Diploma in

IT, Networking and Cloud

Module 5

Business Data Analytics methods and tools

Lab Manual

Learning Outcome

After completing this module, the student should be **able to understand the Business Analytics**

To meet the learning outcome, a student has to complete the following activities

- Use Excel for understanding different types of data (Integer, double, text, date) (5 Hrs)
- Perform operations on different data types. (5Hrs)
- Segregate data in different sheets. (5Hrs)
- Calculate arithmetic mean, geometric mean and Harmonic mean (5Hrs)
- Calculate median from raw & grouped data (5Hrs)
- Calculate mode for row & grouped data (5Hrs)

Activity 1

Aim: Use Excel for understanding different types of data (Integer, double, text, date)

Learning outcome: Able to understand the Business Analytics

Duration: 5 Hours

List of Hardware/Software requirements:

1. Windows 7/Windows 10
2. MS Office 2010 with Excel or Latest Version

Code/Program/Procedure (with comments):

Integer

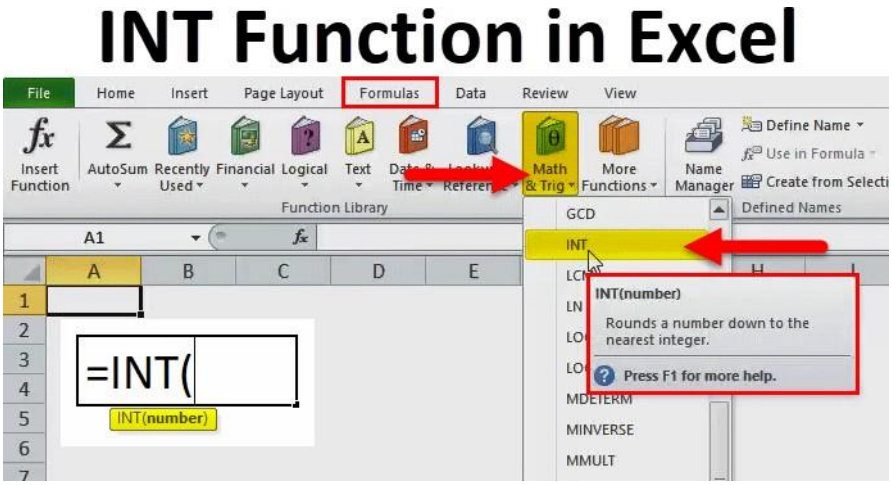
Integer values are written as a sequence of digits, possibly prefixed by a + or - sign. The integer values that can be specified range from -2147483648 to 2147483647. If used where a decimal value was expected, the integer values are automatically converted to decimal values.

Note: Hexadecimal values can be used in custom expressions and in calculated columns. They cannot be used when opening data. Hexadecimal-formatted values have a size limitation of 8 characters.

Examples:

- 0
- 101
- -32768
- +55
- 0xff = 255
- 0x7fffffff = 2147483647
- 0x80000000 = -2147483648

INT Excel Function (Integer)



The Microsoft Excel INT Function is a function which is responsible for returning the integer portion of a number. It works by the process of rounding down a decimal number to the integer. The INT Function in Excel is built in Excel function and is categorized as Math & Trig Function in Excel. The INT function in Excel is used either as a worksheet function. Here, negative numbers become more negative because the function rounds down. For example, INT (10.6) returns 10 and INT (-10.6) returns -11.

Parameters

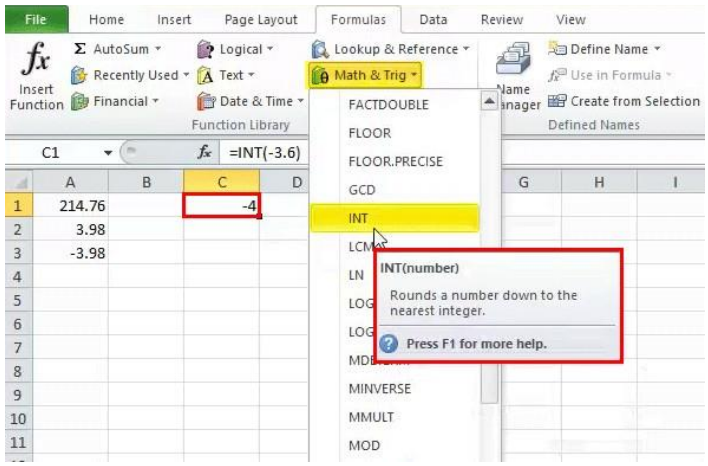
- It accepts the following parameters and arguments:
- number – The number to be entered from which you want an integer.

Return Value

- The return value will be a numeric integer.

Procedure to open INT function in MS Excel

1. You can simply enter the desired Integer excel formula in the required cell to attain areturn value on the argument.
2. You can manually open the INT formula in excel dialogue box in the spreadsheet andenter the logical values to attain a return value.
3. Consider the screenshot below to see the INT Function in excel option under the Math& Trig Function menu.



4. Click on the INT function option. The INT formula in excel dialogue box will openwhere you can put the argument values to obtain a return value.

	A	B	C	D
1	Example	Formula	Result	
2	1	=INT(A1)	214	
3	2	=INT(A2)	3	
4	3	=INT(A3)	-4	
5	4	=INT(-3.6)	-4	
6				

INT Excel Function Errors

If you get any kind of error from the INT Excel Function, then it can be any one of the following.

- **#NAME?** – This error occurs when Excel does not recognize the text in the formula. You may have entered a wrong text in the syntax of the function.
- **#VALUE!** – If you enter a wrong type of argument in the syntax of the function, you will be getting this error in Microsoft Excel
- **.#REF!** – Microsoft Excel will display this error if the formula refers to a cell that is not valid.

Double

Numeric data type with float precision with double precision in calculations.

Code:

Dim x As Integer x = 5.5

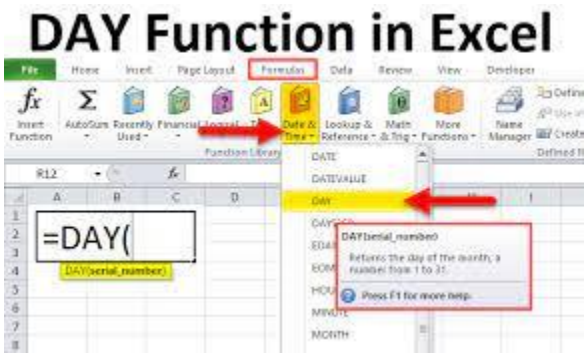
MsgBox "value is " & x

DATE Functions

- DAY
- MONTH
- YEAR
- TODAY()
- DAYS
- DATE

DAY Function

DAY function returns the day number from a valid date. As you know, in Excel, a date is a combination of day, month, and year, DAY function gets the day from the date and ignores the rest of the part.



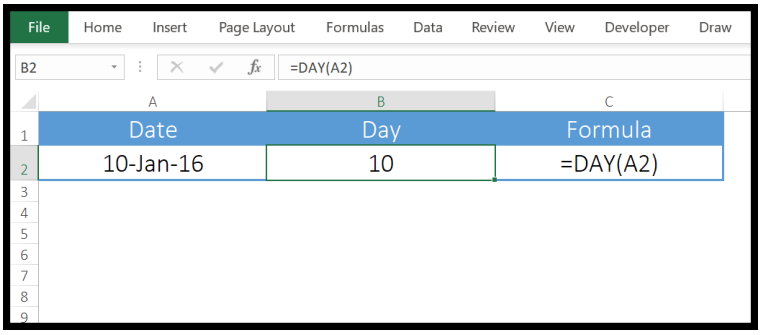
Syntax

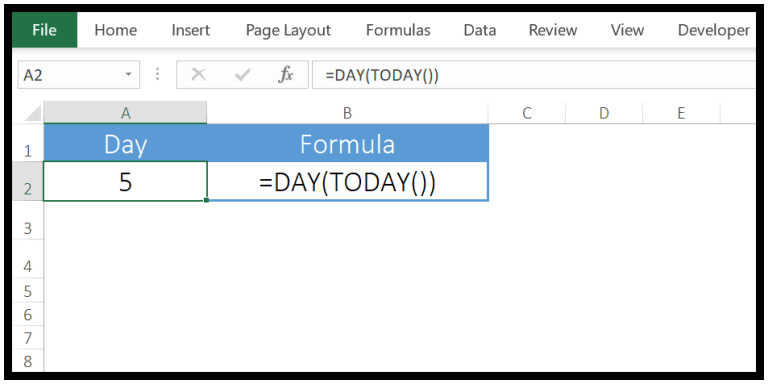
DAY(serial_number)

Arguments

serial_number: A valid serial number of the date from which you want to extract the day number.

Example





we have used DAY with TODAY to create a dynamic formula that returns the current day number and it will update every time you open your worksheet or when you recalculate your worksheet.

MONTH Function

MONTH function returns the month number (ranging from 0 to 12) from a valid date. As you know, in Excel, a date is a combination of day, month, and year, MONTH gets the month from the date and ignores the rest of the part.

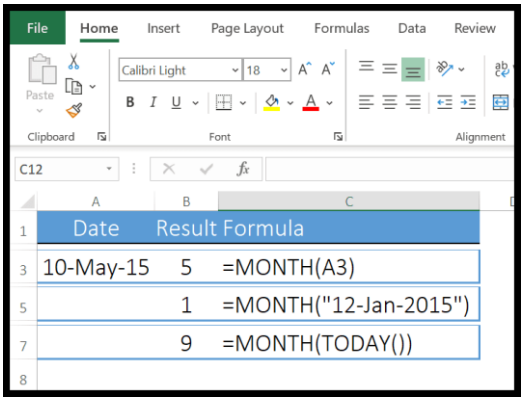
Syntax

MONTH(serial_number)

Arguments

serial_number: A valid date from which you want to get the month number.

Example



- In the FIRST example, we have simply used date and it has returned the 5 in the result which is the month number of MAY.
- In the SECOND example, we have supplied the date directly in the function.
- In the THIRD example, we have used the TODAY function to get the current date and MONTH has returned the month number from it.

YEAR Function

YEAR Function returns the year number from a valid date. As you know, in Excel a date is a combination of day, month, and year, and the YEAR function gets the year from the date and ignores the rest of the part.

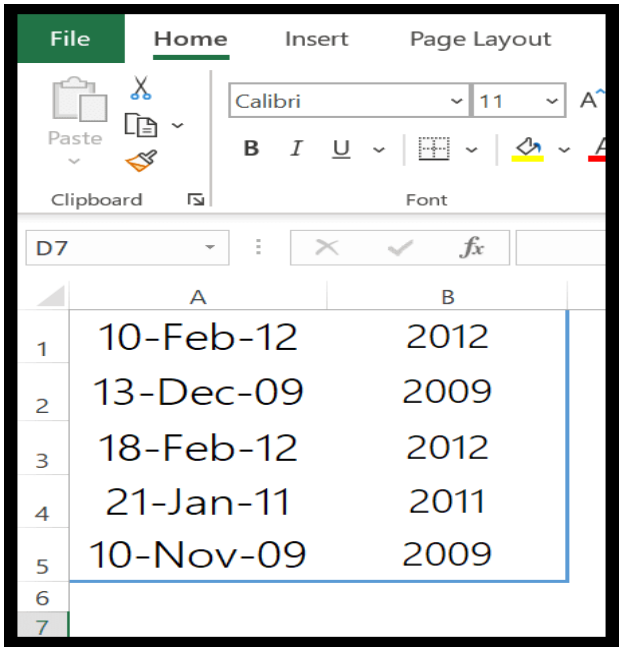
Syntax

YEAR(date)

Arguments

date: A date from which you want to get the year.

Example



we have used the year function to get the year number from the dates. You can use this function where you have dates in your data and you only need the year number.

Example(Day,Month,Year)

Date	Day	Month	Year
Today's Date	8	4	2022
30-04-2021	30	4	2021

TODAY Function

The TODAY function returns the current date and time as per the system's date and time. The date and time returned by the NOW function update continuously whenever you update anything in the worksheet.

Syntax

TODAY()

Arguments

In the TODAY function, there is no argument, all you need to do is enter it in the cell and hit enter, but be careful as TODAY is a volatile function which updates its value every time you update your worksheet calculations.

8	=DAY(TODAY())
4	=MONTH(TODAY())
2022	=YEAR(TODAY())

We have used TODAY with other functions to get the current month number, current year, and current day.

DAYS Function

DAYS function returns the difference between two dates. It takes a start date and an end date and then returns the difference between them in days. This function was introduced in Excel 2013 so not available in prior versions.

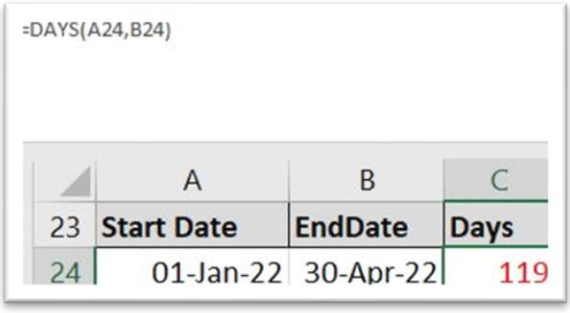
Syntax

DAYS(end_date,start_date)

Arguments

- start_date: It is a valid date from where you want to start the days' calculation.
- end_date: It is a valid date from where you want to end the days' calculation.

Example



we have referred the cell A24 as the start date and B24 as the end date and we have 119 days in the result.

DATE Function

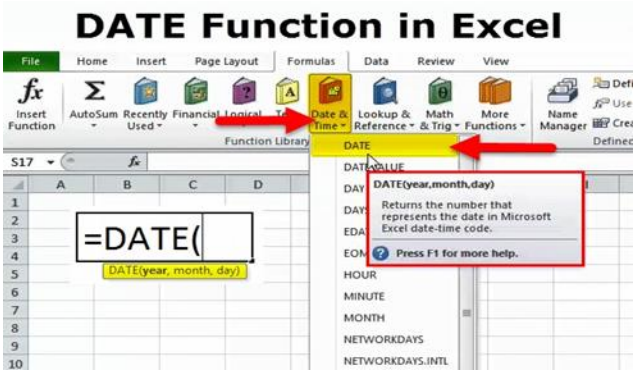
DATE function returns a valid date based on the day, month, and year you input. In simple words, you need to specify all the components of the date and it will create a date out of that.

Syntax

DATE(year,month,day)

Arguments

- year: A number to use as the year.
- month: A number to use as the month.
- day: A number to use as a day.



Example

	A	B	C	D
1	**Year**	**Month**	**Day**	**Result of Date**
2	1900	4	10	=DATE(A2,B2,C2)
3	2015	-1	8	=DATE(A3,B3,C3)
4	2015	0	10	=DATE(A4,B4,C4)
5	0	10	12	=DATE(A5,B5,C5)
6	2016	2	0	=DATE(A6,B6,C6)
7	2015	4	-1	=DATE(A7,B7,C7)

	A	B	C	D
1	Year	Month	Day	Result of Date
2	1900	4	10	10-04-1900
3	2015	-1	8	08-11-2014
4	2015	0	10	10-12-2014
5	0	10	12	12-10-1900
6	2016	2	0	31-01-2016
7	2015	4	-1	30-03-2015

TEXT Function

TEXT in Excel is used to convert a numeric value to a text string in a specific format.

The **syntax** for the Excel TEXT function is as follows:

TEXT(value, format_text)

Where:

- Value - the numeric value to be converted to text. It can be a number, date, reference to a cell containing a numeric value or another function that returns a number or date.
- Format_text - the format that you want to apply. It is supplied in the form of a format code enclosed in the quotation marks, e.g. "mm/dd/yy".

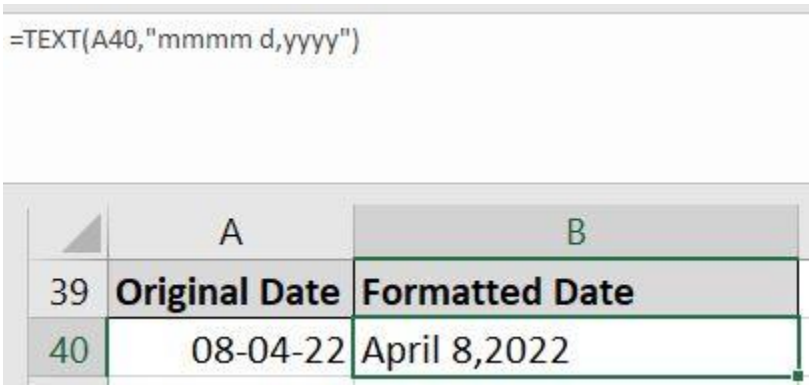
Generally, an Excel TEXT formula is used in the following situations:

- To display numbers in a more readable way or in a format that makes more sense for your users.
- To display dates in a specific format.
- To combine numbers or dates with certain text or characters.

