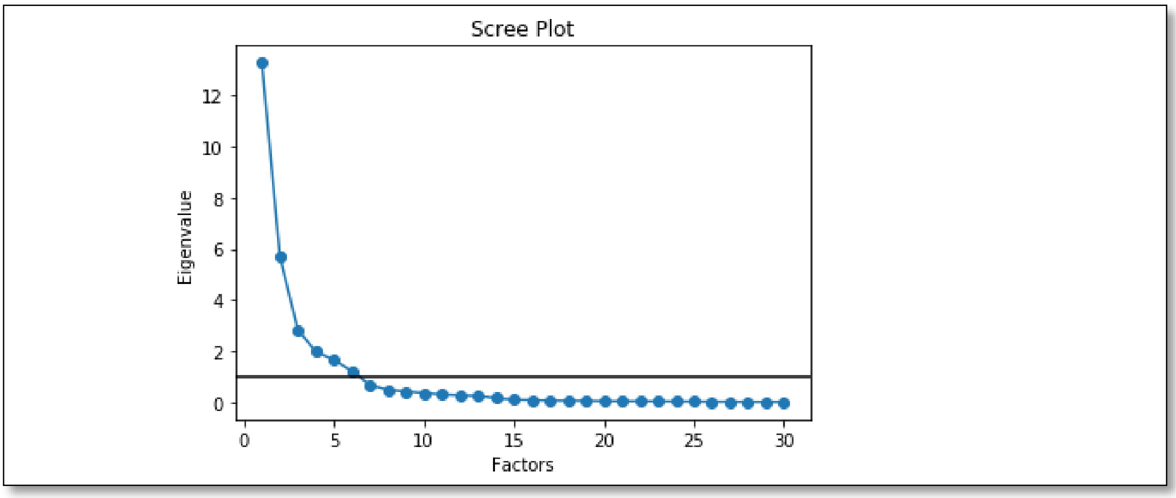
2. Barlett's test of sphericity:

Barlett's test for sphericity tests the null hypothesis that the correlation matrix is an identity matrix. Small p-value indicate that evidence against the null hypothesis (i.e. the variables really are correlated). For p-values much larger than 0.05 indicated that there is insufficient evidence that variables are not correlated, so far factor analysis may not be suitable.

```
In [5]: from factor_analyzer.factor_analyzer import
        calcalate_bartlett_sphericity
        chi_square_value,p_value=calculate_bartlett_sphericity(df)
        chi_square_value, p_value
 Out [5]: (39391.522783630295, 0.0)
```

Here, p-value is less than 0.05 so now we can proceed for factor analysis.
Let's try to find Eigen values which would decide appropriate number of factors to be chosen for our analysis and visualize them.

```
In [6] : fa=FactorAnalyzer(n_factors=30,rotation=”varimax”)
         fa.fit(df)
         ev, v=fa.get_ eigenvalues ()
         ev=pd.DataFrame(list(ev) ,columns=[ 'Eigen_values')
        ,index=[df.columns])
         # Create scree plot using matplotlib
         plt.scatter(range(1,df,shape[1]+1),ev)
         plt.plot(range(1,df,shape[1]+1),ev)
         plt.title('Scree Plot')
         plt.xlabel ('Factors’)
         plt.ylabel('Eigenvalue')
         plt.axhline(y=1, c='k')
Out [6] : <matplotlib.lines.Line2D at 0xbf134d0>
```

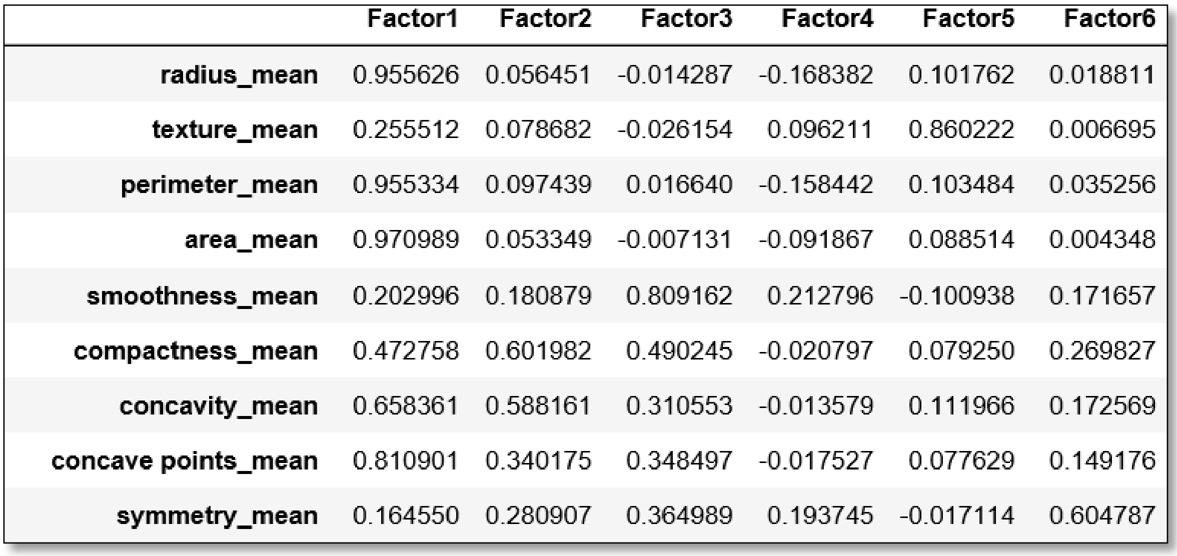


We have to select all those factors which have eigen values greater than 1. Hence, number of factors = 6.

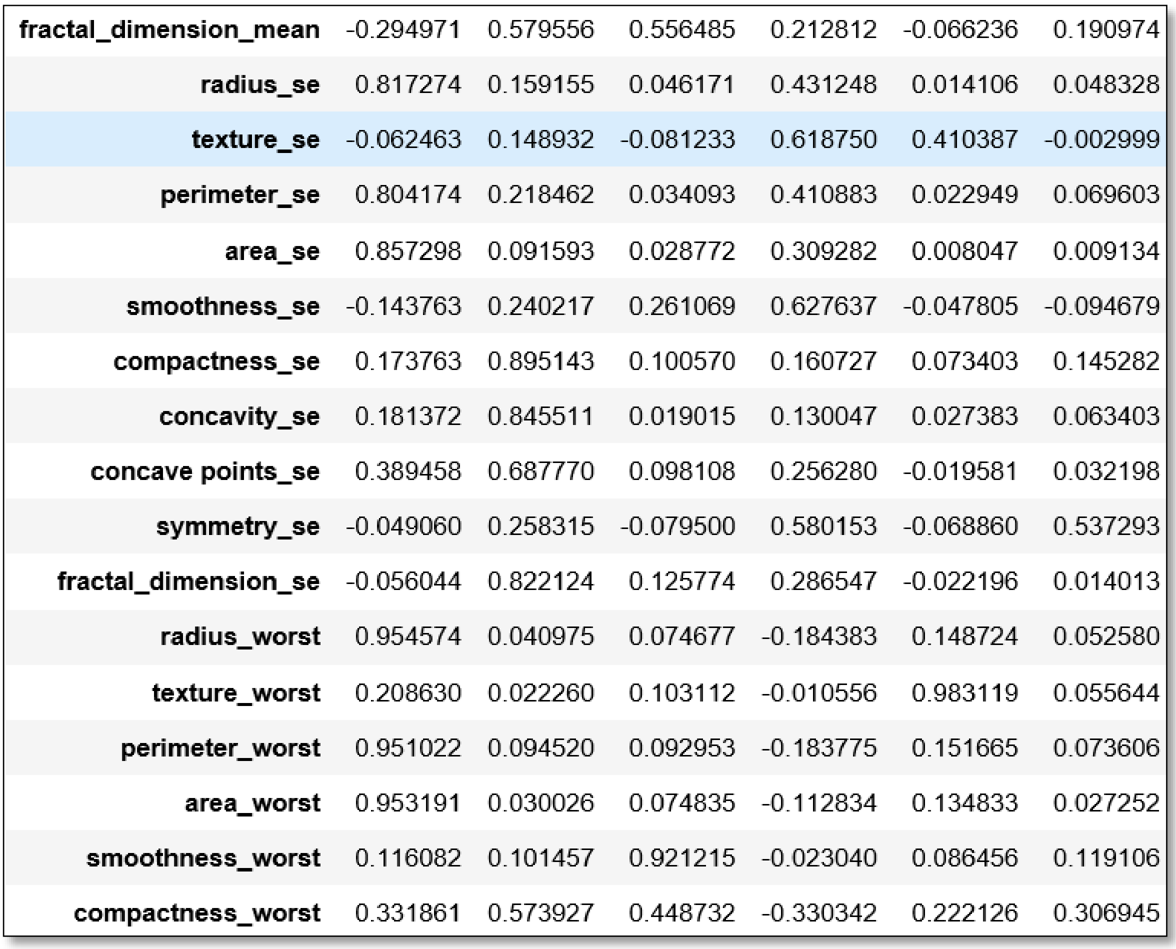
Now we again have to create factor analysis model but selecting n =6 and also estimating loadings.

ISBN 978-93-80788-93-7

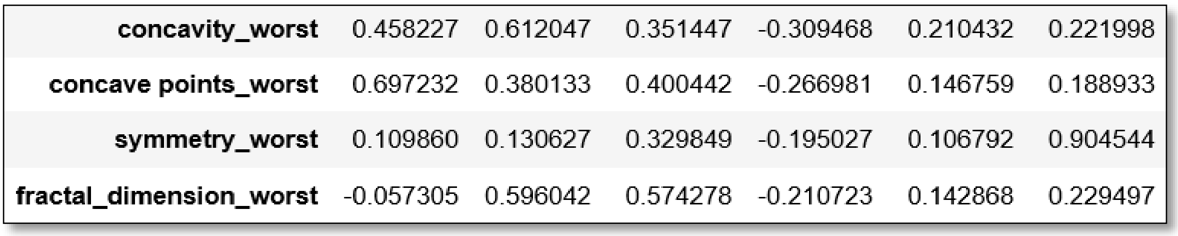
```
In [7]: #Create factor analysis object and perform factor analysis
        fa=FactorAnalyzer(n_factors=6,rotation=”varimax”)
                                #please check definition of varimax at below
        fa.fit(df)
        fa.loadings=fa.loadings_
        fa.loadings=pd.DataFrame(list(fa.loadings),columns=[‘Factor1',
        'Factor2', 'Factor3', 'Factor4','Factor5','Factor6'],index=
        df.columns)
        fa.loadings
```



	Factor1	Factor2	Factor3	Factor4	Factor5	Factor6
radius_mean	0.955626	0.056451	-0.014287	-0.168382	0.101762	0.018811
texture_mean	0.255512	0.078682	-0.026154	0.096211	0.860222	0.006695
perimeter_mean	0.955334	0.097439	0.016640	-0.158442	0.103484	0.035256
area_mean	0.970989	0.053349	-0.007131	-0.091867	0.088514	0.004348
smoothness_mean	0.202996	0.180879	0.809162	0.212796	-0.100938	0.171657
compactness_mean	0.472758	0.601982	0.490245	-0.020797	0.079250	0.269827
concavity_mean	0.658361	0.588161	0.310553	-0.013579	0.111966	0.172569
concave points_mean	0.810901	0.340175	0.348497	-0.017527	0.077629	0.149176
symmetry_mean	0.164550	0.280907	0.364989	0.193745	-0.017114	0.604787



fractal_dimension_mean	-0.294971	0.579556	0.556485	0.212812	-0.066236	0.190974
radius_se	0.817274	0.159155	0.046171	0.431248	0.014106	0.048328
texture_se	-0.062463	0.148932	-0.081233	0.618750	0.410387	-0.002999
perimeter_se	0.804174	0.218462	0.034093	0.410883	0.022949	0.069603
area_se	0.857298	0.091593	0.028772	0.309282	0.008047	0.009134
smoothness_se	-0.143763	0.240217	0.261069	0.627637	-0.047805	-0.094679
compactness_se	0.173763	0.895143	0.100570	0.160727	0.073403	0.145282
concavity_se	0.181372	0.845511	0.019015	0.130047	0.027383	0.063403
concave points_se	0.389458	0.687770	0.098108	0.256280	-0.019581	0.032198
symmetry_se	-0.049060	0.258315	-0.079500	0.580153	-0.068860	0.537293
fractal_dimension_se	-0.056044	0.822124	0.125774	0.286547	-0.022196	0.014013
radius_worst	0.954574	0.040975	0.074677	-0.184383	0.148724	0.052580
texture_worst	0.208630	0.022260	0.103112	-0.010556	0.983119	0.055644
perimeter_worst	0.951022	0.094520	0.092953	-0.183775	0.151665	0.073606
area_worst	0.953191	0.030026	0.074835	-0.112834	0.134833	0.027252
smoothness_worst	0.116082	0.101457	0.921215	-0.023040	0.086456	0.119106
compactness_worst	0.331861	0.573927	0.448732	-0.330342	0.222126	0.306945



concavity_worst	0.458227	0.612047	0.351447	-0.309468	0.210432	0.221998
concave points_worst	0.697232	0.380133	0.400442	-0.266981	0.146759	0.188933
symmetry_worst	0.109860	0.130627	0.329849	-0.195027	0.106792	0.904544
fractal_dimension_worst	-0.057305	0.596042	0.574278	-0.210723	0.142868	0.229497

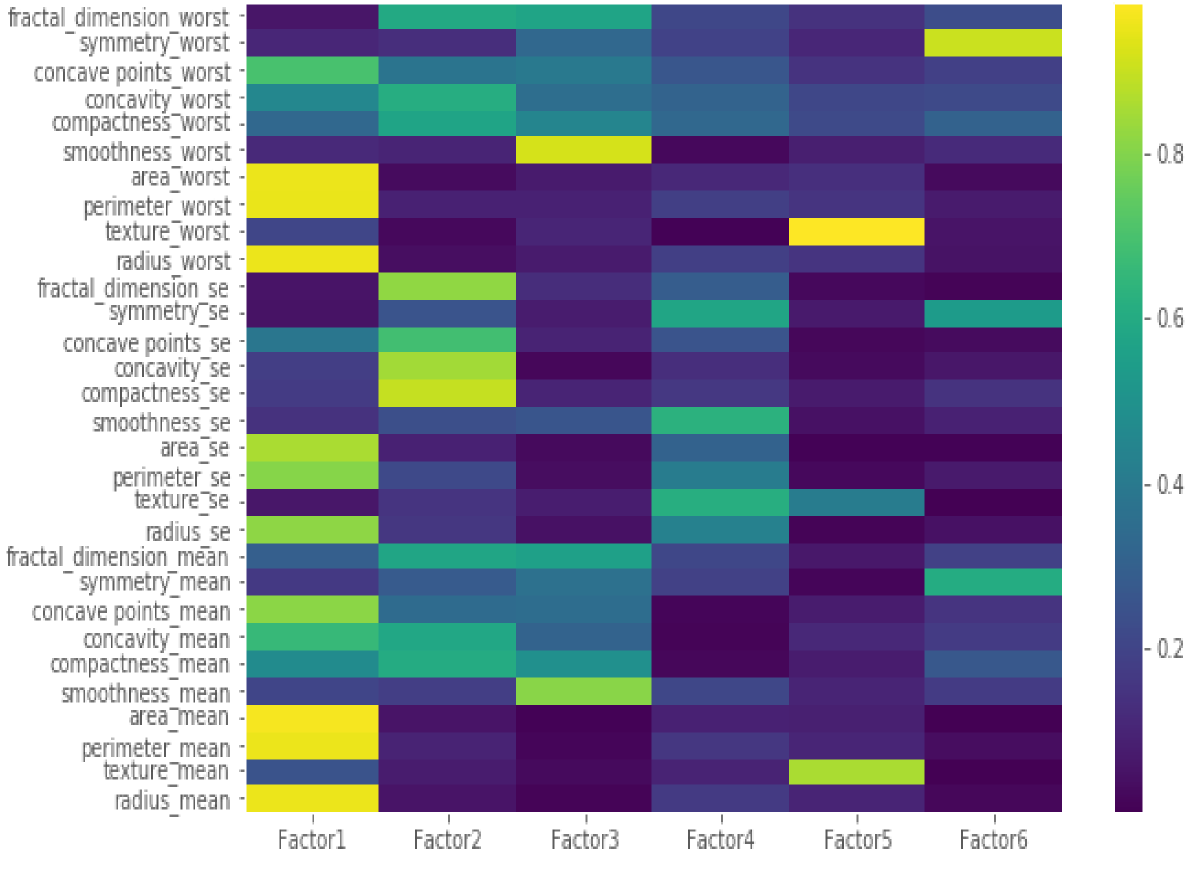
Factor 1 has high factor loading for ‘area_mean’
Factor 2 has high factor loading for ‘compactness_se’
Factor 3 has high factor loading for ‘smoothness_worst’
Factor 4 has high factor loading for ‘smoothness_se’
Factor 5 has high factor for ‘texture worst’
Factor 6 has high factor for ‘symmetry worst’

Let’s try to visualize loadings with variables

```
In [8]: import numpy as np
        Z=np.abs(fa.loadings)
        fig, ax= plt.subplot()
        c = ax.pcolor(Z)
        fig.colorbar(c,ax=ax)
        ax.set_ytricks(np. arange (fa. loading.shape [0] ) +0. 5,
        minor=False)
```

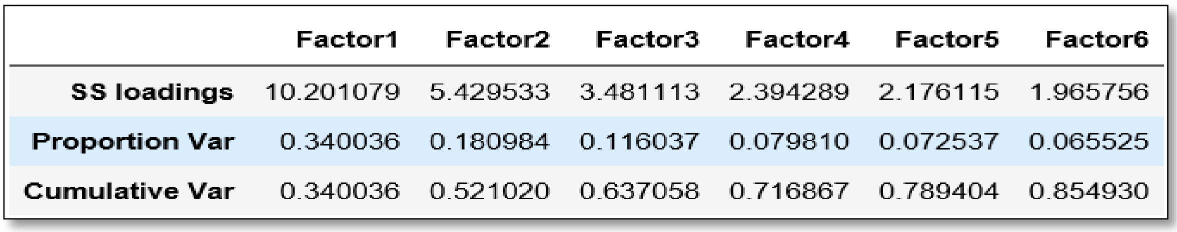
ISBN 978-93-80788-93-7

```
        ax.set_xtricks(np. arange (fa. loading.shape [1] ) +0.5,
        minor=False)
        ax.set_ytricklabels(fa.loadings.index.values)
        ax.set_xtricklabels(fa.loadings.columns.values)
        plt.show()
```



Let us find out how much these factors are explaining variations in data.

```
In [9]: output=fa.get_factor_variance()
      Output=pd.DataFrame(list(output),columns=['Factor1','Factor2',
      'Factor3','Factor4','Factor5','Factor6'),      index=['SS      loadings',
      'ProportionVar', 'Cumulative Var])
       Output
```



	Factor1	Factor2	Factor3	Factor4	Factor5	Factor6
SS loadings	10.201079	5.429533	3.481113	2.394289	2.176115	1.965756
Proportion Var	0.340036	0.180984	0.116037	0.079810	0.072537	0.065525
Cumulative Var	0.340036	0.521020	0.637058	0.716867	0.789404	0.854930

These 6 factors explain more than 85% of the data.

ISBN 978-93-80788-93-7

Other Terminologies:

1. **Varimax rotation:** Varimax rotation (also called Kaiser – Varimax rotation) maximizes the sum of the variance of the squared loadings, where 'loadings' means correlations between variables and factors. This usually results in high factors loadings for a smaller number of variables and low factor loadings for the rest.
2. **Eigen values:** Eigen values shows variance explained by that particular factor out of total variance.
3. **Factor loading:** Factor loading is correlation coefficient for the variable and factor. Factor loading shows the variance explained by the variable on that particular factor.

14.4 References

1. https://iescoders.com/exploratory-factor-analysis
2. https://www.py4e.com/lessons
3. https://scikitlearn.org/stable/modules/generated/sklearn.decomposition.FactorAnalysis.html
4. https://www.statisticssolutions.com/factor-analysis-sem-factor-analysis
5. Dr. Santosh P. Gite (2018), Factoranalysis, Dr. Asha Jindal (ed.) Analysing and visualizing data with R software, ShailajaPrakashan and Kishinchand Chellaram College, pp181-188.
6. Dr. Asha Jindal (2013), Factorinfluencing infant mortality in Uttar Pradesh, International Journal of Multidisciplinary Research, Vol I, Issue 10(I), pp 65-70.

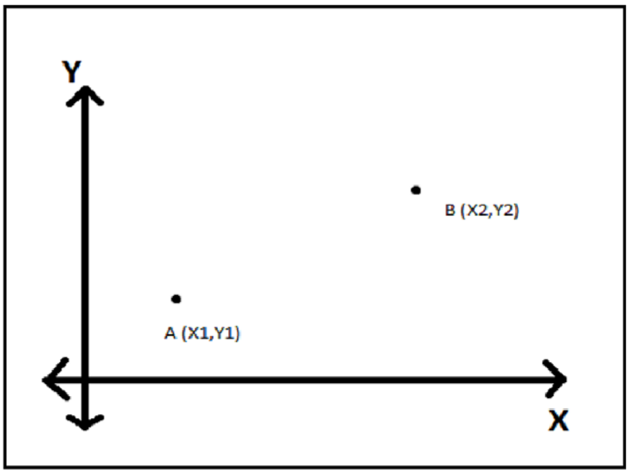
Chapter 15
Cluster Analysis

Mr. Abhay Deshpande*, Freelance Researcher, Alumni 2019, Statistics,*
K. C. College

15.1 Introduction

Cluster analysis or clustering is grouping of cases based on similarities and other numerical attributes of variables in the sample or population under study. Cluster analysis is useful for unsupervised machine learning. Unsupervised learning is used to solve problems in machine learning which have unknown solutions. In this case study, problem is solved using Agglomerative Hierarchical clustering. Idea behind this method is to form different clusters using bottom up approach i.e. considering every observation as a cluster then combining them together until a single cluster is formed starting from the bottom.

For this, a dendrogram is used using the dissimilarities between the observations in the data. A dendrogram is a diagram similar to tree diagram. Using the dendrogram, different clusters are then formed by calculating the distances between the observations.



Simplest way to calculate the distance between any two points in two dimensions is

$$d = \sqrt{(X2 - X1)^2 + (Y2 - Y1)^2}$$

where, d = distance. This method of calculating distance is known as Euclidean method. The general procedure of Agglomerative Hierarchical clustering is as follows:

1. Step 1: Consider each observation in the data as a single cluster so that in general our data has let's says 'k' clusters.
2. Step 2: Take two closest observations and make one cluster of them. So now we have 'k-1' clusters.
3. Step 3: Continue with step 2 procedure until we have a single cluster.

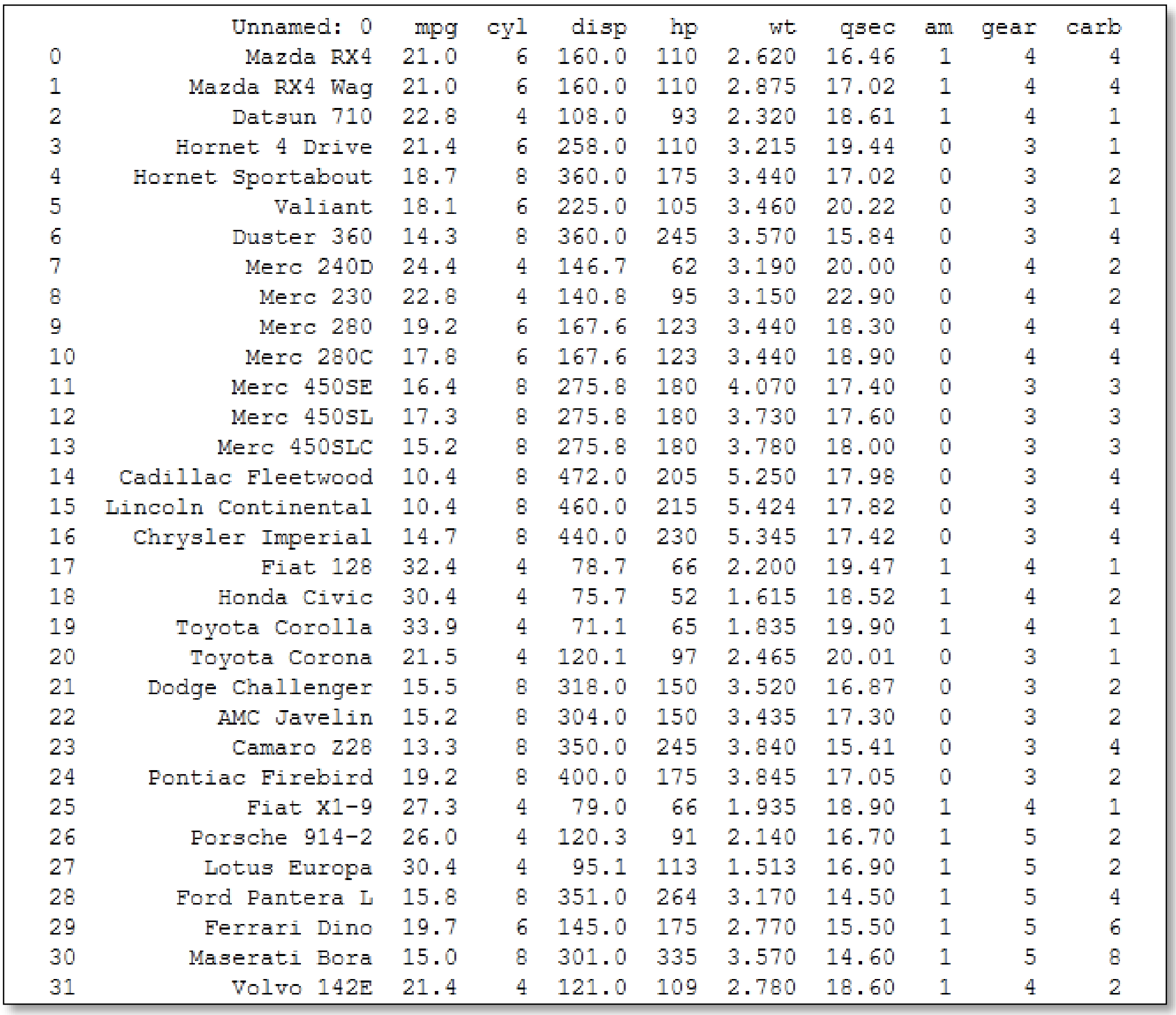
Next part is fitting of clusters to the data provided and visualisation, which we will discuss in the procedure.

15.2 Procedure

```
In [1]: # Importing the libraries
      import numpy as np
      import matplotlib.pyplot as plt
      import pandas as pd
```

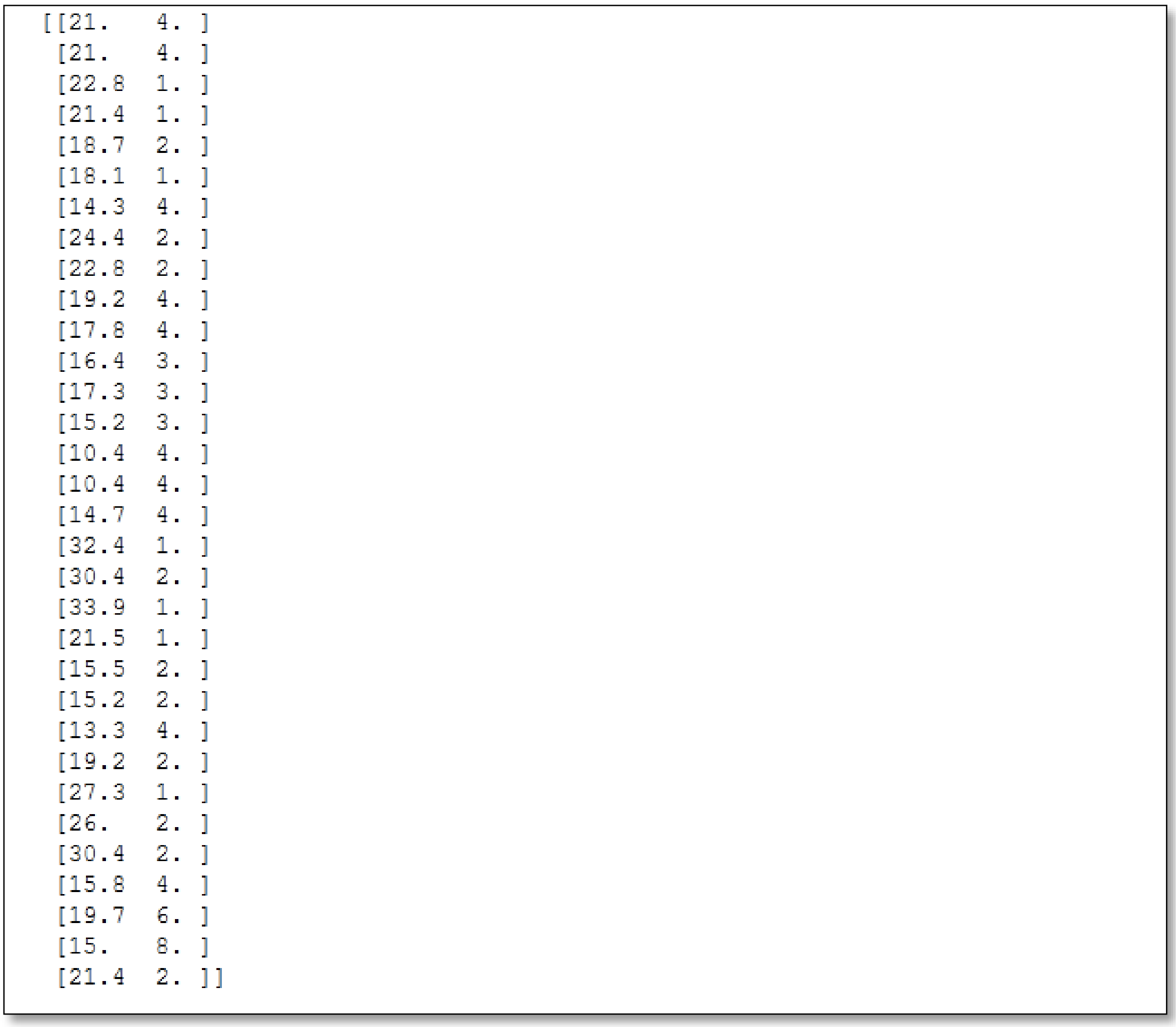
Before importing the dataset it is advised to create a working directory and set it up. It can be done using the os.chdir() command.

```
In [2]: # Importing the dataset with pandas
      dataset = pd.read_csv("C:/User/Admin/Desktop/dataset.csv")
      print(dataset)
      #Indexing the variables to be used from the dataset

      X = dataset.iloc[:,[1,9]].values
      print(X)
```



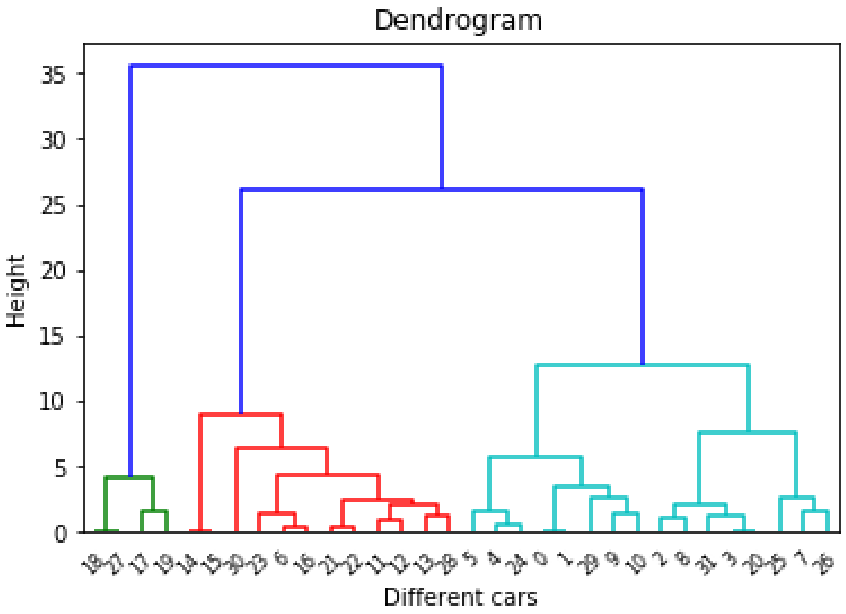
```
            Unnamed: 0   mpg  cyl   disp   hp     wt   qsec  am  gear  carb
0            Mazda RX4  21.0    6  160.0  110  2.620  16.46   1     4     4
1        Mazda RX4 Wag  21.0    6  160.0  110  2.875  17.02   1     4     4
2           Datsun 710  22.8    4  108.0   93  2.320  18.61   1     4     1
3       Hornet 4 Drive  21.4    6  258.0  110  3.215  19.44   0     3     1
4    Hornet Sportabout  18.7    8  360.0  175  3.440  17.02   0     3     2
5              Valiant  18.1    6  225.0  105  3.460  20.22   0     3     1
6           Duster 360  14.3    8  360.0  245  3.570  15.84   0     3     4
7            Merc 240D  24.4    4  146.7   62  3.190  20.00   0     4     2
8             Merc 230  22.8    4  140.8   95  3.150  22.90   0     4     2
9             Merc 280  19.2    6  167.6  123  3.440  18.30   0     4     4
10           Merc 280C  17.8    6  167.6  123  3.440  18.90   0     4     4
11          Merc 450SE  16.4    8  275.8  180  4.070  17.40   0     3     3
12          Merc 450SL  17.3    8  275.8  180  3.730  17.60   0     3     3
13         Merc 450SLC  15.2    8  275.8  180  3.780  18.00   0     3     3
14  Cadillac Fleetwood  10.4    8  472.0  205  5.250  17.98   0     3     4
15 Lincoln Continental  10.4    8  460.0  215  5.424  17.82   0     3     4
16   Chrysler Imperial  14.7    8  440.0  230  5.345  17.42   0     3     4
17            Fiat 128  32.4    4   78.7   66  2.200  19.47   1     4     1
18         Honda Civic  30.4    4   75.7   52  1.615  18.52   1     4     2
19      Toyota Corolla  33.9    4   71.1   65  1.835  19.90   1     4     1
20       Toyota Corona  21.5    4  120.1   97  2.465  20.01   0     3     1
21    Dodge Challenger  15.5    8  318.0  150  3.520  16.87   0     3     2
22         AMC Javelin  15.2    8  304.0  150  3.435  17.30   0     3     2
23          Camaro Z28  13.3    8  350.0  245  3.840  15.41   0     3     4
24    Pontiac Firebird  19.2    8  400.0  175  3.845  17.05   0     3     2
25           Fiat X1-9  27.3    4   79.0   66  1.935  18.90   1     4     1
26       Porsche 914-2  26.0    4  120.3   91  2.140  16.70   1     5     2
27        Lotus Europa  30.4    4   95.1  113  1.513  16.90   1     5     2
28      Ford Pantera L  15.8    8  351.0  264  3.170  14.50   1     5     4
29        Ferrari Dino  19.7    6  145.0  175  2.770  15.50   1     5     6
30       Maserati Bora  15.0    8  301.0  335  3.570  14.60   1     5     8
31          Volvo 142E  21.4    4  121.0  109  2.780  18.60   1     4     2
```

ISBN 978-93-80788-93-7



```
[[21.   4. ]
 [21.   4. ]
 [22.8  1. ]
 [21.4  1. ]
 [18.7  2. ]
 [18.1  1. ]
 [14.3  4. ]
 [24.4  2. ]
 [22.8  2. ]
 [19.2  4. ]
 [17.8  4. ]
 [16.4  3. ]
 [17.3  3. ]
 [15.2  3. ]
 [10.4  4. ]
 [10.4  4. ]
 [14.7  4. ]
 [32.4  1. ]
 [30.4  2. ]
 [33.9  1. ]
 [21.5  1. ]
 [15.5  2. ]
 [15.2  2. ]
 [13.3  4. ]
 [19.2  2. ]
 [27.3  1. ]
 [26.   2. ]
 [30.4  2. ]
 [15.8  4. ]
 [19.7  6. ]
 [15.   8. ]
 [21.4  2. ]]
```

```
In [3]: # using dendogram to find the optimal number of cluster
      import scipy.cluster.hierarchy as sch
      dendrogram = sch.dendrogram(sch.linkage(X, method = ‘ward’) )
      plt.title(‘Dendrogram’)
      plt.xlabel(‘Different cars’)
      plt.ylabel(‘Height’)
      plt.show()
```



Understanding the dendrogram and calculating the number of clusters formed is most important. For that, just see the longest line which is not intersected by a horizontal line i.e. in this case blue line just above green horizontal line. Similar can be seen in case of blue and red lines. After that, see in how many parts those intersected horizontal lines are divided. Here we get a total of five clusters. Next diagram shows how it can be observed.

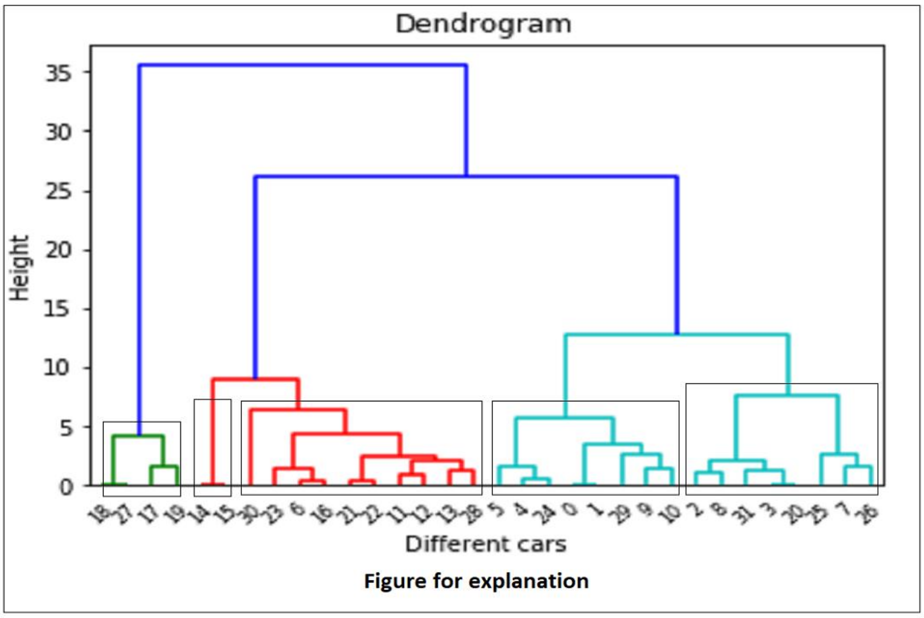


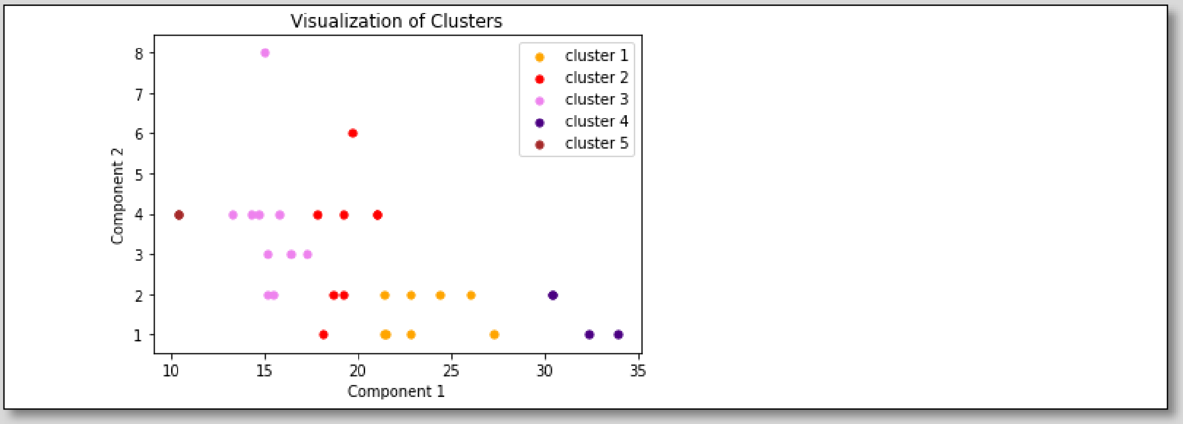
Figure for explanation

```
In [4]: #Fitting hierarchical clustering to data
        from sklearn.cluster import AgglomerativeClustering
        ahc=AgglomerativeClustering(n_cluster=5,   affinity   =   'euclidean',
        linkage='ward')
        pred= ahc.fit_predict(X)
        print(ahc)
```

ISBN 978-93-80788-93-7

```
    print(pred)
    AgglomerativeClustering(affinity='euclidean',
    compute_full_tree='auto',
                      connectivity=None,distance_threshold= none,
                      linkage='ward',memory=none,n_cluster=5,
                      pooling_func='deprecated')
    [1 1 0 0 1 1 2 0 0 1 1 2 2 2 4 4 2 3 3 3 0 2 2 2 1 0 0 3 2 1 2 0]
```

```
In [5]: #Visualizing Clusters

  plt.scatter(X[pred==0,0],X[pred==0,1],s=27, c='orange',label='cluster1')
  plt.scatter(X[pred==1,0],X[pred==1,1],s=27, c='red',label='cluster2')
  plt.scatter(X[pred==2,0],X[pred==2,1],s=27, c='violet',label='cluster3')
  plt.scatter(X[pred==3,0],X[pred==3,1],s=27, c='indigo',label='cluster4')
  plt.scatter(X[pred==4,0],X[pred==4,1],s=27, c='brown',label='cluster5')
  plt.title('Visualization of Clusters')
  plt.xlabel('component 1')
  plt.ylabel('component 2')
  plt.legend()
  plt.show()
```



Interesting part to note is we have plotted multidimensional data in just two dimensions for simplicity. Further one may note that according to the data, types of cars may be categorized in two types as performance heavy/ super cars and pocket friendly. Cars who deliver high end performance are generally having more number of cylinders, greater fuel displacement capacity in engines to generate more horsepower etc. hence buyers pay a premium price. On the other hand, budget cars provide more miles per gallon having less fuel displacement capacity, number of cylinders in the engine etc.

15.3 Acknowledgement

I would like to thank my parents and Dr. Asha Jindal, Associate Professor and Head, Department of Statistics, K. C. College for her encouragement and support.

15.4 References

1. https://docs.anaconda.com/_/downloads/en/latest/pdf/
2. https://cran.r-project.org/doc/manuals/r-release/fullrefman.pdf
3. https://docs.scipy.org/doc/scipy/reference/generated/scipy.cluster.hierarchy.linkage.html
4. https://scikit-learn.org/stable/modules/generated/sklearn.cluster.AgglomerativeClustering.html

Chapter 16
Non-Parametric Test (Choice and Application using Python)

***Dr. S. B. Muley**, Assistant Professor, Department of Statistics,*
K. C. College

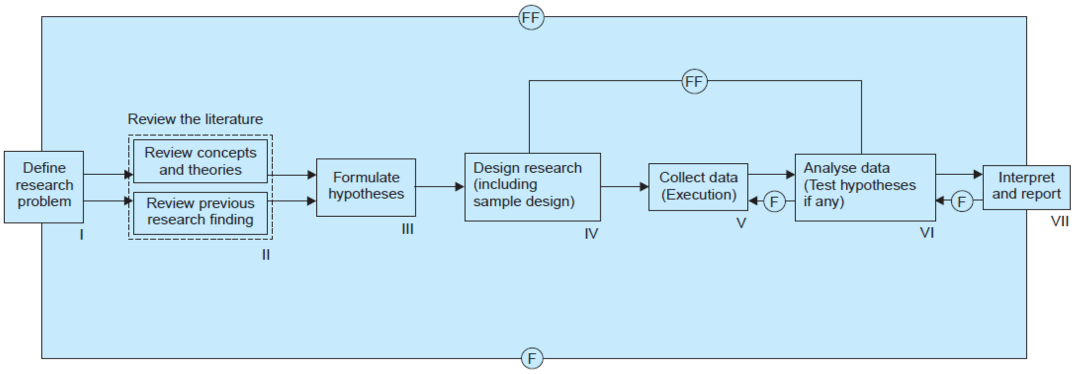
16.1 Introduction

Processed information has become a key to many decision making systems. Data has become an inevitable part of all the business models, research and education. Not only educational Institutions of international repute but business firms like IBM, Microsoft, SAS and many more has come out with courses and training methods on analysis of data. Data is generated by many sources, some of the data is result of planned experiments and some are results of real time processes. Basic types of such data can be of nominal, ordinal and scale type. All these information/data can further be processed for the purpose of

1. To derive some patterns/trends from the populations.
2. To test some assumptions about populations.
3. To develop prediction models. etc.

Hypothesis testing is standard procedure of testing assumptions about population distribution in basic statistics. Due to this reason, statistical inference has become the field of interest of many researchers and data scientists. The inferential type research is very common and frequent type of research. The inferential type research can be a part of many other research types like Experimental, Exploratory, Quantitative, Qualitative, Analytical, Conceptual etc. The steps (Fig.1) in process of research, very precisely, given by Kothari C. R.[1].

Fig.1: Research process flowchart from "**C. R. Kothari** (2009) "**Research Methodology**: **Methods&Techniques**" (Second Revised Edition), New Age International Publishers, New Delhi.


FF
Review the literature
Define research problem
I
Review concepts and theories
Review previous research finding
II
Formulate hypotheses
III
FF
Design research (including sample design)
IV
Collect data (Execution)
V
F
Analyse data (Test hypotheses if any)
VI
F
Interpret and report
VII
F

Where Ⓕ = feed back (Helps in controlling the sub-system to which it is transmitted)
ⒻⒻ = feed forward (Serves the vital function of providing criteria for evaluation)

Analysis of Data is one of the important steps in the research process. In statistical theory of inference this procedure is called as testing of hypothesis if the analysis of data is imperatively for the purpose of testing of hypothesis. This testing of hypothesis process comprises of following steps:

1. Defining objective and problem statement precisely.
2. Formulate null and alternate hypothesis.
3. Decide on level of significance.
4. Choose an appropriate statistical test.
5. Using an appropriate decision rule, decide whether null hypothesis to be accepted or rejected.
6. Interpret and conclude the results with respect to problem statement.

Data analysis requires choosing an appropriate statistical test for analysis from parametric and non-parametric domain. This choice of the test is depending on following criteria:

- Type of Hypothesis (Association or Difference type).
- Number of dependent and independent variables.
- Type of variables (Nominal, Ordinal and Interval or Ratio scale).
- Normal Distribution (test of Normality).

The basic conditions for choosing test from parametric domain are

1. The data should be measured on Interval/Ratio scale.
2. Data should satisfy the condition on normality.

If all of the above conditions are satisfied then we use parametric tests. On the contrary a statistical method is called non-parametric if it makes no assumption on the population distribution or sample size. This approach is less powerful yet more frequent, more flexible, more robust, and applicable to quantitative and non-quantitative data.

On choosing appropriate statistical test for analysis one can use Excel, SPSS, MINITAB, SAS, R-programming and Python programming etc. The packages like SPSS, MINITAB are menu driven packages. SAS, R and Python are programing languages. The R and Python are the important free and open source software used in data analytics and business analytics. In

this chapter we have discussed use of python programming language to apply different non-parametric tests on different types of data sets and different types of cases.
Statsmodels is a Python module that provides classes and functions for conducting statistical tests. The statsmodel functions on nonparametric statistics are used to evaluate nonparametric test. Another platform used for evaluation is SciPy. This is a collection of mathematical algorithms and convenience functions built on the NumPy extension of Python. SciPy is interactive Python platform for data-processing.

16.2 One Sample Tests

The purpose of one sample test is to test the significance of difference between sample and population median.

Sign test:
Basic conditions for applying test: One-sample data and median.
Syntax used:
statsmodels.stats.descriptivestats.sign_test(*samp,mu0=test median*)

Parameters
samp: One dimensional array of data for which you want to perform the signs test.
mu0*: The population median against which data is to be tested.*

Returns

1. The signs test returns M = (N(+) - N(-))/2, where N(+) is the number of values above *mu0*, N(-) is the number of values below. Values equal to *mu0* are discarded.
2. The p-value for M is calculated using the binomial distribution and can be interpreted the same as for a t-test. The test-statistic is distributed Bin(min(N(+), N(-)), n_trials, where n_trials equals N(+) + N(-).

Example1:

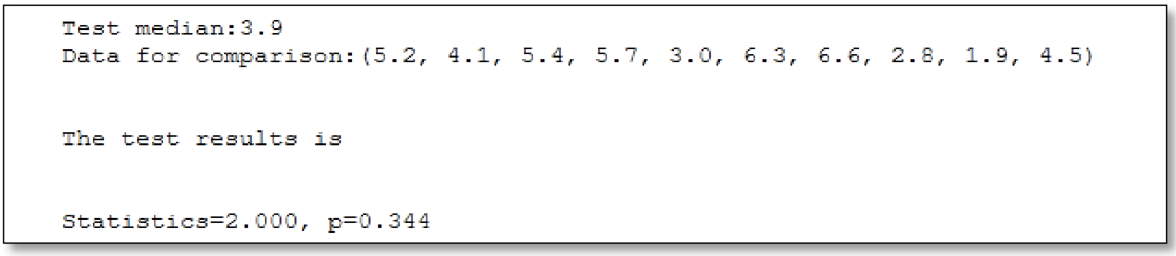
It is known from the past experience that the median length of fish in a particular polluted lake was 3.9 inches. During the past two years the lake was cleaned up and the conjecture is made that now median length is greater than 3.9 inches. A random sample of 10 sunfish selected from this lake showed lengths as 5.2, 4.1, 5.4, 5.7, 3.0, 6.3, 6.6, 2.8, 1.9, 4.5 inches. Will you reject the null hypothesis at 10 % level of significance (l.o.s.) on the basis of Sign Test?

```
In [1] *Sign test tests the hypothesis type of two sided.
      import scipy as sp
      import numpy as np
      import pandas as pd
      import statsmodels.stats.descriptivestats as smsd
```

```
In [2]: # Enter for comparison by using  any of the known of  method and
         entry and choose variable to be tested.
```

```
data-(5.2, 4.1, 5.4, 5.7, 3.0, 8.3, 6.6, 2.2, 4.5 )
```

```
In[3]: # Give test median value.
      med=float(input("Test median:"))
      print ('Data for comparison:'+str(data))
      print('\n')
      print ('The test results is')
      print('\n')
      stat,p=smsd.sign_test(data, mu0=med)
      print('Statistics=%.3f, p=%.3f' %(stat, p))
```



```
Test median:3.9
Data for comparison:(5.2, 4.1, 5.4, 5.7, 3.0, 6.3, 6.6, 2.8, 1.9, 4.5)


The test results is


Statistics=2.000, p=0.344
```

```
In [4] : # interpret
      alpha = 0.05
      if p > alpha:
            print('Fail to reject HO')
      else:
            print('Reject HO')
```

```
Fail to reject HO
```

Interpretation: Since p-value=0.3437> 0.025 indicates one should not reject null hypothesis.

Note: This test can also be used to test the significance of difference between two paired observation by using difference data (d=x-y) as one dimensional data as samp and mu0=0.

16.3 Two Sample Tests

Independent sample comparison:

The two-sample Mann–Whitney U test compares values for two groups. A significant result suggests that the values for the two groups are different. It is equivalent to a two-sample Wilcoxon rank-sum test.

Basic conditions for applying test:

- **Two-sample data**: One-way data with two groups only.
- **Type Dependent variable**: is of the type Ordinal, Interval, or Ratio.
- **Independent variable** is a factor with two levels. That is, two groups.

ISBN 978-93-80788-93-7

- Observations between groups are independent. That is, not paired or repeated measures data.
- In order to be a test of medians, the distributions of values for each group need to be of similar shape and spread; outliers affect the spread. Otherwise the test is a test of distributions.

Syntax:

scipy.stats.mannwhitneyu (x, y, use_continuity=True, alternative=None)

or

stats.mannwhitneyu (x, y, use_continuity=True, alternative=None)

It computes the Mann-Whitney rank test on samples x and y.

Parameters

1. **x, y** like Array of samples, should be one-dimensional.
2. **use_continuity** optional and takes values True or False. Whether a continuity correction (1/2) should be taken into account. Default is True.
3. **alternative** {None, 'two-sided', 'less', 'greater'}, optional. Defines the type of alternative hypothesis.

Returns

1. **U-value:** The Mann-Whitney U statistic, equal to min(U for x, U for y) if *alternative* is equal to None (deprecated; exists for backward compatibility), and U for y otherwise.
2. **p-value:** p-value assuming an asymptotic normal distribution. One-sided or two-sided, depending on the choice of *alternative*.

Example 2:

Consider a Phase II clinical trial designed to investigate the effectiveness of a new drug to reduce symptoms of asthma in children. A total of n=10 participants are randomized to receive either the new drug or a placebo. Participants are asked to record the number of episodes of shortness of breath over a 1 week period following receipt of the assigned treatment. The data is given below.

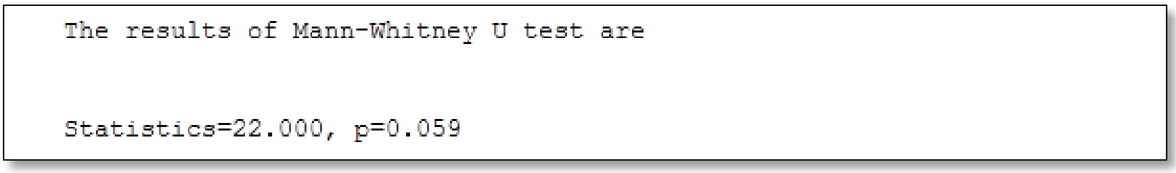
X	7	5	6	4	12
Y	3	6	4	2	1

Is there a difference in the number of episodes of shortness of breath over a 1 week period in participants receiving the new drug as compared to those receiving the placebo? By inspection, it appears that participants receiving the placebo have more episodes of shortness of breath, but is this statistically significant? Use Mann-Whitney-Wilcoxon test at 5% l.o.s. to test H_0 :M_x = M_y against H_1: M_x<M_y (use normal approximation.)

```
In[1]: #comparison of two variables using Mann–Whitney U test
      import scipy.stats as st
      from scipy import stats
      import scipy.stats as mannwhitneyu
      from scipy import stats
```

```
In[2]: #enter data of two variables as x and y
      x=[7,5,6,4,12]
      y=[3,6,4,2,1]
```

```
In  [3]:  stat,  p=stats.mannwhitneyu(x,y,  use_continuity  =  True,  alternative
      = 'two–sided')
      print (" The results of mann-whitney U test are ")
      print ("\n")
      print ('Statistics=%.3f, p=%.3f' %(stat, p))
```



```
The results of Mann-Whitney U test are


Statistics=22.000, p=0.059
```

```
In[6]:  #interpret
      alpha= 0.05
      if p > alpha:
            print ('same distribution (Fail to reject HO)')
      else:
            print('Different distribution (Reject HO)')
      Same distribution (Fail to reject HO)
```

Interpretation: p-value greater than that of 0.05 at 5% level of significance indicates that on should not reject null hypothesis.

Paired sample comparison:

Conditions of paired sample data:

- Two-sample paired data: That is, one-way data with two groups only, where the observations are paired between groups.
- Dependent variable may be of ordinal, interval, or ratio type.
- Independent variable is a factor with two levels of pair like Before-After.
- The distribution of differences in paired samples is symmetric.

Wilcoxon signed-rank test in python:

Syntax: scipy.stats.wilcoxon(x, y=None, zero_method='wilcox', correction=False, alternative='two-sided')
The Wilcoxon signed-rank test is the counter test used against paired sample t-test. This test tests the null hypothesis that two related paired samples come from the same distribution. That is, tests whether the distribution of the differences x - y is symmetric about zero.

Parameters

1. **x**: One dimensional array such that it is either the first set of data when x and y is given or one can use differences between two sets of measurements (d=x-y, as x) [in this case in place of x values difference is entered and in place of y, none is entered.
2. **y: (o**ptional), One dimensional array such that either the second set of data (in case if*x*, the first set of data, is specified), or not specified by mentioning none (in case if *x* is the set data of differences between two sets (d=x-y, as x) of data.)
3. **zero_method: (o**ptional){'pratt', 'wilcox', 'zsplit'}

The following options are available (default is 'wilcox'):

- 'pratt': Includes zero-differences in the ranking process, but drops the ranks of the zeros.
- 'wilcox': Discards all zero-differences, the default.
- 'zsplit': Includes zero-differences in the ranking process and split the zero rank between positive and negative ones.

4. **correction: (Either True or False)**, optional. If True, apply continuity correction by adjusting the Wilcoxon rank statistic by 0.5 towards the mean value when computing the z-statistic. Default is False.
5. **alternative: (o**ptional) {"two-sided", "greater", "less" if the alternative hypothesis type is two-sided, greater than type, less than type respectively}. Default is "two-sided".

Returns

1. **statistic:** If *alternative* is "two-sided", the sum of the ranks of the differences above or below zero, whichever is smaller. Otherwise the sum of the ranks of the differences above zero.
2. **p-value:** The p-value for the test depending on *alternative*.

Example 3:

Test scores of a group of 15 high – school students before &after a training programme are as given below :

Score before	63	75	78	84	58	58	70	76	74	88	74	94	99	79	93
Score after	84	86	75	94	50	95	97	98	72	100	101	98	105	84	90

Use appropriate statistical test at to check if the training has any effect on the test scores.

Hypothesis:

H_0 :M_x =M_y ; H_1: M_x< M_y

X: score before training (Median of corresponding distribution is M_x)

Y: score after training (Median of corresponding distribution is M_y)

```
In [1]: #comparison of two variables (Before-After) using Wilcoxon Signed
      rank test
      from scipy import stats
      from scipy.stats import wilcoxon
```

ISBN 978-93-80788-93-7

```
In [2]: # enter data of two variables as x and y
      #Scores before training program
      x=[63, 75, 78, 84, 58, 58, 70, 76, 74, 88, 74, 94, 99, 79, 93]
      #Scores after training program
      y=[84,86,75,94,50,95,97,98,72,100,101,98,105,84,90]
```

```
In[3]:  print ("comparison of two variables (Before -After) using Wilcoxon
      Signed rank test")
      #test evaluation
      stat,p  =  stats.wilcoxon(x,y,  zero_method='wilcox',  correcticn=False,
      alternative = 'less')
      print ('Results of Wilcoxon Signed rank test test')
      print ('Statistics=%.3f, p=%.3f' %(stat, p))

 comparison two variables (Before-After) using Wilcoxon Signed rank test
 Results of Wilcoxon Signed rank test test
 Statistics=13.000, p=0.004
```

```
In [4]: #interpret
      alpha = 0.05
      if p > alpha:
            print ('(Fail to reject H0)')
      else:
            print('(Reject HO)')
  (Reject HO)
```

Interpretation: Since p-value=0.004< 0.01 indicates that one should reject null hypothesis.

16.4 More than Two Sample Tests

Independent sample comparison (Kruskal–Wallis H):

Kruskal–Wallis H tests the significance of difference between the medians of more than two groups, which may have different sizes. This test is commonly called as Non-parametric ANOVA or Kruskal-Wallis test.If the result of the Kruskal-Wallis test is significant, then we go for post-hoc test. Here we will use Dunn's test to find *which* pair of the groups are showing significance when compared pairwise.

Basic conditions for applying test:

- One-way data.
- Dependent variable is ordinal, interval, or ratio.
- Independent variable is a factor with two or more levels. That is, two or more groups.
- Observations between groups are independent. That is, not paired or repeated measures data.
- In order to be a test of medians, the distributions of values for each group need to be of similar shape and spread. Otherwise the test is a test of distributions.

It performs a Kruskal-Wallis rank sum test.
Syntax: scipy.stats.kruskal(**args, **kwargs*)
Compute the Kruskal-Wallis H-test for independent samples.

Parameters

1. **sample1, sample2, ...**array_like two or more arrays with the sample measurements can be given as arguments.
2. **nan_policy**{'propagate', 'raise', 'omit'}, optional

Defines how to handle when input contains nan. The following options are available (default is 'propagate'):

- 'propagate': returns nan
- 'raise': throws an error
- 'omit': performs the calculations ignoring nan values

Returns

1. **statistic:** The Kruskal-Wallis H statistic, corrected for ties.
2. **p-value:** The p-value for the test using the assumption that H has a chi square distribution.

Example 4:

Test whether there exists a significance of difference between the scores of three groups when compared against each other for the following given data set. Use 5% l. o. s. Also use post-hoc test to find the exact significance.

Group 1	63	75	78	84	58	58	70	76	74	88	74	94	58	79	93
Group 2	84	86	75	94	50	95	97	98	72	100	101	98	105	84	90
Group 3	74	76	65	84	50	85	97	88	72	90	101	98	115	94	90

Hypothesis:

H_0: There is no significance of difference between the median of three groups.

H_1: At least one pair of groups differs significantly in their median when compared.

```
In [1]: #Kruskal-Wallis test for comparison of more than two independent
      variables
      import numpy as np
      import pandas as pd
      from scipy import stats
      from scipy. stats import kruskal

In[2]: #enter data of three variabies as x, y and z
      x=np.array([63,75,78,84,58,58,70,76,74,88,74,94,58,79,93])
      y=np.array([84,86,75,94,50,95,97,98,72,100,101,98,105,84,90])
      z=np.array([74,76,65,84,50,85,97,88,72,90,101,98,115,94,90] )
```

ISBN 978-93-80788-93-7

```
In[3]: #Calculation of Kruskal–Wallis is test
       stat,p=stat3.kruskal(x, y, z)
       print('Results of Kruskal–Wallis test' )
       print('Statistics=%.3f, p=%.3f %(stat, p))

Results of Kruskal–Rallis test
Statistics=8.383, p=0.015

In[4]: # interpret
      alpha = 0.05
      if p > alpha:
        print(' (Fail to reject H0)')
      else:
        print('(Reject H0)')

  (Reject HO)
```

Interpretation: Since p-value =0.015< 0.05 indicates one should reject null hypothesis and conclude that there exists significance of difference between the scores of three group at 5% l.o.s.

Repeated measure comparison (Friedman-Chi-Square):

The Friedman test tests the null hypothesis that repeated measurements of the same individuals have the same distribution. It is considered as the extension of paired t-test.

Syntax: scipy.stats.friedmanchisquare(**args*)

It computes the Friedman test for repeated measurements

Parameters

1. **measurements1, measurements2, measurements3…** : Arrays of measurements. All of the arrays must have the same number of elements. At least 3 sets of measurements must be given.

Returns

1. **statistic** :The test statistic, correcting for ties.
2. **p-value** :The associated p-value assuming that the test statistic has a chi squared distribution.

Example 5

The study is planned to measure a stress level on four consecutive days of week. 10 individuals were randomly selected and their stress is measured on Day-1 to Day-4 using a 10 point stress scale in which score 10 indicates highest stress. Test the significance of difference between the stress levels on four days.

Subject	**Day-1**	**Day-2**	**Day-3**	**Day-4**
1	8	7	6	7

ISBN 978-93-80788-93-7

2	5	8	5	6
3	6	5	3	4
4	6	6	7	3
5	8	10	8	6
6	6	5	6	3
7	6	5	2	3
8	9	9	9	6
9	5	4	3	7
10	7	6	6	5

Hypothesis:

H_0: There is no significance of difference between the median scores on four days.

H_1: There is significance of difference between the median scores on four days.

```
In [1]: #Friedman chi square test to compare repeated measure data
      import numpy as np
      import pandas as pd
      from scipy import stats
      from scipy.stats import friedmanchisquare

In [2]: #enter data of Four time points as Dayl, Day2, Day3 and Day4
      Dayl=np.array([8,5,6,6,8,6,6,9,5,7])
      Day2=np.array([7,8,5,6,10,5,5,9,4,6])
      Day3=np.array([6,5,3,7,8,6,2,9,3,6])
      Day4=np.array([7,6,4,3,6,3,3,6,7,5])


In [3]: #Calculation of Friedman chi square test
      stat,p=stats.friedmanchisquare(Dayl,Day2,Day3,Day4)
      print("The results of fliedman chi square test is ")
      print('Statistics=%.3f,p=%.3f' %(stat, p))

The results of friedman chi square test is
Statistics=7.967, p=0.047

In[4]: # interpret
      alpha = 0.05
      if p > alpha:
            print('(Fail to reject H0)')
      else:
            print('(Reject H0)')

  (Reject HO)
```

Interpretation: Since p-value =0.047< 0.05 indicates one should reject null hypothesis and conclude that There is significance of difference between the median scores on four days at 5% l.o.s.

References

1. C. R. Kothari (2009) "Research Methodology: Methods & Techniques" (Second Revised Edition), New Age International Publishers, New Delhi.
2. Daniel W.W.:Applied Non Parametric Statistics, First edition Boston-Houghton Mifflin Company.
3. Mark Lutz, “Learning Python”, 5th Edition, O'Reilly Media
4. Seabold, Skipper, and Josef Perktold. “statsmodels: Econometric and statistical modeling with python.” *Proceedings of the 9th Python in Science Conference.* 2010.

Case Studies
Case study of Polio in Greater Mumbai

Ms. Anjali Sutar and Ms. Priyanka Chataule
Assistant Professors, Department of Statistics, K.C. College

17.1 Introduction

The data we posse analysis the polio cases in Greater Mumbai from 1960 to 1975. We selected this data set because polio has not only been one of the most infectious and dangerous diseases but was also the corner stone in research in how we fought diseases like this. Polio research led to the usage of vaccinations in modern medicine to combat viruses.

Poliomyelitis, often called polio, is an infectious disease caused by the poliovirus. Once known as Infantile Paralysis, the term "poliomyelitis" is used to identify the disease caused by any of the three types of poliovirus. Poliomyelitis has existed for thousands of years, with depictions of the disease in ancient art. The virus that causes it was first identified in 1908.

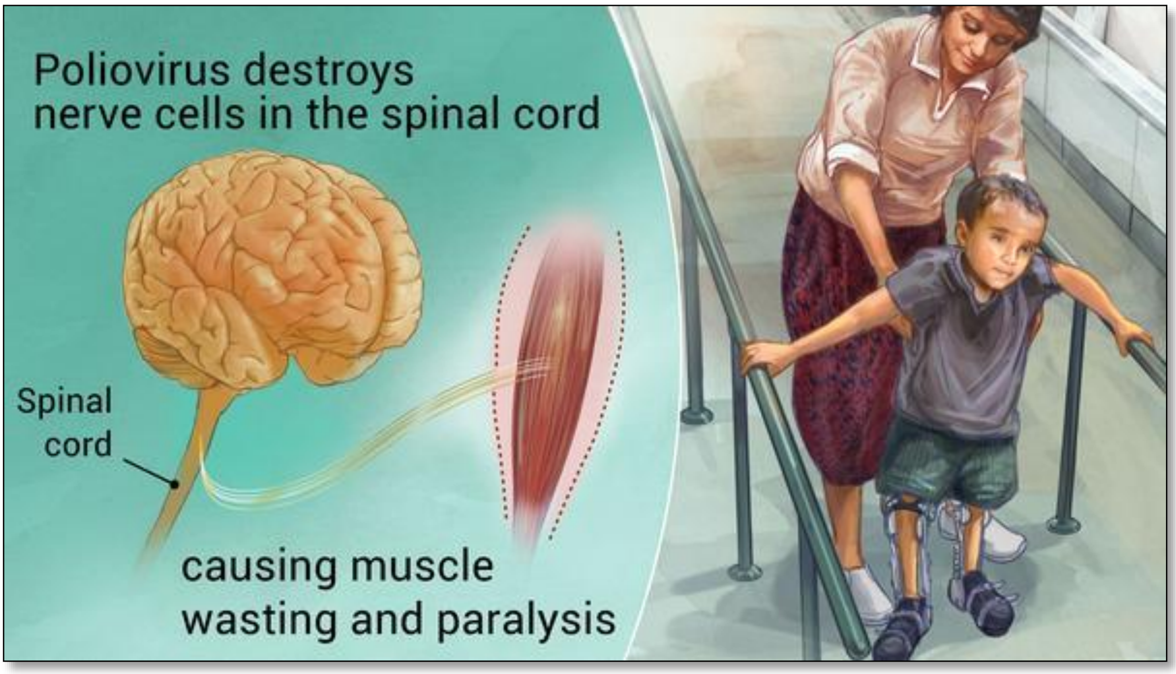
Three serotypes of poliovirus have been identified—poliovirus type 1 (PV1), type 2 (PV2), and type 3 (PV3). All three are extremely virulent and produce the same disease symptoms. PV1 is the most commonly encountered form, and the one most closely associated with paralysis.

17.2 Problem Statement

Poliomyelitis is a crippling disease with dramatically visible impact on the patient. Over the last fifty years the disease has been brought under control by the use of oral vaccine. It is of considerable interest to identify trend, seasonality and other features of data on incidence of polio.

Through this data we wish to analysis how this kind of deadly disease spreads and how world events impact the spread of these diseases.

ISBN 978-93-80788-93-7


Poliovirus destroys nerve cells in the spinal cord
Spinal cord
causing muscle wasting and paralysis

17.3 Objectives

- Features of data on incidence of polio
- To identify trend and seasonality
- To prepare appropriate conclusion from our data analysis

17.4 Data Description

The data **"polio data.csv"** consists of 3 sets of variables for us to analysis.

- Year
- Month
- Number of polio cases

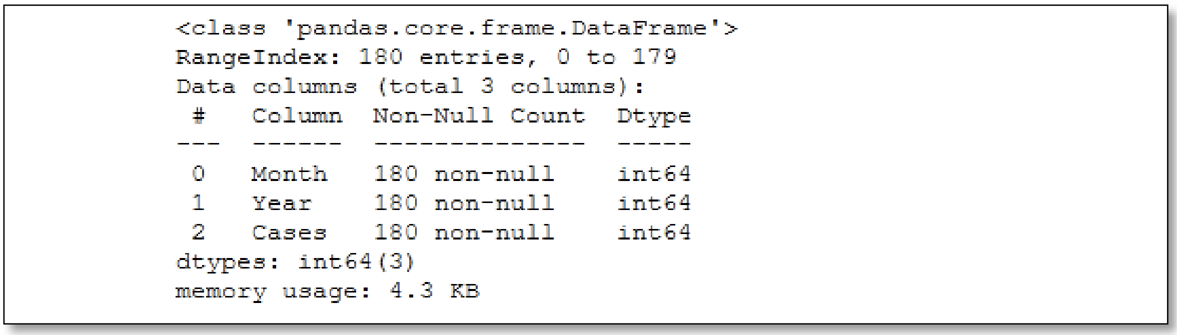
17.5 Methods

The first step we took was to clean the data before analysis. Since there was data missing for 1 month for Year 1960 and 11 months for Year 1975, we had to exclude while considering any yearly based calculations. Since all months had 14 data entries, we didn't have to clean any data while monthly calculations.

We utilised mathematical statistics techniques like mean, median, mode, Time Series and Trend Analysis, graph plotting to analysis the data.