For example, if you want to pull the date from cell A2 and show it in another cell in the traditional date format like "January 1, 2016", you use the following Excel TEXT formula:

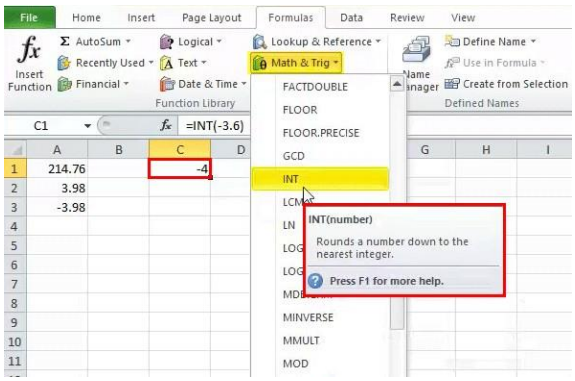
=TEXT(A2, "mmmm d, yyyy")

Example

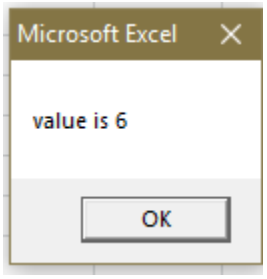


Output/Results snippet:

INT



Double



DATE Functions

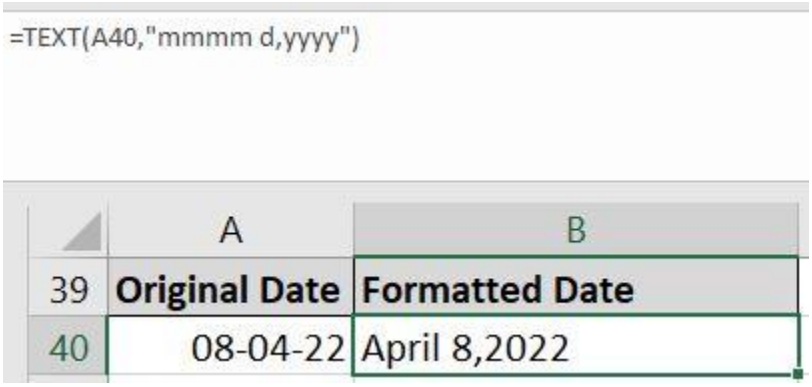
	A	B	C	D
1	**Year**	**Month**	**Day**	**Result of Date**
2	1900	4	10	=DATE(A2,B2,C2)
3	2015	-1	8	=DATE(A3,B3,C3)
4	2015	0	10	=DATE(A4,B4,C4)
5	0	10	12	=DATE(A5,B5,C5)
6	2016	2	0	=DATE(A6,B6,C6)
7	2015	4	-1	=DATE(A7,B7,C7)

	A	B	C	D
1	**Year**	**Month**	**Day**	**Result of Date**
2	1900	4	10	10-04-1900
3	2015	-1	8	08-11-2014
4	2015	0	10	10-12-2014
5	0	10	12	12-10-1900
6	2016	2	0	31-01-2016
7	2015	4	-1	30-03-2015

=DAYS(A24,B24)

	A	B	C
23	**Start Date**	**EndDate**	**Days**
24	01-Jan-22	30-Apr-22	119

TEXT



References:

- https://excelhub.org/how-to-use-excel-int-function/

Activity 2

Aim: Perform operations on different data types.

Learning outcome: Able to understand the Business Analytics

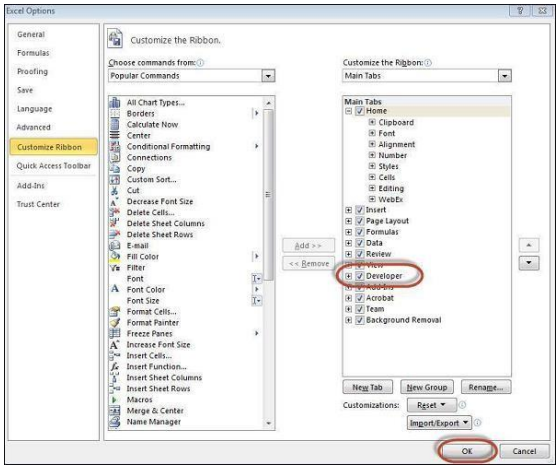
Duration: 5 Hours

List of Hardware/Software requirements:

1. Windows 7/Windows 10
2. MS Office 2010 with Excel or Latest Version
3. VBA Developer

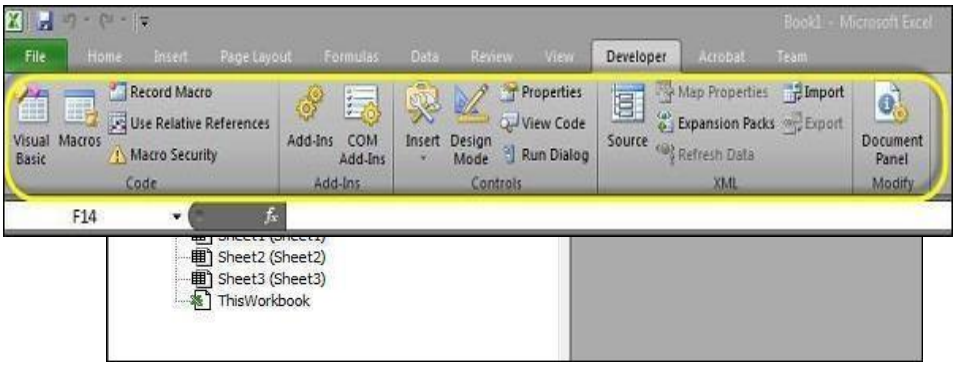
Code/Program/Procedure (with comments):

- Step 1 − First, enable the 'Developer' menu in Excel 20XX. To do the same, click File →Options.
- Step 2 − Click ‘Customize the Ribbon’ tab and check 'Developer'. Click 'OK'.



- Step 3 − The 'Developer' ribbon appears in the menu bar

- Step 4 − Click the 'Visual Basic' button to open the VBA Editor.



Datatypes in Excel

Data type	Length	Description
BYTE	1 byte	Number from 0 to 255 for storing binary data
INTEGER	2	Integer from -32 768 to 32 767.
LONG	4	Integer from -2 147 483 648 to 2 147 483 647
SINGLE	4	Numeric data type with float precision to 6 decimal digits
DOUBLE	8	Numeric data type with float precision with double precision in calculatons
CURRENCY	8	A number with fixed 4 decimal digits
DECIMAL	14	Numeric data type with fixed precision and scale (accuracy upto 28).
STRING		Text strings. Flexible length or 64 kilobytes
BOOLEAN	2	Logical value (true or false)
DATE	8	The date in the range from 1.1.100 to 31.12.9999
OBJECT	4	Reference to an object.
VARIANT	16	Basic type. May contain special value Null, numeric value, text, reference to object or variable array.

Code

Private Sub cmdCalculate_Click()

```
Dim number1, number2, number3 As Single
'Declare Variables
```

```
Dim total, average As Double
number1 = Cells(1, 1).Value
 number2 = Cells(2, 1).Value
 number3 = Cells(3, 1).Value ‘
Total of 3 Values
total = number1 + number2 + number3
average = total / 3
‘Display Total
Cells(5, 1) = "Total:-" & total
‘Display Average
Cells(6, 1) = "Average:-" & average
End Sub
```

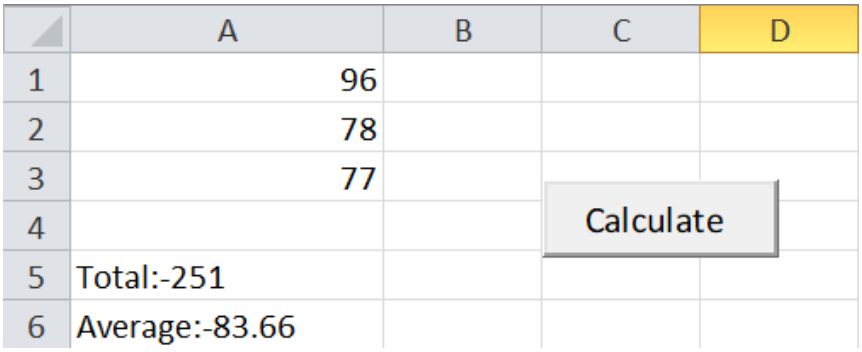
Code

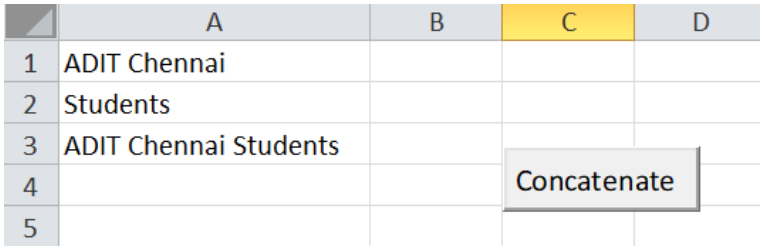
```
Private Sub cmdConcatenate_Click()
‘Declare the String VariablesDim
firstName As String
Dim lastName As String
Dim yourName As String
```

```
firstName = Cells(1, 1).Value

lastName = Cells(2, 1).Value

‘Concatenate with firstName and lastName
yourName = firstName + " " + lastName

‘Result of fullName

Cells(3, 1) = yourName
```

End Sub

Output/Results snippet:





References:

- https://www.automateexcel.com/vba/data-types-variables-constants/

Activity 3

Aim: Segregate data in different sheets.

Learning outcome: Able to understand the Business Analytics

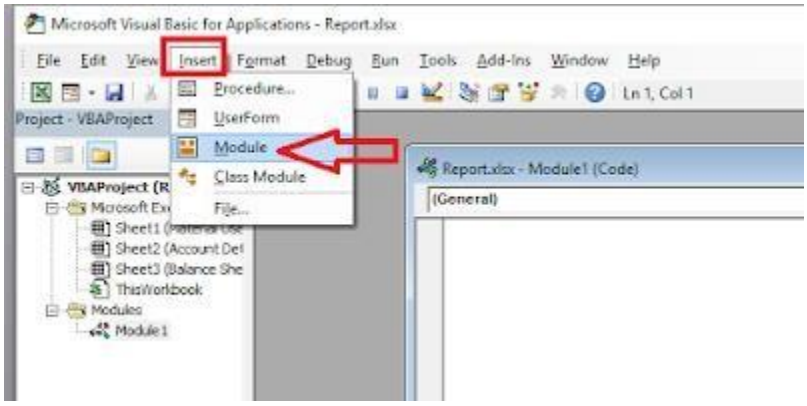
Duration: 5 Hours

List of Hardware/Software requirements:

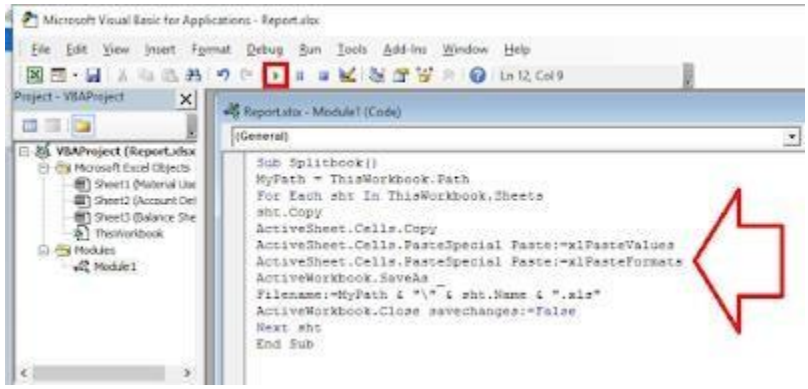
1. Windows 7/Windows 10
2. MS Office 2010 with Excel or Latest Version
3. VBA Developer

Code/Program/Procedure (with comments):

1. Open Excel File and press “Alt+F11”



2. Click on “Insert” than click on “Module



Code

Sub Splitbook()

MyPath = ThisWorkbook.Path

For Each sht In ThisWorkbook.Sheets

‘Copy of the Sheet

sht.Copy ActiveSheet.Cells.Copy

ActiveSheet.Cells.PasteSpecial Paste:=xlPasteValues

ActiveSheet.Cells.PasteSpecial Paste:=xlPasteFormats

‘Save the Sheet Data

ActiveWorkbook.SaveAs _

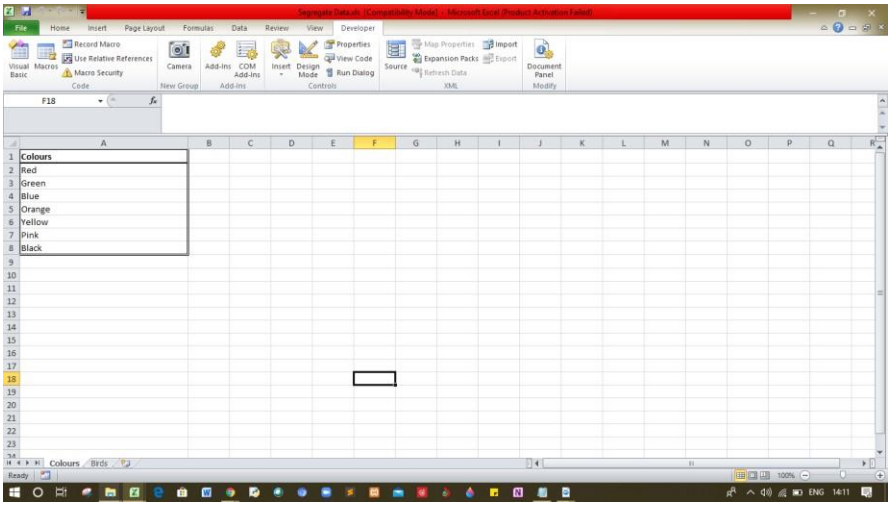
Filename: =MyPath & "\" & sht.Name & ".xls"

ActiveWorkbook.Close savechanges:=False

Next sht

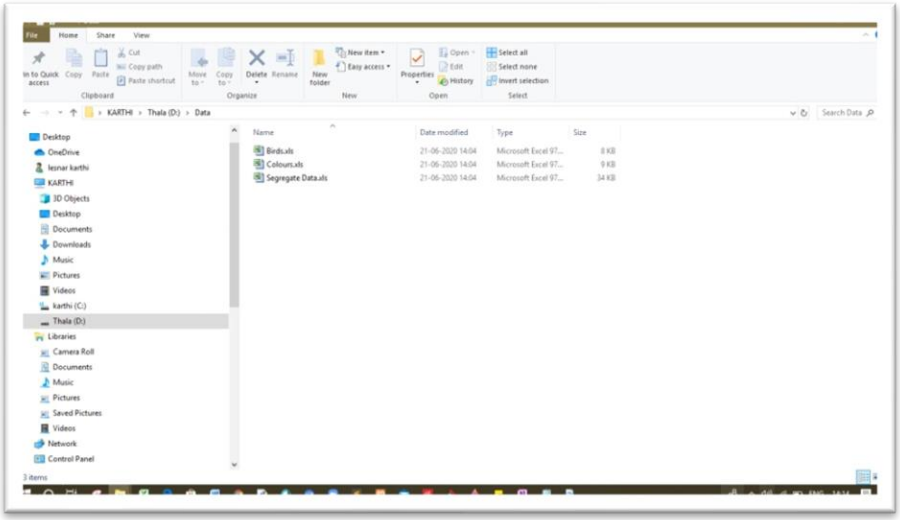
End Sub

Click on “Run” icon



Output/Results snippet:

That's it you're each worksheet will be converted into separate excel file



References:

- http://www.bsocialshine.com/2017/08/how-to-split-each-excel-sheet-into.html

Activity 4

Aim: Calculate arithmetic mean, geometric mean and Harmonic mean (5Hrs)

Learning outcome: Able to understand the Business Analytics

Duration: 5 Hours

List of Hardware/Software requirements:

1. Windows 7/Windows 10
2. MS Office 2010 with Excel or Latest Version

Code/Program/Procedure (with comments):

Arithmetic Mean on Excel

Arithmetic Mean, commonly used term in statistics, is the average of the numerical values set and is calculated by firstly calculating the sum of number in the set and then dividing resultant by count of those numbers.

Arithmetic Mean formula on Excel

$$\text{Arithmetic Mean} = (x_1 + x_2 + ... + x_n) / n$$

or

$$\text{Arithmetic Mean} = \sum x_i / n$$

Where,

x1, x2, x3, xn are the observations

n is the number of observations

Alternatively, it can be symbolically written as shown below-

$$\frac{1}{n} * \sum_{i=1}^{n} xi$$

In the above Equation, the symbol ∑ is known as sigma. It implies the summation of the values

Example

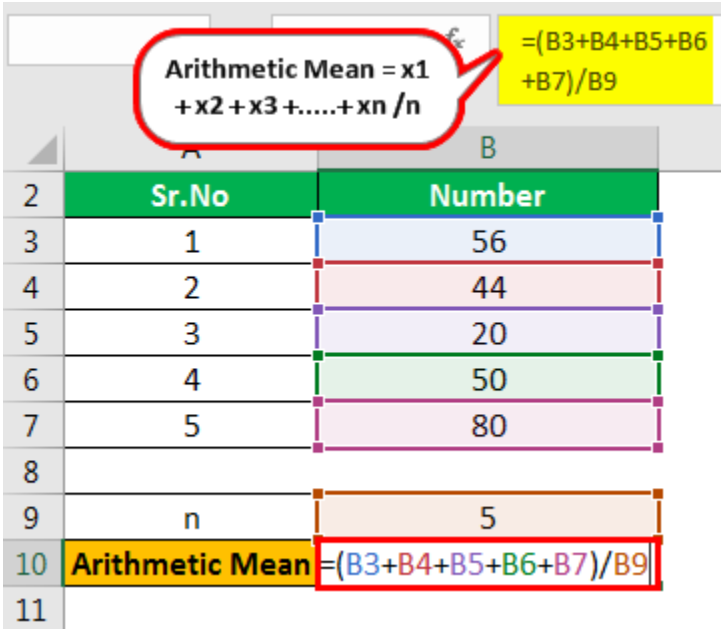
There are five observations. These are 56, 44, 20, 50, 80. Find their arithmetic mean.

Solution

Here, the observations are 56, 44, 20, 50, 80.