17.6 Data Analysis and Coding

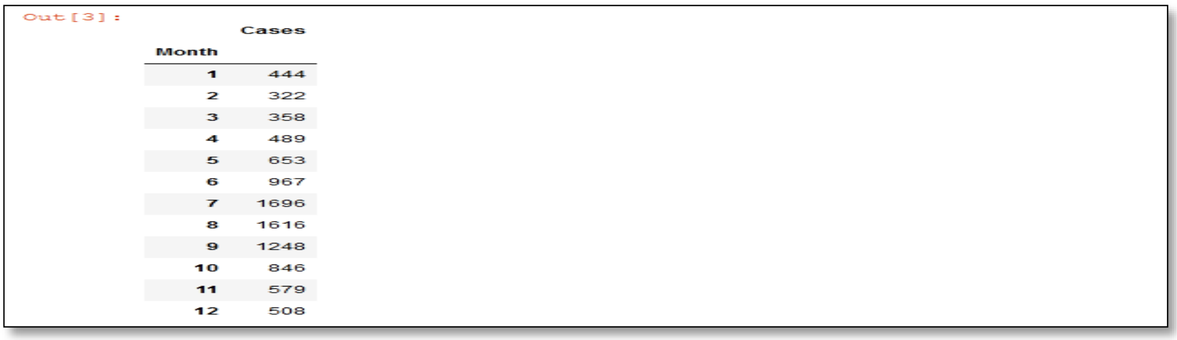
```
In[1]:  #Importing important libraries
       import os                                # helps to change directory
       import pandas as pd      # Used for creating and analyzingdataframes
       import numpy as np                 # Used for numerical calculations
       import matplotlib.pyplot as plt                  # Used for plotting
       import seaborn as sns                            # Used for Plotting
       os.chdir ("C:/Users/Admins/Desktop" )
       po=pd.read_csv( "C:/Users/Admin/Desktop/polio data.csv")
       #Find info on data
       po.info()
```



```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 180 entries, 0 to 179
Data columns (total 3 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   Month   180 non-null    int64
 1   Year    180 non-null    int64
 2   Cases   180 non-null    int64
dtypes: int64(3)
memory usage: 4.3 KB
```

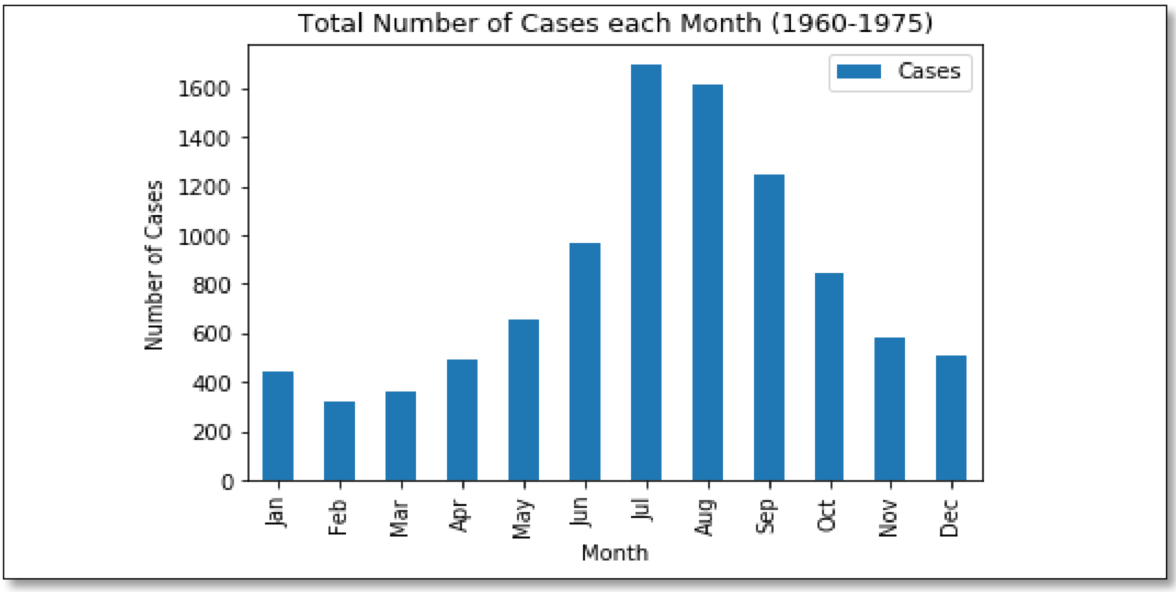
```
In[2]: # To check number of nun values
       print ("Number of missing values in our data are \n{}" .format
       (str(po.isnull)(). sum())))
 Number of missing values in our data are
 Month:        0
 Year :        0
 cases:        0
 dtype: int64
```

```
In [3]: pd.pivot_table(po,values='cases', index=['Month'], aggfunc=np.sum)
```

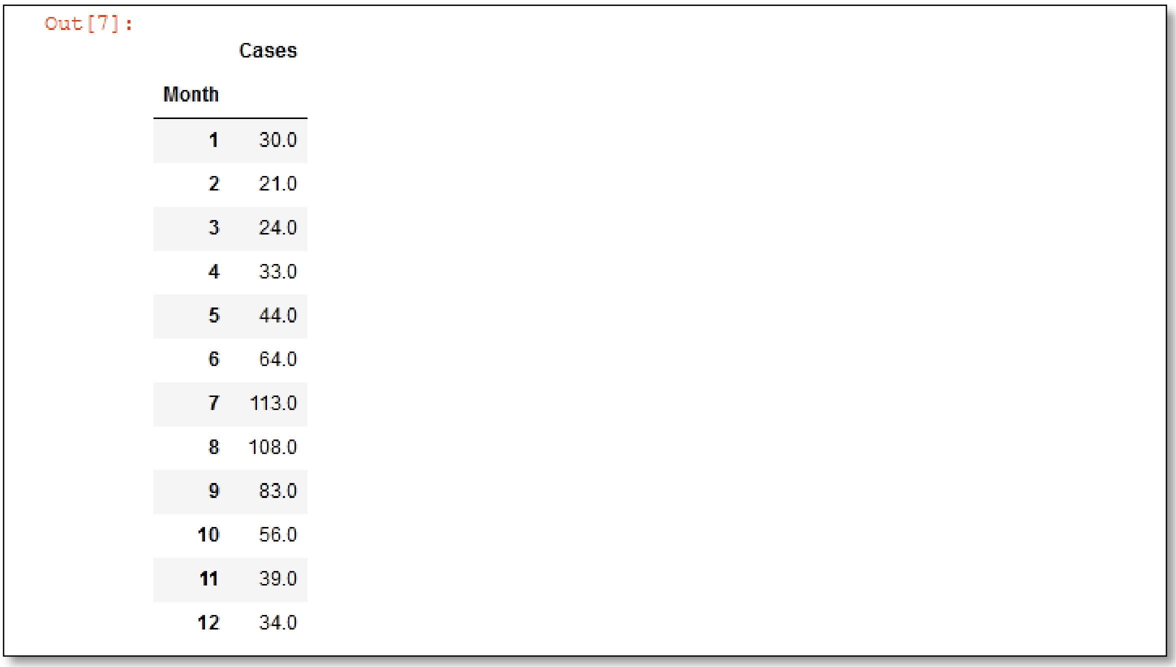


```
Out[3]:
              Cases
     Month
         1      444
         2      322
         3      358
         4      489
         5      653
         6      967
         7     1696
         8     1616
         9     1248
        10      846
        11      579
        12      508
```

```
In [6]: pd.pivot_table(po,values='cases', index=['Month'],
       aggfunc=np.sum).plot.bar()
       plt.xlabel('Month')plt.xticks([0,1,2,3,4,5,6,7,8,9,10,11], ['Jan'
      ,'Feb','Mar','Apr','May','jun','Jul','Aug','Sep','Oct','Nov','Dec'])
       plt.ylabel('Number of Cases')
       plt.title("total Number of Cases each Month (1960-1975)")
       plt.show()
```



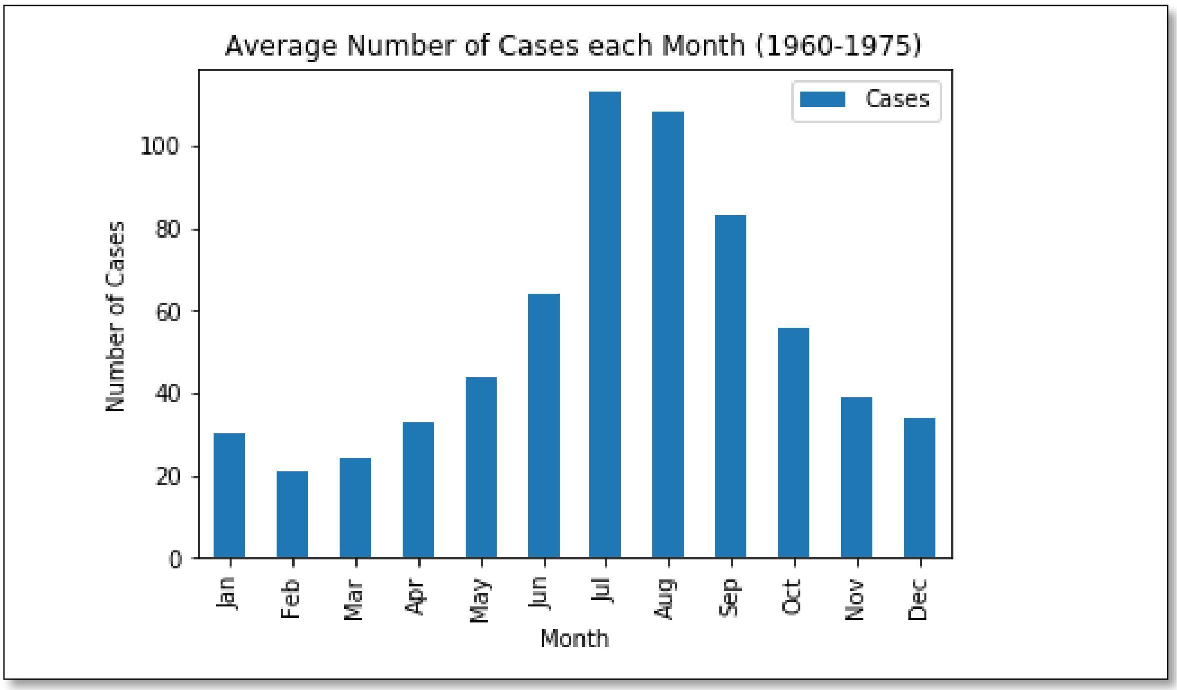
```
In [7]: pd.pivot_table(po,values='cases',index=['Month'],
        aggfunc=np.mean)round()
```



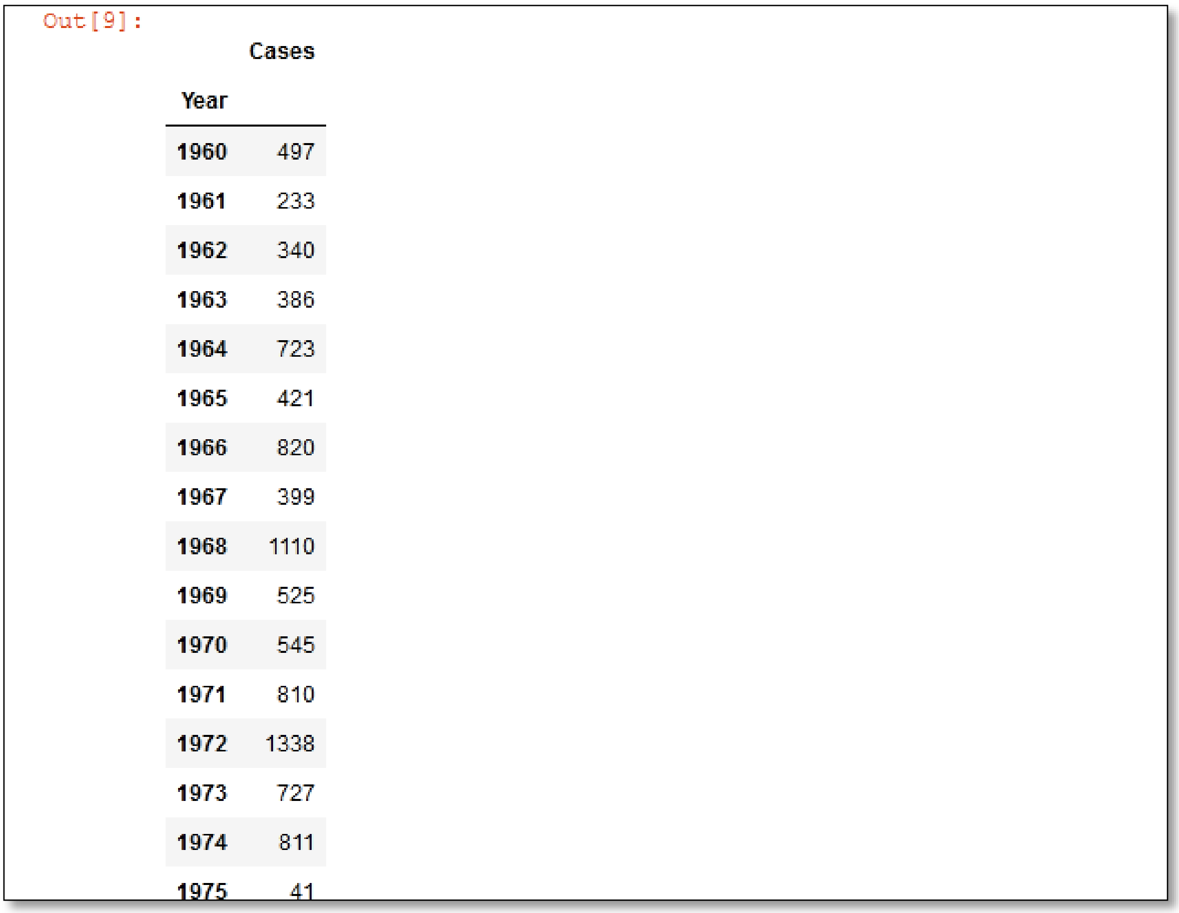
```
Out[7]:
                Cases
        Month
            1    30.0
            2    21.0
            3    24.0
            4    33.0
            5    44.0
            6    64.0
            7   113.0
            8   108.0
            9    83.0
           10    56.0
           11    39.0
           12    34.0
```

```
In[8]:     pd.pivot_table(po,values='cases', index=['Month'],
           aggfunc=np.mean).round().plot.bar()
           plt.xlabel('Month')
           plt.xticks([0,1,2,3,4,5,6,7,8,9,10,11],['Jan','Feb','Mar ','
           Apr','May’,'jun','Jul','Aug','Sep','Oct','Nov','Dec'])
           plt.ylabel('Number of Cases')
           plt.title("Average Number of Cases each Month (1960-1975)")
           plt.show()
```

ISBN 978-93-80788-93-7



```
In [9]: pd.pivot_table(po, values='Cases', index=['Year'], aafunc=np.sum)
```

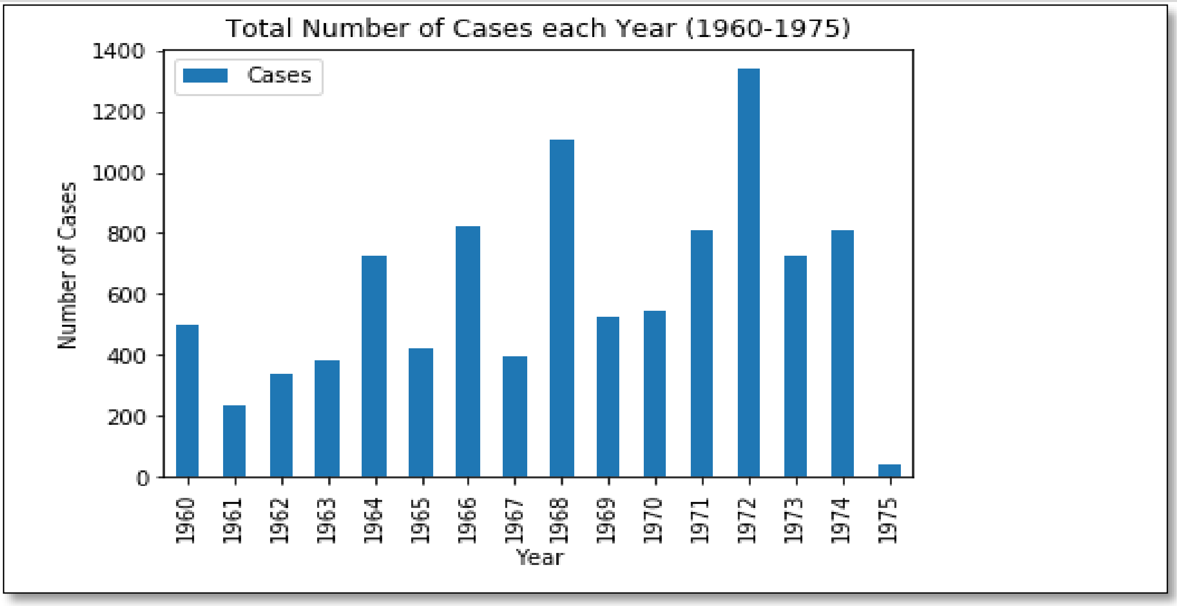


```
Out[9]:
          Cases
Year
1960        497
1961        233
1962        340
1963        386
1964        723
1965        421
1966        820
1967        399
1968       1110
1969        525
1970        545
1971        810
1972       1338
1973        727
1974        811
1975         41
```

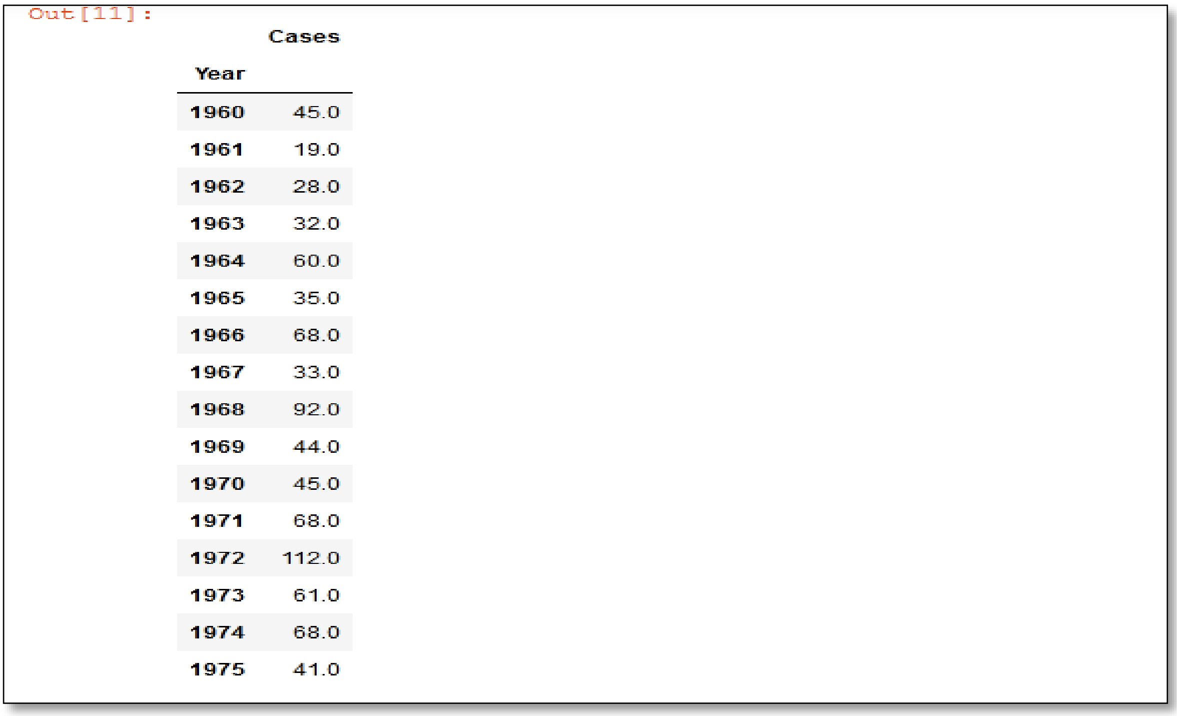
```
In[10]: pd.pivot_table(po,values='cases', index=['Year'],
        aggfunc=np.sum).plot.bar()
        plt.xlabel('Year')
```

ISBN 978-93-80788-93-7

```
        plt.ylabel('Number of Cases')
        plt.title("total Number of Cases each Year (1960-1975)")
        plt.show()
```



```
In[11]: #since there is uneven of cases in year 1975, we will consider you
         be better indicator
         pd.pivot_table(po, values='cases', index=['Year'], aggfunc
            =np.mean)    . round()
```

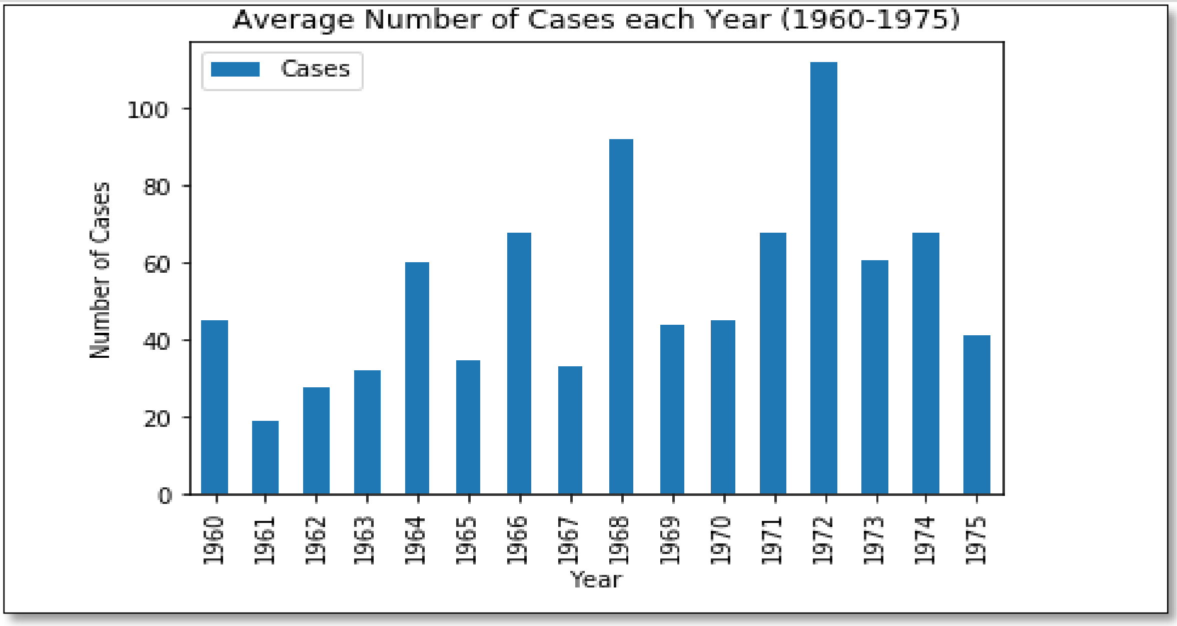


```
Out[11]:
              Cases
      Year
      1960     45.0
      1961     19.0
      1962     28.0
      1963     32.0
      1964     60.0
      1965     35.0
      1966     68.0
      1967     33.0
      1968     92.0
      1969     44.0
      1970     45.0
      1971     68.0
      1972    112.0
      1973     61.0
      1974     68.0
      1975     41.0
```

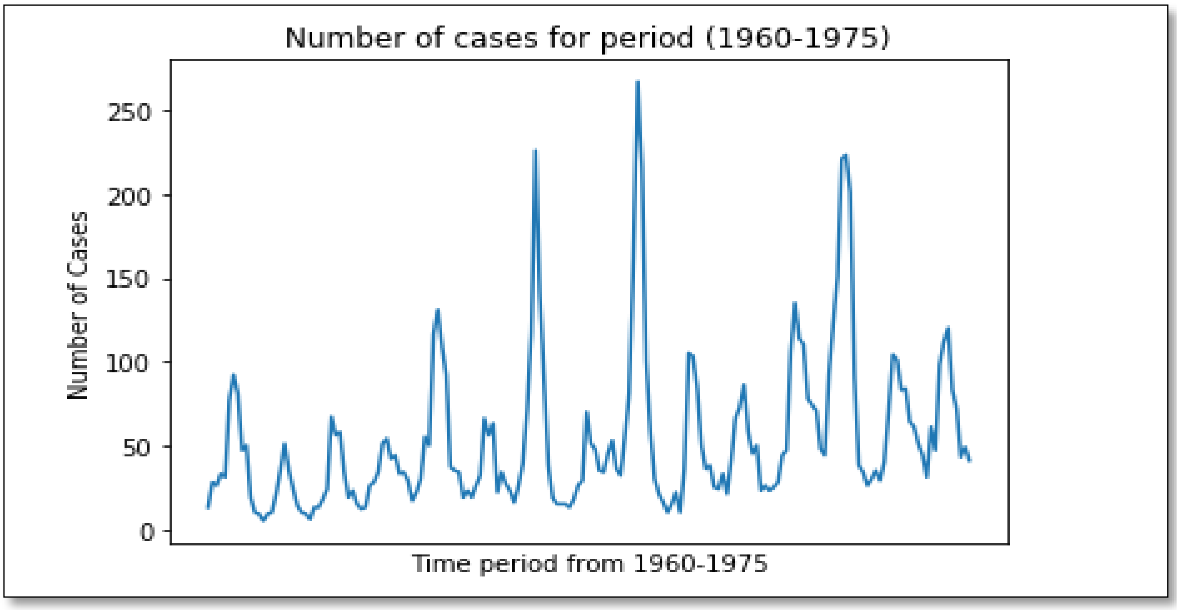
```
In  [12]:  pd.pivot_table(po,  values='cases',  index=['Year'],  aggfunc=  np.
      mean).round().plot.bar()
      plt.xlabel('Year')
      plt.ylabel('Number of Cases')
      plt.title("Average Number of Cases each Month (1960-1975)")
```

ISBN 978-93-80788-93-7

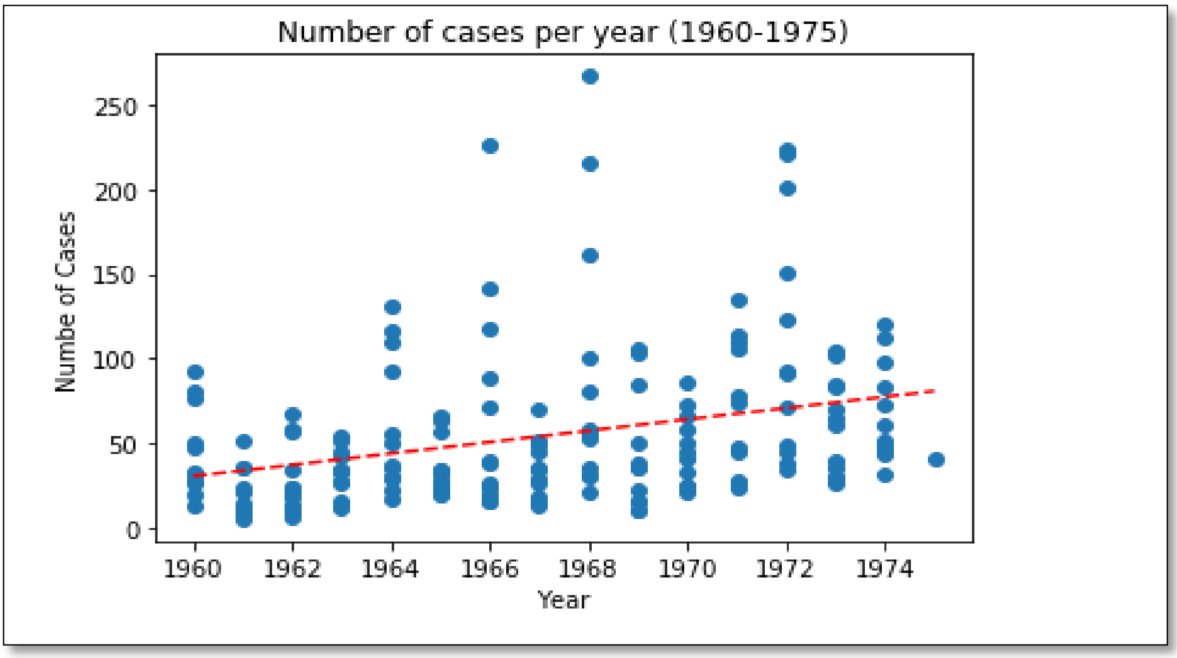
```
      plt.show()
```



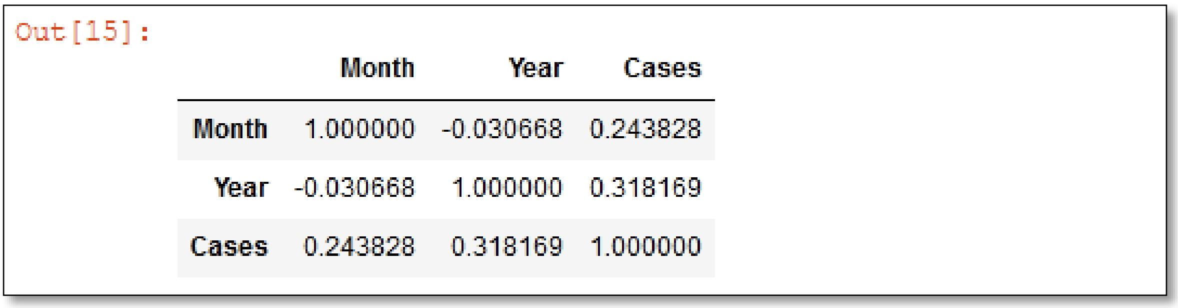
```
In[13]: #Number of cases per month each year
      po.[cases].plot()
      plt.ylabel('Number of Cases')
      plt.xlabel('Time period from 1960-1975')
      plt.xticks([],[])
      plt.title("Number of cases for period (1960-1975)")
      plt.show()
```



```
In[14]: po_year=po['Year']
      po_cases=po['Cases']
      plt.scatter(po_year, po_cases)

      z=np.polyfit(po_year,po_cases, 1)
      p=np.poly1d(z)
      plt.plot(po_year,p(po_year),"r--")

      plt.ylabel('Number of Cases')
      plt.xlabel('Year')
      plt.title("Number of cases per year (1960-1975)")
      plt.show()
#This graph shows a increasing trendline for years showing that number of cases
has increased per year
```



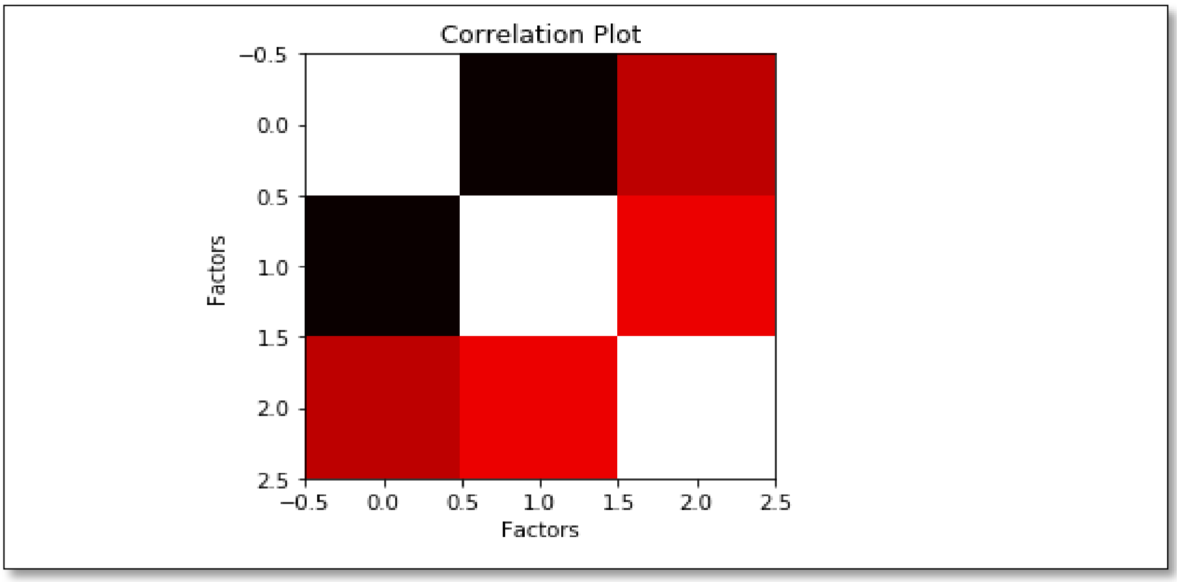
```
In [15]: corr_plot=po.corr()
         corr_plot
```



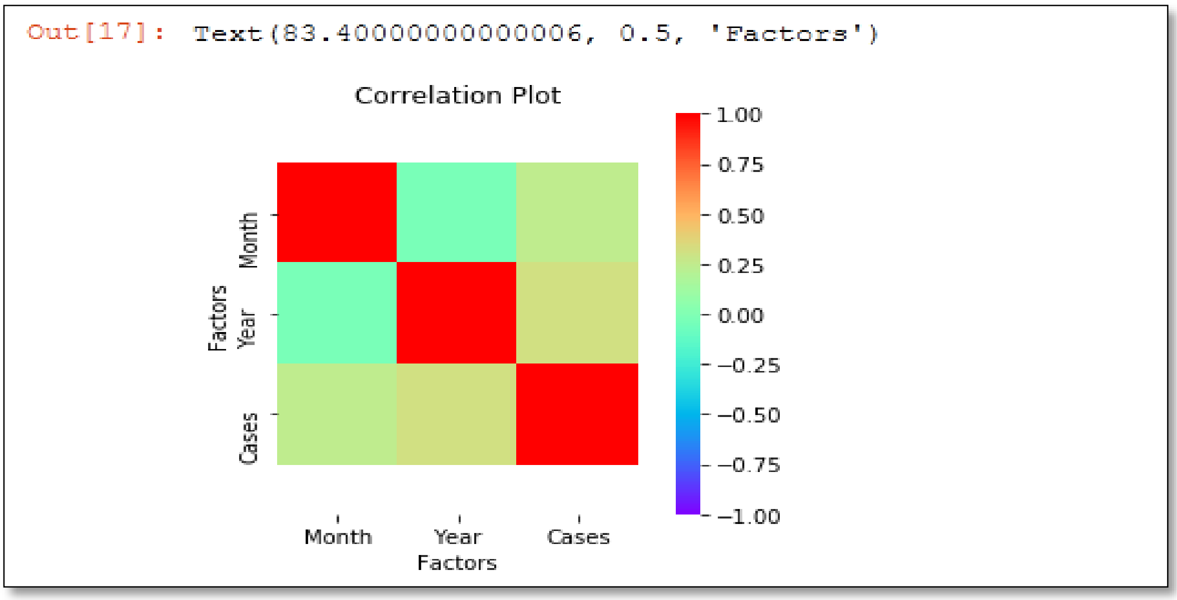
Out[15]:

	Month	Year	Cases
Month	1.000000	-0.030668	0.243828
Year	-0.030668	1.000000	0.318169
Cases	0.243828	0.318169	1.000000

```
In[16]: plt.imshow(corr_plot, cmap='hot',)
      plt.xlabel('Factors')
      plt.ylabel('Factors')
      plt.title("correlation Plot")
      plt.show()
```



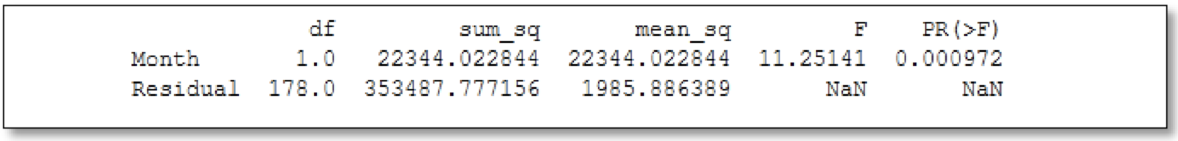
```
In[17]:corr_plot_sns=sns.heatmap(corr_plot,     annot=False,vmax=1     vmin=1,
      center=0, cmap="rainbow", square=True,)
      corr_plot_sns.set_ylim(len(corr_plot)+0.5,-0.5)
      plt.title("correlation plot")
      plt.xlabel("Factors")
      plt.ylabel("Factors")
 # Its clear that cases are not correlaed with month
```



```
Out[17]:  Text(83.40000000000006, 0.5, 'Factors')
```

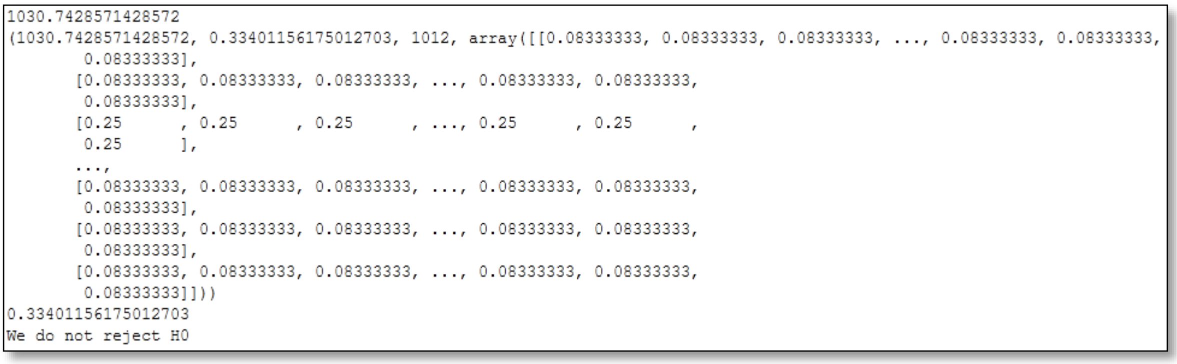
```
In [18]: import statsmcdels.api as sm
      from statsmodels.formula.api import ols
      lm=ols('cases~month',data=po).fit()
      lm
      table=sm.stats.anova_lm(lm)
      print(table)
```

ISBN 978-93-80788-93-7



```
                 df         sum_sq       mean_sq          F    PR(>F)
Month           1.0   22344.022844  22344.022844  11.25141  0.000972
Residual      178.0  353487.777156   1985.886389       NaN       NaN
```

```
In [19]:def_chi_test(df,alpha):
      from scipy import stats

      contingency table=pd.crosstab (df["cases"],df["Month"])
      observed_values=contingency_table.values
      observed_values

      chisq_output=stats.chis2_contingency(contingency_table)
      chisq_output

      expected_values=chisq_output[3]
      chi_squared_stat=(((observed_values-expected_values)**2)/expected_
      values).sum().sum()

      print(chi_squared_stat)
      print(chisq_output)
      print(chisq_output[1])

      if chisq_output[1]>alpha:
            print("we do not reject HO")
      else :
            print("we do reject HO")

chi_ind_test(po,0.05)
```



```
1030.7428571428572
(1030.7428571428572, 0.33401156175012703, 1012, array([[0.08333333, 0.08333333, 0.08333333, ..., 0.08333333, 0.08333333,
         0.08333333],
        [0.08333333, 0.08333333, 0.08333333, ..., 0.08333333, 0.08333333,
         0.08333333],
        [0.25      , 0.25      , 0.25      , ..., 0.25      , 0.25      ,
         0.25      ],
        ...,
        [0.08333333, 0.08333333, 0.08333333, ..., 0.08333333, 0.08333333,
         0.08333333],
        [0.08333333, 0.08333333, 0.08333333, ..., 0.08333333, 0.08333333,
         0.08333333],
        [0.08333333, 0.08333333, 0.08333333, ..., 0.08333333, 0.08333333,
         0.08333333]]))
0.33401156175012703
We do not reject H0
```

```
In[20]: # we can not use chisquare result since frequencies are very small
```

17.7 Conclusion

We observe that polio cases were rising faster than compared to the population. Polio Cases observed a seasonal trend, seeing sharp increase during rainy seasons and seeing sharp decrease in summer seasons.

World events like war cause sharp increase in polio cases. This shows lack of proper health facilities for emergencies. Similarly, polio outbreak in other countries causes sharp increase in polio cases in India too. Thus we fail to prevent entry of diseases from other containment countries. We lack facilities to prevent entry of infected personnel's Polio cases increased in period 1960 to 1975.

17.8 Limitations

The data is from a secondary source. It is unclear whether the rise in case is because of increase in people getting infected with polio or due to better methods to detect polio.

We assume that the polio cases in a month or year where infected in the same month or year. We assume that there is no gap between when a person was infected and when the infection was detected. Their might exist a latent period between when a person is infected, when the infection is detected and when the case has been recorded.

17.9 References

1. Training Programme notes conducted by Pravesh Tiwari
2. https://towardsdatascience.com/train-test-split-and-cross-validation-in-python-80b61beca4b6
3. https://heartbeat.fritz.ai/seaborn-heatmaps-13-ways-to-customize-correlation-matrix-visualizations-f1c49c816f07

Chapter 18
Analysis of Air Pollution in New Delhi

Mr. Rajesh Kalal and Mr. Shubham Gupta
DBT- Star College Status Scheme Researchers, Department of Statistics

18.1 Abstract

There is increasingly growing evidence linking urban air pollution to acute and chronic illnesses amongst all age groups. Therefore, monitoring of ambient concentrations of various air pollutants as well as quantification of the dose inhaled becomes quite important, especially in view of the fact that in many countries, policy decisions for reducing pollutant concentrations are mainly taken on the basis of their health impacts. The dose when gets combined with the likely responses, indicates the ultimate health risk (HR). Thus, as an extension of our earlier studies, HR has been estimated for three pollutants, namely, suspended particulate matter (SPM), nitrogen dioxide (NO2) and sulfur dioxide (SO2) for Delhi City in India. For estimation and analyses, three zones have been considered, viz., residential, industrial and commercial.