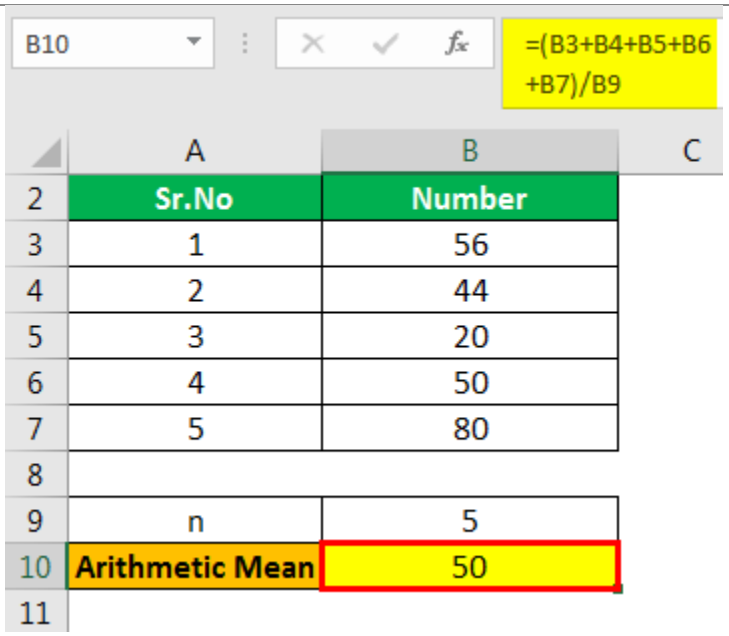
n = 5

Therefore, the calculation is as follows,



	A	B
2	Sr.No	Number
3	1	56
4	2	44
5	3	20
6	4	50
7	5	80
8		
9	n	5
10	Arithmetic Mean	=(B3+B4+B5+B6+B7)/B9
11		

=56+44+20+50+80/5



	A	B	C
2	Sr.No	Number	
3	1	56	
4	2	44	
5	3	20	
6	4	50	
7	5	80	
8			
9	n	5	
10	Arithmetic Mean	50	
11			

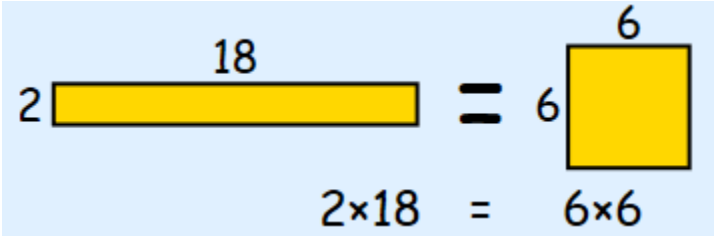
Arithmetic Mean=**50**

The GEOMEAN function

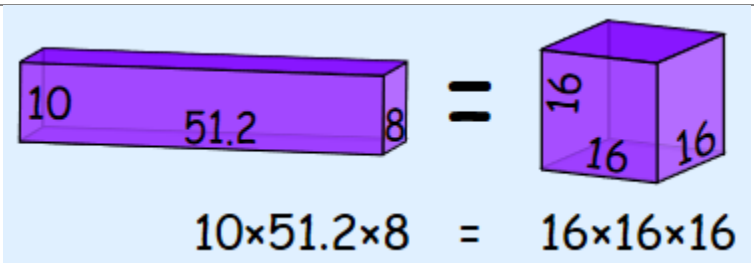
In mathematics, the geometric mean is a mean or average, which indicates the central tendency or typical value of a set of numbers by using the product of their values (as opposed to the arithmetic mean which uses their sum). The geometric mean is defined as the **n**th root of the product of **n** numbers, *i.e.* for a set of numbers $\mathbf{x_1}$, $\mathbf{x_2}$, ..., $\mathbf{x_n}$, the geometric mean is defined as

$$\left(\prod_{i=1}^{n} x_i\right)^{\frac{1}{n}} = \sqrt[n]{x_1 x_2 \cdots x_n}$$

In two dimensions, it is the equivalent of finding the equivalent square with the same area as the rectangle given by the two dimensions cited:



In three dimensions, it is the equivalent of finding the equivalent cube with the same volume as the given hexahedron with the three dimensions cited:



The idea continues in **n** dimensions.

The Excel function **GEOMEAN** returns the geometric mean of an array or range of positive data. For example, you can use **GEOMEAN** to calculate average growth rate given compound interest with variable rates. It has the following syntax:

GEOMEAN(number1, [number2], ...)

The **GEOMEAN** function has the following arguments:

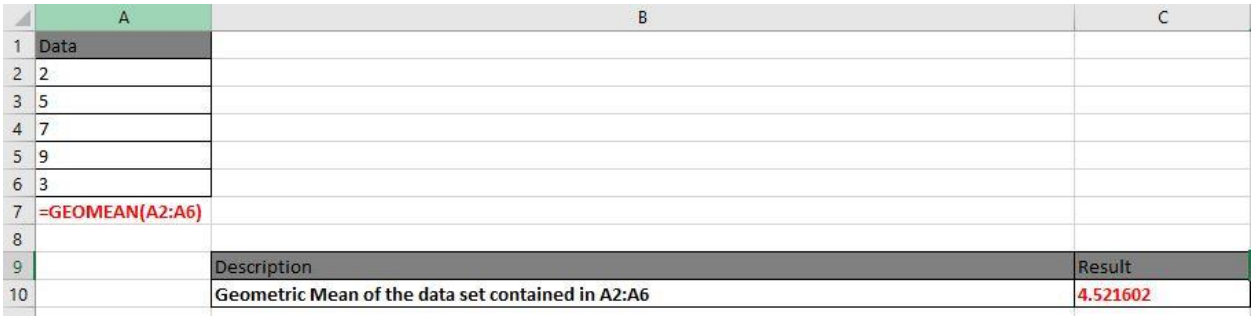
- **number1**, **number2**,...where **number1** is required, and subsequent numbers are optional. There can be between one (1) and 255 numbers. You can also use a single array or a reference to an array instead of arguments separated by commas.

It should be further noted that:

- arguments can either be numbers or names, arrays, or references that contain numbers
- logical values and text representations of numbers that you type directly into the list of arguments are counted
- of an array or reference argument contains text, logical values or empty cells, those values are ignored; however, cells with the value zero are included
- arguments that are error values or text that cannot be translated into numbers cause errors
- if any data point ≤ 0, **GEOMEAN** returns the *#NUM!* error value
- the equation for the geometric mean is:

$$GM_{\bar{y}} = \sqrt[n]{y_1 y_2 y_3 \cdots y_n}$$

Example



	A	B	C
1	Data		
2	2		
3	5		
4	7		
5	9		
6	3		
7	=GEOMEAN(A2:A6)		
8			
9		Description	Result
10		Geometric Mean of the data set contained in A2:A6	4.521602

Harmonic mean on Excel

The Excel HARMEAN function returns the harmonic mean for a set of numeric values. The harmonic mean is the reciprocal of the arithmetic mean of reciprocals. Harmonic mean can beused to calculate a mean that reduces the impact of outliers.

Harmonic mean Formula on Excel

Harmonic Mean = n / (1/X1 + 1/X2 + 1/X3 ..... 1/Xn)

=HARMEAN (number1, [number2], ...)

- number1 - First value or reference.
- number2 - [optional] Second value or reference.Where:

X1, X2,…Xn – Data Points

n – Total number of data points

When to use harmonic mean?

This average is used when data values are expressed in relative units, e.g. speed unit (miles/ h) or salary per hour of work (USD / h).

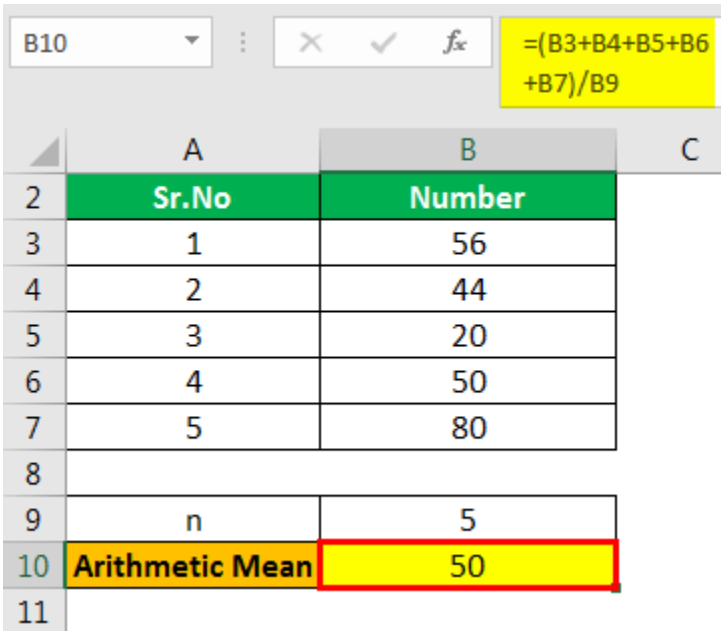
The use of the harmonic mean gives equal weight to each data. Using the arithmetic mean in this case would give more weight to the higher-valued data, and thus the mean would be overstated.

Example

	A	B
1	**Employee**	**Salary/h**
2	Employee 1	17
3	Employee2	19
4	Employee 3	25
5	Employee 4	16.7
6	Employee 5	34.2
7	**Harmonic Mean**	**=HARMEAN(B2:B6)**
8	**Result**	20.78353

Output/Results

Arithmetic mean



Geometric Mean

	A	B	C
1	Data		
2	2		
3	5		
4	7		
5	9		
6	3		
7	=GEOMEAN(A2:A6)		
8			
9		Description	Result
10		Geometric Mean of the data set contained in A2:A6	4.521602

Harmonic Mean

	A	B
1	**Employee**	**Salary/h**
2	Employee 1	17
3	Employee2	19
4	Employee 3	25
5	Employee 4	16.7
6	Employee 5	34.2
7	**Harmonic Mean**	**=HARMEAN(B2:B6)**
8	**Result**	20.78353

References:

- https://www.educba.com/arithmetic-mean-formula/
- https://www.exceltip.com/statistical-formulas/excel-geomean-function.html
- https://www.educba.com/harmonic-mean-formula/

Activity 5

Aim: Calculate median from raw & grouped data

Learning outcome: Able to understand the Business Analytics

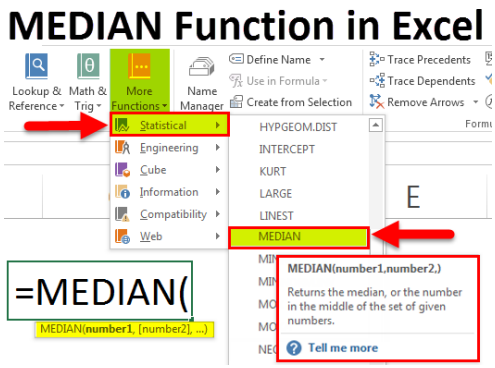
Duration: 5 Hours

List of Hardware/Software requirements:

1. Windows 7/Windows 10
2. MS Office 2010 with Excel or Latest Version

Code/Program/Procedure (with comments):

Median Function in Excel



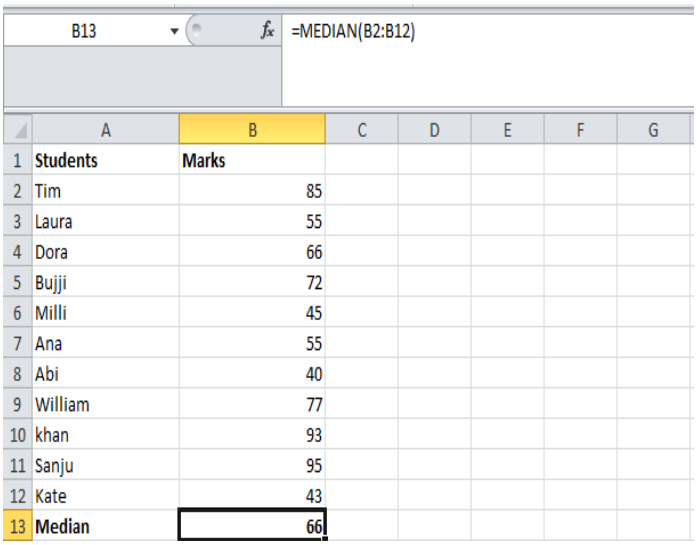
1. Number 1 (required argument) – The number arguments are a set of single or more numeric values (or arrays of numeric values), for which you want to calculate the median.

2. Number 2 (optional argument)

Median is a function which is used to find the middle number in each range of numbers. When you are finding median manually, you need to sort the data in ascending order but in Excel, you can simply use the Median function and select the range and you will find your median. We take the same example as above to find the median of marks obtained by students. So, we use = MEDIAN (B2: B12).

Output/Results snippet:

Median



	A	B	C	D	E	F	G
1	**Students**	**Marks**					
2	Tim	85					
3	Laura	55					
4	Dora	66					
5	Bujji	72					
6	Milli	45					
7	Ana	55					
8	Abi	40					
9	William	77					
10	khan	93					
11	Sanju	95					
12	Kate	43					
13	**Median**	66					

References:

- https://www.educba.com/excel-median-function/

Activity 6

Aim: Calculate mode for row & grouped data

Learning outcome: Able to understand the Business Analytics

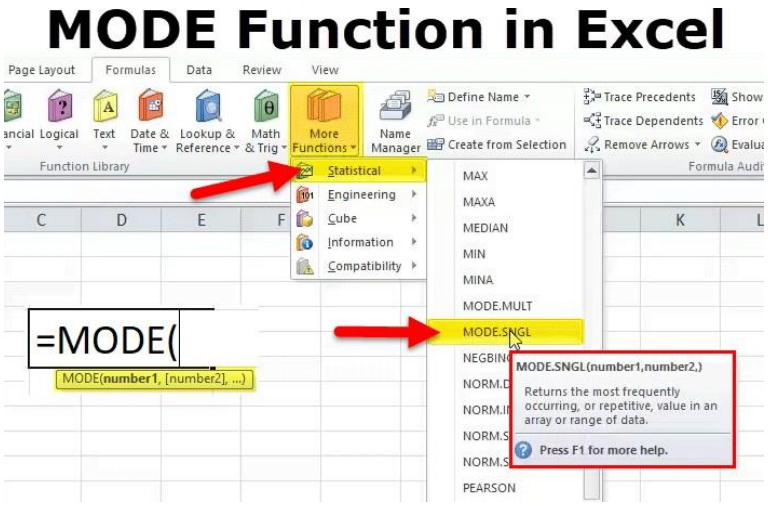
Duration: 5 Hours

List of Hardware/Software requirements:

1. Windows 7/Windows 10
2. MS Office 2010 with Excel or Latest Version

Code/Program/Procedure (with comments):

MODE in Excel



1. number1 (compulsory OR required argument) – Arrays of cell reference or numericvalues (set of one or more numeric values) for which we have to calculate the mode.

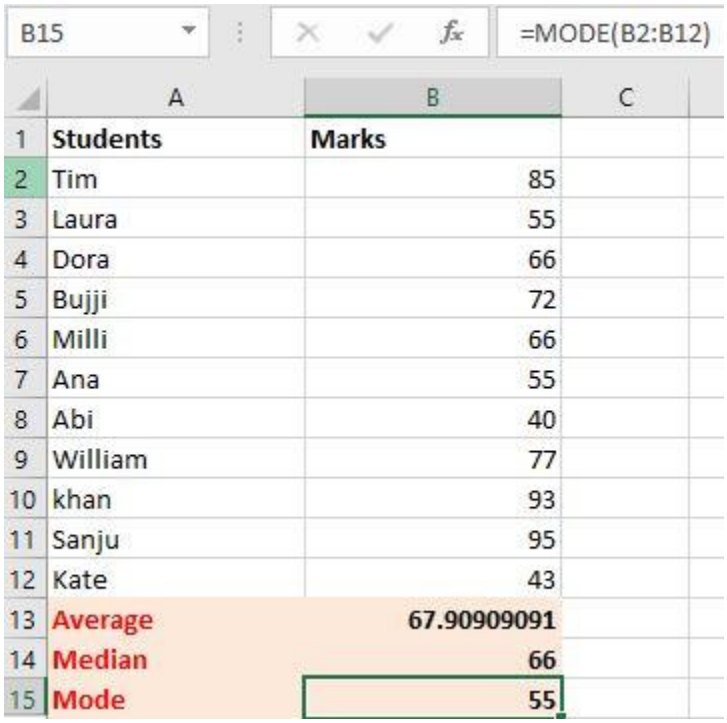
2. number2 (Optional OR not required) – Arrays of cell reference or numeric values (set ofone or more numeric values) for which we have to calculate the mode.

Mode helps you to find out the value that occurs the most number of times. When you are workingon a large amount of data, this function can be a lot of help. To find the most occurring value in

Excel, use the MODE function and select the range you want to find the mode of. In our examplebelow, we use =MODE (B2: B12) and since 2 students have scored 55, we get the answer as 55.

Output/Results snippet:

Mode



	A	B	C
1	**Students**	**Marks**	
2	Tim	85	
3	Laura	55	
4	Dora	66	
5	Bujji	72	
6	Milli	66	
7	Ana	55	
8	Abi	40	
9	William	77	
10	khan	93	
11	Sanju	95	
12	Kate	43	
13	**Average**	**67.90909091**	
14	**Median**	**66**	
15	**Mode**	**55**	

References:

- https://www.educba.com/mode-in-excel/

Learning Outcome

After completing this module, the student should be **able to understand business analytics and develop business intelligence.**

To meet the learning outcome, a student has to complete the following activities

1. Calculate standard deviation for set of data
2. Calculate standard variance for a set of data
3. Using VLOOKUP in excel for searching operation
4. Plot basic charts in excel over numeric data series
5. Plot uniform and binomial distributions in excel
6. Implement Central limit theorem in excel
7. Generate data table and find chi-square analysis

Activity 1

Aim: Calculate standard deviation for set of data (2.5Hrs)

Learning outcome: Able to business analytics and develop business intelligence.

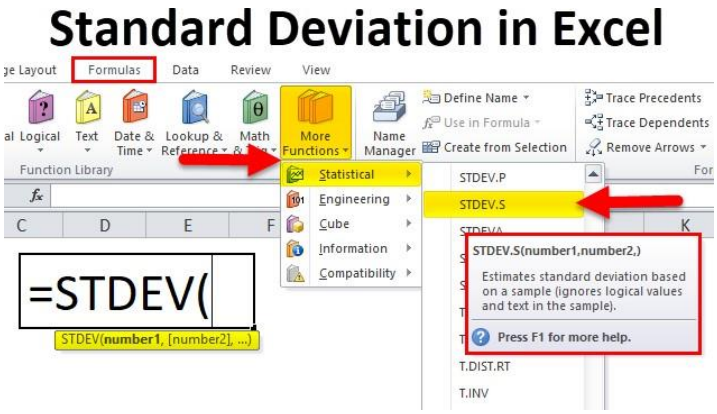
Duration: 2.5 hour

List of Hardware/Software requirements:

1. Windows 7/Windows 10
2. MS Office 2010 with Excel or Latest Version

Code/Program/Procedure (with comments):

Standard Deviation in Excel



Standard deviation formula in Excel

i. number1: (Compulsory or mandatory argument) It is the first element of the sample of a

ii. number2: (Optional argument) It is a number of arguments from 2 to 254 corresponding to a sample of a population.

The standard deviation in Excel helps you to understand, how much your values deviate from the Average or Mean that is it tells you whether your data is somewhere close to the average or fluctuates a lot. If the value received is on the higher side then that means that your data has a lot

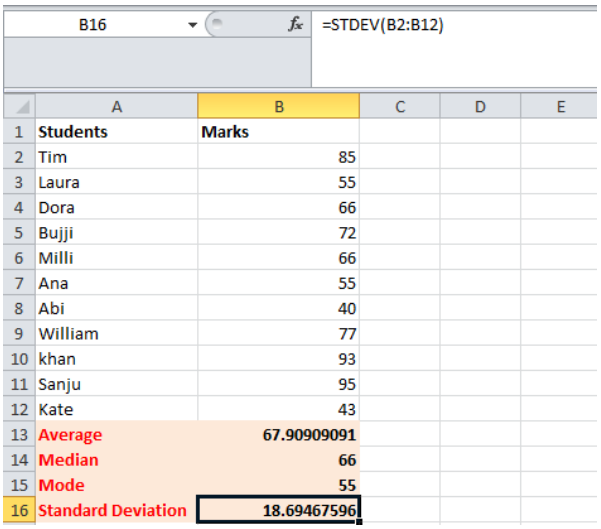
of fluctuations and vice versa. To calculate standard deviation in excel we use the STDEV function. In the same example, we shall use the STDEV function so our formula will be

= STDEV (B2: B12).

Our answer is around 20 which indicates that the marks of the students fluctuate a lot

Output/Results snippet:

Standard Deviation



	A	B	C	D	E
1	**Students**	**Marks**			
2	Tim	85			
3	Laura	55			
4	Dora	66			
5	Bujji	72			
6	Milli	66			
7	Ana	55			
8	Abi	40			
9	William	77			
10	khan	93			
11	Sanju	95			
12	Kate	43			
13	**Average**	**67.90909091**			
14	**Median**	**66**			
15	**Mode**	**55**			
16	**Standard Deviation**	**18.69467596**			

References:

- https://www.educba.com/sample-standard-deviation-formula/

Activity 2

Aim: Calculate standard variance for a set of data (2.5Hrs)

Learning outcome: Able to business analytics and develop business intelligence.

Duration: 2.5 hour

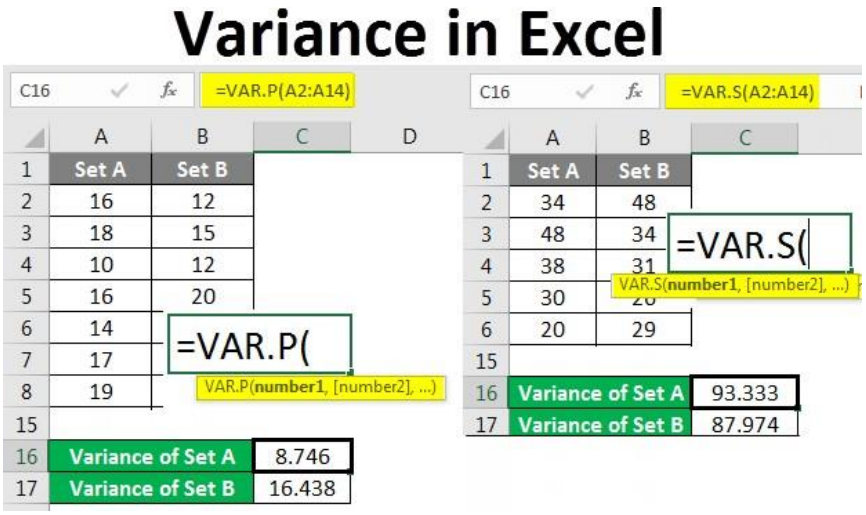
List of Hardware/Software requirements:

1. Windows 7/Windows 10
2. MS Office 2010 with Excel or Latest Version

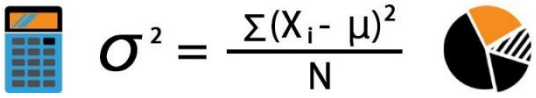
Code/Program/Procedure (with comments):

Variance in Excel

The term “variance” refers to the extent of dispersion of the data points of a data set from its mean, which is computed as the average of the squared deviation of each data point from the population mean.



Variance Formula



Variance formula in Excel

The formula for a variance can be derived by summing up the squared deviation of each data point and then dividing the result by the total number of data points in the data set.

Mathematically, it is represented as,

σ2 = ∑ (Xi – μ)^2 / N

where,

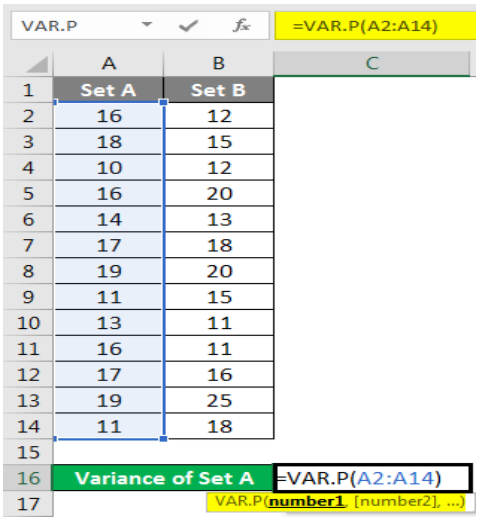
Xi = ith data point in the data set μ = Population mean

N = Number of data points in the population

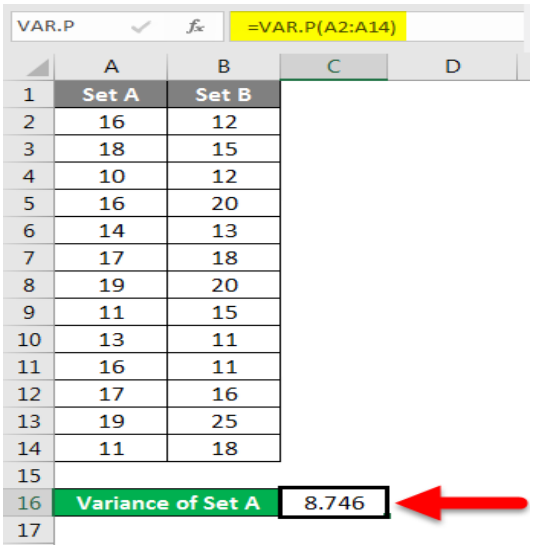
- Step 1 – Enter the data set in the columns.

	A	B	C
1	Set A	Set B	
2	16	12	
3	18	15	
4	10	12	
5	16	20	
6	14	13	
7	17	18	
8	19	20	
9	11	15	
10	13	11	
11	16	11	
12	17	16	
13	19	25	
14	11	18	
15			

- Step 2 – Insert the VAR.P function and choose the range of the data set. Here one thing should be noted that if any cell has an error, then that cell will be ignored.



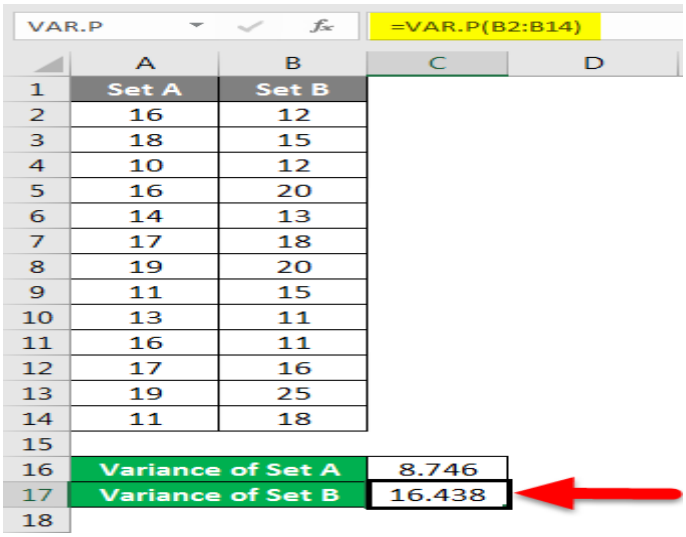
- Step 3 – After pressing the Enter key we will get the variance.



We have calculated the variance of Set B by following the same steps given above.

Output/Results snippet:

Var.p



	A	B	C	D
1	Set A	Set B		
2	16	12		
3	18	15		
4	10	12		
5	16	20		
6	14	13		
7	17	18		
8	19	20		
9	11	15		
10	13	11		
11	16	11		
12	17	16		
13	19	25		
14	11	18		
15				
16	Variance of Set A		8.746	
17	Variance of Set B		16.438	
18				

References:

- https://www.educba.com/excel-variance/

Activity 3

Aim: Calculate standard variance for a set of data (5Hrs)

Learning outcome: Able to business analytics and develop business intelligence.

Duration: 5 hours

List of Hardware/Software requirements:

1. Windows 7/Windows 10
2. MS Office 2010 with Excel or Latest Version

Code/Program/Procedure (with comments):

VLOOKUP function

The VLOOKUP function in Excel is a tool for looking up a piece of information in a table or data set and extracting some corresponding data/information. In simple terms, the VLOOKUP function says the following to Excel: "Look for this piece of information (e.g., bananas), in this data set (a table), and tell me some corresponding information about it (e.g., the price of bananas)".

VLOOKUP Formula

=VLOOKUP(lookup_value, table_array, col_index_num, [range_lookup])

To translate this to simple English, the formula is saying, "Look for this piece of information, in the following area, and give me some corresponding data from another column".

The VLOOKUP function uses the following arguments:

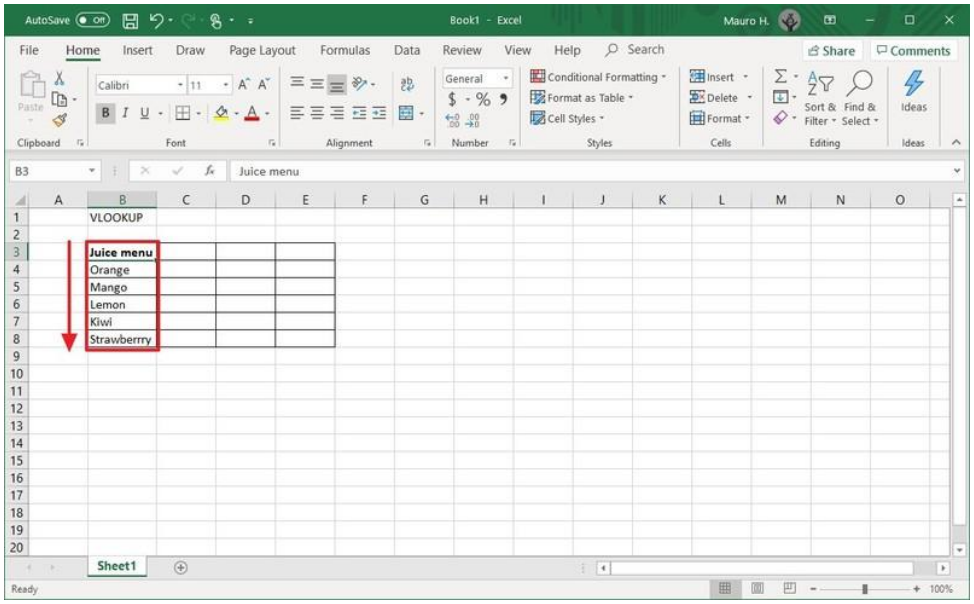
- Lookup_value (required argument) – Lookup_value specifies the value that we want to look up in the first column of a table.
- Table_array (required argument) – The table array is the data array that is to be searched. The VLOOKUP function searches in the left-most column of this array.
- Col_index_num (required argument) – This is an integer, specifying the column number of the supplied table_array, that you want to return a value from.
-

- Range_lookup (optional argument) – This defines what this function should return in the event that it does not find an exact match to the lookup_value. The argument can be set to TRUE or FALSE, which means:
- TRUE – Approximate match, that is, if an exact match is not found, use the closest match below the lookup_value.
- FALSE – Exact match, that is, if an exact match not found, then it will return an error.

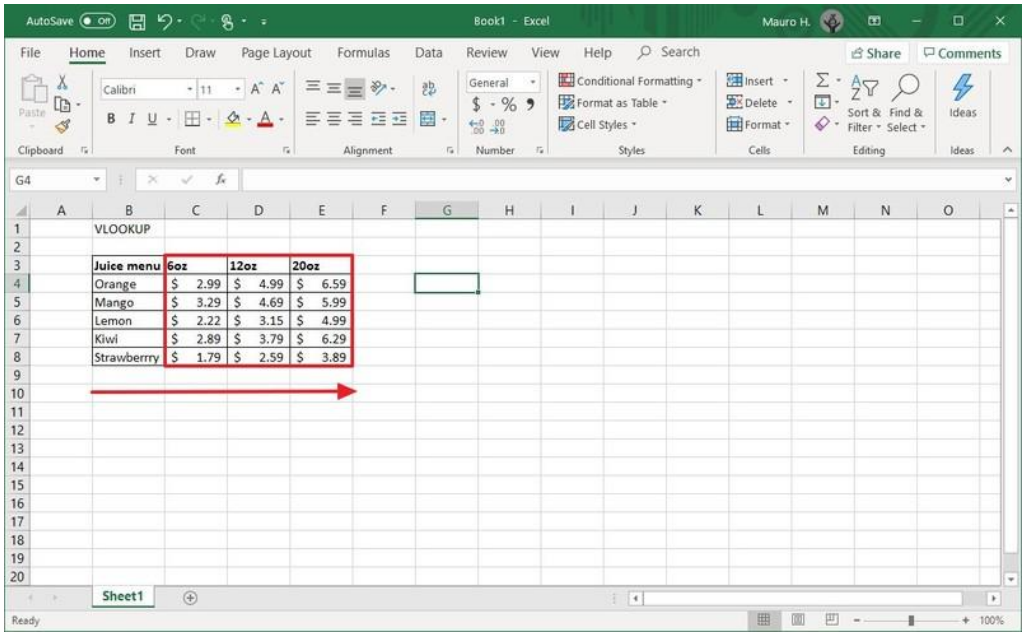
Write VLOOKUP function in Excel

To write a VLOOKUP function manually in Excel, use these steps:

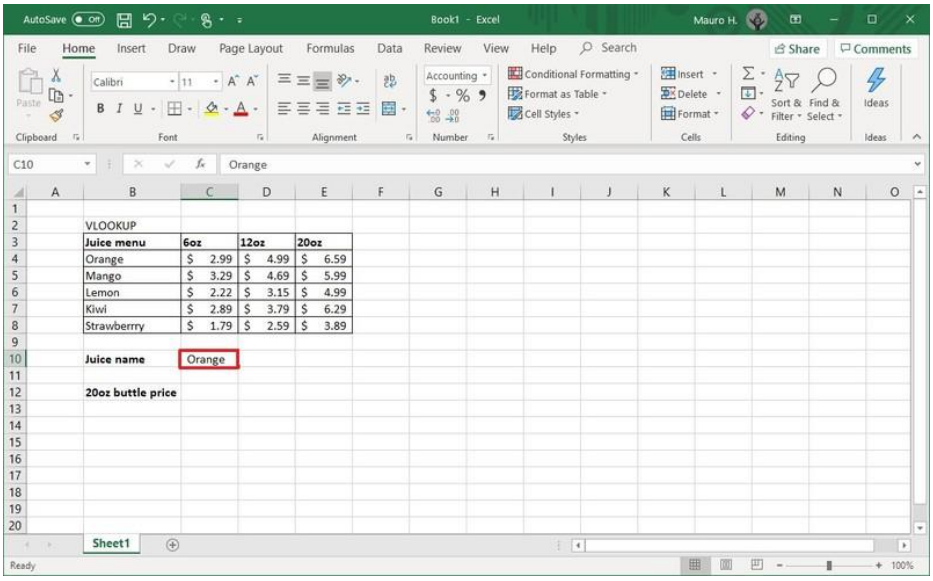
1. Open Excel.
2. Create the first column with items that will work as unique identifiers (required).



3. Create one or more additional columns (on the right side) with the different values for each item from the first column (on the left side).

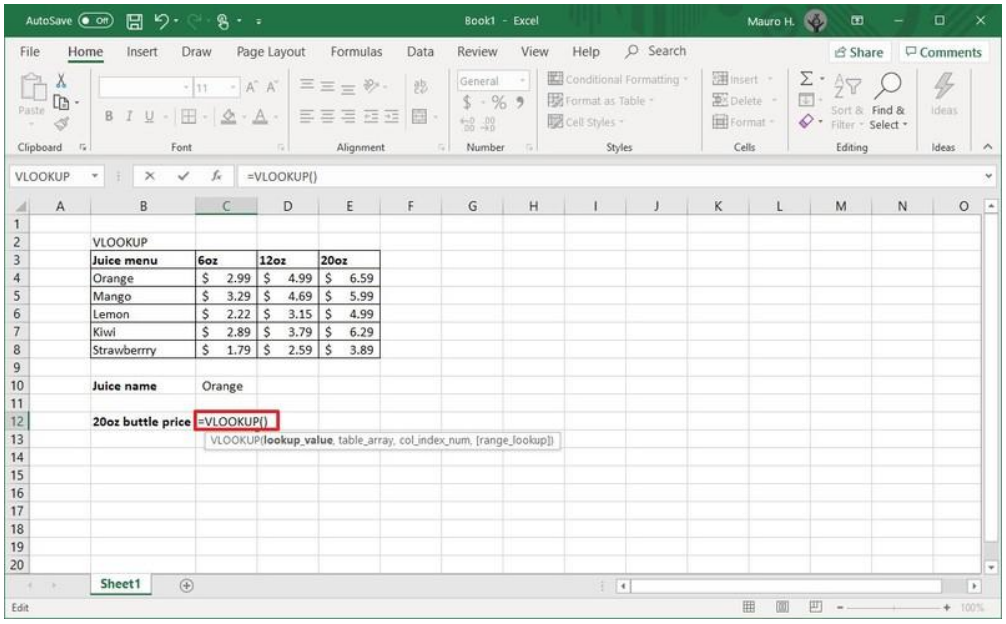


4. Select an empty cell in the spreadsheet and specify the name of the item you want to find an answer to—for example, Orange.