The total population has been divided into three age classes (infants, children and adults) with different body weights and breathing rates. The exercise takes into account age-specific breathing rates, body weights for different age categories and occupancy factors for different zones. Results indicate that health risks due to air pollution in Delhi are highest for children. For all age categories, health risks due to SO2 (HR_SO2) are the lowest. Hence, HR_SO2 has been taken as the reference with respect to which HR values due to SPM and NO2 have been compared. Taking into account all the age categories and their occupancy in different zones, average HR values for NO2 and SPM turn out to be respectively 22.11 and 16.13 times more than that for SO2. The present study can be useful in generating public awareness as well as in averting and mitigating the health risks.

18.2 Data

The dataset contains the following features:

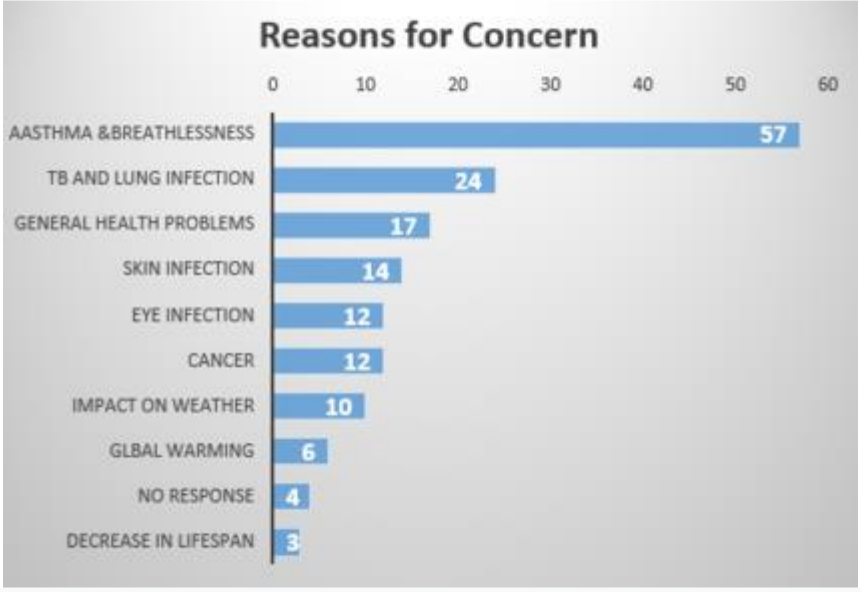
1. stn_code : Station code. A code given to each station that made the measurements.
2. sampling_date : The date when the data was recorded.
3. state : It represents the states whose air quality data is measured.
4. location : It represents the city whose air quality data is measured.
5. agency : Name of the agency that measured the data.
6. type : The type of area where the measurement was made.

7. so2 : The amount of Sulphur Dioxide measured.
8. no2 : The amount of Nitrogen Dioxide measured.
9. rspm :Respirable Suspended Particulate Matter measured.
10. spm : Suspended Particulate Matter measured.
11. location_monitoring_station : It indicates the location of the monitoring area.
12. pm2_5 : It represents the value of particulate matter measured.
13. date : It represents the date of recording (It is cleaner version of 'sampling_date' feature)

18.3 Effects

Health costs of air pollution:

Asthma is the leading health problem faced by Indians. Not surprisingly, it accounts for more than 50% of the health problems caused by air pollution.



The most important reason for concern over the worsening air pollution in the country is its effect on the health of individuals. Exposure to particulate matter for a long time can lead to respiratory and cardiovascular diseases such as asthma, bronchitis, COPD, lung cancer and heart attack. The Global Burden of Disease Study for 2010, published in 2013, had found that outdoor air pollution was the fifth-largest killer in India and around 620,000 early deaths occurred from air pollution-related diseases in 2010. According to a WHO study, 13 of the 20 most-polluted cities in the world are in India; however, the accuracy and methodology of the WHO study was questioned by the Government of India. India also has one of the highest number of COPD patients and the highest number of deaths due to COPD.

Over a million Indians die prematurely every year due to air pollution, according to the non-profit Health Effects Institute. Over two million children—half the children in Delhi—have abnormalities in their lung function, according to the Delhi Heart and Lung Institute. Over the past decade air pollution has increased in India significantly. Asthma is the most common

ISBN 978-93-80788-93-7

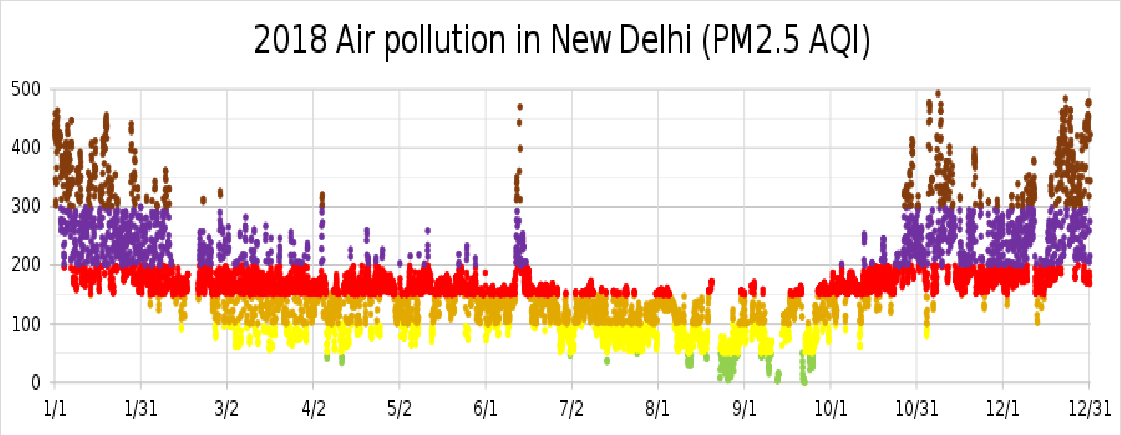
health problem faced by Indians and it accounts for more than half of the health issues caused by air pollution.

Ambient air pollution in India is estimated to cause 670,00 deaths annually and particularly aggravates respiratory and cardiovascular conditions including chronic bronchitis, lung cancer and asthma. Ambient air pollution is linked to an increase in hospital visits, with a higher concentration of outdoor pollution particulates resulting in emergency room visit increases of between 20-25% for a range of conditions associated with higher exposure to air pollution. Approximately 76% of households in rural India are reliant on solid biomass for cooking purposes which contributes further to the disease burden of ambient air pollution experienced by the population of India.

State-Wide Trends:

According to the WHO, India has 14 out of the 15 most polluted cities in the world in terms of PM 2.5 concentrations.

Air Quality Index (AQI) is a number used to communicate the level of pollution in the air and it essentially tells you the level of pollution in the air in a given city on a given day. The AQI of Delhi was placed under the "severe-plus category" when it touched 574, by the System of Air Quality and Weather Forecasting and Research. In May 2014, the World Health Organization announced New Delhi as the most polluted city in the world. In November 2016, the Great smog of Delhi was an environmental event which saw New Delhi and adjoining areas in a dense blanket of smog, which was the worst in 17 years.



2018 Air Pollution in New Delhi (PM2.5 AQI).

A surge on June 14 was caused by dust storms brought on by a combination of extreme heat and powerful downdraft winds.

🟫 Hazardous
🟪 Very Unhealthy
🟥 Unhealthy
🟧 Unhealthy for Sensitive Groups

ISBN 978-93-80788-93-7

🟨 Moderate
🟩 Good

- ➢ The average annual SOx and NOx emissions level and periodic violations in industrial areas of India were significantly and surprisingly lower than the emission and violations in residential areas of India.
- ➢ Of the four major Indian cities, air pollution was consistently worse in Delhi, every year over 5-year period (2004–2018). Kolkata was a close second, followed by Mumbai. Chennai air pollution was least of the four.

Steps Taken:

- ➢ The government in Delhi launched an Odd-Even Rule in November, 2017 which is based on the Odd-Even rationing method: This meant that cars running with number plates ending in Odd digits could only be driven on certain days of the week, while the even digit cars could be driven on the remaining days of the week.
- ➢ The Indian government has committed to a 50% reduction in households using solid fuel for cooking
- ➢ Some goals set for future are:
- Clean up the transportation sector by introducing 1,000 electric public transport buses to its 5, 50-string fee.
- Identify effective ways to inform the public about air pollution data.
- Launch new citizen science programs to better document exposures.
- Reduce Carbon Emissions: "According to Inter-governmental Panel on Climate Change, to limit warming well below 2 degree Celsius, CO2 emissions should decline by about 20 per cent by 2030 and reach net zero around 2075; to limit warming below 1.5 degree Celsius, CO2 emissions should decline by 50 per cent by 2030 and reach net zero by around 2050.

18.4 Analysis

We use **"dataset (3) (1).csv"**. We began by importing the libraries we are going to need.

```
In [1]:  # importing important libraries
      import cs                                 # Helps to change directory
      import numpy as np                  # Used for numerical calculations
      import pandas as pd      # Used for creating and analyzing dataframes
      import seaborn as sns                             # Used for plotting
      import metplotlib as plt                          # Used for plotting
      from skIearn import linear_model          # Used for modeling purpose
      from pandas import DataFrame                # Used for importing data
      import statsmodels.api as sm              # Used for modeling purpose
      from sklearn.linear_model import linearRegression
                                                # Used for modeling purpose
```

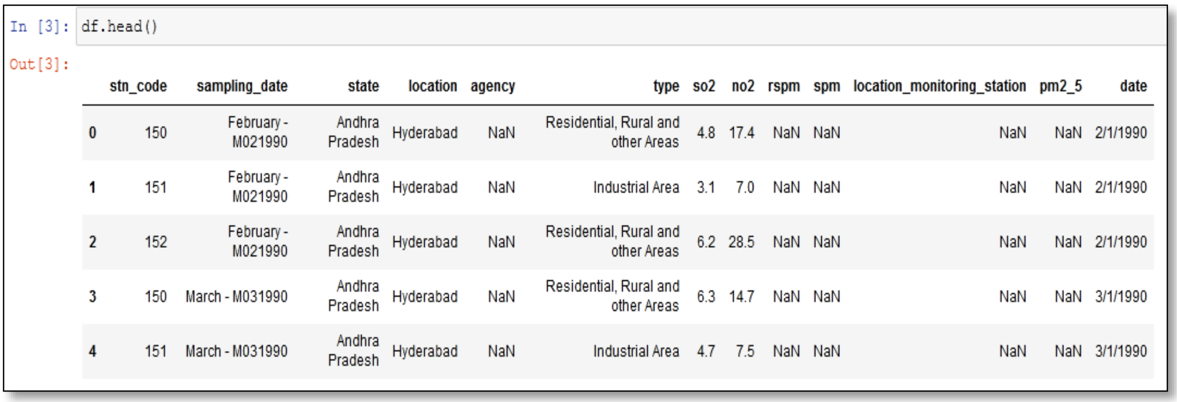
ISBN 978-93-80788-93-7

Then we are going to load the datafiles

```
In [2]: # read dataset using read_csv()-data.csv
        dataset=pd.read_csv("C:/Users/Admin/Desktop/dataset   (3)   (1).csv"   ,
        encoding="ISO-8859-1")
        df=dataset.copy()
```

Initial level of investigation on dataset:

df.head()

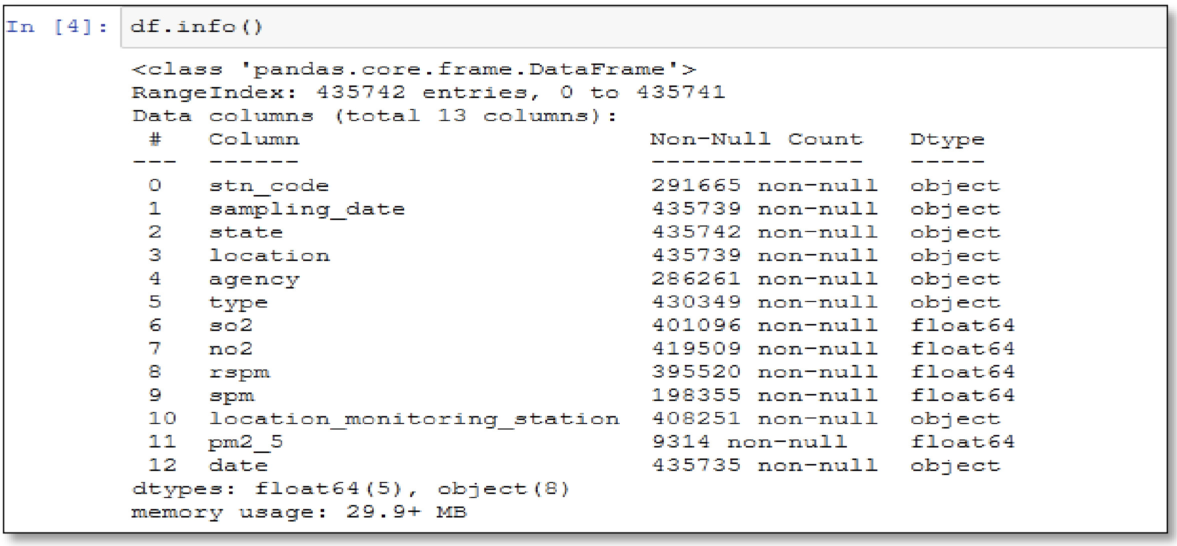


```
In [3]: df.head()
Out[3]:
```

	stn_code	sampling_date	state	location	agency	type	so2	no2	rspm	spm	location_monitoring_station	pm2_5	date
0	150	February - M021990	Andhra Pradesh	Hyderabad	NaN	Residential, Rural and other Areas	4.8	17.4	NaN	NaN	NaN	NaN	2/1/1990
1	151	February - M021990	Andhra Pradesh	Hyderabad	NaN	Industrial Area	3.1	7.0	NaN	NaN	NaN	NaN	2/1/1990
2	152	February - M021990	Andhra Pradesh	Hyderabad	NaN	Residential, Rural and other Areas	6.2	28.5	NaN	NaN	NaN	NaN	2/1/1990
3	150	March - M031990	Andhra Pradesh	Hyderabad	NaN	Residential, Rural and other Areas	6.3	14.7	NaN	NaN	NaN	NaN	3/1/1990
4	151	March - M031990	Andhra Pradesh	Hyderabad	NaN	Industrial Area	4.7	7.5	NaN	NaN	NaN	NaN	3/1/1990

df.info()

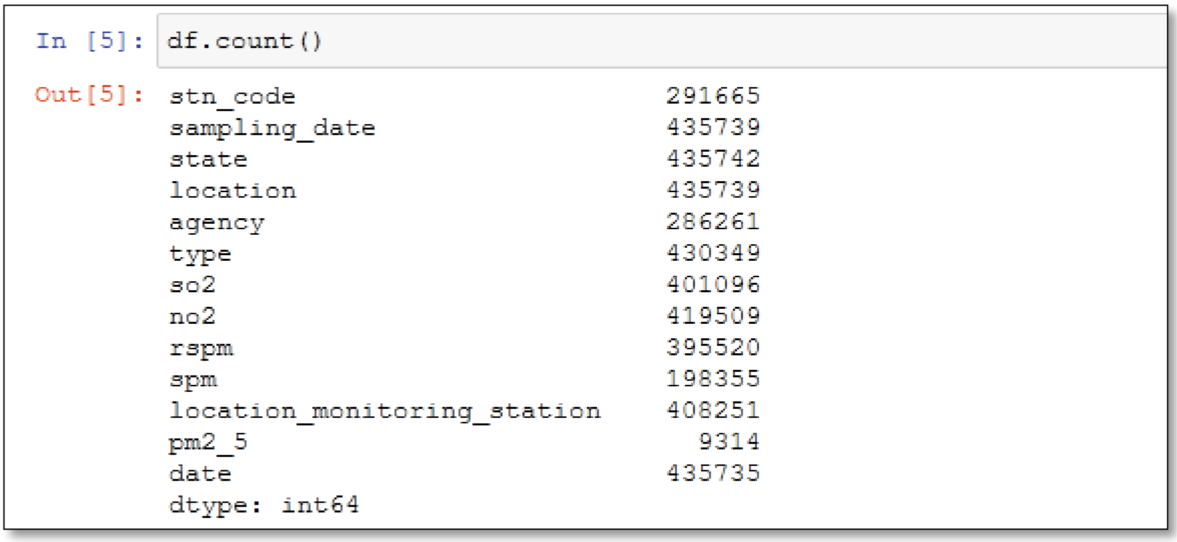
It returns range, column, number of non-null objects of each column, datatype and memory usage.



```
In [4]: df.info()

        <class 'pandas.core.frame.DataFrame'>
        RangeIndex: 435742 entries, 0 to 435741
        Data columns (total 13 columns):
         #   Column                       Non-Null Count   Dtype
        ---  ------                       --------------   -----
         0   stn_code                     291665 non-null  object
         1   sampling_date                435739 non-null  object
         2   state                        435742 non-null  object
         3   location                     435739 non-null  object
         4   agency                       286261 non-null  object
         5   type                         430349 non-null  object
         6   so2                          401096 non-null  float64
         7   no2                          419509 non-null  float64
         8   rspm                         395520 non-null  float64
         9   spm                          198355 non-null  float64
         10  location_monitoring_station  408251 non-null  object
         11  pm2_5                        9314 non-null    float64
         12  date                         435735 non-null  object
        dtypes: float64(5), object(8)
        memory usage: 29.9+ MB
```

df.count()

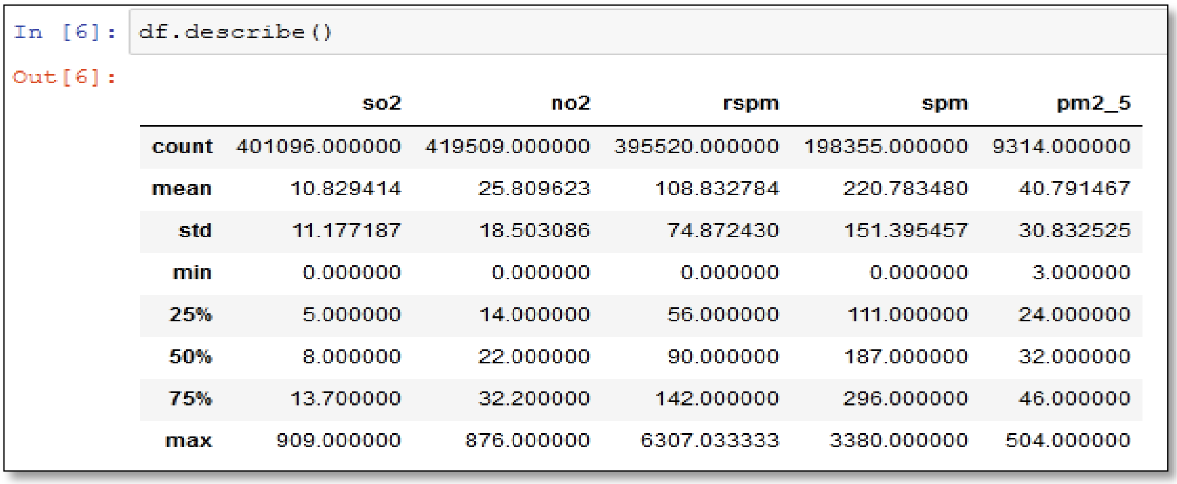
It results in a number of non null values in each column.



```
In [5]:  df.count()

Out[5]:  stn_code                          291665
         sampling_date                     435739
         state                             435742
         location                          435739
         agency                            286261
         type                              430349
         so2                               401096
         no2                               419509
         rspm                              395520
         spm                               198355
         location_monitoring_station       408251
         pm2_5                               9314
         date                              435735
         dtype: int64
```

df.describe()

Generate descriptive statistics that summarize the central tendency, dispersion, and shape of a dataset's distribution, excluding NaN values.

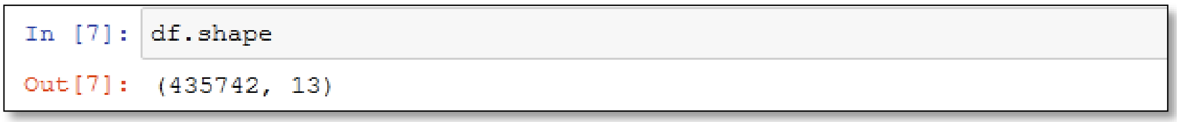


```
In [6]:  df.describe()

Out[6]:
```

	so2	no2	rspm	spm	pm2_5
count	401096.000000	419509.000000	395520.000000	198355.000000	9314.000000
mean	10.829414	25.809623	108.832784	220.783480	40.791467
std	11.177187	18.503086	74.872430	151.395457	30.832525
min	0.000000	0.000000	0.000000	0.000000	3.000000
25%	5.000000	14.000000	56.000000	111.000000	24.000000
50%	8.000000	22.000000	90.000000	187.000000	32.000000
75%	13.700000	32.200000	142.000000	296.000000	46.000000
max	909.000000	876.000000	6307.033333	3380.000000	504.000000

df.shape

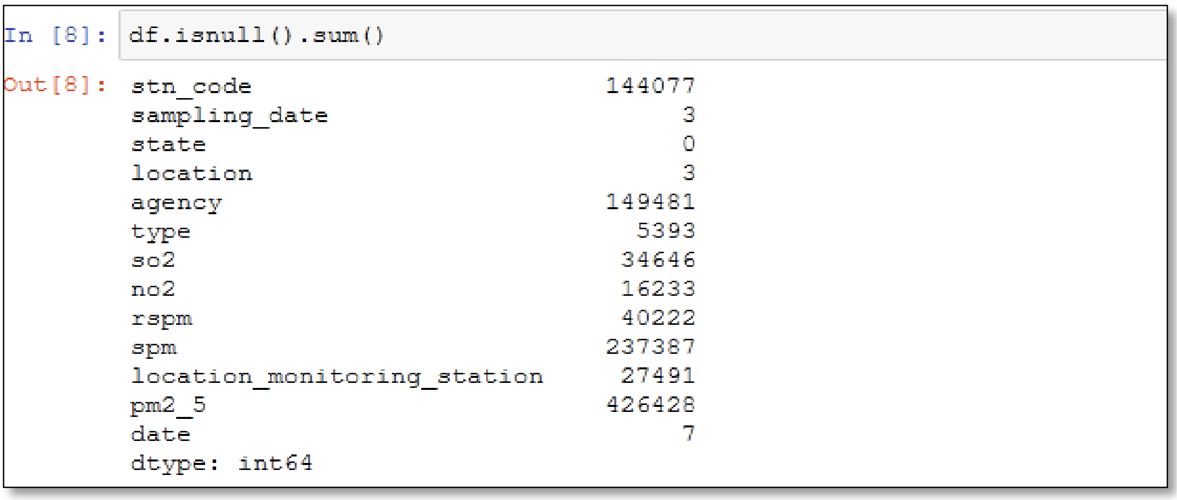
It returns a number of rows and columns in a dataset.



```
In [7]:  df.shape

Out[7]:  (435742, 13)
```

df.isnull().sum()

It returns a number of null values in each column.



```
In [8]: df.isnull().sum()

Out[8]: stn_code                        144077
        sampling_date                        3
        state                                0
        location                             3
        agency                          149481
        type                              5393
        so2                              34646
        no2                              16233
        rspm                             40222
        spm                             237387
        location_monitoring_station      27491
        pm2_5                           426428
        date                                 7
        dtype: int64
```

Dropping of less valued columns:

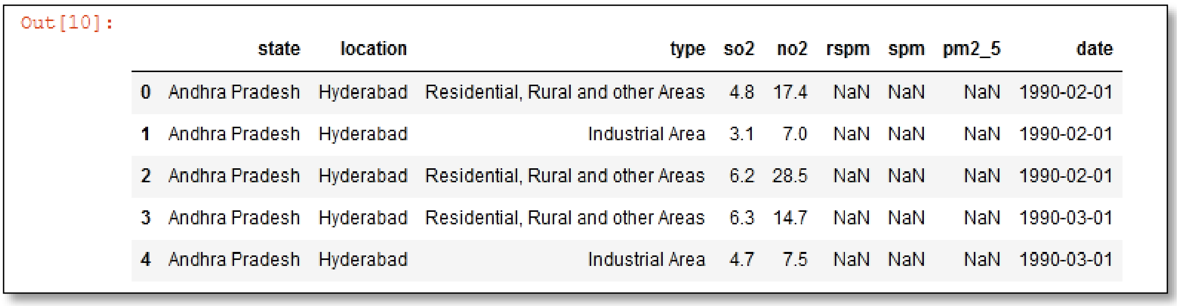
1. stn_code, agency, sampling_date, location_monitoring_agency do not add much value to the dataset in terms of information. Therefore, we can drop those columns.

2. Dropping rows where no date is available.

```
In  [9]:  df.drop(['stn_code',  'agency',’sampling_date’  'location_monitoring_
      station'],  axis=1, inplace=True)
      df= df.dropna(subset=['date'] )
      df.columns

Out[9]:  Index(['state',  'location',  'type',  'so2',  'no  2',  'rspm',  'spm',
      'pm2_5','date'],dtype=' object')
```

Creating a ‘year’ column:

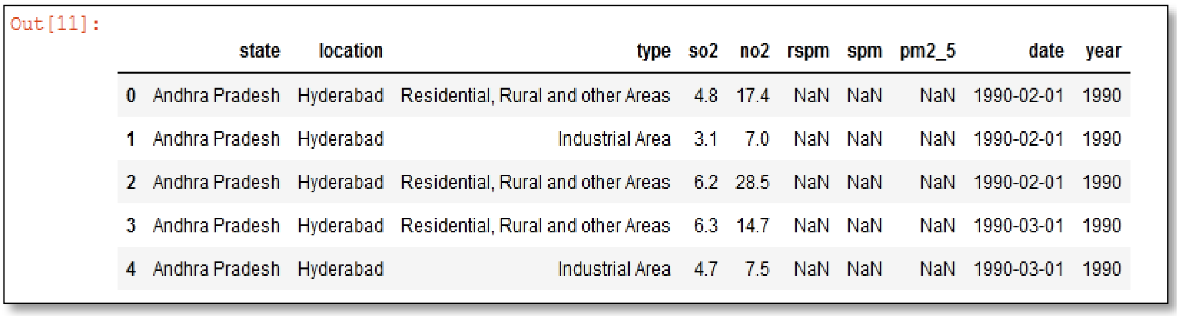
```
In [10]: df['date']=pd.to_datetime(df['date'])
         df.head(5)
```



Out[10]:

	state	location	type	so2	no2	rspm	spm	pm2_5	date
0	Andhra Pradesh	Hyderabad	Residential, Rural and other Areas	4.8	17.4	NaN	NaN	NaN	1990-02-01
1	Andhra Pradesh	Hyderabad	Industrial Area	3.1	7.0	NaN	NaN	NaN	1990-02-01
2	Andhra Pradesh	Hyderabad	Residential, Rural and other Areas	6.2	28.5	NaN	NaN	NaN	1990-02-01
3	Andhra Pradesh	Hyderabad	Residential, Rural and other Areas	6.3	14.7	NaN	NaN	NaN	1990-03-01
4	Andhra Pradesh	Hyderabad	Industrial Area	4.7	7.5	NaN	NaN	NaN	1990-03-01

```
In [11]: dft ['year'] =df.date.dt.year
      df. head (5)
```



Out[11]:

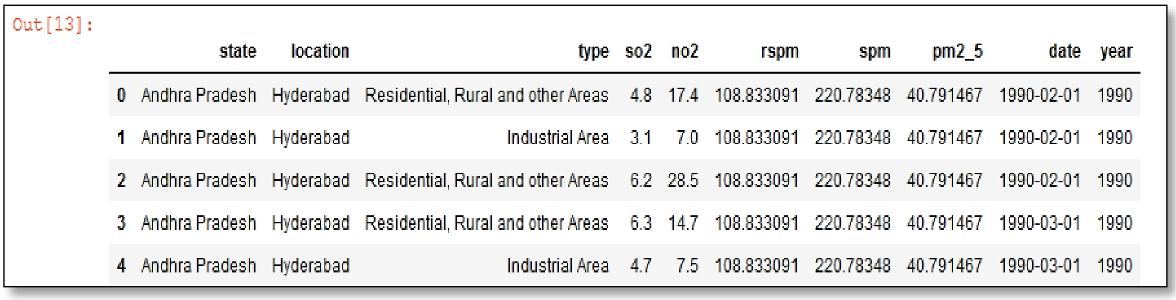
	state	location	type	so2	no2	rspm	spm	pm2_5	date	year
0	Andhra Pradesh	Hyderabad	Residential, Rural and other Areas	4.8	17.4	NaN	NaN	NaN	1990-02-01	1990
1	Andhra Pradesh	Hyderabad	Industrial Area	3.1	7.0	NaN	NaN	NaN	1990-02-01	1990
2	Andhra Pradesh	Hyderabad	Residential, Rural and other Areas	6.2	28.5	NaN	NaN	NaN	1990-02-01	1990
3	Andhra Pradesh	Hyderabad	Residential, Rural and other Areas	6.3	14.7	NaN	NaN	NaN	1990-03-01	1990
4	Andhra Pradesh	Hyderabad	Industrial Area	4.7	7.5	NaN	NaN	NaN	1990-03-01	1990

Handling missing values:

The columns such as so2, no2, rspm, spm, pm2_5 are the ones which contribute much to our analysis. So, we need to remove null from those columns to avoid inaccuracy in the prediction. We use the Imputer from sklearn.preprocessing to fill the missing values in every column with the mean.

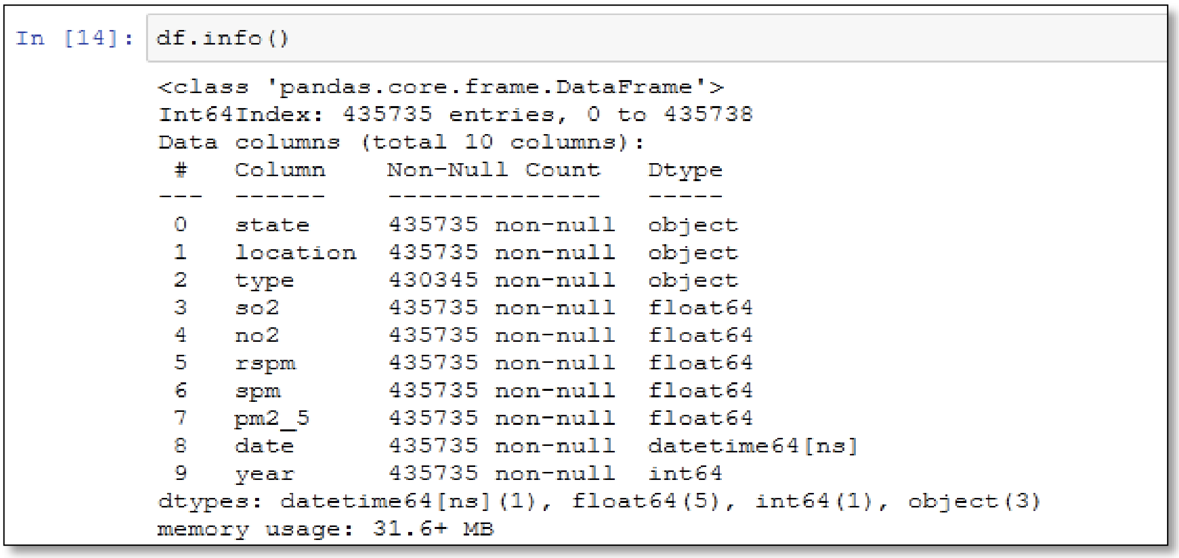
```
In [12]: COLS=['so2', 'no2', 'rspm', 'spm', 'pm2_5']
```

```
In [13]: from sklearn.preprocessing import Imputer
      # invoking SimplerInmputer to fill missing values
      imputer = Imputer(missing_values=np.nan, strategy='mean')
      df [COlS] = imputer.fit_transform(df[COLS])
      df.head(5)
```



Out[13]:

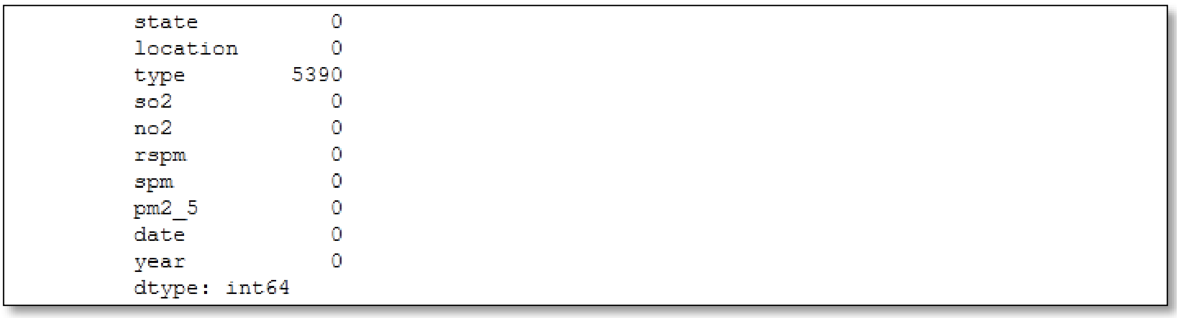
	state	location	type	so2	no2	rspm	spm	pm2_5	date	year
0	Andhra Pradesh	Hyderabad	Residential, Rural and other Areas	4.8	17.4	108.833091	220.78348	40.791467	1990-02-01	1990
1	Andhra Pradesh	Hyderabad	Industrial Area	3.1	7.0	108.833091	220.78348	40.791467	1990-02-01	1990
2	Andhra Pradesh	Hyderabad	Residential, Rural and other Areas	6.2	28.5	108.833091	220.78348	40.791467	1990-02-01	1990
3	Andhra Pradesh	Hyderabad	Residential, Rural and other Areas	6.3	14.7	108.833091	220.78348	40.791467	1990-03-01	1990
4	Andhra Pradesh	Hyderabad	Industrial Area	4.7	7.5	108.833091	220.78348	40.791467	1990-03-01	1990



```
In [14]: df.info()

         <class 'pandas.core.frame.DataFrame'>
         Int64Index: 435735 entries, 0 to 435738
         Data columns (total 10 columns):
          #   Column    Non-Null Count   Dtype
         ---  ------    --------------   -----
          0   state     435735 non-null  object
          1   location  435735 non-null  object
          2   type      430345 non-null  object
          3   so2       435735 non-null  float64
          4   no2       435735 non-null  float64
          5   rspm      435735 non-null  float64
          6   spm       435735 non-null  float64
          7   pm2_5     435735 non-null  float64
          8   date      435735 non-null  datetime64[ns]
          9   year      435735 non-null  int64
         dtypes: datetime64[ns](1), float64(5), int64(1), object(3)
         memory usage: 31.6+ MB
```

ISBN 978-93-80788-93-7

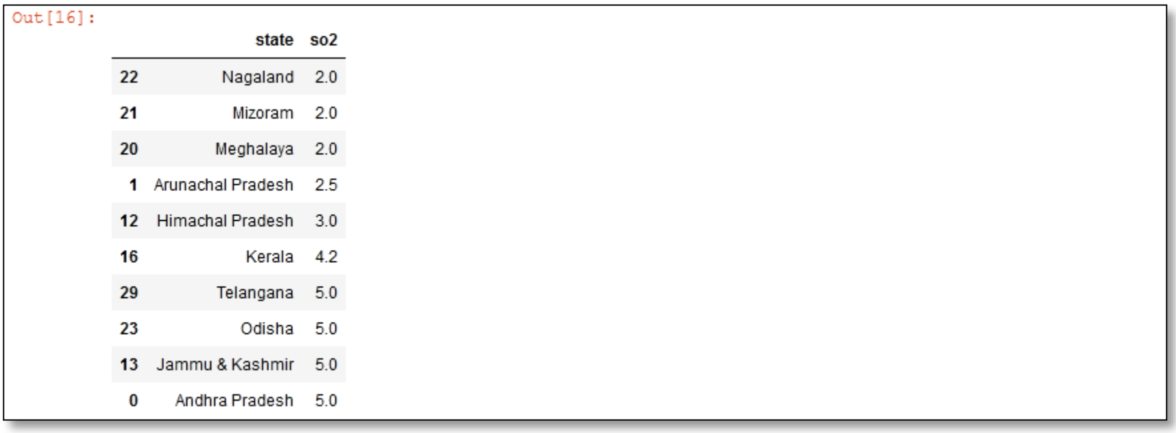
```
In [15]: # checking to see if the dataset has any null values left over
      and the format
      print (df.isnull().sum())
```



```
state          0
location       0
type        5390
so2            0
no2            0
rspm           0
spm            0
pm2_5          0
date           0
year           0
dtype: int64
```

so2 status:

```
In[16]:   statewise_so2   =   df[['so2','state']].groupby('state'   ,as_index=
      False).median().sort_values(by='so2')statewise_s02.head(10)
```

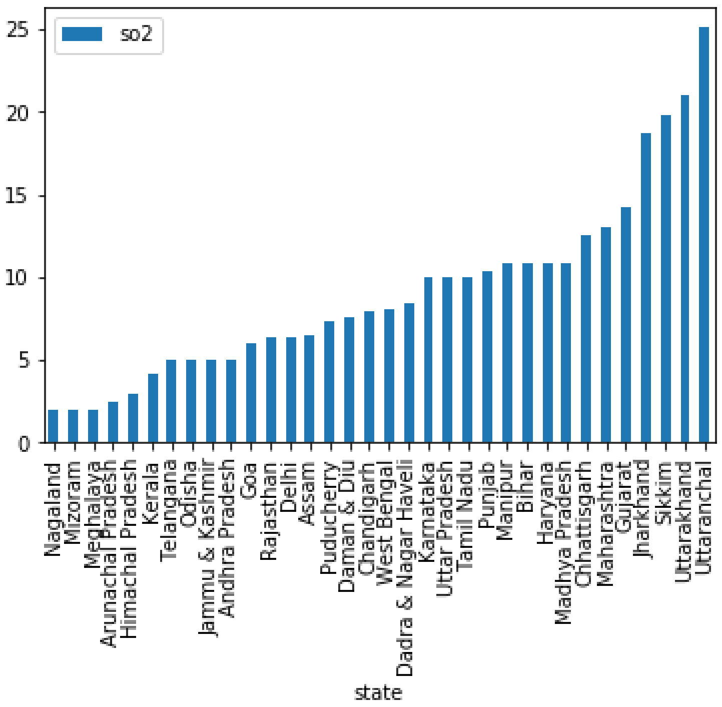


Out[16]:

	state	so2
22	Nagaland	2.0
21	Mizoram	2.0
20	Meghalaya	2.0
1	Arunachal Pradesh	2.5
12	Himachal Pradesh	3.0
16	Kerala	4.2
29	Telangana	5.0
23	Odisha	5.0
13	Jammu & Kashmir	5.0
0	Andhra Pradesh	5.0

A collection of estimates of past and future anthropogenic global so2 emissions. The cofala et al. estimates are for sensitivity studies on so2 emission policies. Those most at risk of developing problems if they are exposed to so2 are people with asthama or similar conditions.