5. Select an empty cell to store the formula and returned value.

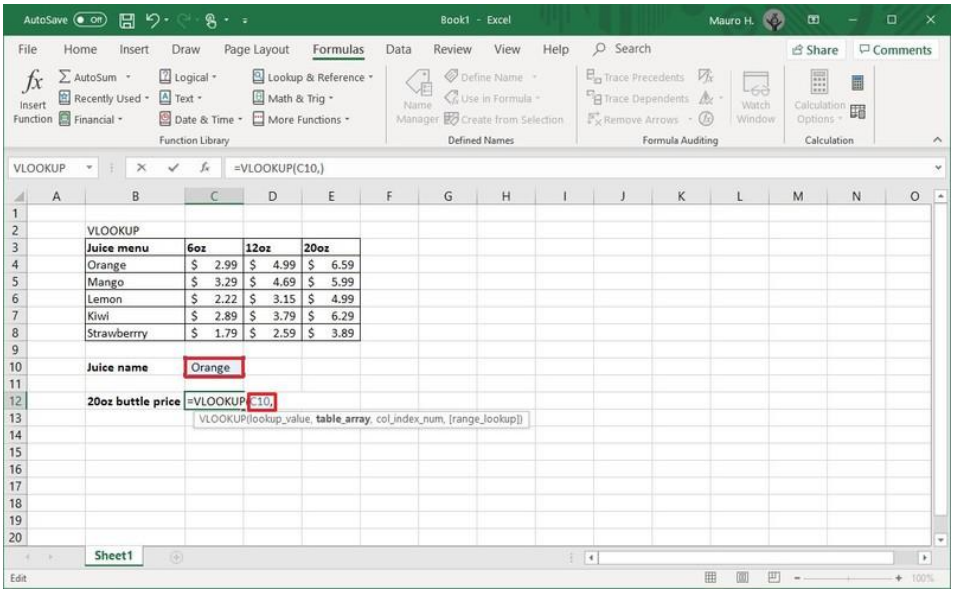
6. In the empty cell, type the following syntax to create a VLOOKUP formula and press Enter: =VLOOKUP()



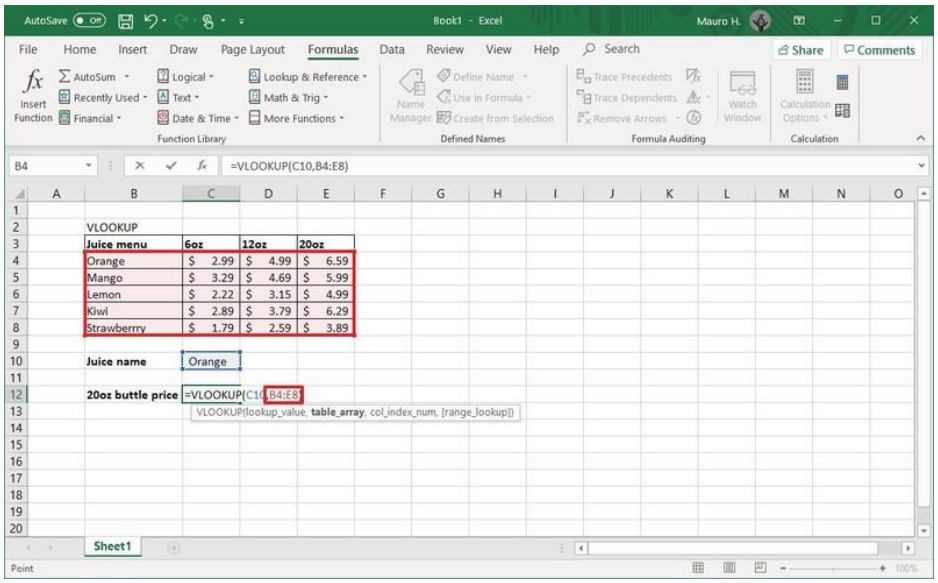
7. Type the following arguments inside the parenthesis "()" to write the function and press Enter:

=VLOOKUP(lookup_value,table_array,col_index_num,range_lookkup)

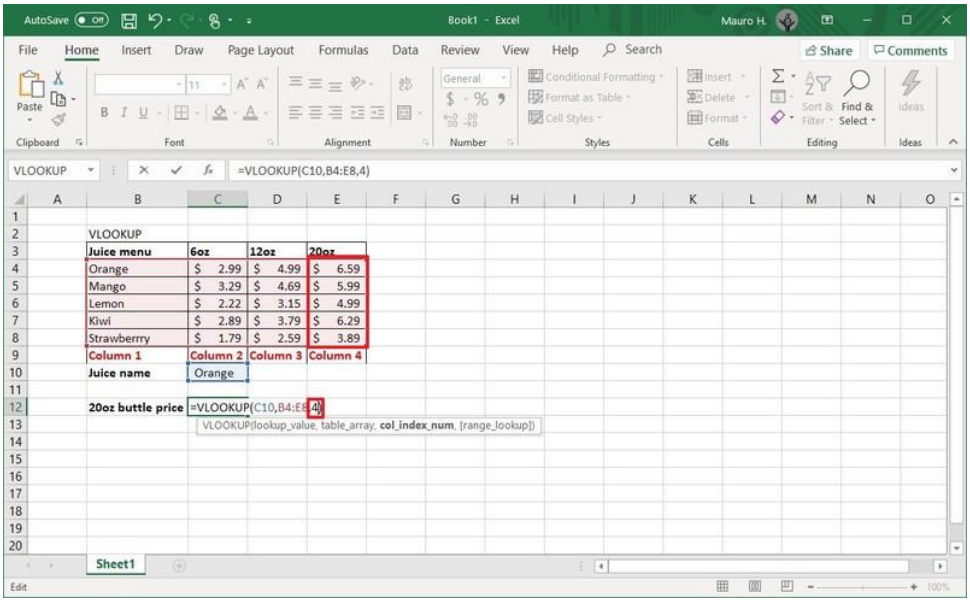
- lookup_value: defines the cell that includes the product identifier from the first column on the left.



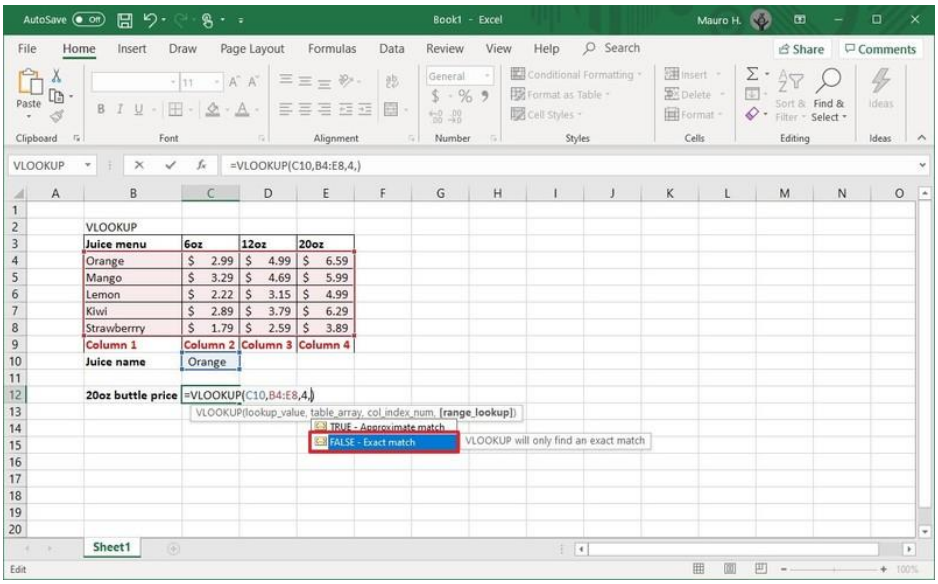
- table_array: defines the range of data where you want to perform a search. Typically, you would select the entire Excel table.



- col_index_num: defines the column number that the function will look to find a value. When specifying multiple columns, you should do from left to right.



- range_lookkup: includes two options: "false" for exact match or "true" for an approximate match. Usually, you want to use the false option.

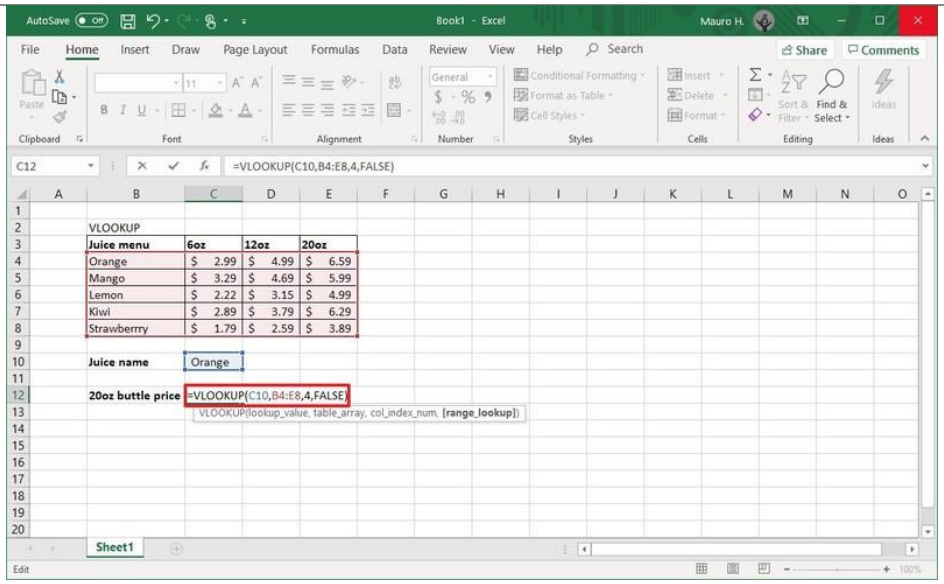


- Quick note: If you don't specify a value, then the "true" option will be applied by default. Sometimes, when using the "true" option, the first column needs to be shorted, which may cause an unexpected result. If you're not getting the correct value, you should use the "false" option or sort the first column alphabetically or numerically.

In the command, make sure to update the variables inside the parenthesis with the information you want to query. Also, remember to use a comma to separate each value in the function. You do not need a space between each comma.

Here's an example that returns the price for the 20oz bottle of orange juice:

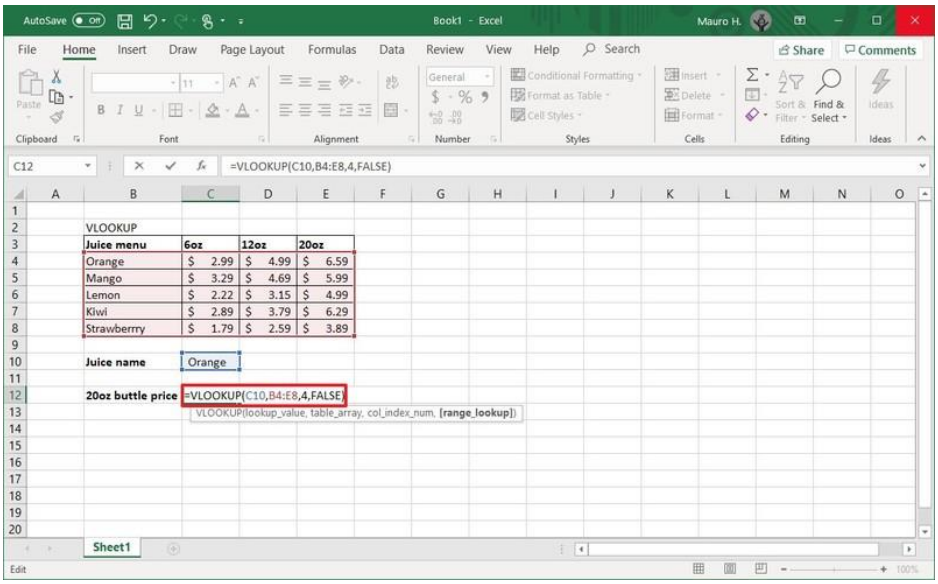
=VLOOKUP(C10,B4:E8,4,FALSE)



Once you complete the steps, the feature will return the value for the item you specified on step No. 4. If you receive the "#NAME?" error value, then it means that the formula is missing one or multiple quotes.

If you are trying to find data for another item, update the name of the cell on step No. 4. For example, if you want to see the price for the "20oz" bottle of Kiwi juice, then replace "Orange" with "Kiwi" in the "lookup_value" cell and press Enter to update the result.

Output/Results snippet:



References:

- https://www.windowscentral.com/how-use-vlookup-function-excel-office

Activity 4

Aim: Plot basic charts in excel over numeric data series (5Hrs)

Learning outcome: Able to business analytics and develop business intelligence.

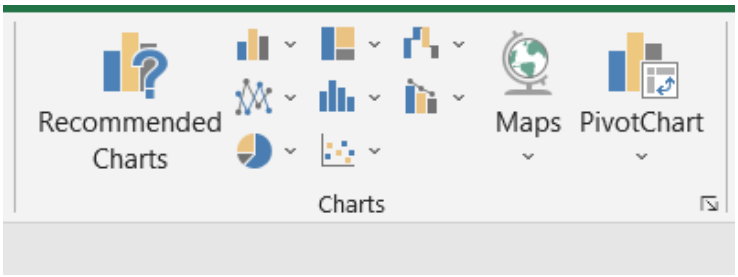
Duration: 5 hours

List of Hardware/Software requirements:

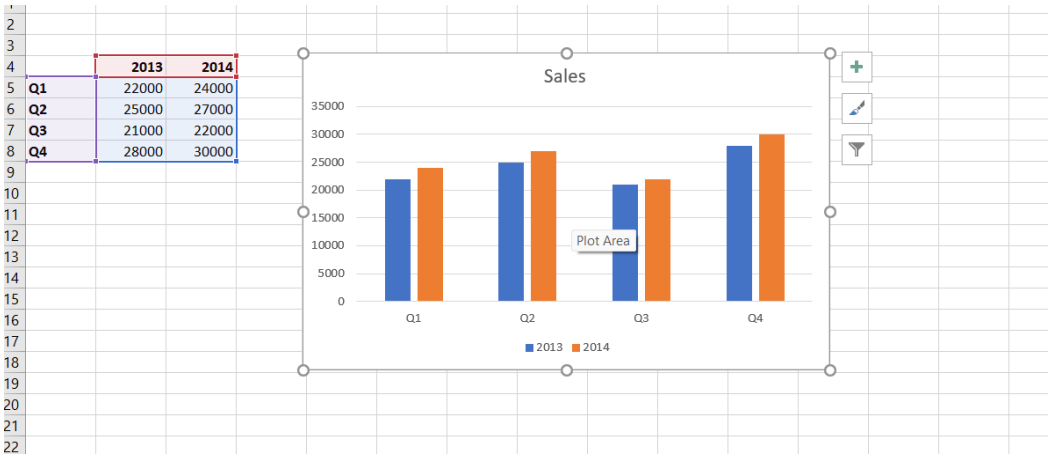
1. Windows 7/Windows 10
2. MS Office 2010 with Excel or Latest Version

Code/Program/Procedure (with comments):

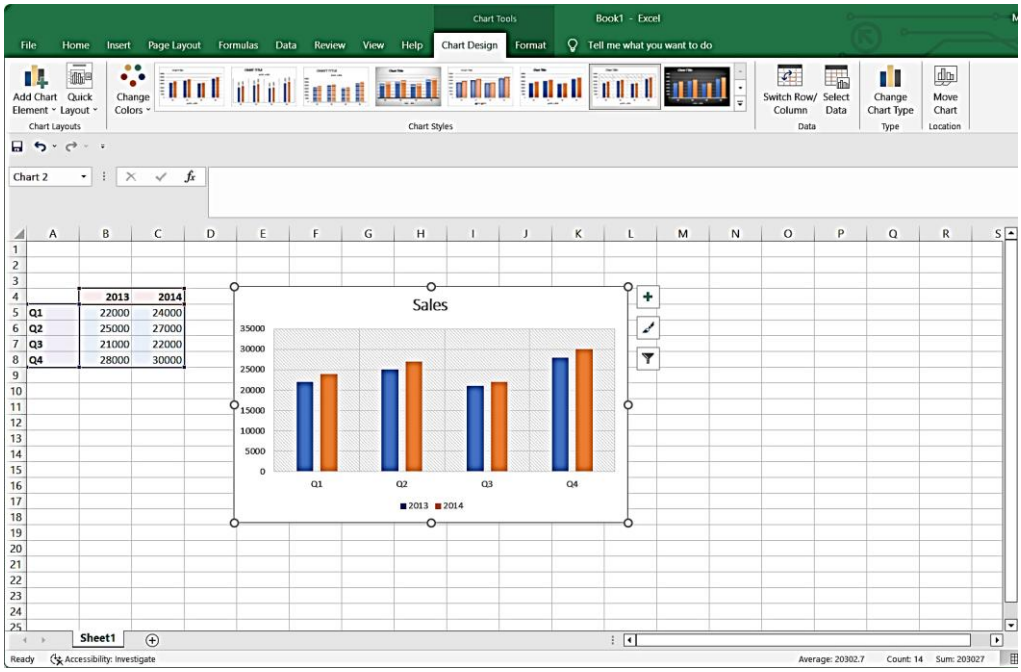
1. Open your Excel worksheet and create numeric data series.
2. Select data series click on insert and select chart.



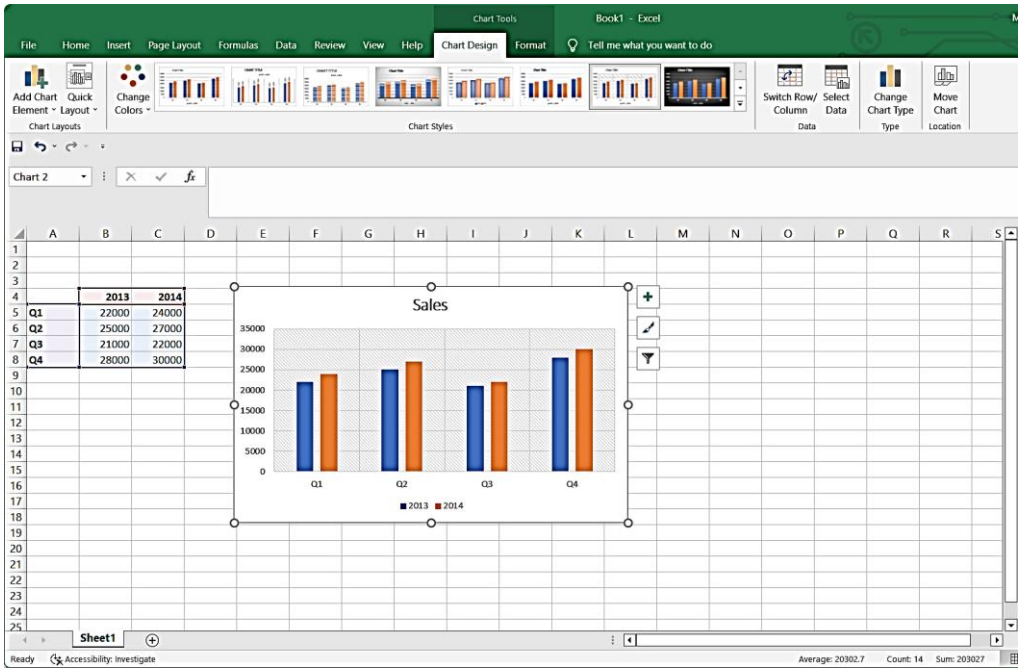
3. Choose required chart type.



4. Customize the chart by clicking the "Design," "Layout" and "Format" tabs of the Ribbon. Change the color with the Chart Styles options of the Design tab, add data labels, titles and shapes from the Layout tab and modify the colors, fill and effects from the Format tab. Save your Excel spreadsheet when complete.



Output/Results snippet:



References:

- https://smallbusiness.chron.com/turbotax-taxes-13771756.html

Activity 5

Aim: Plot uniform and binomial distributions in excel.(5hour)

Learning outcome: Able to business analytics and develop business intelligence.

Duration: 5 hours

List of Hardware/Software requirements:

1. Windows 7/Windows 10
2. MS Office 2010 with Excel or Latest Version

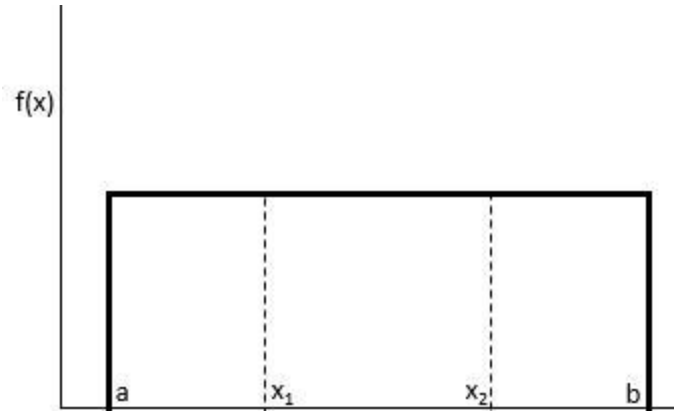
Code/Program/Procedure (with comments):

Uniform distribution

A uniform distribution is a probability distribution in which every value between an interval from a to b is equally likely to be chosen.

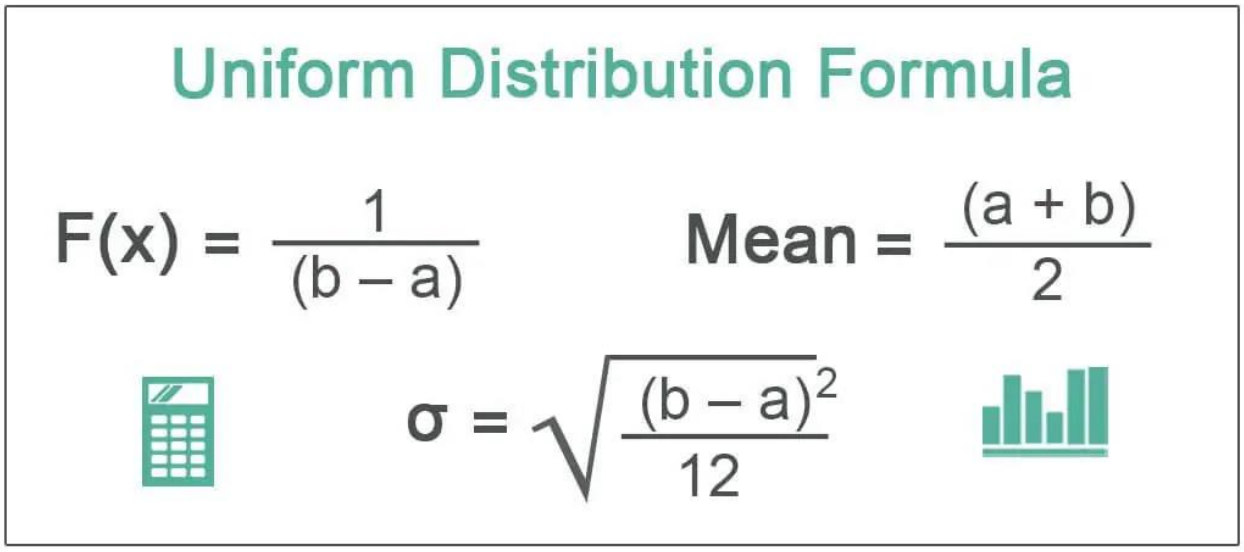
The probability that we will obtain a value between x1 and x2 on an interval from a to b can be found using the formula

P(obtain value between x1 and x2) = (x2 – x1) / (b – a)



The uniform distribution has the following properties:

- The mean of the distribution is $\mu = (a + b) / 2$
- The variance of the distribution is $\sigma 2 = (b – a)2 / 12$
- The standard deviation of the distribution is $\sigma = \sqrt{\sigma 2}$



The following examples show how to calculate probabilities for uniform distributions in Excel.

Note: You can double check the solution to each example below using the Uniform Distribution Calculator.

Uniform Distribution in Excel

Let us take the example of an employee of company ABC. He normally takes up the services of the cab or taxi for the purpose of traveling from home and office. The duration of the wait time of the cab from the nearest pickup point ranges from zero and fifteen minutes.

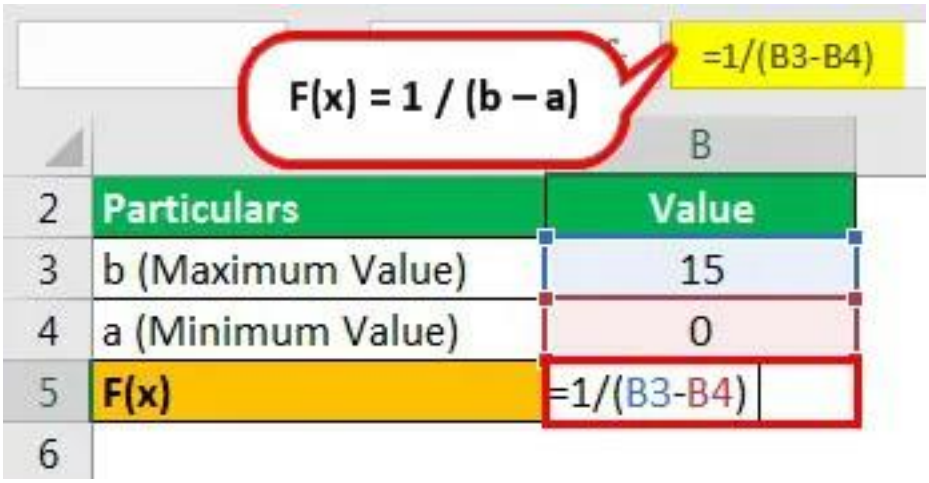
Help the employee determine the probability that he would have to wait for approximately less than 8 minutes. Additionally, determine the mean and standard deviation with respect to the wait time. Determine the probability density function as displayed below wherein for a variable X; the following steps should be performed:

Solution

Use the given data for the calculation of uniform distribution.

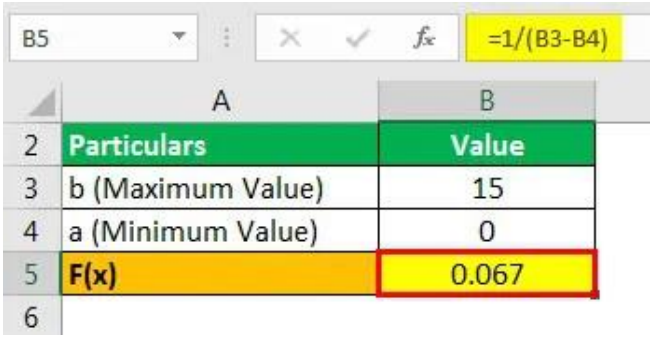
	A	B
2	**Particulars**	**Value**
3	b (Maximum Value)	15
4	a (Minimum Value)	0
5		

Calculation of the probability of the employee waiting for less than 8 minutes.



Uniform Distribution Formula Example 1.1

= 1 / (15 – 0)



Uniform Distribution Formula Example 1.2

F(x) = 0.067

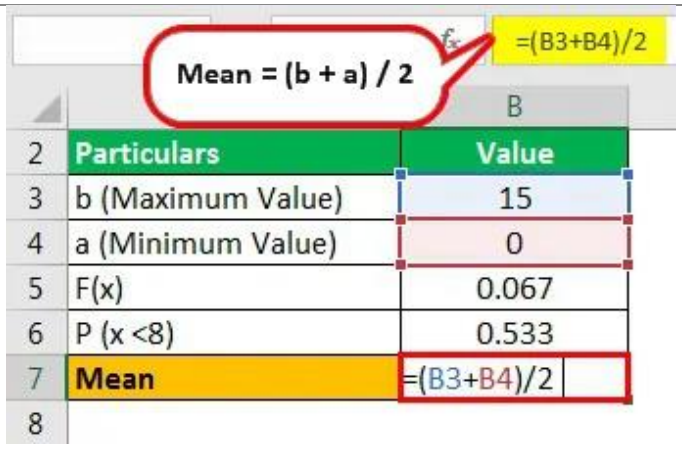
P (x < k) = base x height

P (x <8) = (8) x 0.067

P (x <8) = 0.533

Therefore, for a probability density function of 0.067, the probability that the waiting time for the individual would be less than 8 minutes is 0.533.

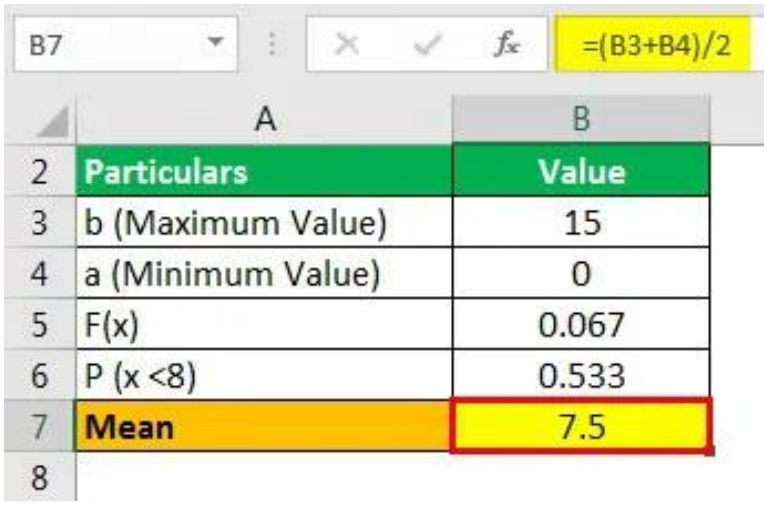
Calculation of mean of the distribution –



Uniform Distribution Formula Example

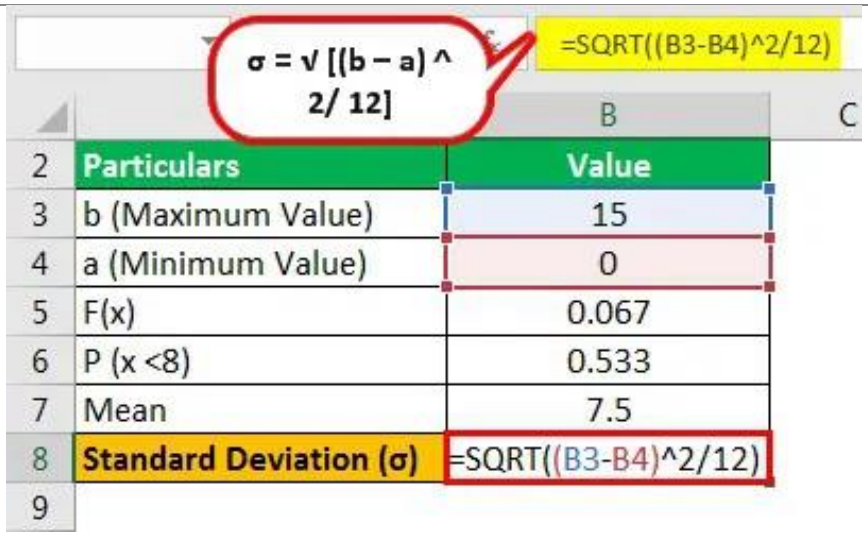
= (15 + 0) /2

Mean will be –



Mean = 7.5 minutes.

Calculation of standard deviation of the distribution –



σ = √ [(b – a) ^ 2/ 12]

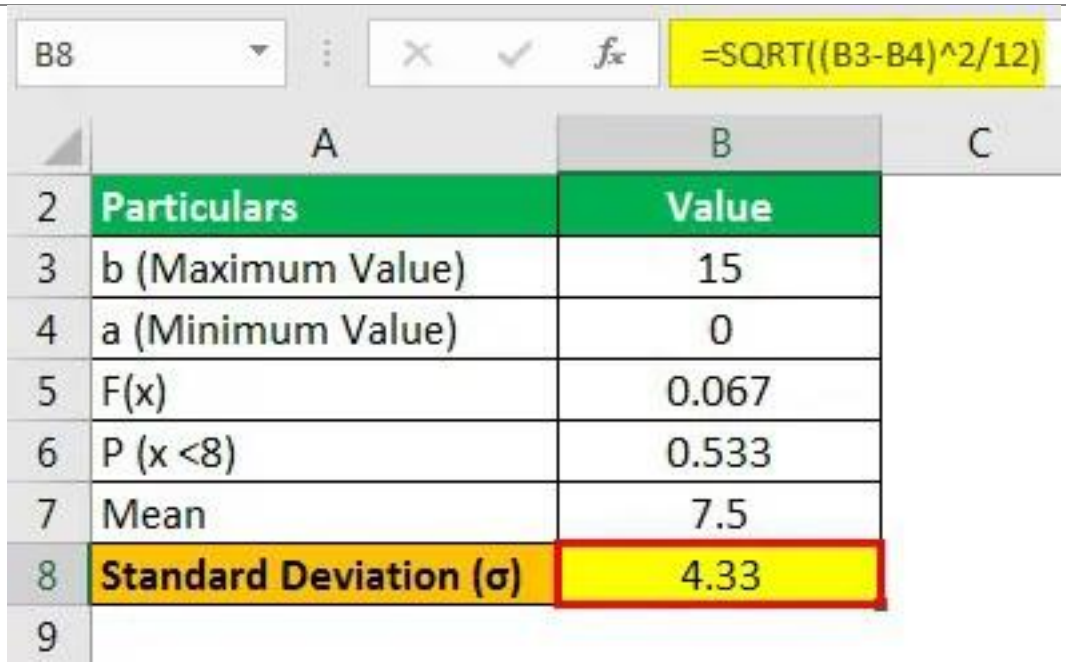
= √ [(15 – 0) ^ 2/ 12]

= √ [(15) ^ 2/ 12]

= √ [225 / 12]

= √ 18.75

Standard Deviation will be –



$\sigma = 4.33$

Therefore, the distribution shows a mean of 7.5 minutes with a standard deviation of 4.3 minutes.

Binomial distributions

The BINOM.DIST function is categorized under Excel Statistical functions. It calculates the binomial distribution probability for the number of successes from a specified number of trials. This binomial distribution Excel guide will show you how to use the function, step by step.

The binomial distribution is a statistical measure that is frequently used to indicate the probability of a specific number of successes occurring from a specific number of independent trials. The two forms used are:

The Probability Mass Function – Calculates the probability of there being exactly x successes from n independent trials

The Cumulative Distribution Function – Calculates the probability of there being at most x successes from n independent trials

In financial analysis, the BINOM.DIST function can be useful in finding out, for example, the probability of publishing a best-selling book from a range of books to be published by a company.

BINOM.DIST function is an updated version of the BINOMDIST function.