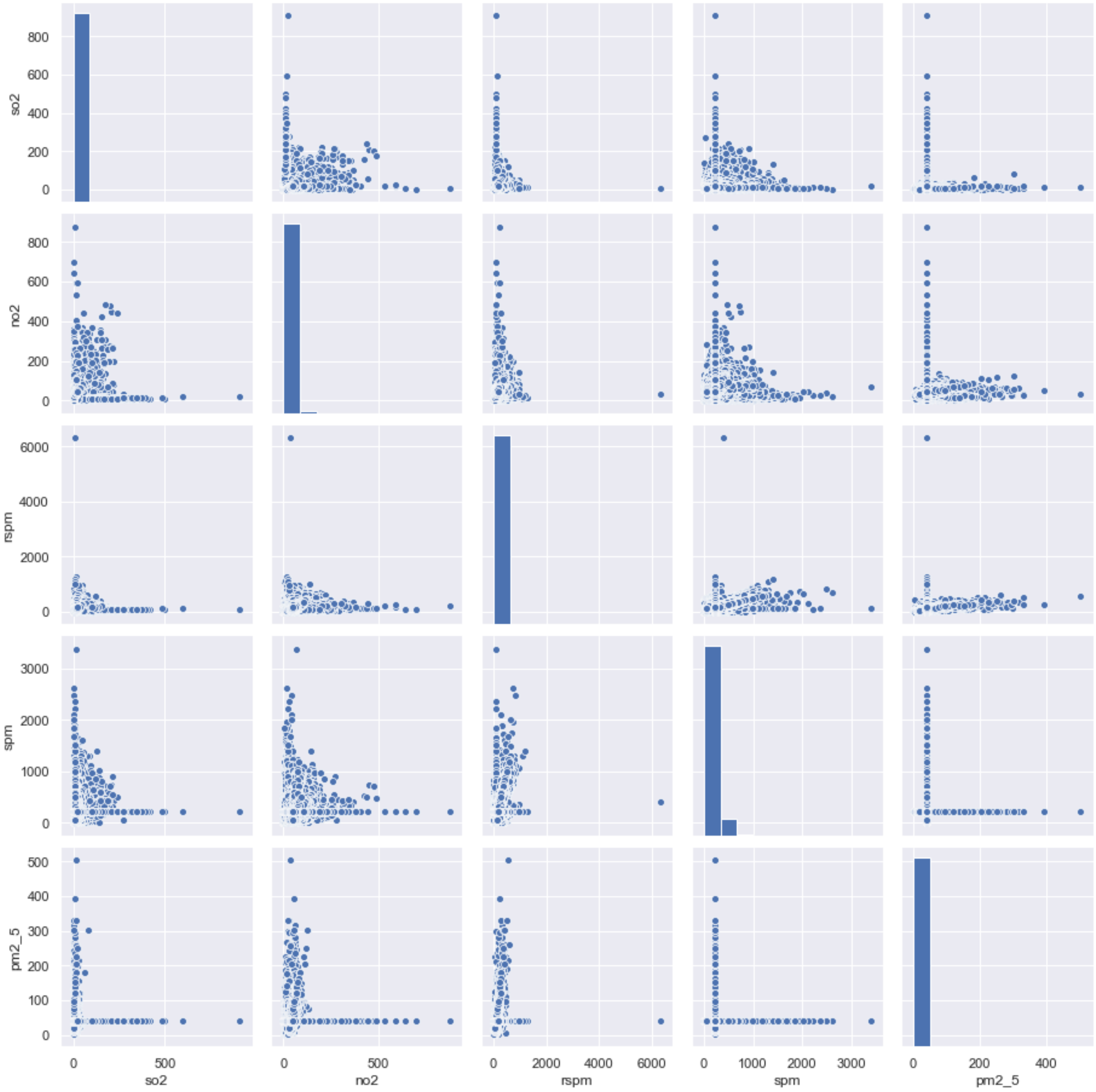
```
In [17]:#bar plot of so2 vs state
      Statewise_so2.plot(kind=’bar’,x=’state’,y=’so2’)
```



The plot shows the states with highest so2 levels in ascending order. We can see that Uttaranchal has the highest so2 concentration. Nagaland has the least concentrations of so2 among the states.

Scatter plots of all columns:

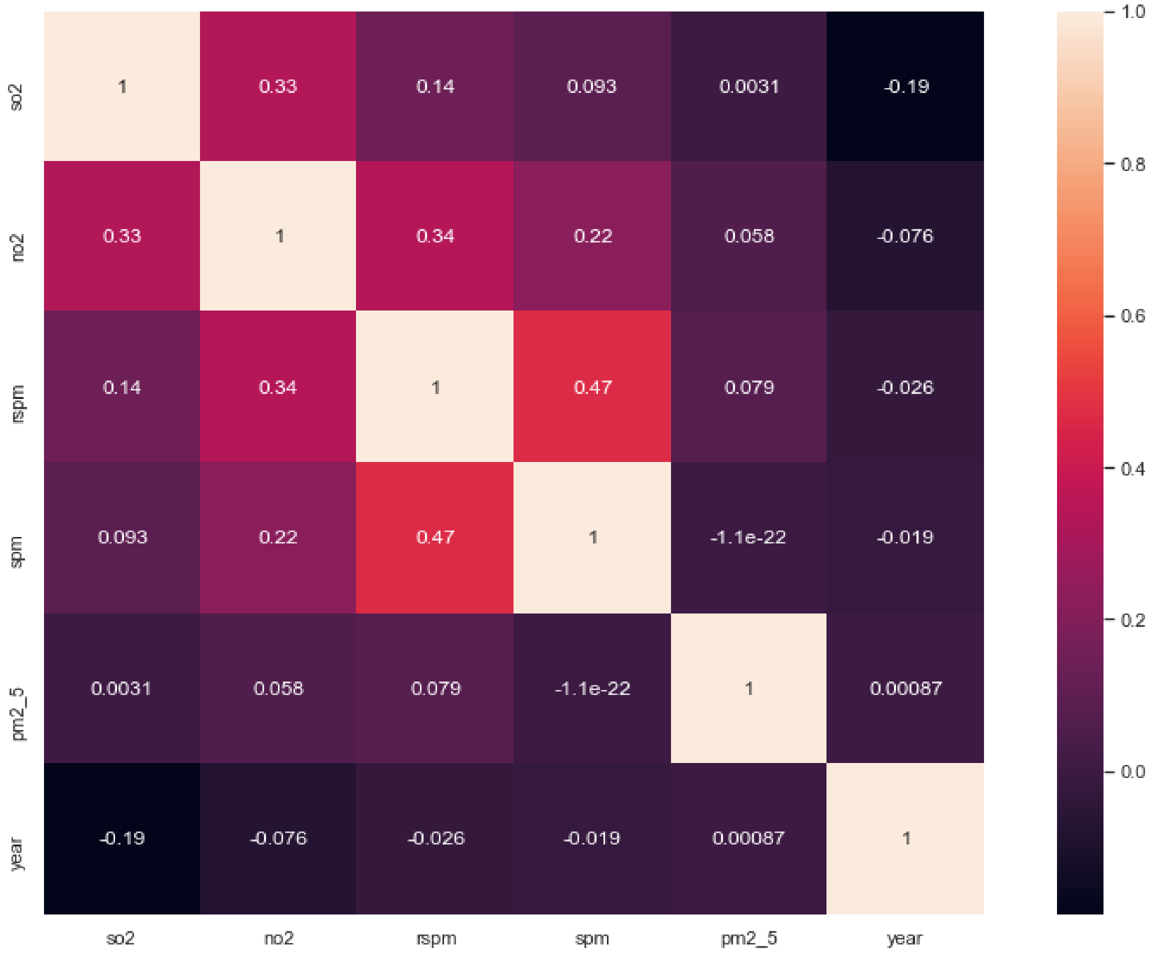
```
In [18]: sns.set()
      cols=['so2', 'no2', 'rspm', 'spm', 'pm2_5']
      sns.pairplot(df[cols], size = 2.5)
      plt.show()
```



The relationship between pm2_5 and any other feature is simply because pm2_5 has tons of null values. So2 and no2 values are highly concentrated near to the origin. spm and rspm share somewhat linear relationship, rest all features are not entirely related.

Correlation matrix:

```
In [19]: corrmat=df.corr()
         f, ax plt.subplots(figsize= (15,10))
         sns.heatmap(corrmat,vmax = 1,square =  True, annot = True)
```



It is clear from the correlation matrix that we have some correlation between spm and rspm, which supports our scatter plot analysis.

Performing the Simple Linear Regression:

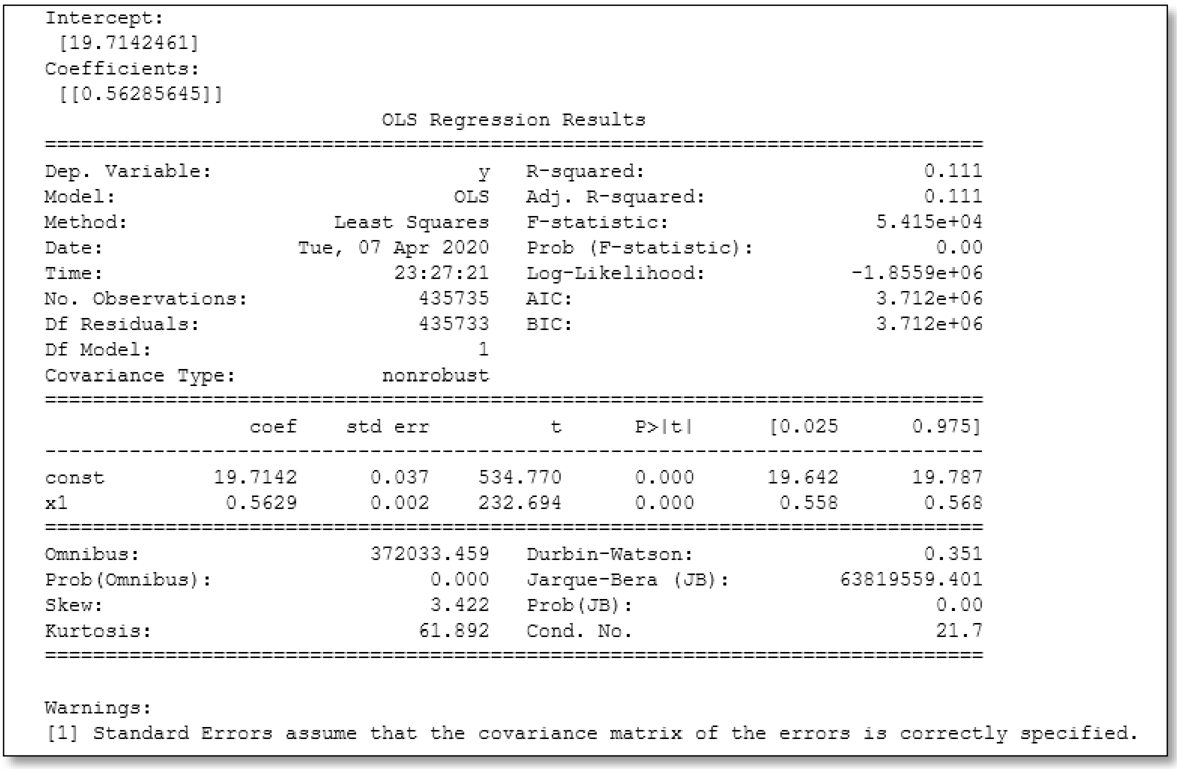
By applying linear regression we can take multiple X's and predict the corresponding Y values.

```
In [20]: X = np.array (df['so2']).reshape(–1, 1)
      y = np.array (df['n02']).reshape(–1, 1)
      df.dropna(inplace = True)
      regr = LinearRegression()
      regt.fit(X,y)
      print(regr.score(X, y))
      0.11053015931253296
```

```
In [21]: print('Intercept: \n' regr.intercept_)
      print('Coefficients: \n', regr.coef_)
      x = sm.add_constant (X) # adding a constant

      model = sm.OLS(y, X).fit()
      predictions = model . predict(X)
```

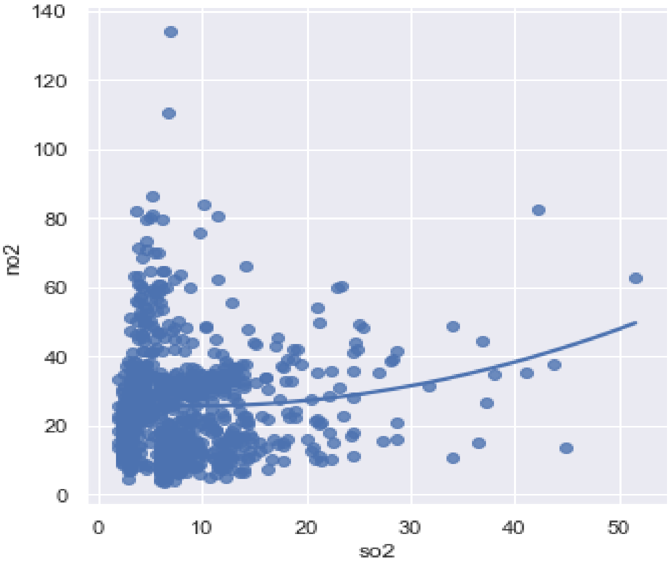
ISBN 978-93-80788-93-7

```
    print_model = model.summary()
    print(print_model)
```



```
Intercept:
 [19.7142461]
Coefficients:
 [[0.56285645]]
                            OLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:                       0.111
Model:                            OLS   Adj. R-squared:                  0.111
Method:                 Least Squares   F-statistic:                 5.415e+04
Date:                Tue, 07 Apr 2020   Prob (F-statistic):               0.00
Time:                        23:27:21   Log-Likelihood:            -1.8559e+06
No. Observations:              435735   AIC:                         3.712e+06
Df Residuals:                  435733   BIC:                         3.712e+06
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const         19.7142      0.037    534.770      0.000      19.642      19.787
x1             0.5629      0.002    232.694      0.000       0.558       0.568
==============================================================================
Omnibus:                   372033.459   Durbin-Watson:                   0.351
Prob(Omnibus):                  0.000   Jarque-Bera (JB):         63819559.401
Skew:                           3.422   Prob(JB):                         0.00
Kurtosis:                      61.892   Cond. No.                         21.7
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

```
In[22] : # Selecting the 1st 1000 row of the data
        df1000 = df[:] [:1000]
        sns.lmplot(x = "so2", y ="no2",data = df1000, order = 2, ci = None)
```



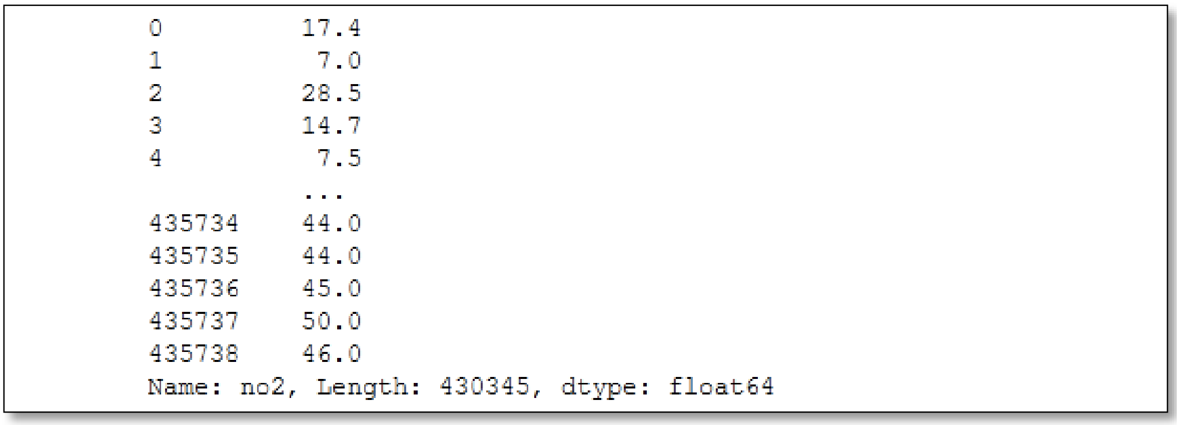
In the graph, the dots are the true data and the line is linear model. The plot shows that the data is highly positively skewed.

ISBN 978-93-80788-93-7

Normality check using Shapiro-Wilk test:

```
In [28]: import pandas as pd
         from scipy. stats import shapirc
```

```
In [29]: x=(df['no2'])
         print(x)
```



```
0         17.4
1          7.0
2         28.5
3         14.7
4          7.5
          ...
435734    44.0
435735    44.0
435736    45.0
435737    50.0
435738    46.0
Name: no2, Length: 430345, dtype: float64
```

```
In [30]: stat, p=shapiro(x)
```

```
In [31]: print(stat)
       0.7809625864028931
```

```
In [32] print(p)
      0.0
```

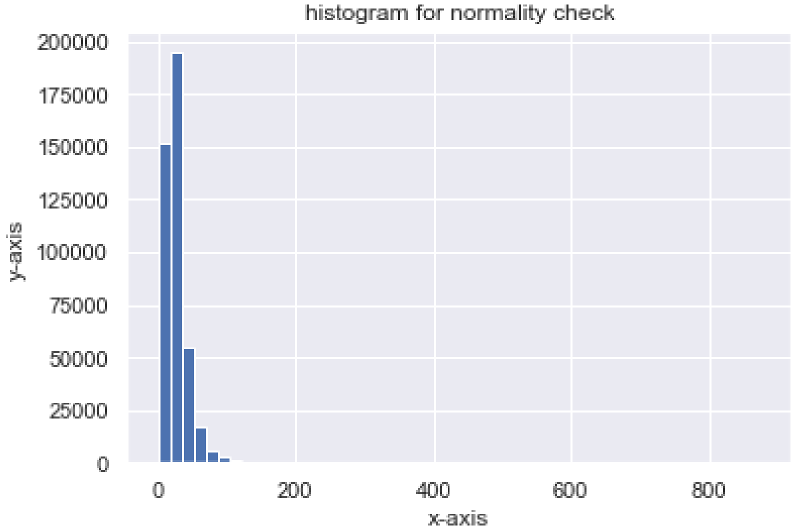
```
In [33]: alpha=0.05
       if p>alpha:
             print('sample looks Guass±an (fail to reject H0)')
       else:
            print('sample does not look Guass±an (reject H0)')

   sample does not lock Guassian (retject HO)
```

The p-value returned is less than 0.05 and finds that the data is not likely drawn from a Gaussian distribution (normal distribution).

Visualization of data using Histogram Plot:

```
In [34]: from numpy. random import randn
      import matplotlib.pyplot as plt
```

```
In [35]: plt.title("histogram for normality check")
      plt.xlanel('x–axis')
      plt.ylabel'(y–axis')
      plt.hist(x,50)
      plt.show()
```

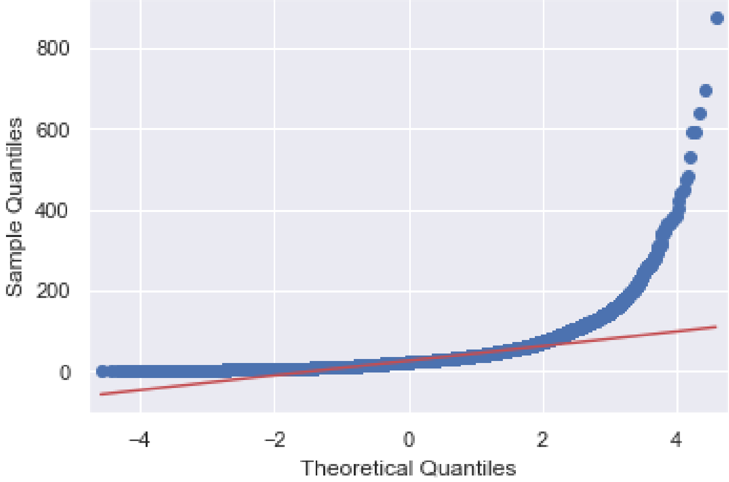


The histogram plot shows the number of observations in each bin. We cannot see a Gaussian-like shape to the data.

Visualization of data using QQ plot(Quantile-Quantile Plot):

```
In [36]: from statsmodels.graphics.gofplots import qqplot
```

```
In [37]: qqplot(x,line='s')
          plt.show()
```



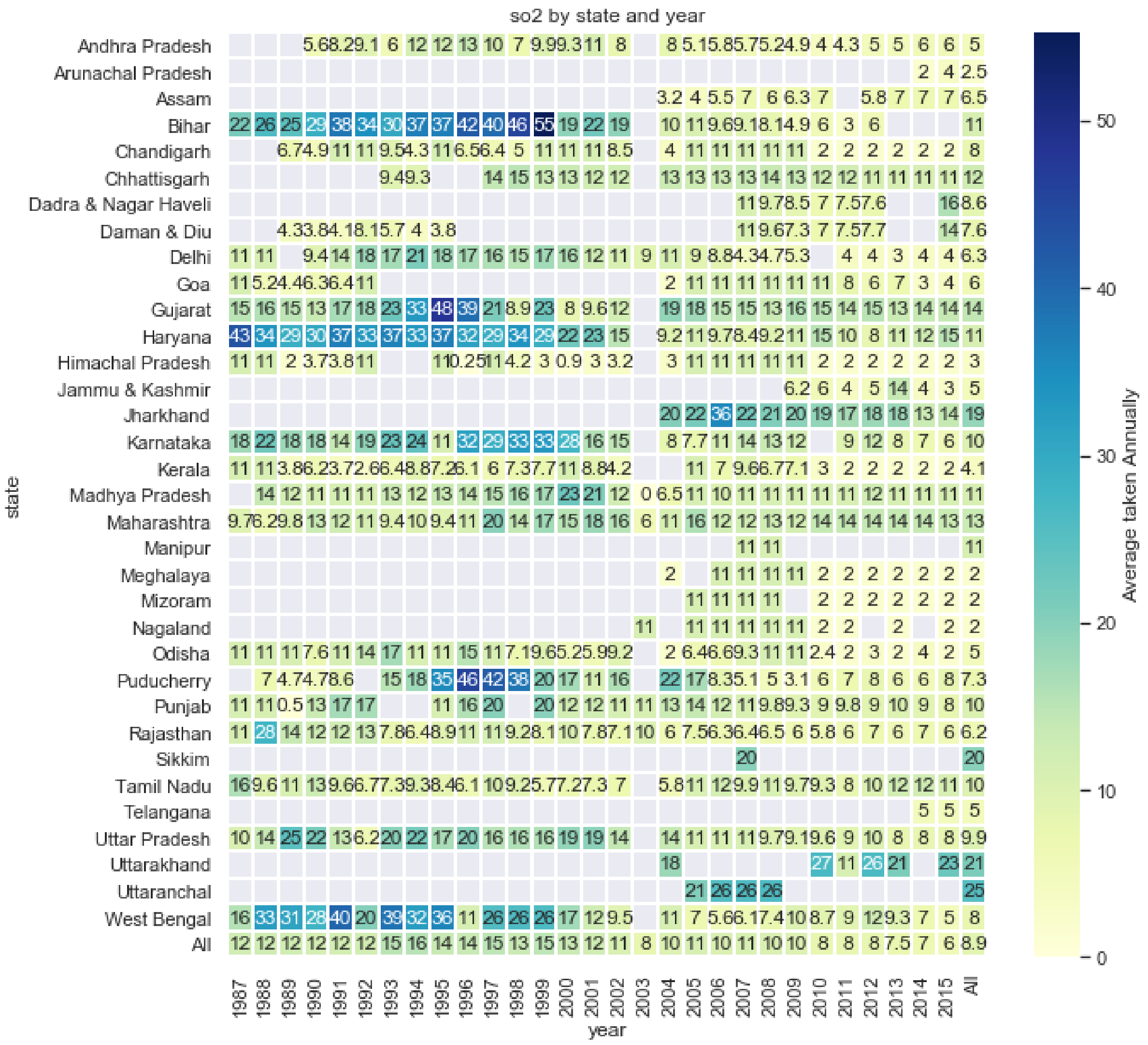
The QQ plot shows the scatter plot of points is not in a diagonal line, closely fitting the pattern for a sample from a positively skewed distribution.

Heatmap Pivot with state as Row, year as Col, no2 as value:

```
In [23]: f, ax = plt.subplots(figsize = (10,10))
      ax.set_title('{} by state and year'.fomat('so2'))
      sns.heatmap(df.pivot_table(so2', index = 'state',
            columns = ['year'], aggfunc = 'median', margins=True),
```

ISBN 978-93-80788-93-7

```
            annot = True, cmap = 'YIGnBu', linewidths = 1, ax = ax,
            cbar_kws = {'label': 'Average taken Annually'))
```

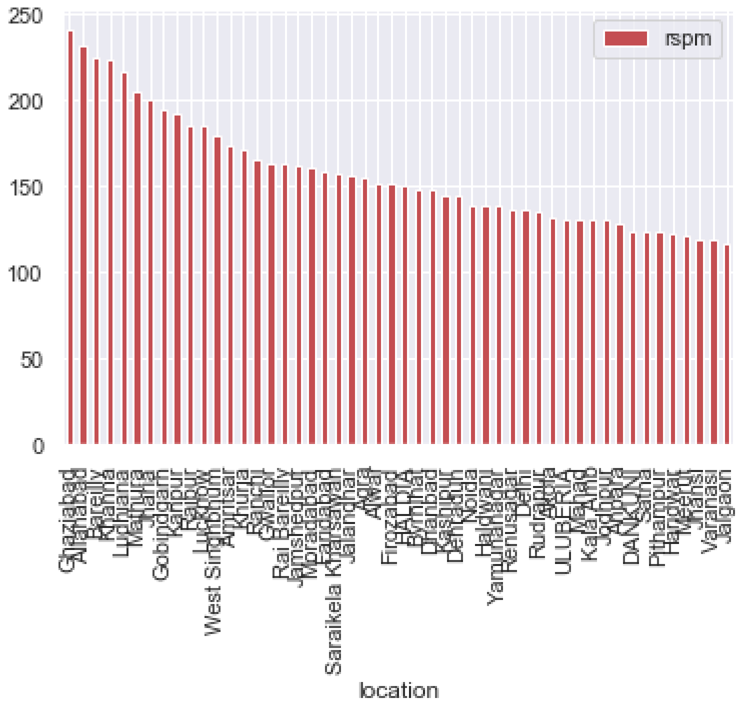


There has been a gradual increase of so2 concentration in Bihar from 1987 to 1999. The presence of so2has been high from 1980 to 2000 in some states but has decreased in the new country (from 2000).

rspm = PM10 - location wise - first 50:

```
In[24]:     df[['rspm',     'location']].groupby(['location'])     .median().
      sort_values("rspm", ascending = False).head(50).plot.bar(color = 'r')
```

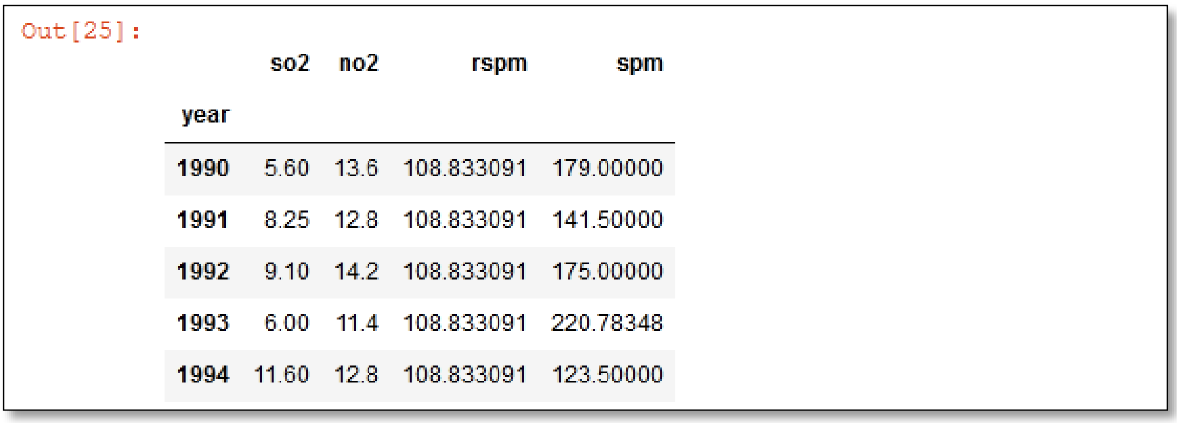
ISBN 978-93-80788-93-7



What is the yearly trend in a particular state, say ‘Andhra Pradesh’?

We have created a new dataframe containing the NO2, SO2, rspm, and spm data regarding state ‘Andhra Pradesh’ only and group it by ‘year’.

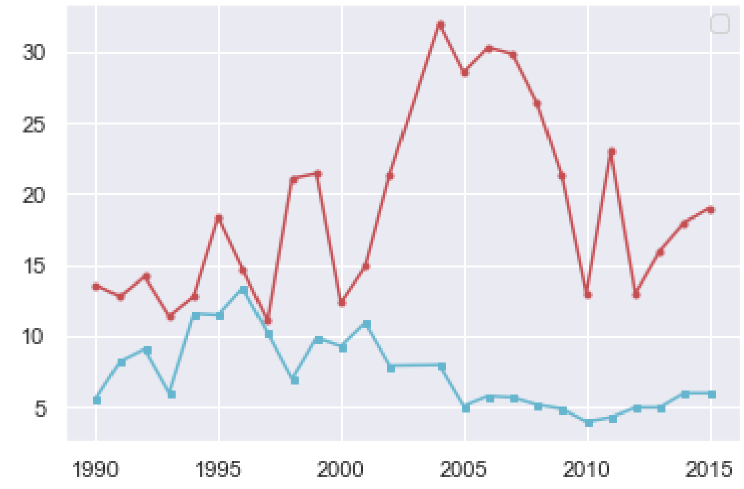
```
In [25]: andhra=df['state']=='Andhra Pradesh']
      year_wise_AP=andhra[['so2','no2','rspm','spm','year']].groupby
      ('year') .median()
      year_wise_ÄP.head()
```



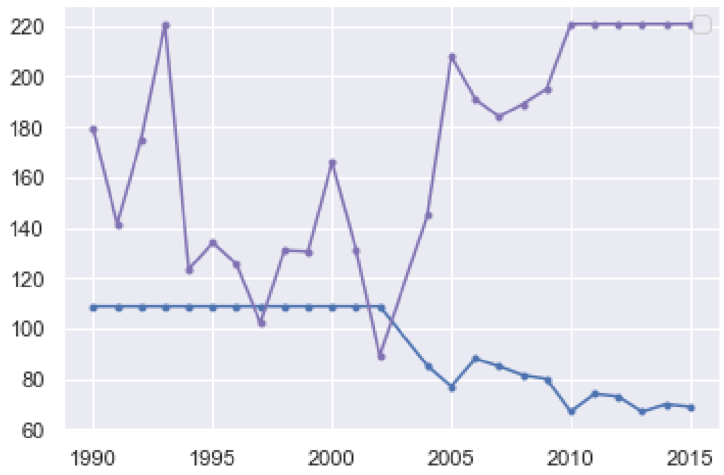
Out[25]:

	so2	no2	rspm	spm
year				
1990	5.60	13.6	108.833091	179.00000
1991	8.25	12.8	108.833091	141.50000
1992	9.10	14.2	108.833091	175.00000
1993	6.00	11.4	108.833091	220.78348
1994	11.60	12.8	108.833091	123.50000

```
In [26]: plt.plot(year_wise_AP['so2],'-sc',markersize = 3)
         plt.plot(year_wise_AP['so2],'-or', markersize = 3)
         plt.legend()
```



```
In [27]: plt.plot(year_wise_AP['rspm],'-ob',markersize = 3)
         plt.plot(year_wise_AP['spm],'-om', markersize = 3)
         plt.legend()
```



This gave an alarming signal that the value spm in Andhra Pradesh is hiking. It's 220 µg/m3 for the past 6 years (2010–2015).

Result:

Based on the various data visualization and analysis done it can be evaluated that most of the pollutants vary within acceptable limits and do not have significant correlation amongst other pollutants. However PM10 value is quite high for most of the months. As these readings are for the Kadubeesanahalli sensor the reason for this excess PM10 level might be due to high vehicular density in the vicinity. Along with that, other factors such as wind speed, road repairs, building construction that result in high dust particles might also have some impact on the variation in pollutant levels.

18.5 Conclusion

Based on data study and research carried out for the air pollutant dataset collected by PAQS sensor device it can be inferred that the pollutant levels of some harmful particulate matters such as PM10 are quite high in the air. One of the key sources for this seems to be the high vehicle density in the location where this data was collected. As the pollutants are mostly varying in a constant manner throughout the day the approximate pollutant levels can be predicted for the next day for a particular temperature and relative humidity factor. Also based on the observation it can be predicted that the pollutant levels vary based on the vehicle density i.e. during peak hours it is quite high. Also, air pollutants are influenced by the seasonal changes. In monsoon, the pollutant levels are low when compared to winter and spring seasons where it is high and again in summer the pollutant levels go down.

18.6 References

1. https://www.kaggle.com/sharmamanali/air-quality-index-analysis-ml-visualisation
2. https://towardsdatascience.com/india-air-pollution-data-analysis-bd7dbfe93841
3. Training Programme notes conducted by Pravesh Tiwari

Chapter 19
Analyzing Happiness Development Index using Python

Mr. Satvik Tandon, Mr. Alpesh Rathod and Mr. Gaurav Jadhav
DBT- Star College Status Scheme Researchers, Department of Statistics

19.1 Abstract

One of the most important macroeconomics indicator is Growth Development Product. However, recent breakthroughs in economics have proved that while GDP might be a great indicator of economic growth, it doesn't actually indicate whether a country has observed real growth or not. Since growth of a nation is not simply about economic growth but also physical, mental and cultural growth, and aims to maximize happiness for its citizens; Happiness Development index has been observed to be a better indicator.

Happiness Development Index considers multiple variables to decide whether people in a nation are Happy or not. These variables are Family, Economics, Freedom, Government Trust, and Generosity of people and Health (Life Expectancy).

The data I have is of HDI for years 2015, 2016 and 2017. The data consists of 6 indicators. I have done analysis on the date using Python Software. We have analyze the data to find the best country in each category, find the country with biggest gain, mean changes, correlation between various factors and created a model to predict Happiness Score and find on what factors does Happiness Score depend on. In this report I am going to share my findings and give interpretations. Furthermore, I have added variable Ranks for each year to analyze how many countries people are happy or not.

19.2 Understanding the Data

The data is for Happiness Index for years 2015, 2016 and 2017 for 158, 157 and 155 countries respectively. The variables in this dataset are:

1. **Country** – The country for which the values belong.
2. **Region** – The region to which the country it belongs to.
3. **Happiness Score** – The happiness Score the country had.
4. **Economy (GDP per Capita)** – Economy score of the country.
5. **Family** – The Family score of the country.
6. **Health (Life Expectancy)** – The health score of the country.

ISBN 978-93-80788-93-7

7. **Freedom** – How free do the people of the country feel.
8. **Trust (Government Corruption)** – The amount of trust people have in the government in the country.
9. **Generosity** – How much people believe other citizens in their country are generous in nature.

We have also added the variable 'Ranks' to the dataset. Ranks is a categorical variable telling whether people of a country are happy, okay or sad according to the happiness score.

19.3 Objectives

The aim of our study is to achieve the following goals:

1. To find the best country in various categories.
2. To find the country with the biggest gain and biggest decrease in happiness scores.
3. Add a variable indicating whether people in a country are happy or not.
4. To find how many countries are happy, okay and sad.
5. To test whether proportion of happy, okay and sad countries has changed or not.
6. To find correlation between various factors.
7. To make a model with Happiness Score has dependent variable and various independent variables.

19.4 Methods

We utilized statistical techniques like linear regression modeling, measures of central tendency, graphical illustrations, statistical tests like chi square and z proportion test, etc. We created our own model using ordinary least square and tested by dividing our data into 2 parts. Python concepts like modeling, statistical testing, graphical representations, refactoring, creating and calling functions, etc.

19.5 Analysis

Importing libraries:

We are using **"Fifteen.csv"**, **"sixteen.csv"** and **"seventeen.csv"**.

We began by importing the libraries we are going to need.