Formula for Binomial Distribution

=BINOM.DIST(number_s,trials,probability_s,cumulative)

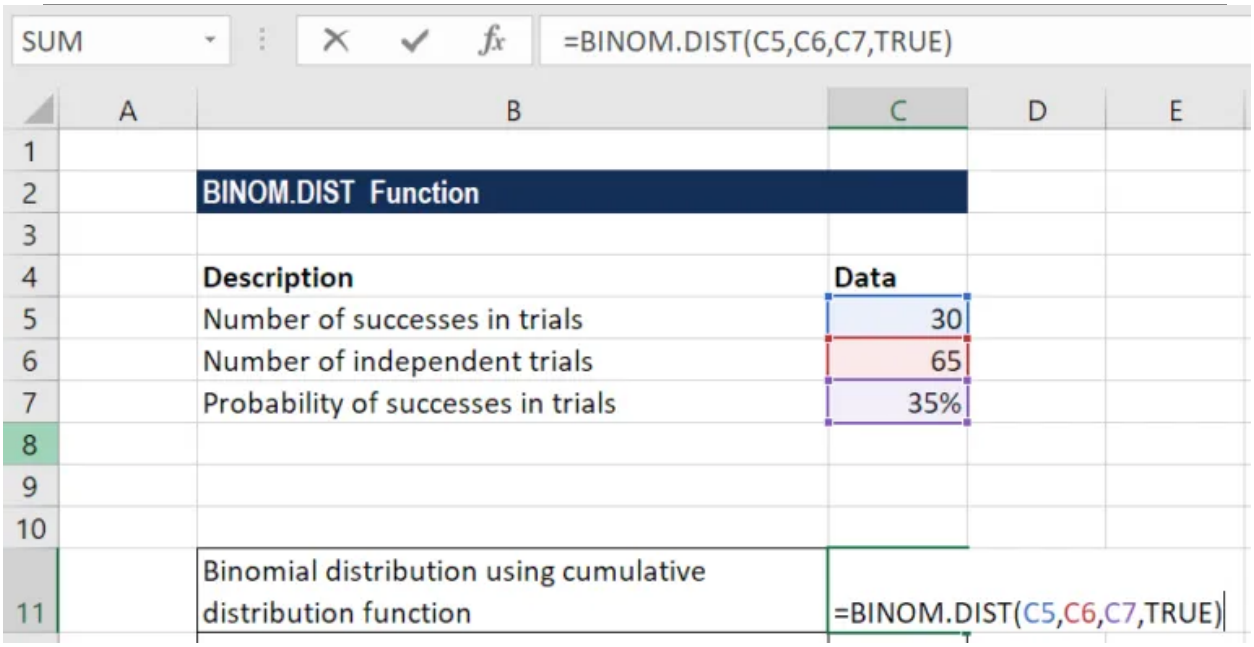
The BINOM.DIST uses the following arguments:

- Number_s (required argument) – This is the number of successes in trials.
- Trials (required argument) – This is the number of independent trials. It must be greater than or equal to 0.
- Probability_s (required argument) – This is the probability of success in each trial.
- Cumulative (required argument) – This is a logical value that determines the form of the function. It can either be:
- TRUE – Uses the cumulative distribution function.
- FALSE – Uses the probability mass function.

1. Suppose we are given the following data:

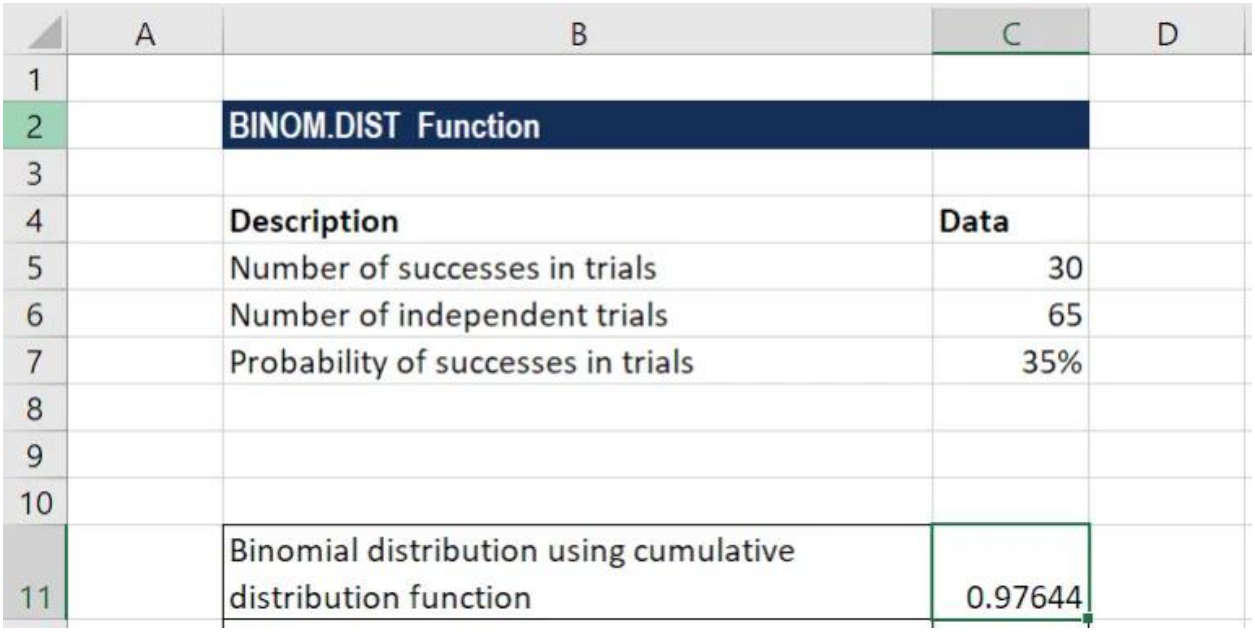
	A	B	C
1			
2		**BINOM.DIST Function**	
3			
4		**Description**	**Data**
5		Number of successes in trials	30
6		Number of independent trials	65
7		Probability of successes in trials	35%
8			

2. Apply the formula for calculating binomial distribution using the cumulative distribution function as shown below:

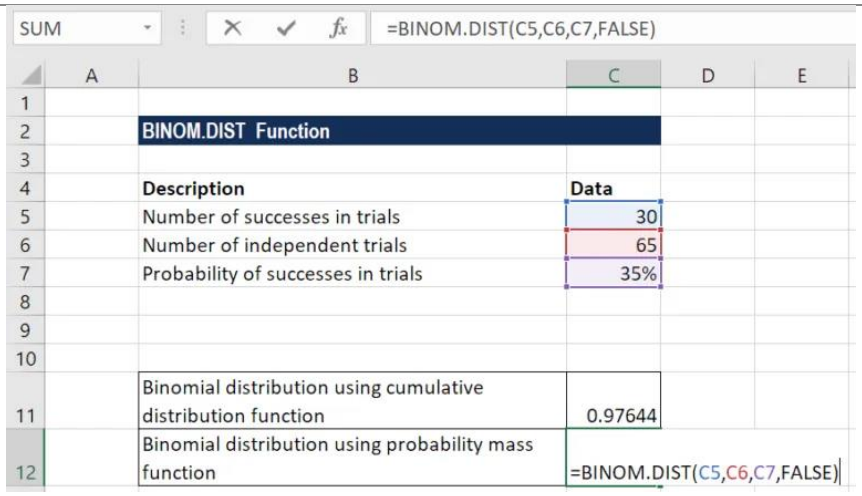


3. After applying the formula, we get the result below:

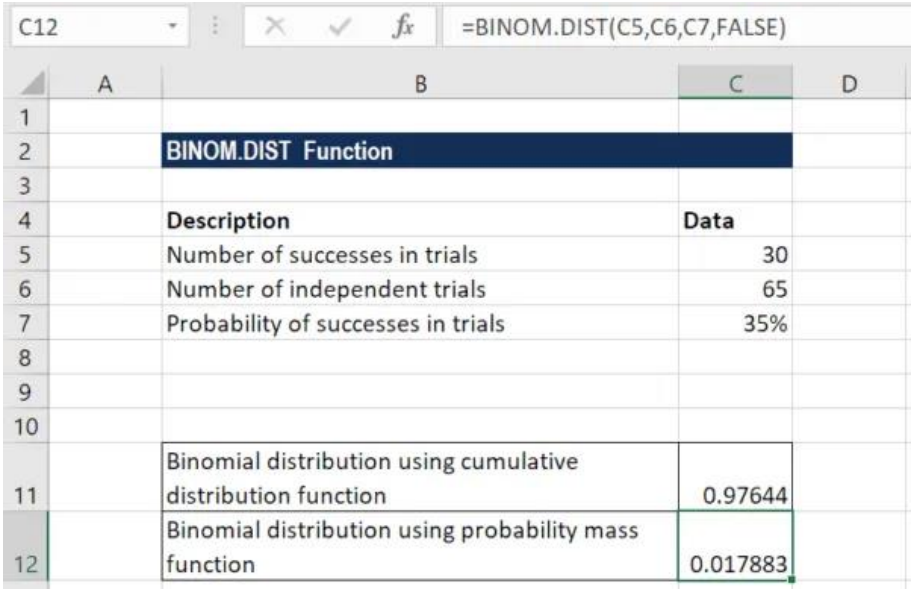
4. The



formula for calculating binomial distribution using the probability mass function is shown below:



We get the result below:



References:

- https://www.wallstreetmojo.com/uniform-distribution/
- https://corporatefinanceinstitute.com/resources/excel/functions/binomial-distribution-excel/

Activity 6

Aim: Implement Central limit theorem in excel. (5 hour)

Learning outcome: Able to business analytics and develop business intelligence.

Duration: 5 hours

List of Hardware/Software requirements:

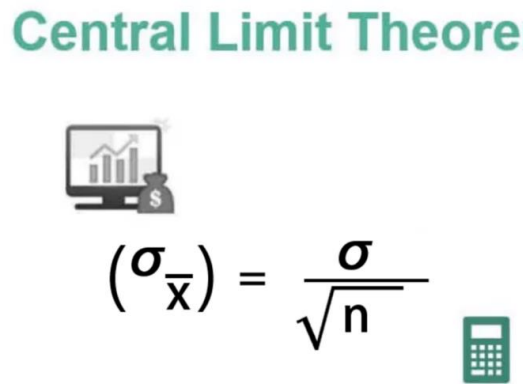
1. Windows 7/Windows 10
2. MS Office 2010 with Excel or Latest Version

Code/Program/Procedure (with comments):

Central Limit Theorem

The central limit theorem is a sampling distribution theory. It states that normal distribution can be attained by increasing sample size. Thus, the population mean is represented by the average of random sample means.

Central Limit Theorem Formula



The central limit theorem sets forth that the average of the sample means gives the population mean.

$\mu_{\bar{x}} = \mu$

Here $\mu_{\bar{x}}$ is an average of repetitive sample means.

And μ is the population mean.

The central limit theorem is calculated using the following formula.

The sample's standard deviation is computed by dividing the population's standard deviation by the square root of sample size:

$$\sigma_x = \frac{\sigma}{\sqrt{n}}$$

Here,

σ is the population standard deviation,

σx is the sample standard deviation; and

n is the sample size

Example:

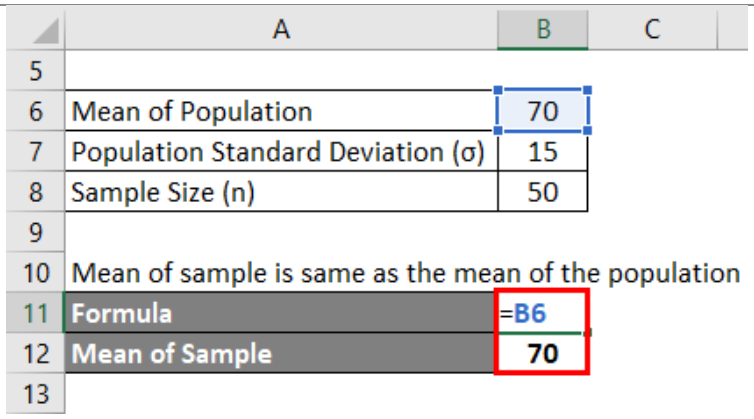
In a country located in the middle east region, the recorded weights of the male population are following a normal distribution. The mean and the standard deviations are 70 kg and 15 kg respectively. If a person is eager to find the record of 50 males in the population then what would mean and the standard deviation of the chosen sample?

	A	B
5		
6	Mean of Population	70
7	Population Standard Deviation (σ)	15
8	Sample Size (n)	50
9		

Solution:

- Calculation of Mean of Sample for example 1

Mean of Sample is the same as the mean of the population.



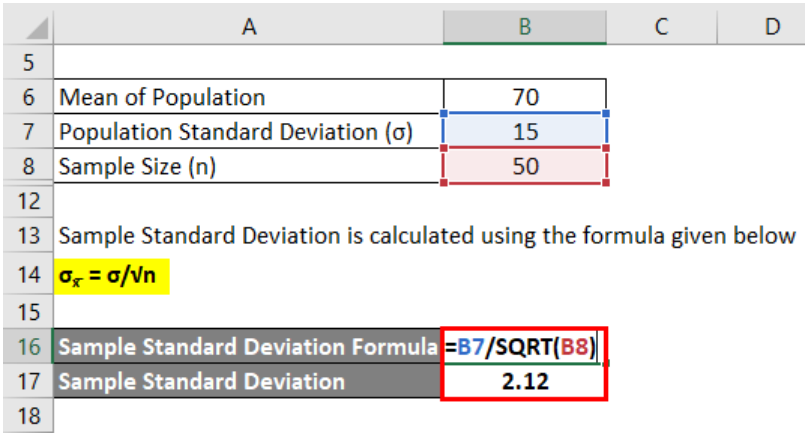
	A	B	C
5			
6	Mean of Population	70	
7	Population Standard Deviation (σ)	15	
8	Sample Size (n)	50	
9			
10	Mean of sample is same as the mean of the population		
11	**Formula**	=B6	
12	**Mean of Sample**	**70**	
13			

The mean of the population is 70 since the sample size > 30.

- Calculation of Sample Standard Deviation for example

Sample Standard Deviation is calculated using the formula given below

$$\sigma x = \sigma / \sqrt{n}$$



	A	B	C	D
5				
6	Mean of Population	70		
7	Population Standard Deviation (σ)	15		
8	Sample Size (n)	50		
12				
13	Sample Standard Deviation is calculated using the formula given below			
14	$\sigma_{\bar{x}} = \sigma/\sqrt{n}$			
15				
16	**Sample Standard Deviation Formula**	=B7/SQRT(B8)		
17	**Sample Standard Deviation**	**2.12**		
18				

Sample Standard Deviation = 15 / √50

Sample Standard Deviation = 2.12

Reference:

- https://www.educba.com/central-limit-theorem-formula/
- https://www.wallstreetmojo.com/central-limit-theorem/

Activity 7

Aim: Implement Central limit theorem in excel. (5hour)

Learning outcome: Able to business analytics and develop business intelligence.

Duration: 5 hours

List of Hardware/Software requirements:

1. Windows 7/Windows 10
2. MS Office 2010 with Excel or Latest Version

Code/Program/Procedure (with comments):

A Chi-Square Test of Independence is used to determine whether or not there is a significant association between two categorical variables.

Chi-Square Test of Independence in Excel

Suppose we want to know whether or not gender is associated with political party preference. We take a simple random sample of 500 voters and survey them on their political party preference. The following table shows the results of the survey:

	A	B	C	D	E	F
1		Republican	Democrat	Independent	Total	
2	Male	120	90	40	250	
3	Female	110	95	45	250	
4	Total	230	185	85	500	
5						
6						
7						
8						
9						
10						
11						
12						
13						

Steps to perform a Chi-Square test of independence to determine if gender is associated with political party preference.

- Step 1: Define the hypotheses.

We will perform the Chi-Square test of independence using the following hypotheses:

H0: Gender and political party preference are independent.

H1: Gender and political party preference are not independent.

- Step 2: Calculate the expected values.

Next, we will calculate the expected values for each cell in the contingency table using the following formula:

Expected value = (row sum * column sum) / table sum.

For example, the expected value for Male Republicans is: (230*250) / 500 = 115.

We can repeat this formula to obtain the expected value for each cell in the table:

	A	B	C	D	E	F	G	H	I	J	K
1		Republican	Democrat	Independent	Total						
2	Male	120	90	40	250						
3	Female	110	95	45	250						
4	Total	230	185	85	500						
5											
6	Expected values						Formulas				
7											
8		Republican	Democrat	Independent	Total			Republican	Democrat	Independent	Total
9	Male	115	92.5	42.5	250		Male	=B$4*$E2/$E$4	=C$4*$E2/$E$4	=D$4*$E2/$E$4	250
10	Female	115	92.5	42.5	250		Female	=B$4*$E3/$E$4	=C$4*$E3/$E$4	=D$4*$E3/$E$4	250
11	Total	230	185	85	500		Total	230	185	85	500
12											
13											
14											
15											
16											
17											

- Step 3: Calculate (O-E)2 / E for each cell in the table.

Next, we will calculate (O-E)2 / E for each cell in the table where:

O: observed value

E: expected value

For example, Male Republicans would have a value of: (120-115)2 /115 = 0.2174.

We can repeat this formula for each cell in the table:

	A	B	C	D	E	F	G	H	I	J	K
1		**Republican**	**Democrat**	**Independent**	**Total**						
2	**Male**	120	90	40	250						
3	**Female**	110	95	45	250						
4	**Total**	230	185	85	500						
5											
6	Expected values						Formulas				
7											
8		**Republican**	**Democrat**	**Independent**	**Total**			**Republican**	**Democrat**	**Independent**	**Total**
9	**Male**	115	92.5	42.5	250		**Male**	=B$4*$E2/$E$4	=C$4*$E2/$E$4	=D$4*$E2/$E$4	250
10	**Female**	115	92.5	42.5	250		**Female**	=B$4*$E3/$E$4	=C$4*$E3/$E$4	=D$4*$E3/$E$4	250
11	**Total**	230	185	85	500		**Total**	230	185	85	500
12											
13											
14	$(O-E)^2/E$						Formulas				
15											
16		**Republican**	**Democrat**	**Independent**	**Total**			**Republican**	**Democrat**	**Independent**	**Total**
17	**Male**	0.2174	0.0676	0.1471	250		**Male**	=(B2-B9)^2/B9	=(C2-C9)^2/C9	=(D2-D9)^2/D9	250
18	**Female**	0.2174	0.0676	0.1471	250		**Female**	=(B3-B10)^2/B10	=(C3-C10)^2/C10	=(D3-D10)^2/D10	250
19	**Total**	230	185	85	500		**Total**	230	185	85	500
20											
21											

- Step 4: Calculate the test statistic X2 and the corresponding p-value.