```
In[1]: # Importing important -libraries
      import os                                 # Helps to change directory
      import pandas as pd      # Used for creating and analyzing dataframes
      import numpy as np                  # Used for numerical calculations
      import matplotlib.pyplot as plt                   # Used for plotting
      import seaborn as sns                             # Used for plotting
      from scipy.stats import binom_test  # Used for Z Test for proportions
      from sklearn import linear_model         # Used for modelling purpose
      import statsmodels.api as sm             # Used for modelling purpose
```

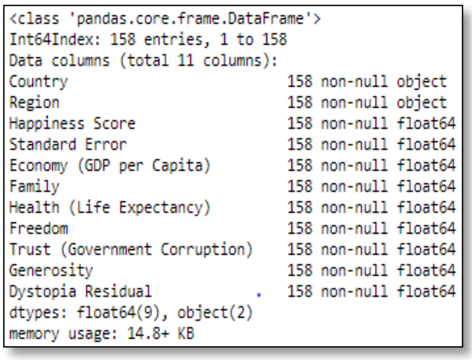
Importing Dataset:

Then we are going to load the datafiles. Since we have ranks given for year 2015 and 2016 in columns 2, we will declare those columns as index columns for respective datasets.

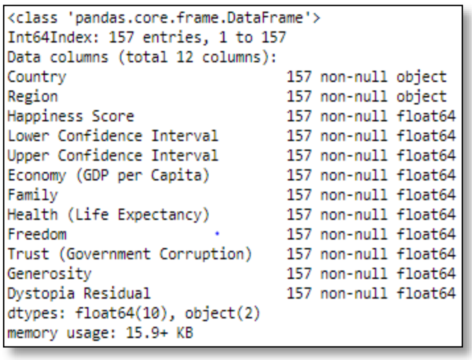
```
In [2]: os.chdir("C:/Users/Ädmin/Desktop")
 y2015 = pd.read_csv("C:/Users/Admin/Desktcp/Fifteen.csv", index col =2)
 y2016 = pd.read_csv("C:/Users/Admin/Desktop/3ixteen.csv", index col =2)
 y2017 = pd.read_csv("C:/Users/Admin/Desktop/seventeen.csv")
```

Using info () to find brief summary of the data. We will start by seeing a brief overview of our data using formula dataset.info().

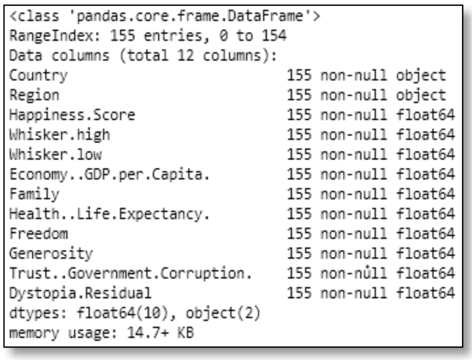
```
In [3]: # Find info on Data
      y2015.info()
      y2016.info()
      y2017.info()
```



```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 158 entries, 1 to 158
Data columns (total 11 columns):
Country                          158 non-null object
Region                           158 non-null object
Happiness Score                  158 non-null float64
Standard Error                   158 non-null float64
Economy (GDP per Capita)         158 non-null float64
Family                           158 non-null float64
Health (Life Expectancy)         158 non-null float64
Freedom                          158 non-null float64
Trust (Government Corruption)    158 non-null float64
Generosity                       158 non-null float64
Dystopia Residual                158 non-null float64
dtypes: float64(9), object(2)
memory usage: 14.8+ KB
```



```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 157 entries, 1 to 157
Data columns (total 12 columns):
Country                          157 non-null object
Region                           157 non-null object
Happiness Score                  157 non-null float64
Lower Confidence Interval        157 non-null float64
Upper Confidence Interval        157 non-null float64
Economy (GDP per Capita)         157 non-null float64
Family                           157 non-null float64
Health (Life Expectancy)         157 non-null float64
Freedom                          157 non-null float64
Trust (Government Corruption)    157 non-null float64
Generosity                       157 non-null float64
Dystopia Residual                157 non-null float64
dtypes: float64(10), object(2)
memory usage: 15.9+ KB
```

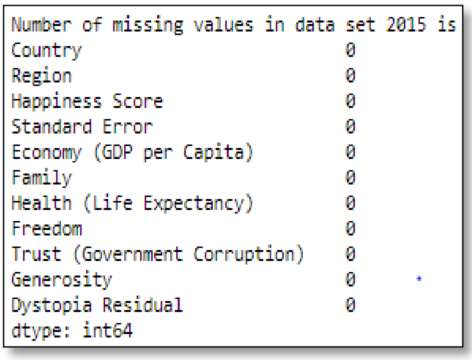


```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 155 entries, 0 to 154
Data columns (total 12 columns):
Country                          155 non-null object
Region                           155 non-null object
Happiness.Score                  155 non-null float64
Whisker.high                     155 non-null float64
Whisker.low                      155 non-null float64
Economy..GDP.per.Capita.         155 non-null float64
Family                           155 non-null float64
Health..Life.Expectancy.         155 non-null float64
Freedom                          155 non-null float64
Generosity                       155 non-null float64
Trust..Government.Corruption.    155 non-null float64
Dystopia.Residual                155 non-null float64
dtypes: float64(10), object(2)
memory usage: 14.7+ KB
```

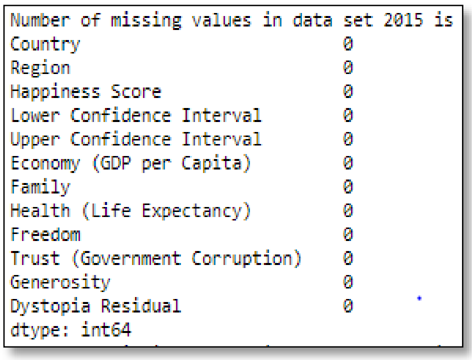
Checking whether the dataset has any null values:

```
In [3] #To Check number of null values
      print('Number of missing values in data set 2015 is \n{}'.format
      (str(y2015.isnull ().sum()))
      print('Number of missing values in data set 2015 is \n{}'.format
      (str(y2016.isnull ().sum()))
      print('Number of missing values in data set 2015 is \n{}'.format
      (str(y2017.isnull ().sum()))
```

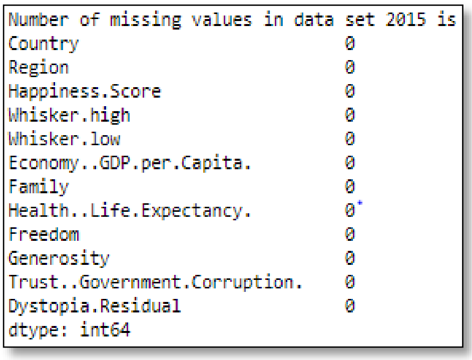
ISBN 978-93-80788-93-7



```
Number of missing values in data set 2015 is
Country                           0
Region                            0
Happiness Score                   0
Standard Error                    0
Economy (GDP per Capita)          0
Family                            0
Health (Life Expectancy)          0
Freedom                           0
Trust (Government Corruption)     0
Generosity                        0
Dystopia Residual                 0
dtype: int64
```



```
Number of missing values in data set 2015 is
Country                          0
Region                           0
Happiness Score                  0
Lower Confidence Interval        0
Upper Confidence Interval        0
Economy (GDP per Capita)         0
Family                           0
Health (Life Expectancy)         0
Freedom                          0
Trust (Government Corruption)    0
Generosity                       0
Dystopia Residual                0
dtype: int64
```



```
Number of missing values in data set 2015 is
Country                           0
Region                            0
Happiness.Score                   0
Whisker.high                      0
Whisker.low                       0
Economy..GDP.per.Capita.          0
Family                            0
Health..Life.Expectancy.          0
Freedom                           0
Generosity                        0
Trust..Government.Corruption.     0
Dystopia.Residual                 0
dtype: int64
```

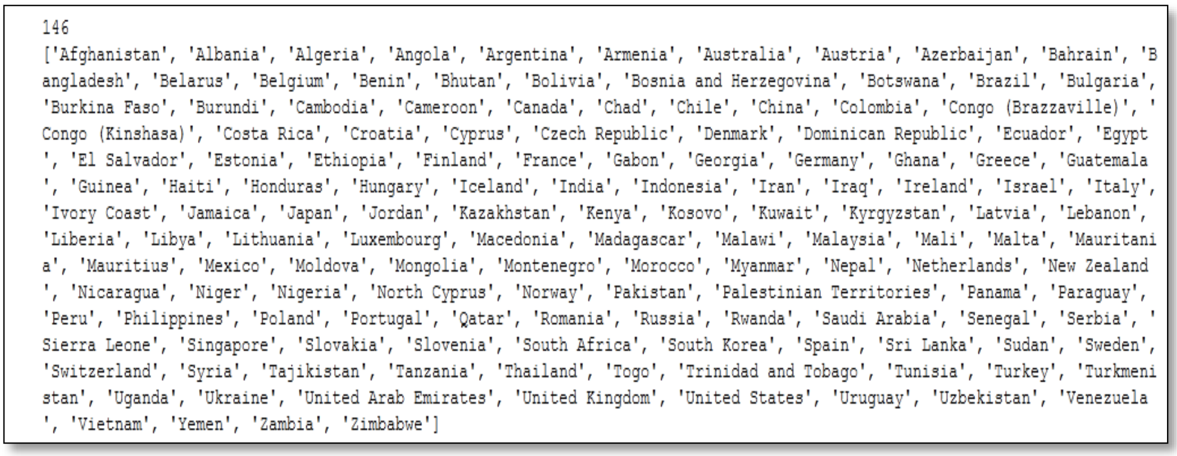
None of the dataset has any null values.

Countries similar in all three dataset:

However, each data set has unequal number of countries in it. We will find out those countries which are available in all data set.

```
In [5]: # ==================================================================
#The below code check vhether a country which exists in 2015 also exists in
#2016 and 2017
#The above code thus checks whether a country in all three database
# ==========================================================================
      same_countries = 0
                    #same countries will store number of similiar countries
      same_countries_list = [ ]
                 # same_countries_list contains names of similiar countries
      for x in np.unique(y2015['Country']) :
            if x in np.unique(y2016['Country']):
                  if x in np.unigue(y2017['Country']) :
                        same_countries = same_countries+l
                        same_countries list.append (x)
 print (same_countries)
 print (same_countries_list
```

ISBN 978-93-80788-93-7



```
146
['Afghanistan', 'Albania', 'Algeria', 'Angola', 'Argentina', 'Armenia', 'Australia', 'Austria', 'Azerbaijan', 'Bahrain', 'B
angladesh', 'Belarus', 'Belgium', 'Benin', 'Bhutan', 'Bolivia', 'Bosnia and Herzegovina', 'Botswana', 'Brazil', 'Bulgaria',
'Burkina Faso', 'Burundi', 'Cambodia', 'Cameroon', 'Canada', 'Chad', 'Chile', 'China', 'Colombia', 'Congo (Brazzaville)', '
Congo (Kinshasa)', 'Costa Rica', 'Croatia', 'Cyprus', 'Czech Republic', 'Denmark', 'Dominican Republic', 'Ecuador', 'Egypt
', 'El Salvador', 'Estonia', 'Ethiopia', 'Finland', 'France', 'Gabon', 'Georgia', 'Germany', 'Ghana', 'Greece', 'Guatemala
', 'Guinea', 'Haiti', 'Honduras', 'Hungary', 'Iceland', 'India', 'Indonesia', 'Iran', 'Iraq', 'Ireland', 'Israel', 'Italy',
'Ivory Coast', 'Jamaica', 'Japan', 'Jordan', 'Kazakhstan', 'Kenya', 'Kosovo', 'Kuwait', 'Kyrgyzstan', 'Latvia', 'Lebanon',
'Liberia', 'Libya', 'Lithuania', 'Luxembourg', 'Macedonia', 'Madagascar', 'Malawi', 'Malaysia', 'Mali', 'Malta', 'Mauritani
a', 'Mauritius', 'Mexico', 'Moldova', 'Mongolia', 'Montenegro', 'Morocco', 'Myanmar', 'Nepal', 'Netherlands', 'New Zealand
', 'Nicaragua', 'Niger', 'Nigeria', 'North Cyprus', 'Norway', 'Pakistan', 'Palestinian Territories', 'Panama', 'Paraguay',
'Peru', 'Philippines', 'Poland', 'Portugal', 'Qatar', 'Romania', 'Russia', 'Rwanda', 'Saudi Arabia', 'Senegal', 'Serbia', '
Sierra Leone', 'Singapore', 'Slovakia', 'Slovenia', 'South Africa', 'South Korea', 'Spain', 'Sri Lanka', 'Sudan', 'Sweden',
'Switzerland', 'Syria', 'Tajikistan', 'Tanzania', 'Thailand', 'Togo', 'Trinidad and Tobago', 'Tunisia', 'Turkey', 'Turkmeni
stan', 'Uganda', 'Ukraine', 'United Arab Emirates', 'United Kingdom', 'United States', 'Uruguay', 'Uzbekistan', 'Venezuela
', 'Vietnam', 'Yemen', 'Zambia', 'Zimbabwe']
```

There are 146 countries similar in all dataset. The list of those countries is given above. Because of uneven number of countries and the data belonging to three different years, I will not combine the dataset into one for analysis but rather keep it separately while conducting analysis. Without knowing how the countries affect the dataset and analysis, it wouldn't be right to discard the data points. So, to maintain fair analysis the dataset is kept separated.

Best Country in Each Category:

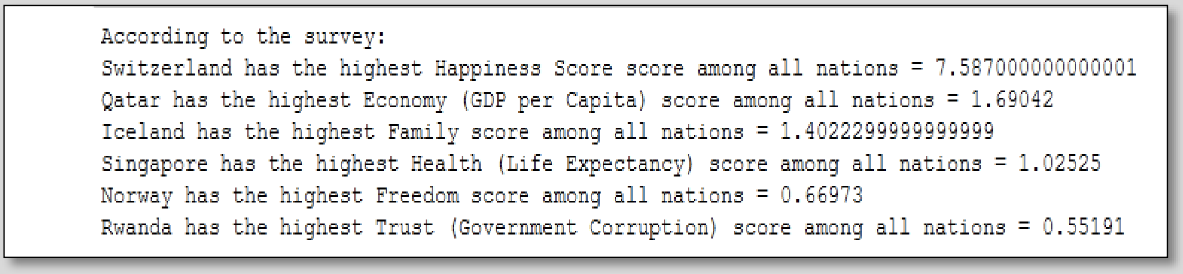
We will now begin with Exploratory Data Analysis and focus on finding best countries in various categories for all 3 years.

```
In [6]: def best_survey(df):
      to_test = to_test = range(2,8)
      print('According to the survey:')
      for y in to test:
# This for loop changes the variable for which we are finding the best
            for x in range(0,len(df)):
#This for Loop changes the country for which we are testing the condition
                  if df.iloc[x,y] == max(df.iloc[:,y])
#This checks whether this country has the highest value for y variable
                        print('{} has the highest {} score among all
                        nations ={}'.format(df.iloc[x,0],df.columns[y]
                        ,str(df..iloc[x,y])))
# This prints the variable and the country and the value
```

The above function will take the three datasets as input and then print the best country in various categories for each year. By creating a function not only does the memory required decrease but it also makes executing the same set of codes for three different countries much easier. This also allows us to add new datasets without having to perform cumbersome code additions. We will call the function using command best survey (DataFrame).

Results for 2015 survey

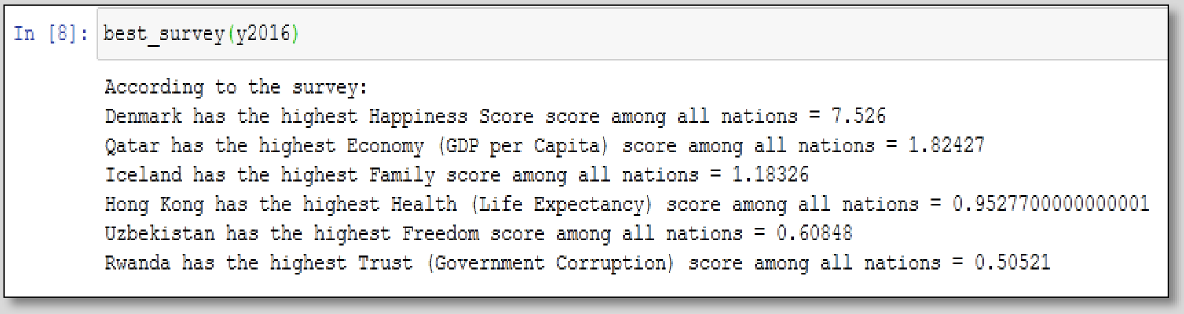
```
In [7]: best_survey(y2015)
```



```
According to the survey:
Switzerland has the highest Happiness Score score among all nations = 7.587000000000001
Qatar has the highest Economy (GDP per Capita) score among all nations = 1.69042
Iceland has the highest Family score among all nations = 1.4022299999999999
Singapore has the highest Health (Life Expectancy) score among all nations = 1.02525
Norway has the highest Freedom score among all nations = 0.66973
Rwanda has the highest Trust (Government Corruption) score among all nations = 0.55191
```

Results for 2016 survey

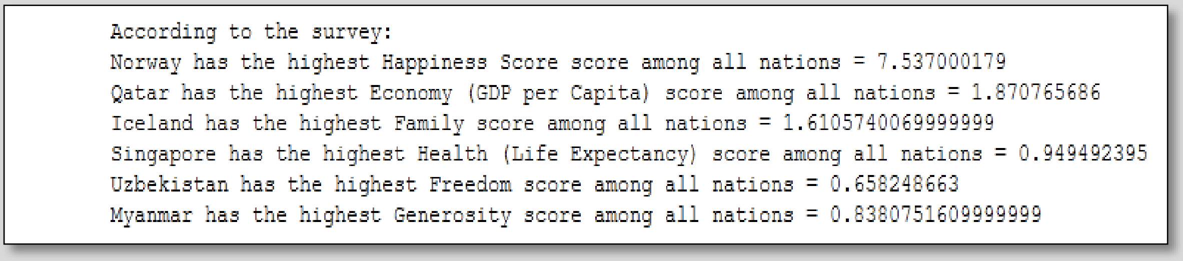
```
In [8]: best_survey(y2016)
```



```
In [8]: best_survey(y2016)

        According to the survey:
        Denmark has the highest Happiness Score score among all nations = 7.526
        Qatar has the highest Economy (GDP per Capita) score among all nations = 1.82427
        Iceland has the highest Family score among all nations = 1.18326
        Hong Kong has the highest Health (Life Expectancy) score among all nations = 0.9527700000000001
        Uzbekistan has the highest Freedom score among all nations = 0.60848
        Rwanda has the highest Trust (Government Corruption) score among all nations = 0.50521
```

Results for 2017 survey

```
In [9]: best_survey(y2017)
```



```
According to the survey:
Norway has the highest Happiness Score score among all nations = 7.537000179
Qatar has the highest Economy (GDP per Capita) score among all nations = 1.870765686
Iceland has the highest Family score among all nations = 1.6105740069999999
Singapore has the highest Health (Life Expectancy) score among all nations = 0.949492395
Uzbekistan has the highest Freedom score among all nations = 0.658248663
Myanmar has the highest Generosity score among all nations = 0.8380751609999999
```

Finding the countries with the highest improvement and downfall:

The next goal was to find out which countries observed the most improvement and most downfalls for each set of years. The code I created for this was:

```
In [10]: def change_in_scores(dfl,df2):
      max_p_change_country =""
              # Stores the name of the country with maximum positive change
      max_p_change_score = 0
                                 # Stores the most positive change in score

      max_n_change_country = ""
              # Stores the name of the country Vith maximum positive change
      max_n_change_score = 0
                                 # Stores the most positive change in score

      change scores = []                            # Stores the change in score
      for x in range (0, len (dfl)):
            for y in range (0, len(df2)):
                  if dfl.iloc[x, 0] == df2.iloc[y,0]:
                                 # Checks the data is for the sane country
                        change_scores.append(df2.iloc[y,2] – dfl.iloc[x,2]
                                                  # Store the changed value
                  if (df2.iloc[y,2] - dfl.iloc[x, 2]) > max_p_change_score:
                        max_p_change_score =df2.iloc[y,2] - dfl.iloc[x, 2]
                    # This variable is to calculate the biggest improvement
                        max_p_change_country = df2.iloc[y,0]
                  elif (df2.iloc[y,2]–dfl.iloc[x, 2] < max_n_change score:
                        max_n_change_score = df2.iloc[y,2] -df2.iloc[x,2]
                       # This variable is to calculate the biggest downfall
                        max_n_change_country = df2.iloc[y,0]

      print ("{} saw the biggest improvement in score ={}".format(max
      p_change_country,max p_change_score))
      print ("{} saw the biggest downfall in score ={}". format (max
      n_change country,max n_change_score))
```

We will call the function using command change_in_scores(DataFrame1, DataFrame2).

Results for 2015 to 2016

```
In [11]:change_in_scores(y2015,y2016)
        Algeria saw the biggest improvement in score = 0.75
        Liberia saw the biggest downfall in score = -0.9490000000000007
```

Results for 2016 to 2017

```
In [12]:change_in_scores(y2016,y2017)
      Bulgaria saw the biggest improvement in score = 0.49700022500000074
      Venezuela saw the biggest downfall in score = -0.8339999999999996
```

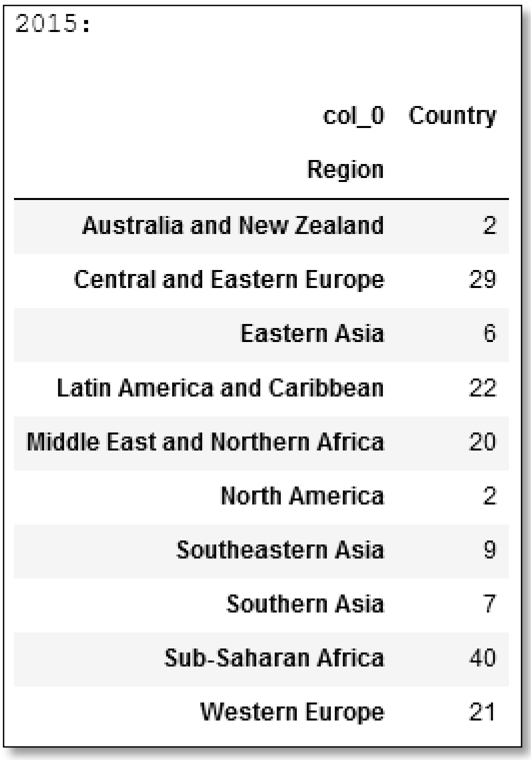
Results for 2015 to 2017

```
In [13]:change_in_scores (y2015, y2017)
      Latvia saw the biggest improvement in score = 0.7519999049999999
      Venezuela saw the biggest downfall in score = -1.5599999999999996
```

Distribution of Country in various Regions:

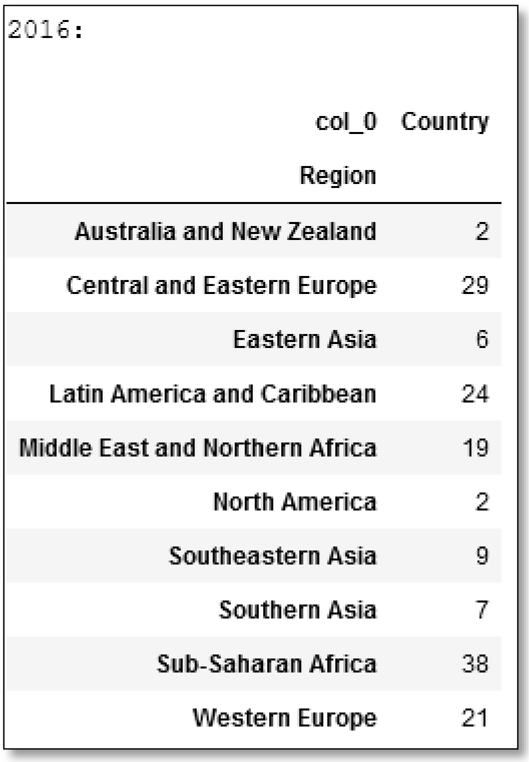
Finding out which regions do the country belongs to.

```
In [18]:print("2015:")
      pd.crosstab(y2015['Region'],columns = 'Country',dropna = True)

In [19]:print("2016:")
      pd.crosstab(y2016['Region'],columns = 'Country',dropna = True)

In [20]:print ("2017:")
      pd.crosstab(y2017 ['Region'],columns = 'Country',dropna = True)
```



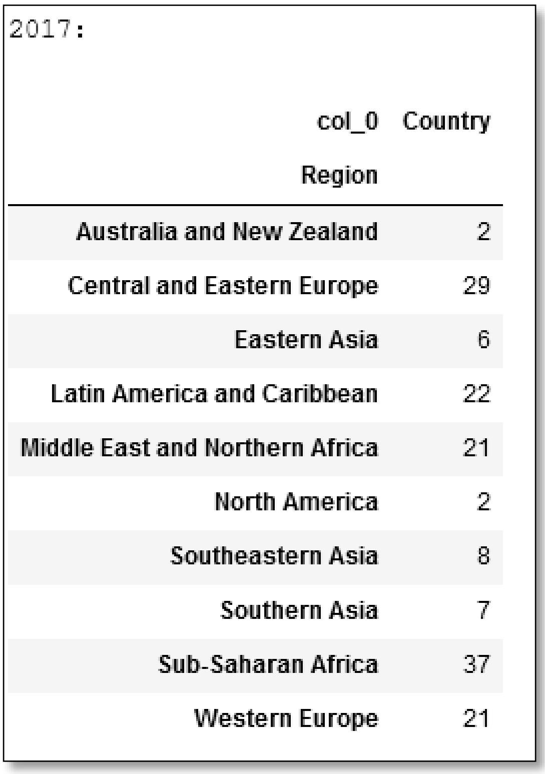
2015:

col_0	Country
Region	
Australia and New Zealand	2
Central and Eastern Europe	29
Eastern Asia	6
Latin America and Caribbean	22
Middle East and Northern Africa	20
North America	2
Southeastern Asia	9
Southern Asia	7
Sub-Saharan Africa	40
Western Europe	21



2016:

col_0	Country
Region	
Australia and New Zealand	2
Central and Eastern Europe	29
Eastern Asia	6
Latin America and Caribbean	24
Middle East and Northern Africa	19
North America	2
Southeastern Asia	9
Southern Asia	7
Sub-Saharan Africa	38
Western Europe	21



2017:

col_0	Country
Region	
Australia and New Zealand	2
Central and Eastern Europe	29
Eastern Asia	6
Latin America and Caribbean	22
Middle East and Northern Africa	21
North America	2
Southeastern Asia	8
Southern Asia	7
Sub-Saharan Africa	37
Western Europe	21

Categorizing the country according to citizen’s happiness:

We will divide the countries into different categories (ranks) according to happiness score. This will allow also performing better analysis and understanding which country’s citizens are happy, okay or sad. The code used for this is:

```
In [21]: def_dividing_category(df) :
# we will divide the scores into different categories and add those categories
into each database
 # More than 7 indicates Happy, between 7 and 5 indicates Okay and less than 5
indicates Sad

      cate= []                             #Stores the category for country
      for x in range(0,len(df)) :
            if df.iloc[x,2] >= 7:
                  cate.append('Happy')
            elif df.iloc[x,2] >= 5 and df.iloc[x,2]< 7:
                  cate.append('Okay')
            elif df.iloc[x,2] < 5:
                  cate.append(Sad)
      return(cate)
```

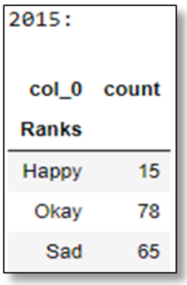
```
In [22]: y2015["Ranks"] = dividing_category(y2015)
         y2016["Ranks"] = dividing_category(y2016)
       y2017["Ranks"] = dividing_category(y2017)
```

Checking how many people are Happy, Okay and Sad for each year:
We will now see how many countries fall in each category.

```
In [23]: print("2015:")
      pd.crosstab(y2015['Ranks], columns = 'count')
```

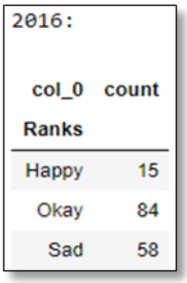
```
In [25]: print("2016:")
      pd.crosstab(y2016['Ranks], columns = 'count')
```

```
In [26]: print("2017:")
      pd.crosstab(y2017['Ranks], columns = 'count')
```



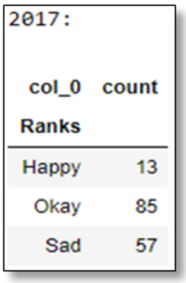
2015:

col_0	count
Ranks	
Happy	15
Okay	78
Sad	65



2016:

col_0	count
Ranks	
Happy	15
Okay	84
Sad	58



2017:

col_0	count
Ranks	
Happy	13
Okay	85
Sad	57

By looking at the results we can see that the number of happy, okay and sad nations have not changed over time. Its laregly been the same. We will check this later through Hypothesis testing.

Checking how many people are Happy, Okay and Sad in each Region for every year:

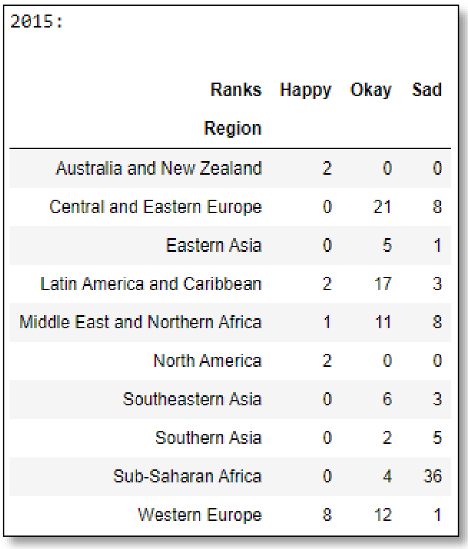
Next we will look at how many happy, okay and sad countries exist in each region. We will also graph this to better understand the results.

ISBN 978-93-80788-93-7

```
In [28]:print("2015:")
      pd.crosstab(y2015['Region'],y2015['Ranks']
```

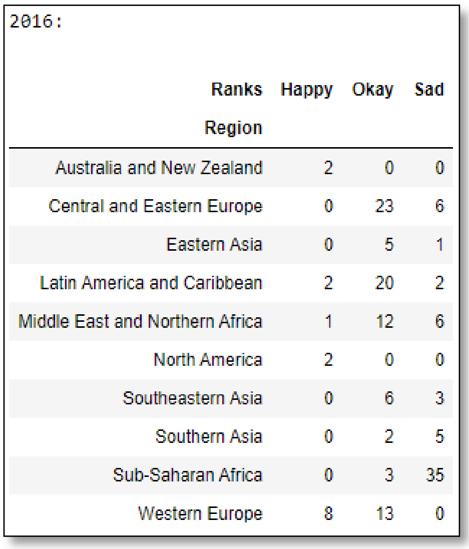
```
In [29]:print("2016:")
      pd.crosstab(y2016['Region'],y2016['Ranks']
```

```
In [30]:print("2017:")
      pd.crosstab(y2017['Region'],y2017['Ranks']
```



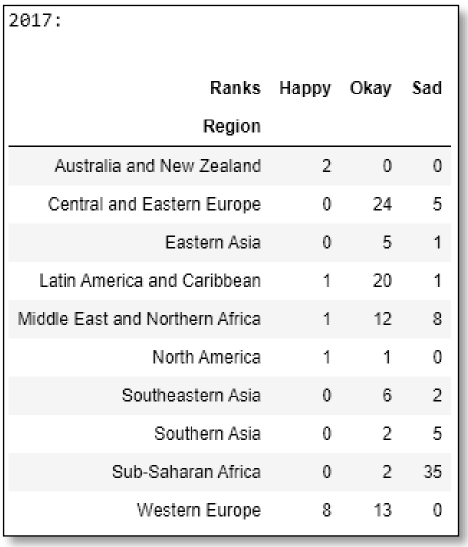
2015:

Ranks	Happy	Okay	Sad
Region			
Australia and New Zealand	2	0	0
Central and Eastern Europe	0	21	8
Eastern Asia	0	5	1
Latin America and Caribbean	2	17	3
Middle East and Northern Africa	1	11	8
North America	2	0	0
Southeastern Asia	0	6	3
Southern Asia	0	2	5
Sub-Saharan Africa	0	4	36
Western Europe	8	12	1



2016:

Ranks	Happy	Okay	Sad
Region			
Australia and New Zealand	2	0	0
Central and Eastern Europe	0	23	6
Eastern Asia	0	5	1
Latin America and Caribbean	2	20	2
Middle East and Northern Africa	1	12	6
North America	2	0	0
Southeastern Asia	0	6	3
Southern Asia	0	2	5
Sub-Saharan Africa	0	3	35
Western Europe	8	13	0



2017:

Ranks	Happy	Okay	Sad
Region			
Australia and New Zealand	2	0	0
Central and Eastern Europe	0	24	5
Eastern Asia	0	5	1
Latin America and Caribbean	1	20	1
Middle East and Northern Africa	1	12	8
North America	1	1	0
Southeastern Asia	0	6	2
Southern Asia	0	2	5
Sub-Saharan Africa	0	2	35
Western Europe	8	13	0

By looking at this data we can see that certain region has no happy countries and certain regions have no sad countries. This can be better understood through bar plot graphs.

Graphing the results:

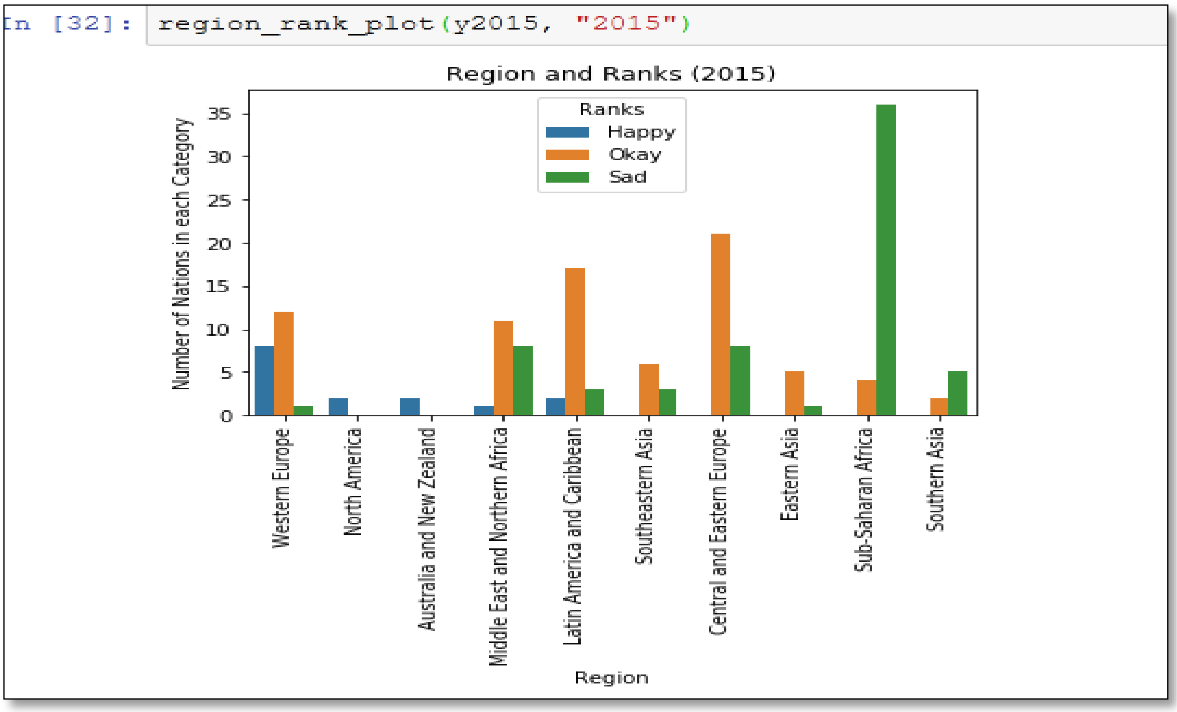
We will plot average happiness score for each region for all year. The codes used for this are:

```
In [31]: def region_rank_plot(df,year):
      sns.countplct( Region' , data = df, hue = 'Ranks')
```

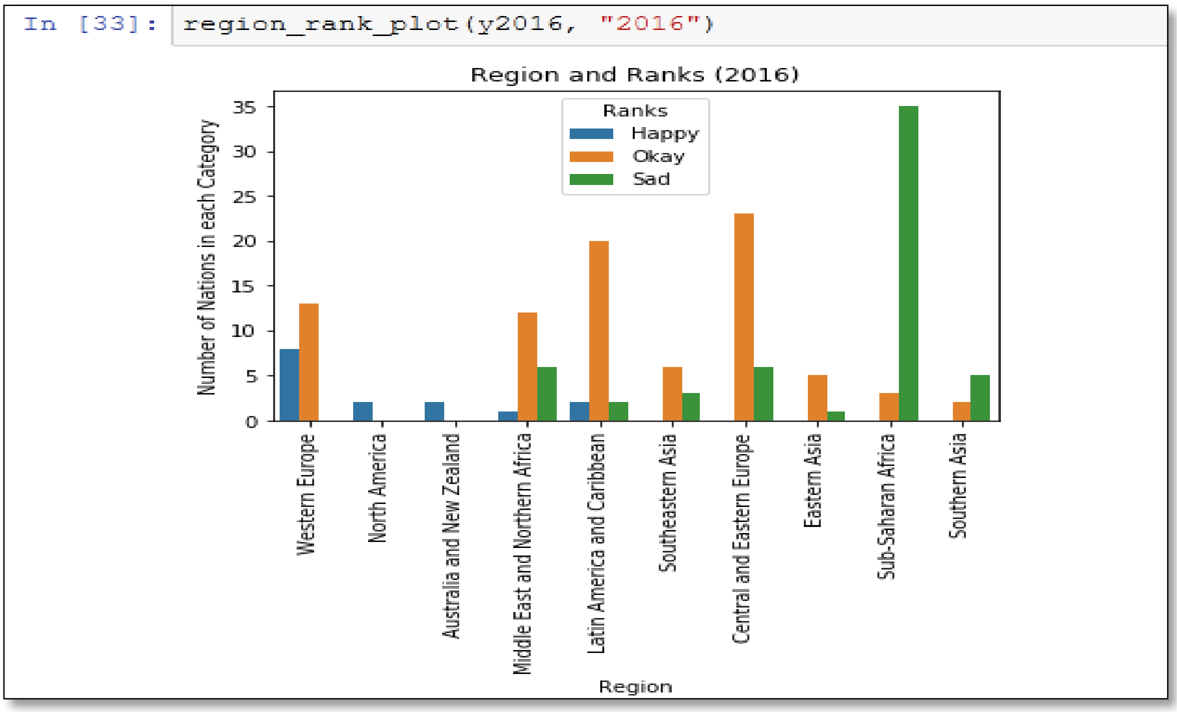
ISBN 978-93-80788-93-7

```
    plt.xticks(rotation = 90)
    plt.xlabel('Region')
    plt.ylabel('Number of Nations in each Category')
    plt.title("Region and Ranks ({})".format(year))
```

Calling the function using commands:

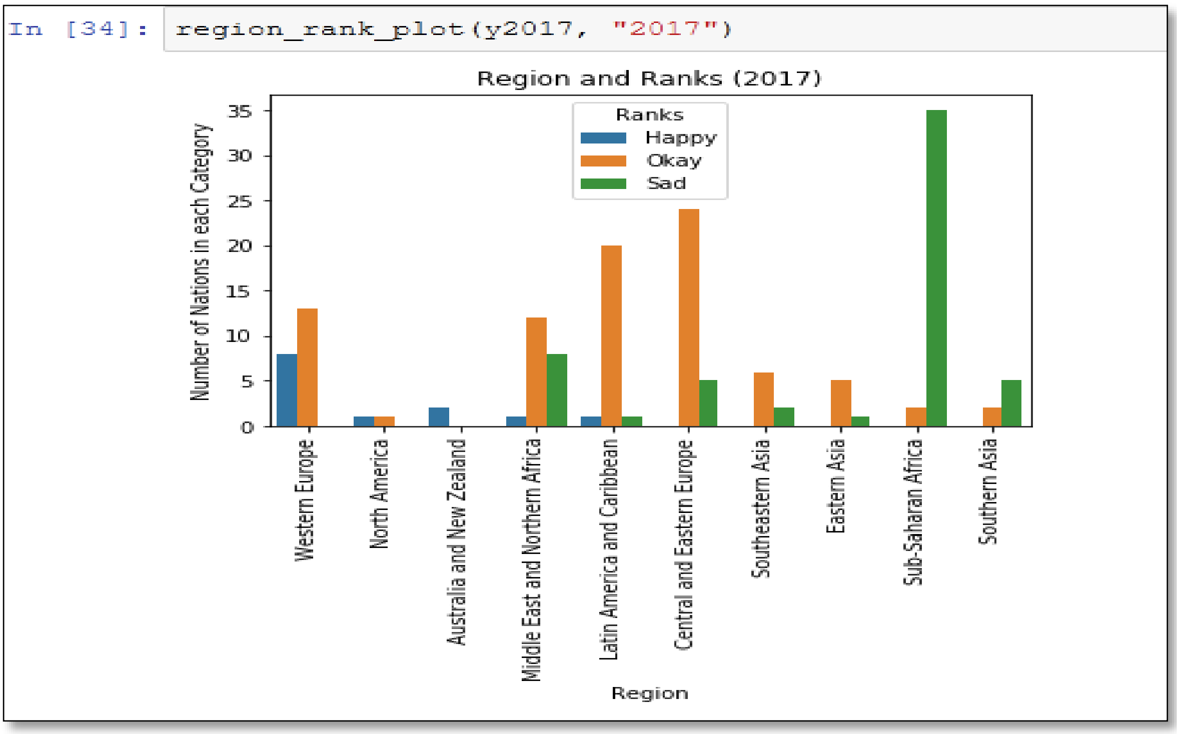


```
In [32]: region_rank_plot(y2015, "2015")
```



```
In [33]: region_rank_plot(y2016, "2016")
```

ISBN 978-93-80788-93-7



```
In [34]: region_rank_plot(y2017, "2017")
```

For 2015, 2016 and 2017, it is observed that happy countries are seen only in Western Europe, North America, Australia and New Zealand, Middle East and Latin America. All of which have either developed nation or nation with a lot of wealth. Most happy countries are in Western Europe. On the other hand, majority of sad countries are present in Sub-Sahara Africa. The countries in Sub-Sahara Africa are under developed countries and the area also prone to civil wars. This has caused these nations to be unhappy. The proportion of nation in this region is sad. Regions with developing countries like Central and Eastern Europe, Latin America and Western Europe sees most people being okay, i.e., neither happy nor sad. North America, Australia and New Zealand only have happy countries. But we must also remember that these regions have very few nations. For 2016 and 2017, Western Europe no longer has any sad countries in it.

Testing whether Proportion of Happy, Okay and Sad people have changed:

We will conduct a hypothesis test to understand whether the proportion of happy, sad and okay families has changed or not. Level of Significance is 1%. The code for this is:

```
In [35]: # Happy, Sad and Okay
        g15 = 78
        g16 = 65
        g17 = 13
        a15 = 84
        a16 = 58
        a17 = 13
        b15 = 85
        b16 = 57
        b17 = 13
```

```
In [36]: def proportion_test(pl,t,p2):
            # pl is the number of countries in year 2015
            # tl is the total number of countries in year 2015
            # p2 is the proportion of countries in year 2016
      pval = binom_test(p1,t,p2)
      alpha = 0.01
      if pval> alpha:
            print("We do not reject HO")
      else:
            print ("We reject HO")
```

2015 to 2016

H_{01}: Proportion of Happy Countries has remained unchanged, against
H_{11}: Proportion of Happy Countries has changed.
H_{02}: Proportion of Sad Countries has remained unchanged, against
H_{12}: Proportion of Sad Countries has changed.
H_{03}: Proportion of Okay Countries has remained unchanged, against
H_{13}: Proportion of Okay Countries has changed.

```
In [37]: proportion_test(g15, (g15+a15+b15), g16/(g16+b16+a16))
      we do not reject HO
```

```
In [38]: proportion_test(b15, (g15+a15+b15), g16/(g16+b16+a16))
      we do not reject HO
```

```
In [39]: proportion_test(a15, (g15+a15+b15), g16/(g16+b16+a16))
      we do not reject HO
```

Conclusion: We can thus conclude from the test that proportion of happy countries, sad countries and okay countries have not changed from 2015 to 2016.

2016 to 2017

H_{01}: Proportion of Happy Countries has remained unchanged, against
H_{11}: Proportion of Happy Countries has changed.
H_{02}: Proportion of Sad Countries has remained unchanged, against
H_{12}: Proportion of Sad Countries has changed.
H_{03}: Proportion of Okay Countries has remained unchanged, against
H_{13}: Proportion of Okay Countries has changed.

```
In [40]: proportion_test(g16, (g16+a16+b16), g17/(g17+b17+a17))
      we do not reject HO
```

```
In [41]: proportion_test(b16, (g16+a16+b16), b17/(g17+b17+a17))
      we do not reject HO
```

```
In [42]: proportion_test(a16, (g16+a16+b16), a17/(g17+b17+a17))
```

```
      we do not reject HO
```

Conclusion: We can thus conclude from the test that proportion of happy countries, sad countries and okay countries have not changed from 2016 to 2017.

2015 to 2017

H_{01}: Proportion of Happy Countries has remained unchanged, against
H_{11}: Proportion of Happy Countries has changed.
H_{02}: Proportion of Sad Countries has remained unchanged, against
H_{12}: Proportion of Sad Countries has changed.
H_{03}: Proportion of Okay Countries has remained unchanged, against
H_{13}: Proportion of Okay Countries has changed.

```
 In [43]: proportion_test(g15, (g15+a15+b15), g17/(g17+b17+a17))
      we do not reject HO
```

```
In [44]: proportion_test(b15, (g15+a15+b15), b17/(g17+b17+a17))
      we do not reject HO
```

```
In [45]: proportion_test(a15, (g15+a15+b15), a17/(g17+b17+a17))
      we do not reject HO
```

Conclusion: We can thus conclude from the test that proportion of happy countries, sad countries and okay countries have not changed from 2015 to 2017.

A combined dataset was created. This was done using the formula

```
In [46]: combined_dataset = y2015.copy()
         combined_dataset =combined_dataset.append(y2016, sort = False)
         combined_dataset =combined_dataset.append(y2017, sort = False)
```

Testing whether Happiness is dependent on the Region in which the Country belongs to:

Since countries in the same region have political, geographically and cultural similarity, so logically speaking happiness of country should depend the region in which the country belongs. Country in the same regions should have similar amount of happiness. We will check this using Chi Square test.

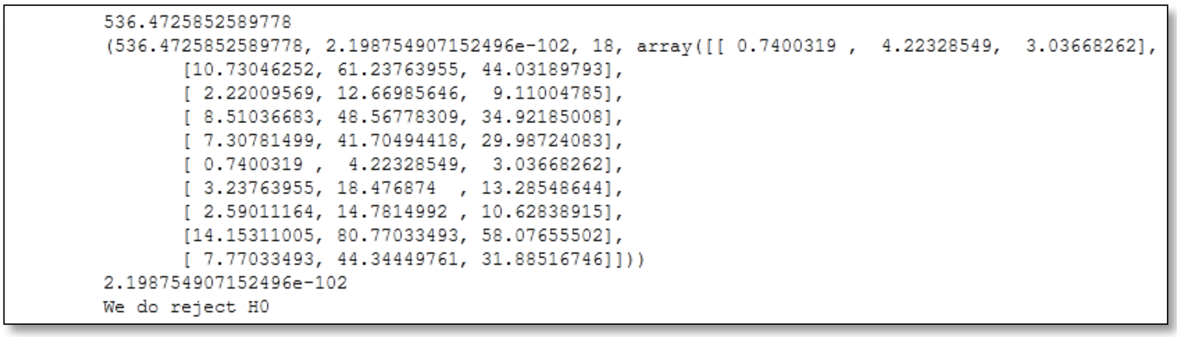
H_0: Happiness of people in a country is independent of the Region the country belongs to.
H_1: Happiness of people in a country is not independent of the Region the country belongs to.

Level of significance: 1%

Decision Criteria: If p- value is less than level of signifiance or chi square calculated is more than chi square tabulated, we will reject H_0.

```
In [53]:def chi _ ind_test (df,alpha) :

            from scipy import stats

            contingency_table = pd.crosstab(df ["Region"],df["Ranks"])
            observed values = contingency_table.values
            observed values

            chisq_output = stats.chi2_contingency(contingency table)
            chisq_output

            expected_values = chisq_output [3]
            chi_squared_stat = (((observed_values–expected_values)**2)/
      expected_values).sum().sum()

            print(chi_squared–stat)
            print(chisq_output)
            print(chisq_output[1])

            if chisq_output[1]>alpha:
                  print("We do not reject HO")
            else :
                  print("We do reject HO")
```

```
In[54]:chi_ind_test(combined_dataset,alpha = 0.01)
```



```
536.4725852589778
(536.4725852589778, 2.198754907152496e-102, 18, array([[ 0.7400319 ,  4.22328549,  3.03668262],
       [10.73046252, 61.23763955, 44.03189793],
       [ 2.22009569, 12.66985646,  9.11004785],
       [ 8.51036683, 48.56778309, 34.92185008],
       [ 7.30781499, 41.70494418, 29.98724083],
       [ 0.7400319 ,  4.22328549,  3.03668262],
       [ 3.23763955, 18.476874  , 13.28548644],
       [ 2.59011164, 14.7814992 , 10.62838915],
       [14.15311005, 80.77033493, 58.07655502],
       [ 7.77033493, 44.34449761, 31.88516746]]))
2.198754907152496e-102
We do reject H0
```

Conclusion: We can thus conclude that happiness of people in a country is dependent on the Region the country belongs to.

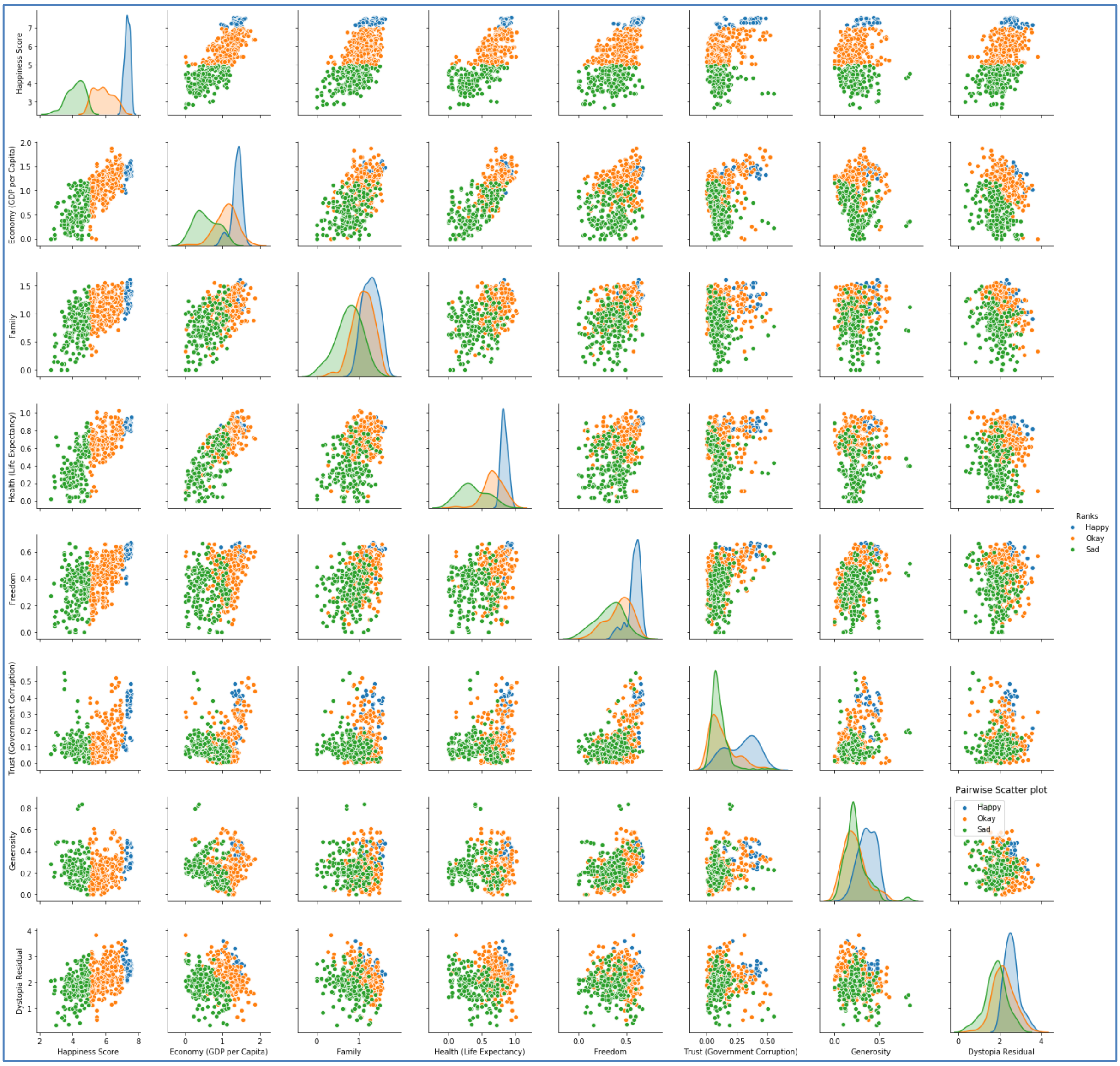
Limitations: Since some of the expected frequencies is less than 5, we cannot fully trust the chi square results.

Correlation Plotting:

We will check the correlation between the models through pairwise scatter plot and correlation plot of the combined dataset using matlibplot and seaborn libraries.
Plotting Scatter Plot of different Factors

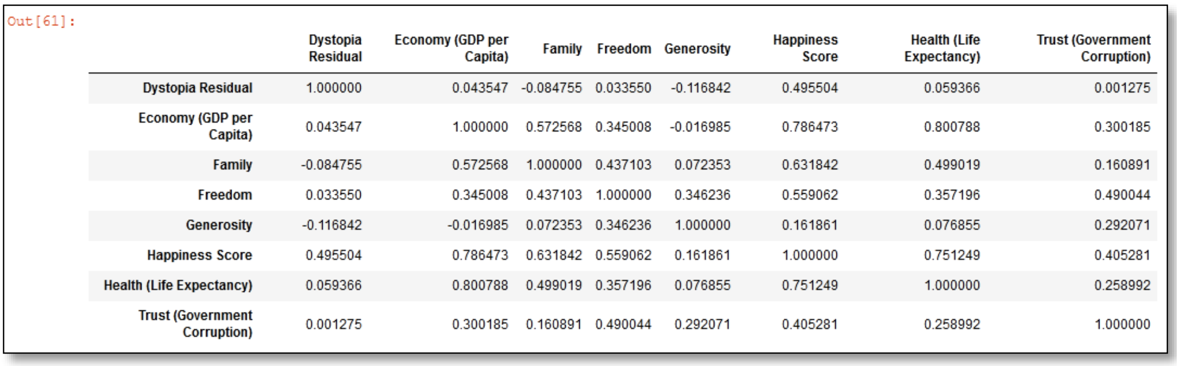
```
In [60]: sns.pairplot(combined_dataset, kind = 'scatter', hue = "Ranks")
      plt.xlabel("Scores")
```

```
    plt.ylabel("Scores")
    plt.title("pairwise Scatter plot")
    plt.legend(loc='upper left')
```



Correlation between the different factors is

```
In [61]: corr_plot = combined_dataset.corr()
         corr_plot
```

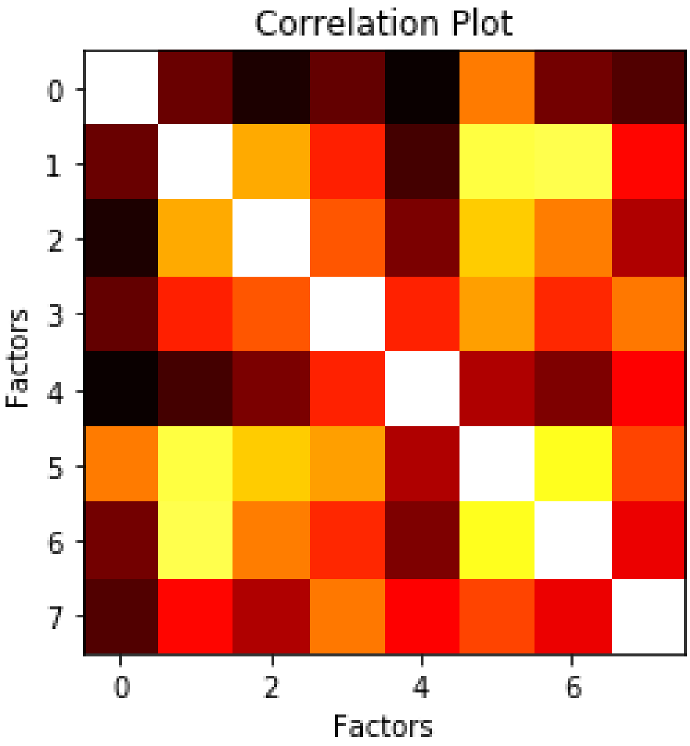

Out[61]:

	Dystopia Residual	Economy (GDP per Capita)	Family	Freedom	Generosity	Happiness Score	Health (Life Expectancy)	Trust (Government Corruption)
Dystopia Residual	1.000000	0.043547	-0.084755	0.033550	-0.116842	0.495504	0.059366	0.001275
Economy (GDP per Capita)	0.043547	1.000000	0.572568	0.345008	-0.016985	0.786473	0.800788	0.300185
Family	-0.084755	0.572568	1.000000	0.437103	0.072353	0.631842	0.499019	0.160891
Freedom	0.033550	0.345008	0.437103	1.000000	0.346236	0.559062	0.357196	0.490044
Generosity	-0.116842	-0.016985	0.072353	0.346236	1.000000	0.161861	0.076855	0.292071
Happiness Score	0.495504	0.786473	0.631842	0.559062	0.161861	1.000000	0.751249	0.405281
Health (Life Expectancy)	0.059366	0.800788	0.499019	0.357196	0.076855	0.751249	1.000000	0.258992
Trust (Government Corruption)	0.001275	0.300185	0.160891	0.490044	0.292071	0.405281	0.258992	1.000000

Plotting Correlation using MatLibPlot Library

We will plot the correlation. The first correlation plot is made using matlibplot library.

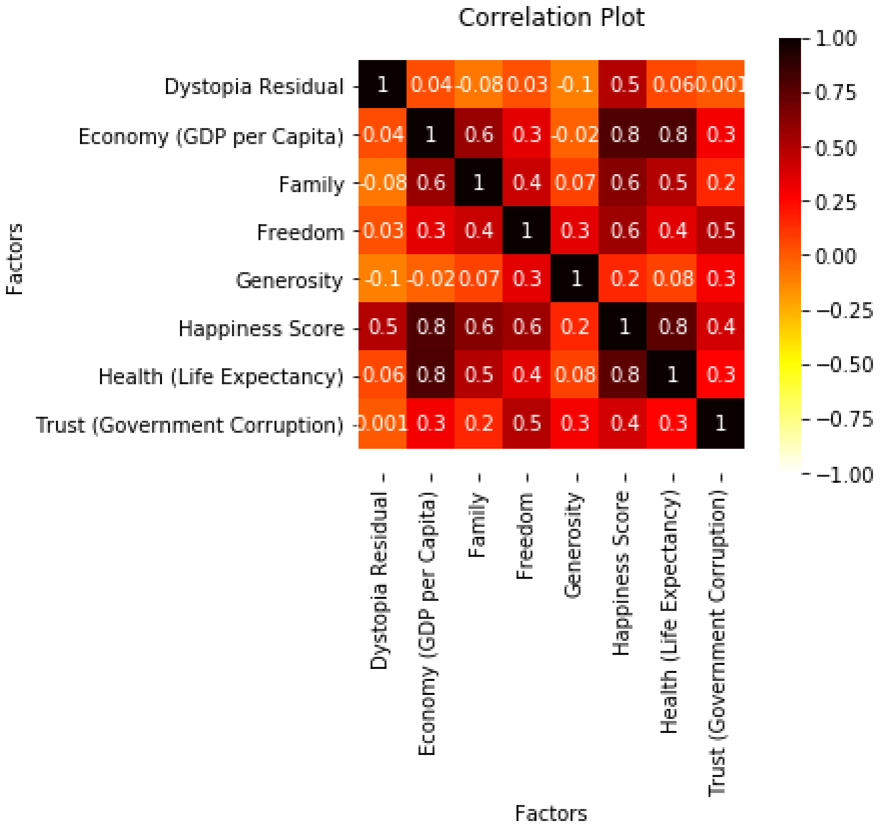
```
In[62]: plt.imshow(corr_plot, cmap='hot', )
      plt.xlabel("Factors")
      plt.ylabel("Factors")
      plt.title("Correlation Plot")
      plt.show()
```



Plotting Correlation using Seaborn Library

The next correlation plot is using seaborn library.

```
In  [63]:corr_plot_sns  =  sns.heatmap(corr_plot,  annot  =  True,  fmt='.1g',
         vmin=-1,vmax=l, center= 0, cmap= "hot_r", square= True,)
         corr_plot_sns.set_ylim(len(corr_plot)+0.5, -0.5)
         plt.title("Correlation Plot")
         plt.xlabel("Factors")
         plt.ylabel("Factors")
```



Its clear from the above model that happiness of country is highly correlated to Economic and Health conditions and weakly correlated to Generosity of people in the nation and the people's trust on the government.

Predictive Modeling:

Dividing the dataset into Test and Train

The following code will create a copy of data frame of Combined Dataset in a randomized order. The new dataset is divided into 2 parts train and test. 70% of data points are in train and 30% in test.

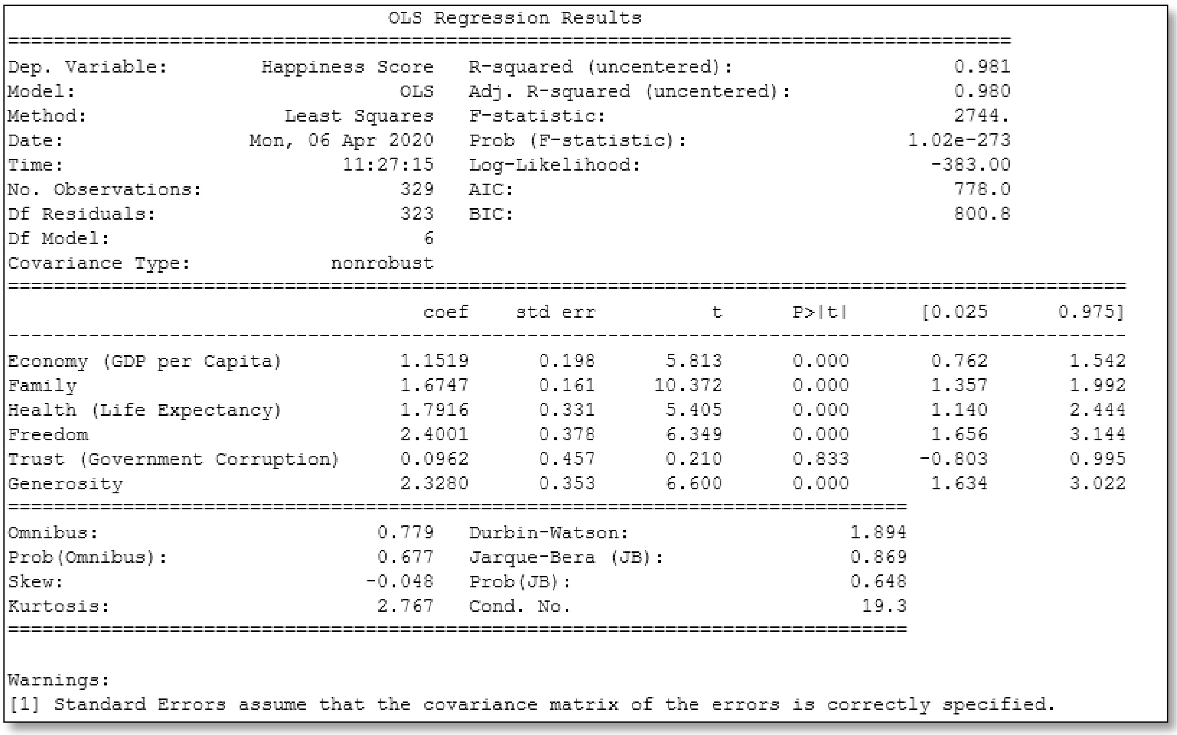
```
In[55]: # We Will store a randomized order in a new dataframe
      comb = combined_dataset.sample(frac=l).reset_index(drop=True)
      # I have divided it into 2 parts
      # 70% in test and 30% in train
      train = comb [0: 329]
      test = comb [329:]
```

Creating the model

Variables y_train and x_train is used for training and represents the dependent and independent variables respectively. Next we will create a model. The codes are:

```
In [56]: # We first import the libraries needed
      from sklearn import datasets, linear_model
      # Our dependent variable is Happiness Score
      # Our independend variable are Economy, Family, Health, Freedom, rust
      and Generosity

       y_train ="Happiness Score"]
       x_train =train[["Economy (GDP per Capita)", "Family", " Health (Life
      Expectancy)", "Freedom", "Trust (Government Corruption)",
      "Generosity"]]

      # we will now create the model

      model = sm.OLS(y_train, x_train).fit()

      print_model = model.summary()
      print(print_model)
```

The model summary is:



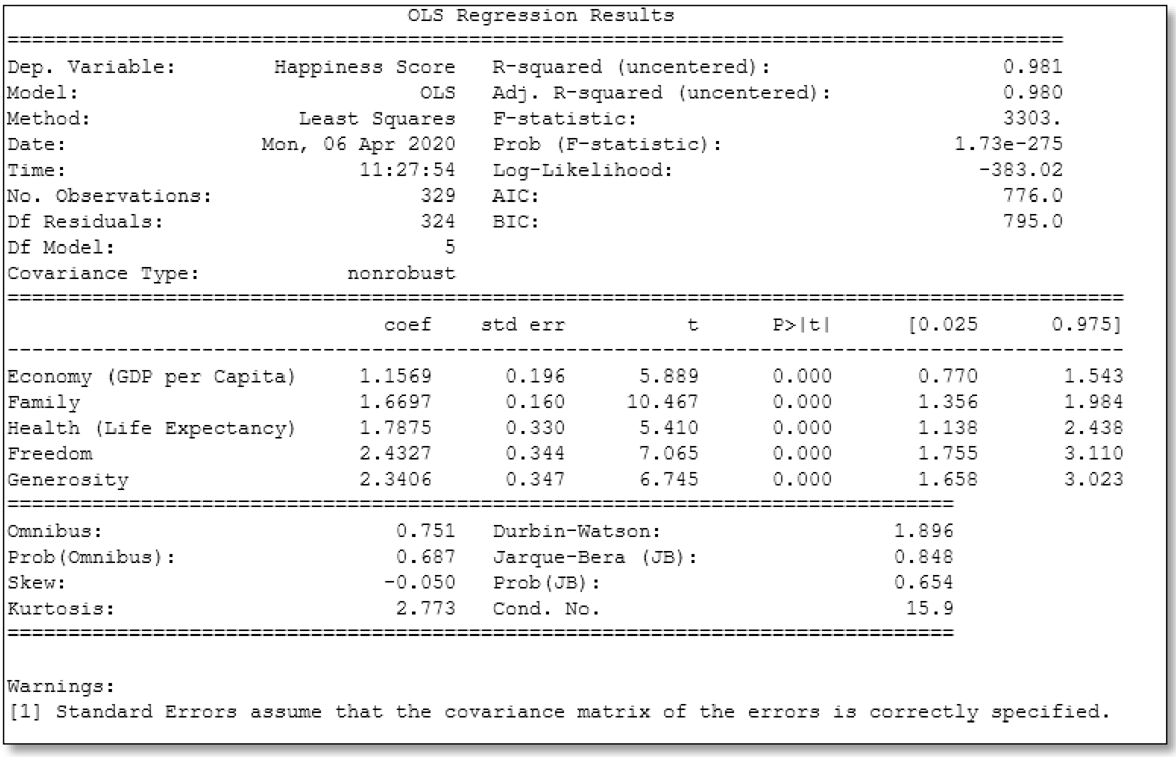
```
                                 OLS Regression Results
=======================================================================================
Dep. Variable:       Happiness Score   R-squared (uncentered):                   0.981
Model:                           OLS   Adj. R-squared (uncentered):              0.980
Method:                Least Squares   F-statistic:                              2744.
Date:               Mon, 06 Apr 2020   Prob (F-statistic):                   1.02e-273
Time:                       11:27:15   Log-Likelihood:                         -383.00
No. Observations:                329   AIC:                                      778.0
Df Residuals:                    323   BIC:                                      800.8
Df Model:                          6
Covariance Type:           nonrobust
=================================================================================================
                                  coef    std err          t      P>|t|      [0.025      0.975]
-------------------------------------------------------------------------------------------------
Economy (GDP per Capita)        1.1519      0.198      5.813      0.000       0.762       1.542
Family                          1.6747      0.161     10.372      0.000       1.357       1.992
Health (Life Expectancy)        1.7916      0.331      5.405      0.000       1.140       2.444
Freedom                         2.4001      0.378      6.349      0.000       1.656       3.144
Trust (Government Corruption)   0.0962      0.457      0.210      0.833      -0.803       0.995
Generosity                      2.3280      0.353      6.600      0.000       1.634       3.022
==============================================================================
Omnibus:                       0.779   Durbin-Watson:                   1.894
Prob(Omnibus):                 0.677   Jarque-Bera (JB):                0.869
Skew:                         -0.048   Prob(JB):                        0.648
Kurtosis:                      2.767   Cond. No.                         19.3
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

The model accuracy is given by Adjusted R Squared. The accuracy of the model is given as 98.1%. Variables for which P > |t| is more than 0.05 are not significant and are rejected. Happiness is not dependent on them. Since Trust has P value 0.145, Trust does not have a significant effect on Happiness. Happiness is dependent on Economy, Family, Health, Freedom and Generosity.

We will create a new model after not considering Trust has a dependent variable.

```
In [57]: # We first import the libraries needed
      from sklearn import datasets, linear_model
      # Our dependent variable is Happiness Score
      # Our independend variable are Economy, Family, Health, Freedom, rust
      and Generosity
      y_train ="Happiness Score"]
      x_train =train[["Economy (GDP per Capita)", "Family", " Health (Life
      Expectancy)", "Freedom", "Generosity"]]
      # we will now create the model

      model = sm.OLS(y_train, x_train).fit()

      print_model = model.summary()
      print(print_model)
```

The summary is:



```
                                 OLS Regression Results
=======================================================================================
Dep. Variable:       Happiness Score   R-squared (uncentered):                   0.981
Model:                           OLS   Adj. R-squared (uncentered):              0.980
Method:                Least Squares   F-statistic:                              3303.
Date:               Mon, 06 Apr 2020   Prob (F-statistic):                   1.73e-275
Time:                       11:27:54   Log-Likelihood:                         -383.02
No. Observations:                329   AIC:                                      776.0
Df Residuals:                    324   BIC:                                      795.0
Df Model:                          5
Covariance Type:           nonrobust
============================================================================================
                              coef    std err          t      P>|t|      [0.025      0.975]
--------------------------------------------------------------------------------------------
Economy (GDP per Capita)    1.1569      0.196      5.889      0.000       0.770       1.543
Family                      1.6697      0.160     10.467      0.000       1.356       1.984
Health (Life Expectancy)    1.7875      0.330      5.410      0.000       1.138       2.438
Freedom                     2.4327      0.344      7.065      0.000       1.755       3.110
Generosity                  2.3406      0.347      6.745      0.000       1.658       3.023
==============================================================================
Omnibus:                       0.751   Durbin-Watson:                   1.896
Prob(Omnibus):                 0.687   Jarque-Bera (JB):                0.848
Skew:                         -0.050   Prob(JB):                        0.654
Kurtosis:                      2.773   Cond. No.                         15.9
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

The accuracy of the model is given as 98.1%. Variables for which P > |t| is more than 0.05 are not significant and are rejected. Happiness is not dependent on them. Happiness is dependent on Economy, Family, Health, Freedom and Generosity.

Since this model has a similar accuracy to the previous model and variables in this model are significant variables, we will accept this model.

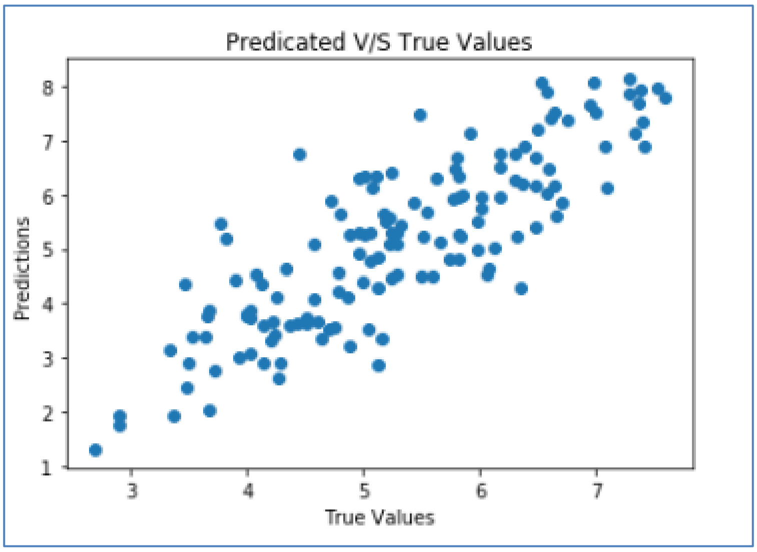
Our model thus is:

$$\begin{aligned} \boldsymbol{Happiness\ Score} = {} & \mathbf{1.1196} \times \boldsymbol{Economy} + \mathbf{1.6687} \times \boldsymbol{Family} + \mathbf{1.6127} \times \boldsymbol{Health} \\ & + \mathbf{2.8593} \times \boldsymbol{Freedom} + \mathbf{1.8650} \times \boldsymbol{Generosity} \end{aligned}$$

Testing the model:

We will test this on the remaining 30% dataset.

```
In [58]:x_test = test[["Economy (GDP per Capita)", "Family", " Health (Life
      Expectancy)", "Freedom", "Generosity"]]
      y_test = test["Happiness Score"]

      predictions = model.predict(x_test)
      plt.scatter(y_test, predictions)
      plt.xlabel("True Values")
      plt.ylabel("Predictions")
      plt.title("Pridiction V/S True Values")
```



The above graph proves that residuals follow a normal distribution.

Using Shapiro test to check Normality

We will verify this through Shapiro Test

```
In [59]:from scipy import stats
      residuals = predictions -  y_test
      pval = stats.shapiro(residuals)[1]

      alpha = 0.05
      if pval > alpha:
            print("Error terms follow a noraml distribution")
      else :
            print ( "Error terms do not follow a normal distribution")

Error terms follow a noraml distribution
```

This proves that the residuals follow normal distribution.

19.6 Conclusion

We were able to obtain best countries in all of the six categories for all the three years. Switzerland, Denmark and Norway were the happiest country in 2015, 2016 and 2017 respectively. Other results can be seen on page 6.

From year 2015 to 2016, Algeria saw the biggest improvement whereas Liberia saw the biggest downfall; from year 2016 to 2017, Bulgaria saw the biggest improvement whereas Venezuela saw the biggest downfall and for year 2015 to 2017, Latvia saw the biggest downfall and Venezuela saw the biggest downfall.

Venezuela saw the biggest downfall and most change in score, this is because from 2016, Venezuela's economy collapsed and the country saw huge riots. Currently Venezuela is in an economic crisis. This led to such a huge downfall in its score.
At an average 15 nations are happy, 83 are okay and 60 are sad. Western Europe is the happiest region and Sub Sahara Region is the saddest.

Happiness of a country is dependent on the region in which the country belongs to. This may be because countries in the same region tend to have similar political, social and economical conditions.

The proportion of happy, sad and okay countries has not changed in the 3 year time period.
Happiness is highly correlated to Economy and Health and weakly correlated to Generosity.
We were also able to construct a model for predicting happiness score:

$$\begin{aligned} Happiness\ Score = {} & 1.1196 \times Economy + 1.6687 \times Family + 1.6127 \times Health \\ & +2.8593 \times Freedom + 1.8650 \times Generosity \end{aligned}$$

19.7 Limitation

The data we have is for only 3 years. Such a short period will not provide adequate answers. A long period would have given us a better idea. While frequency testing some of the frequencies was less than 5, because of this the results we obtained cannot be fully tested.

Only 146 countries data are available in all 3 dataset. This leads to inadequate results will finding the countries with biggest change since data for some countries are missing to do proper comparison.

The dataset is for a preference survey. Preference surveys do not always provide the right results. A country's economic conditions may be bad but through propaganda people might be influenced to give better results. Thus, propaganda might have influenced the survey results.

The data is secondary and thus we have no idea how this survey was conducted and data collected. The dataset does not consider difference in happiness and results due to gender, religion, sexuality, etc. A particular group may be happier compared to another. It does not consider cultural impact, entertainment impact, etc. on happiness.

19.8 Acknowledgment

We acknowledge our Principal Dr. Hemalata Bagla of K. C. College for encouraging Project Work and Research. We sincerely acknowledge Dr. Asha Jindal, Head, Department of Statistics, K. C. College for organizing the Python Software Schem Training with DBT- Star College Status Scheme Fund in which we could acquire the skills to complete this project. We would like to especially acknowledge Mr. Pravesh Tiwari for teaching us Python Software effectively in short span of time.

19.9 References

1. https://towardsdatascience.com/train-test-split-and-cross-validation-in-python-80b61beca4b6
2. https://heartbeat.fritz.ai/seaborn-heatmaps-13-ways-to-customize-correlation-matrix-visualizations-f1c49c816f07
3. Training Programme material provided by Pravesh Tiwari Sir

Chapter 20
Analysis of Diabetes

Mr. Aditya Shrivastava and Mr. Hruturaj Nikam
DBT- Star College Scheme Researchers, Department of Computer Science

20.1 Introduction

Diabetes is a very common metabolic disease. Usually onset of type 2 diabetes happens in middle age and sometimes in old age. But nowadays incidences of this disease are reported in children as well. There are several factors for developing diabetes like genetic susceptibility, body weight, food habit and sedentary lifestyle. Undiagnosed diabetes may result in very high blood sugar level referred as hyperglycemia which can lead to complication like diabetic retinopathy, nephropathy, neuropathy, cardiac stroke and foot ulcer. So, early detection of diabetes is very important to improve quality of life of patients and enhancement of their life expectancy

20.2 Role of analytics in Diabetes

In today's competitive world talented people are the most worthwhile treasure for the company and at the same time burdensome to hold down such valuable resources in organization. During last year's, large investments were put into tools and information systems to manage performance, hiring, compliance and employees' development in order to enhance its capabilities.

The learning process starts with the gathering of data by different means, from various resources. Then the next step is to prepare the data, that is pre-process it in order to fix the data related issues and to reduce the dimensionality of the space by removing the irrelevant data (or selecting the data of interest). Since the amount of data that is being used for learning is large, it is difficult for the system to make decisions, so algorithms are designed using some logic, probability, statistics, control theory etc. to analyse the data and retrieve the knowledge from the past experiences. Next step is testing the model to calculate the accuracy and performance of the system. And finally optimization of the system, *i.e.* improvising the model by using new rules or data set. The techniques of machine learning are used for classification, prediction and pattern recognition. Machine learning can be applied in various areas like: search engine, web page ranking, email filtering, face tagging and recognizing, related advertisements, character recognition, gaming, robotics, disease prediction and traffic management The biggest struggles in achieving better utilization of data resources and information systems are inefficient use of the data, asking wrong

ISBN 978-93-80788-93-7

questions and lack of analytical ability in HR environment in general. HR departments are in need for analytically capable people enabled to provide right insights combining reporting skills and domain knowledge.

20.3 Problem Statement

1. The goal of the case study is to find out which are the most influential factors leading to diabetes.
2. Which person will get diabetes next?

20.4 Methodology

1. **Visualization: -** The first step is to visualize and perform univariate analysis to explore data to find useful insights.
2. **Model: -** Next step is to model the data in order to confirm or reject our hypothesis that certain variables are significant in determining person diabetes.
3. **Actionable Insights: -** The final step is to review and build onto our analysis by drawing new insights or further enhancing existing insights.

20.5 Data Dictionary

Variable Name	**Variable Definition**
Pregnancies	Number of times pregnant
Glucose	Plasma glucose concentration a 2 hours in an oral glucose tolerance test.
BloodPressure	Number of projects completed while at work
SkinThickness	Triceps skin fold thickness (mm)
Insulin	2-Hour serum insulin (mu U/ml)
BMI	Body mass index (weight in kg/(height in m)^2)
DiabetesPedigreeFunction	Diabetes pedigree function
Age	Age (years)
Outcome	Class variable (0 or 1) 268 of 768 are 1, the others are 0

20.6 Data Analysis

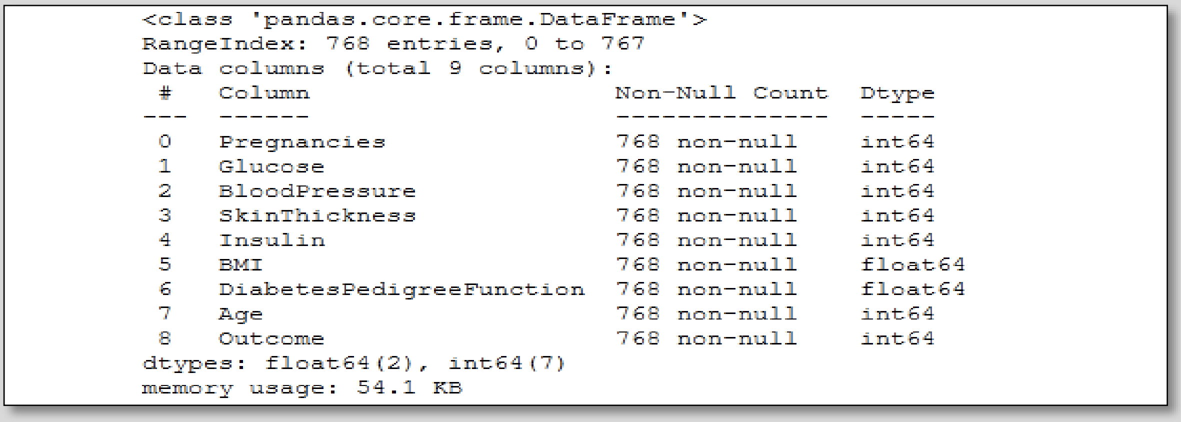
Structure of the data

We are using **“diabetes.csv”**.

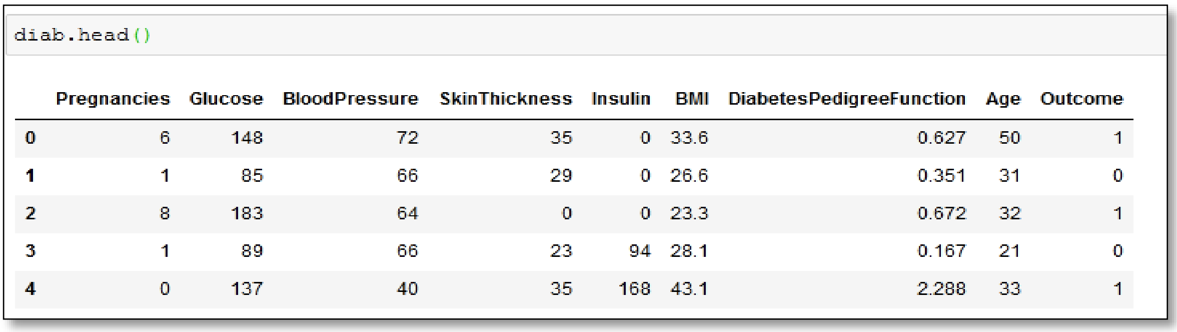
```
In [1]:import numpy as np
      import pandas as pd
      import matplotlib.pyplot as plt
      import seaborn as sns
      %matplotlib inline
```

```
In [2]: diab = pd.read_csv(“C:/Users/Admin/Desktop/diabetes.csv”)
In [3]: diab.shape
Out [3]: (768, 9)
```

```
In [4]: diab.info()
```

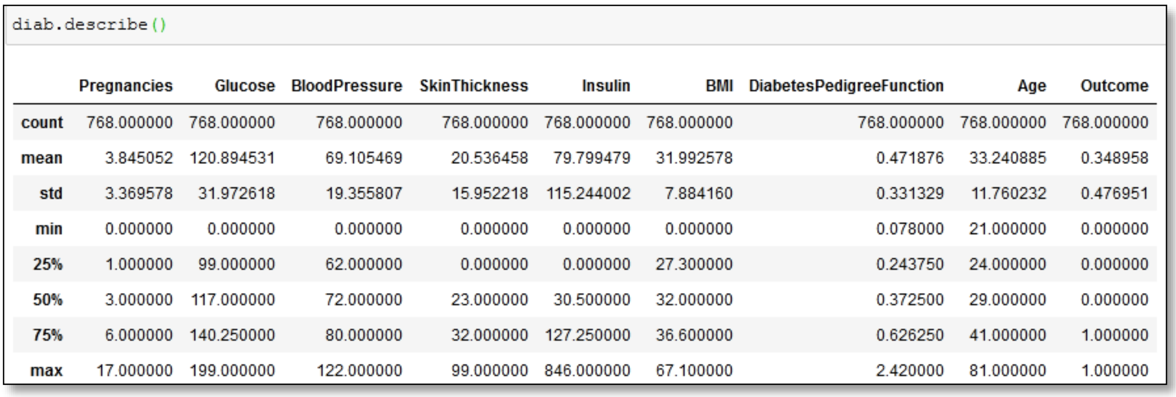


```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   Pregnancies               768 non-null    int64
 1   Glucose                   768 non-null    int64
 2   BloodPressure             768 non-null    int64
 3   SkinThickness             768 non-null    int64
 4   Insulin                   768 non-null    int64
 5   BMI                       768 non-null    float64
 6   DiabetesPedigreeFunction  768 non-null    float64
 7   Age                       768 non-null    int64
 8   Outcome                   768 non-null    int64
dtypes: float64(2), int64(7)
memory usage: 54.1 KB
```



```
diab.head()
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1



```
diab.describe()
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000
mean	3.845052	120.894531	69.105469	20.536458	79.799479	31.992578	0.471876	33.240885	0.348958
std	3.369578	31.972618	19.355807	15.952218	115.244002	7.884160	0.331329	11.760232	0.476951
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.078000	21.000000	0.000000
25%	1.000000	99.000000	62.000000	0.000000	0.000000	27.300000	0.243750	24.000000	0.000000
50%	3.000000	117.000000	72.000000	23.000000	30.500000	32.000000	0.372500	29.000000	0.000000
75%	6.000000	140.250000	80.000000	32.000000	127.250000	36.600000	0.626250	41.000000	1.000000
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	2.420000	81.000000	1.000000

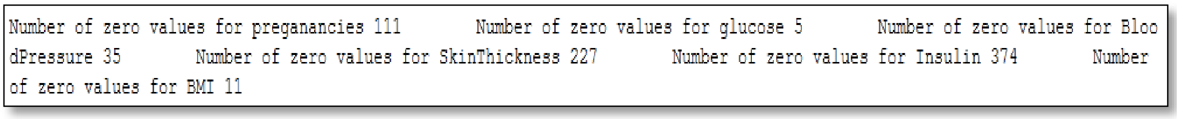
From the structure of the data we can see that sample size of the data is 768 and there are 9 variables and also whether a variable is categorical or continuous.

Removing Null Values

```
In [7]: diad.isnull().sum()
Out[7]:Pregnancies                 0
       Glucose                     0
       Blood Pressure              0
       SkinThickness               0
       Insulin                     0
       BNI                         0
       DiabetesPedigreeFunction    0
       Age                         0
       Outcome                     0
       dtype: int64
```

```
In [8]: diab.duplicated().value_counts()
Out [8]: Fales    768
         dtype:   int64
```

```
In[9]:print(f”Number of zero values for pregnancies { len( diab
            [‘Pregnancies’]==0            ])}\
            Number of zero values for glucose {len(diab[‘Glucose’]==0])}\
            Number of zero values for Blood Pressure {len (diab [‘Blood
      Pressure’]==0])}\
            Number of zero values for SkinThickness {len(diab
            [‘SkinThickness’]==0])}\
            Number of zero values for Insulin {len(diab[‘Insulin’]==0])}\
            Number of zero values for BMI {len(diab[‘BMI’]==0])}\“)
```



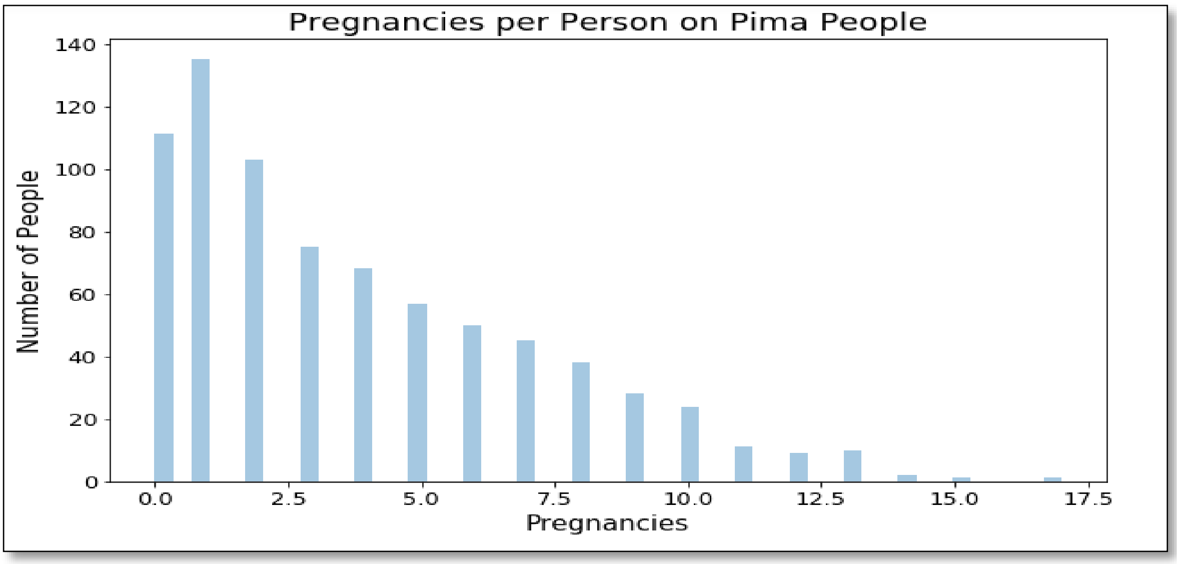
```
Number of zero values for preganancies 111       Number of zero values for glucose 5       Number of zero values for Bloo
dPressure 35       Number of zero values for SkinThickness 227       Number of zero values for Insulin 374       Number
of zero values for BMI 11
```

Pregnant women who suffer from the Diabetes

```
print(f”Average amount of children had by a Pima woman: {diab [‘Pregancies’]
,mean()}”)
```

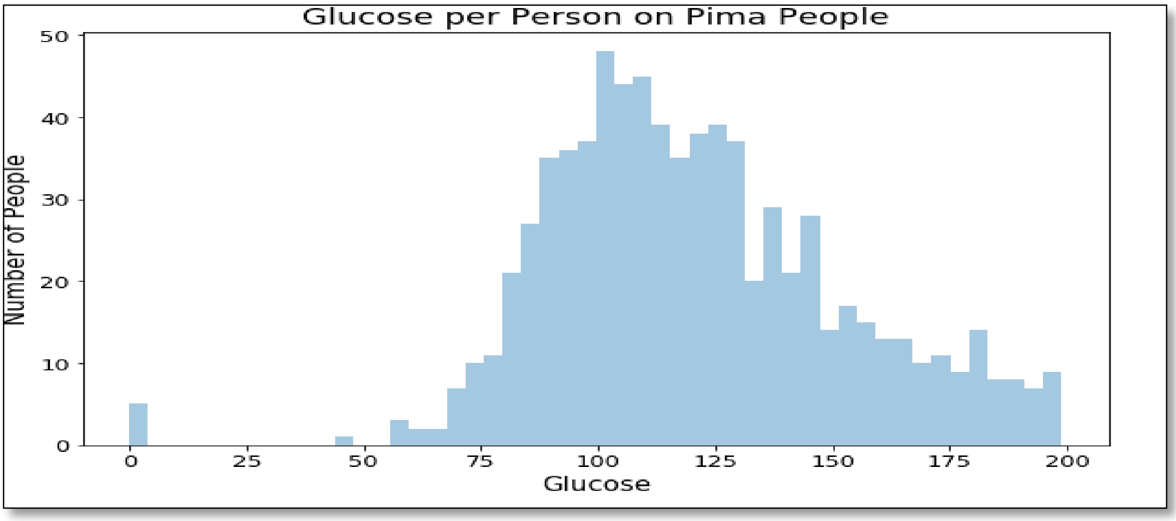
```
In [10]:#unvivarite analyasis
      plt.figure(figsize=(10,6))
      sns.distplot(diab['Pregancies'],kde=False,bins=50)
      plt.title('Pregancies per Person on Pima People',fontsize=18)
      plt.xticks(fontsize=13)
      plt.yticks(fontsize=13)
      plt.xlabel('Pregnancies'fontsize=15)
      plt.ylabel('Number of people',fontsize=15)
      plt.show()
```

ISBN 978-93-80788-93-7



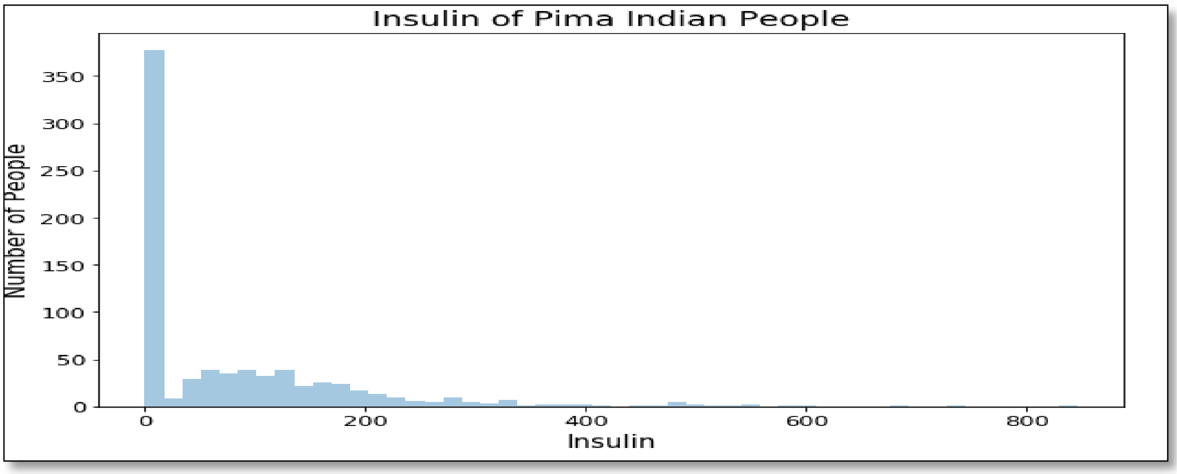
Level of Glucose per Person on Pima People

```
In [11]:plt.figure(figsize=(10,6))
      sns.displot(diab['Glucose'],kde=False,bins=50)
      plt.title('Glucose per Person on Pima People',fontsize=18)
      plt.xticks(fontsize=13)
      plt.yticks(fontsize=13)
      plt.xlabel('Glucose'fontsize=15)
      plt.ylabel('Number of people',fontsize=15)
      plt.show()
```



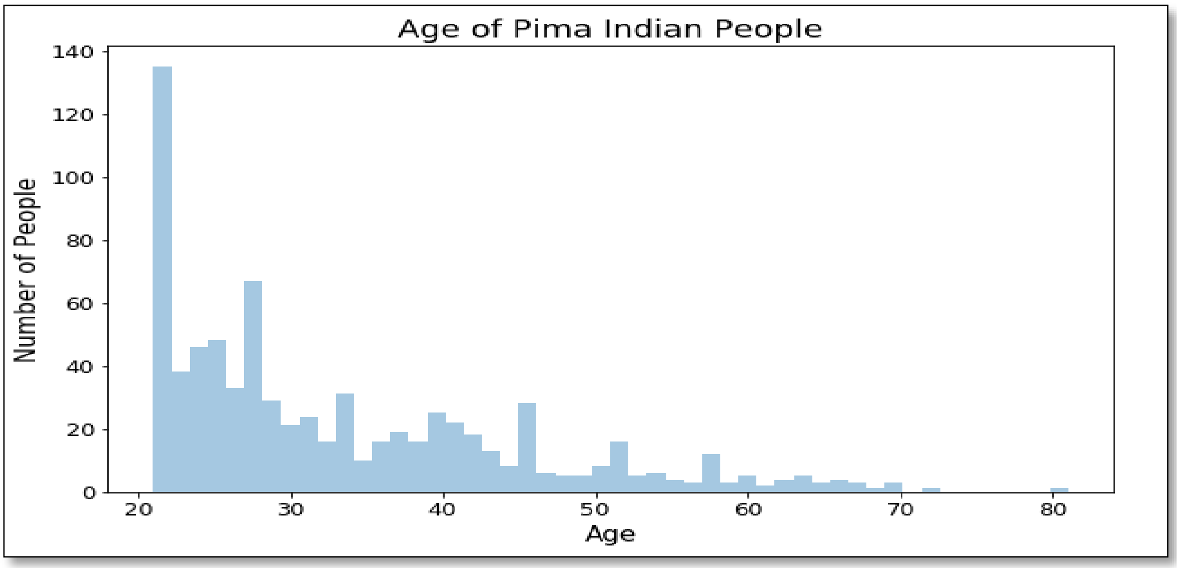
Insulin of Pima Indian People

```
In [13]:plt.figure(figsize=(10,6))
      sns.displot(diab['Insulin'],kde=False,bins=50)
      plt.title('Glucose per Person on Pima People',fontsize=18)
      plt.xticks(fontsize=13)
      plt.yticks(fontsize=13)
      plt.xlabel('Insulin'fontsize=15)
      plt.ylabel('Number of people',fontsize=15)
      plt.show()
```



Age of Pima Indian People

```
In [14]:plt.figure(figsize=(10,6))
      sns.displot(diab['Age'],kde=False,bins=50)
      plt.title('Glucose per Person on Pima People',fontsize=18)
      plt.xticks(fontsize=13)
      plt.yticks(fontsize=13)
      plt.xlabel('Age'fontsize=15)
      plt.ylabel('Number of people',fontsize=15)
      plt.show()
```



Predictive Modeling:

Logistic Regression

Logistic Regression is a classification algorithm. It is used to predict a binary outcome (1 / 0, Yes / No, True / False) given a set of independent variables. In simple words, it predicts the probability of occurrence of an event by fitting data to a logit function. Logistic regression is

ISBN 978-93-80788-93-7

used to describe data and to explain the relationship between one dependent binary variable and one or more nominal, ordinal, interval or ratio-level independent variables.

The logistic regression model

$\ln[p/(1-p)] = a + BX + e$ or

$[p/(1-p)] = \exp(a + BX + e)$

where,

ln is the natural logarithm, $\log_{exp}$, where exp=2.71828…

p is the probability that the event Y occurs, $p(Y=1)$

$p/(1-p)$ is the "odds ratio"

$\ln[p/(1-p)]$ is the log odds ratio, or "logit"

All other components of the model are the same.

The logistic regression model is simply a non-linear transformation of the linear regression. The "logistic" distribution is an S-shaped distribution function which is similar to the standard-normal distribution (which results in a probit regression model) but easier to work with in most applications (the probabilities are easier to calculate). The logit distribution constrains the estimated probabilities to lie between 0 and 1.

For instance, the estimated probability is:

$p = 1/[1 + \exp(-a - BX)]$

With this functional form:

if you let $a + BX = 0$, then $p = .50$

as $a + BX$ gets really big, p approaches 1

as $a + BX$ gets really small, p approaches 0.

Data splitting

Separating data into training and testing sets is an important part of evaluating data mining models. Typically, when you separate a data set into a training set and testing set, most of the data is used for training, and a smaller portion of the data is used for testing. Analysis Services randomly samples the data to help ensure that the testing and training sets are similar. By using similar data for training and testing, you can minimize the effects of data discrepancies and better understand the characteristics of the model.

After a model has been processed by using the training set, you test the model by making predictions against the test set. Because the data in the testing set already contains known values for the attribute that you want to predict, it is easy to determine whether the model's guesses are correct.

Dependent Variable for Modeling: (Whether the person is diabetic or not in future)

Independent Variables: Pregnancies, Glucose, BloodPressure, Insulin, BMI, DiabetesPedigreeFunction, Age

Model Building and Output Interpretation

Step 1: Create a logistic model

```
In [15]: # Logistic Regession
In [16]: from Sklearn.model_selection import train_test_split
      from sklern.linear_model import LogisticRegression
      from sklearn.metrics import classification_report, confusion_matrix
```

```
In [17]: x = diab.drop(['Outcome'],axis=1)
       y=diab['Outcome']
```

```
In[18]:  x_train,x_test,  y_train  =  train_test_split(x,  y,  test_size=0.3,
      random_state=101)
      lr=logisticRegression('12',solver='newton-cg')
      lr=fit(x_train,y_train)
Out [18]: LogisticRegression(c=1.0, claas_weight=none, dual=False,
                  fit_intercept=True,
                  intercept_scaling=1, 11_ratio=none, max_iter=100,
                  multi_class='warn', n_jobs=None, penalty='12',
                  random_state=None, solver='newton-cg', tol=0.0001,
                  verbose=0,
                  warm_start=False)
```

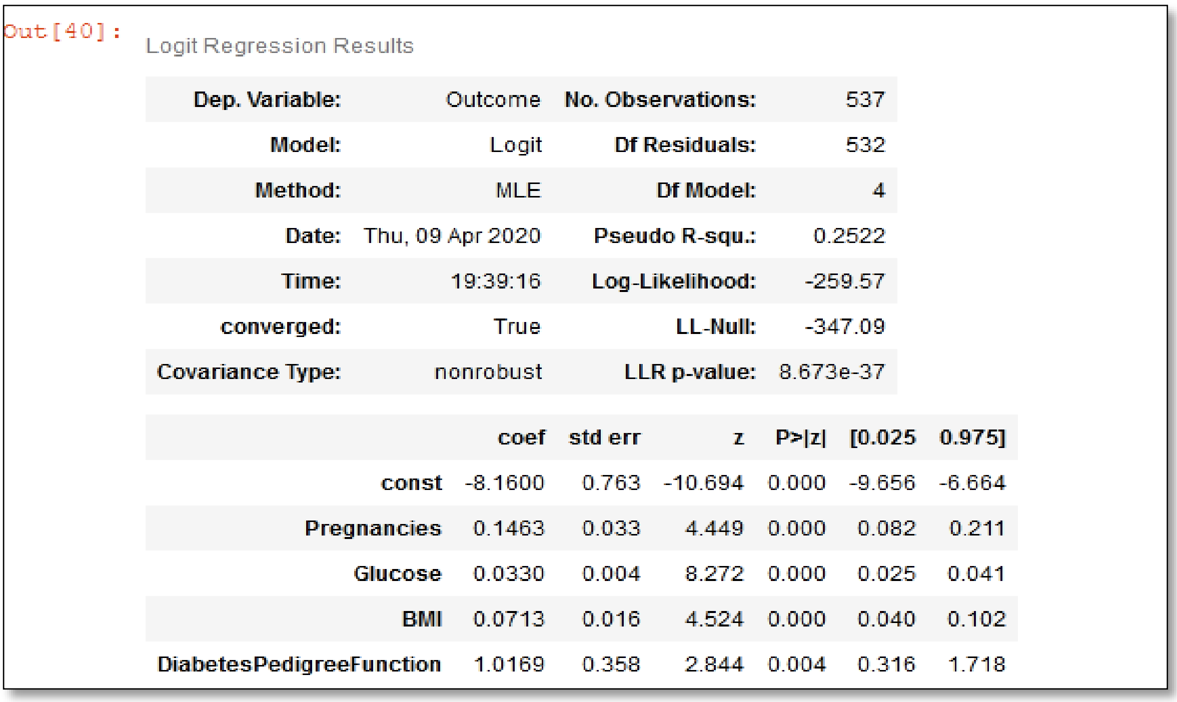
```
In [38]: #Alternative way
      from sklearn.feature_selection import RFE
      import statsmodels.api as sm
      logreg = LogisticRegression()
      rfe = RFE(logre, )
      rfe = rfe.fit(X_train, y_train.values.ravel())
      list(zip(x_train.columns,rfe.support_,rfe.ranking_))
```

```
Out [38]:[('Pregnancies', True, 1),
      ('Glucose', True, 1),
      ('BloodPressure', False, 2),
      ('SkinThickness', False, 4) ,
      ('Inaulin', False, 5),
      ('BMI', True, 1),
      ('DiabetesPedigreePunction', True, 1),
      ('Age', False, 3)]
```

```
 In [39]: print(x_train.columns[rfe.support_])
      x_train_rfe = x_train[x_train.columns[rfe.support_]]
      x_train_rfe = sm.add_constant(x_train_rfe)
      Index(['Pregnancies','Glucose','BMI','DiabetesPedigreePunction'],dty
      pe='object'
```

```
In [40]: model = sm.logit(y_train, x_train_rfe)
      result = model.fit()
      result.summary()
      optimization terminated successfully.
            Current function value: 0.483368
            Iteration 6
```

ISBN 978-93-80788-93-7



```
Out[40]:
          Logit Regression Results

    Dep. Variable:          Outcome   No. Observations:        537
           Model:             Logit       Df Residuals:        532
          Method:               MLE           Df Model:          4
            Date:  Thu, 09 Apr 2020     Pseudo R-squ.:      0.2522
            Time:          19:39:16     Log-Likelihood:     -259.57
       converged:              True            LL-Null:     -347.09
 Covariance Type:         nonrobust        LLR p-value:   8.673e-37

                               coef   std err         z   P>|z|   [0.025   0.975]
                    const   -8.1600     0.763   -10.694   0.000   -9.656   -6.664
              Pregnancies    0.1463     0.033     4.449   0.000    0.082    0.211
                  Glucose    0.0330     0.004     8.272   0.000    0.025    0.041
                      BMI    0.0713     0.016     4.524   0.000    0.040    0.102
 DiabetesPedigreeFunction    1.0169     0.358     2.844   0.004    0.316    1.718
```

Step 2: Global testing

H_0: $b_1 = b_2 = \dots = b_k = 0$ OR (H_0: None of the variables has significant impact)

H_1: At least one coefficient is not zero

Test Statistic:

$\chi^2 = L_1 - L_2$ which follows Chi-square distribution with k df.

$L_1 = -2 \log L$ with only constant term L2 = $-2 \log L$ with k variables and constant term

Reject H_0 for large value of χ^2 or Reject H_0 if p-value < 0.05

Since p-value is less than 0.05 we reject H_0 and conclude that the variable is making impact on dependent variable.

Step 3: Obtaining confusion matrix

```
In [41]: logreg = LogisticRegression()
      logreg.fit(x_train_rfe, y_train)
      x_test = sm.add_constant(x_test)
      x_test_rfe = x_test.filter(list(x_train_rfe.columns))
      y_pred = logreg.predict(x_test_rfe)
```

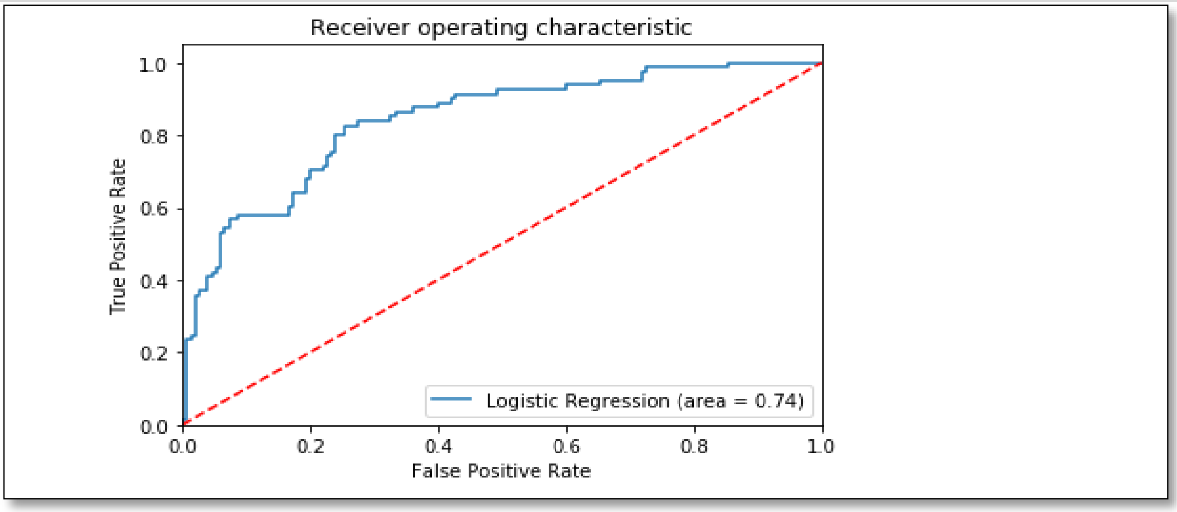
```
In [42]: from sklearn.metrics import confusion_matrix
       confusion_matrix = confusion_matrix(y_test, y_pred)
       print(confusion_matrix)
       [[134      16]
       [34        47]]
```

Step 4: Measuring accuracy of the model

```
In [43]: from sklearn.metrics import roc_auc_score
      from sklearn.metrics import roc_curva
```

ISBN 978-93-80788-93-7

```
      logit_roc_auc                   =                  roc_auc_score(y_test,
      logreg.predict_proba(x_test_rfe)[:,1])
      plt.figure(),
      plt.plot(fpr,  tpr,  label=  'Logistic  Regression  (area  =  %0.2f)'
      %logit_roc_auc)
      plt.plot([0,1],[0,1],'r--')
      plt.xlim([0.0, 1.0])
      plt.ylim([0.0,1.05])
      plt.xlabel('False Positive Rate')
      plt.ylabel('True Positive Rate')
      plt.title('Receiver operating charateristic')
      plt.legend(loc="lower right")
      #plt.savefig('Log_ROC')
      plt.show()
```



Area inside the curve which indicates accuracy of the model is **74%.**

20.7 References

1. https://realpython.com/logistic-regression-python/
2. https://towardsdatascience.com/logistic-regression-python-7c451928efee
3. Training Programme notes conducted by Pravesh Tiwari

Graphics & Design Team, K. C. College

Mr. Roshan Khilnani, Coordinator, Graphics & Design
Mr. Monish Jeswani, Student, Dept of Computer Science
Mr. Kartik Sharma, Student, Dept of Computer Science
Ms. Roshini Gupta, Student, Dept of Computer Science

ISBN 978-93-80788-93-7

Kishinchand Chellaram College

Vidyasagar Principal K.M. Kundnani Chowk,
124, Dinshaw Wachha Road,
Churchgate, Mumbai 400020.
Tel: +91-22-2285 5726; +91-22-6698 1000;
Fax: +91-22-2202 9092;
Email: office@kccollege.edu.in
Website: www.kccollege.edu.in

Shailja Prakashan

57–P, Kunj Vihar–II,
Yashoda Nagar, Kanpur-11
Ph.: 0512-2633004
Email: shailjaprakashan@gmail.com

ISBN 978-93-80788-93-7
9 789380 788937 >

ISBN 978-93-80788-93-7

Designed by Roshan P. Khilnani, email: rpkhilnani@yahoo.in