The test statistic X^2 is simply the sum of the values in the last table.

The p-value that corresponds to the test statistic X^2 can be found by using the formula:

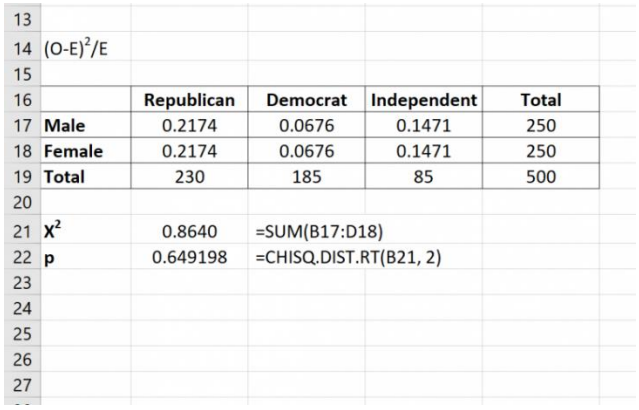
=CHISQ.DIST.RT(x, deg_freedom)

where:

x: test statistic X2

deg_freedom: degrees of freedom, calculated as (#rows-1) * (#columns-1)

The test statistic X2 turns out to be 0.8640 and the corresponding p-value is 0.649198.



13					
14	$(O-E)^2/E$				
15					
16		**Republican**	**Democrat**	**Independent**	**Total**
17	**Male**	0.2174	0.0676	0.1471	250
18	**Female**	0.2174	0.0676	0.1471	250
19	**Total**	230	185	85	500
20					
21	X^2	0.8640	=SUM(B17:D18)		
22	**p**	0.649198	=CHISQ.DIST.RT(B21, 2)		
23					
24					
25					
26					
27					

- Step 5: Draw a conclusion.

Since this p-value is not less than 0.05, we fail to reject the null hypothesis. This means we do not have sufficient evidence to say that there is an association between gender and political party preference.

Reference:

- https://www.statology.org/chi-square-test-of-independence-excel/

Learning Outcome

After completing this module, the student should be able to install and different operation in python

To meet the learning outcome, a student has to complete the following activities

1. Install NumPy, pandas, matplotlib, Seaborn, sklearn in python 3
2. Creating arrays in NumPy
3. Creating multidimensional array in NumPy
4. Numpy Operations, methods and attributes
5. Numpy case studies
6. Understanding Pandas series and dataframe
7. Pandas ingestion of data from csv, json, html, excel, text files
8. Pandas functionalities for Series & Data Frames
9. Grouping, Merging, concatenating, joining, segregation

Activity 1

Aim: Install NumPy, pandas, matplotlib, Seaborn, sklearn in python 3

Learning outcome: Able to install and different operation in python

Duration: 3.5 hour

List of Hardware/Software requirements:

1. Laptop/Computer with Windows 10/11
2. Internet connection
3. Python

Program / Procedure:

Installing pandas in python 3.

Installing with pip

It is a package installation manager that makes installing Python libraries and frameworks straightforward.

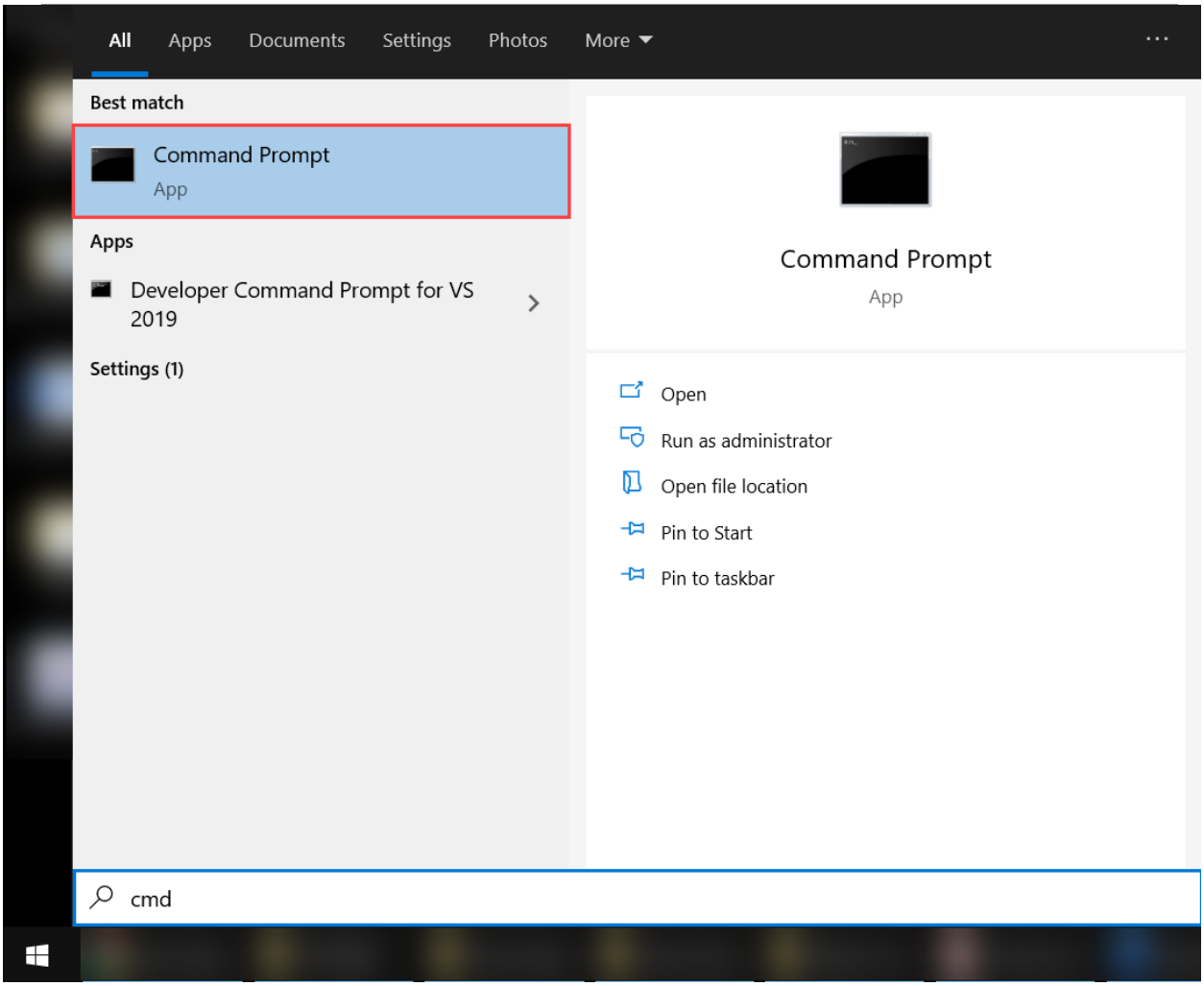
As long as you have a newer version of Python installed (> Python 3.4), pip will be installed on your computer along with Python by default.

However, if you're using an older version of Python, you will need to install pip on your computer before installing Pandas. The easiest way to do this is to upgrade to the latest version of Python available on https://www.python.org.

Step #1: Launch Command Prompt

Press the Windows key on your keyboard or click on the Start button to open the start menu. Type "cmd," and the Command Prompt app should appear as a listing in the start menu.

Open up the command prompt so you can install Pandas.

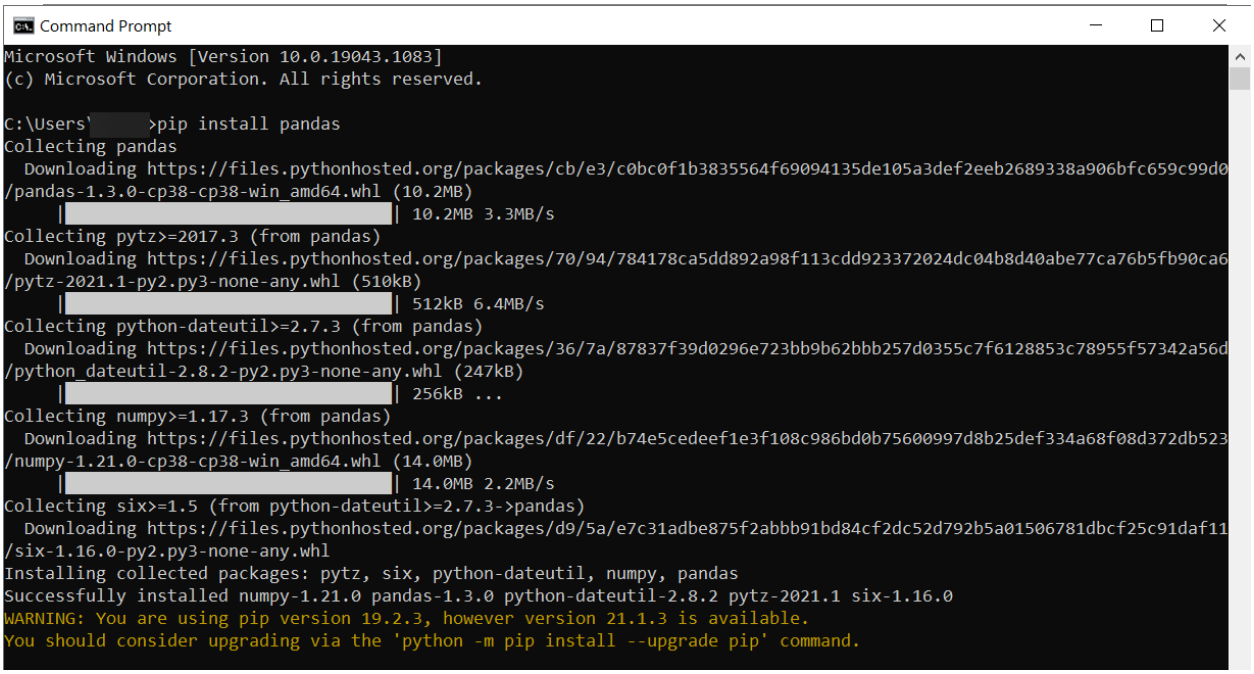


https://www.pythoncentral.io/wp-content/uploads/2021/07/Opening-Command-Prompt-768x626.png

Step #2: Enter the Required Command

After you launch the command prompt, the next step in the process is to type in the required command to initialize pip installation.

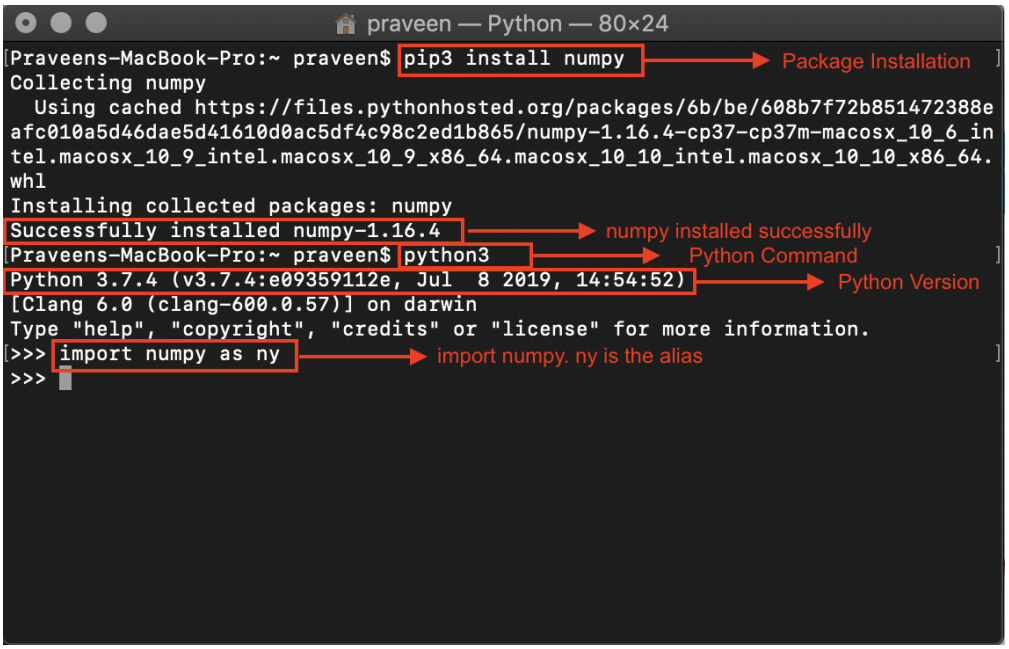
Enter the command “pip3 install pandas” on the terminal. This should launch the pip installer. The required files will be downloaded, and Pandas will be ready to run on your computer.



https://www.pythoncentral.io/wp-content/uploads/2021/07/Installing-Pandas-with-pip.png

After the installation is complete, you will be able to use Pandas in your Python programs.

Enter the command “pip3 install numpy” on the terminal. This should launch the pip installer. The required files will be downloaded, and numpy will be ready to run on your computer.



https://d1jnx9ba8s6j9r.cloudfront.net/blog/wp-content/uploads/2019/09/5Output-Numpy-installation-Edureka.png

After the installation is complete, you will be able to use numpy in your Python programs.

Matplotlib can be installed using pip. The following command is run in the command prompt to install Matplotlib.

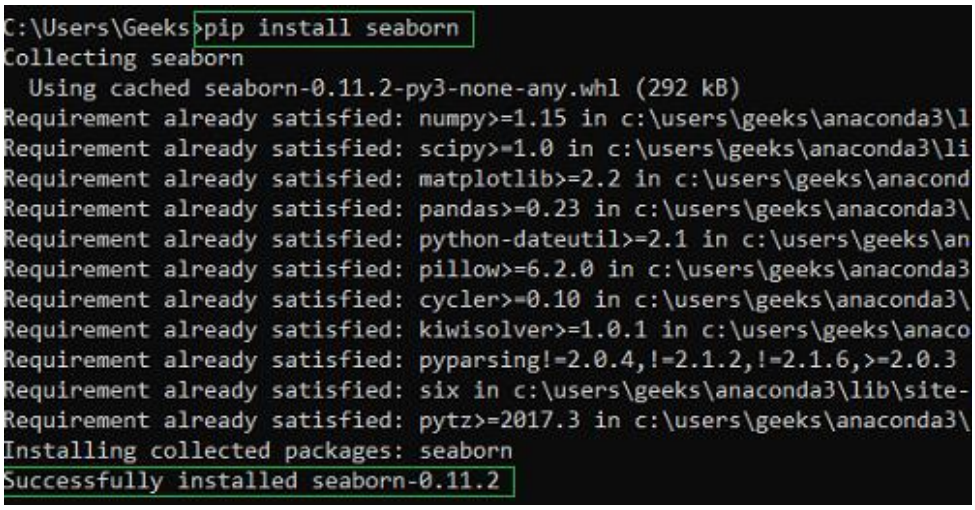
pip install matplotlib

This command will start downloading and installing packages related to the matplotlib library. Once done, the message of successful installation will be displayed.

PIP users can open up the command prompt and run the below command to install Python Seaborn Package on Windows:

pip install Seaborn

The following message will be shown once the installation is completed:



https://media.geeksforgeeks.org/wp-content/uploads/20210907232107/fgjghkyh.PNG

PIP users can open up the command prompt and run the below command to install Python sklearn Package on Windows:

pip install --pre -U scikit-learn

References:

https://www.pythoncentral.io/how-to-install-pandas-in-python/

https://www.geeksforgeeks.org/how-to-install-seaborn-on-windows/

https://www.tutorialspoint.com/how-to-install-matplotlib-in-python

Activity 2

Aim: Creating arrays in NumPy

Learning outcome: Able to install and different operation in python

Duration: 3.5 hour

List of Hardware/Software requirements:

1. Laptop/Computer with Windows 10/11
2. Internet connection
3. Python
4.

Program / Procedure:

Create a NumPy ndarray Object

NumPy is used to work with arrays. The array object in NumPy is called ndarray.

We can create a NumPy ndarray object by using the array() function.

```
import numpy as np

arr = np.array([1, 2, 3, 4, 5])

print(arr)

print(type(arr))
```

References:

https://www.w3schools.com/python/numpy/numpy_creating_arrays.asp

Activity 3

Aim: Creating multidimensional array in NumPy

Learning outcome: Able to install and different operation in python

Duration: 3.5 hour

List of Hardware/Software requirements:

1. Laptop/Computer with Windows 10/11
2. Internet connection
3. Python

Program / Procedure:

Multidimension Arrays

A dimension in arrays is one level of array depth (nested arrays).

0-D Arrays

0-D arrays, or Scalars, are the elements in an array. Each value in an array is a 0-D array.

```
import numpy as np

arr = np.array(42)

print(arr)
```

1-D Arrays

An array that has 0-D arrays as its elements is called uni-dimensional or 1-D array.

These are the most common and basic arrays.

```
import numpy as np

arr = np.array([1, 2, 3, 4, 5])

print(arr)
```

2-D Arrays

An array that has 1-D arrays as its elements is called a 2-D array.

These are often used to represent matrix or 2nd order tensors.

```
import numpy as np

arr = np.array([[1, 2, 3], [4, 5, 6]])

print(arr)
```

3-D arrays

An array that has 2-D arrays (matrices) as its elements is called 3-D array.

These are often used to represent a 3rd order tensor

```
import numpy as np

arr = np.array([[[1, 2, 3], [4, 5, 6]], [[1, 2, 3], [4, 5, 6]]])

print(arr)
```

Reference:

- https://www.w3schools.com/python/numpy/numpy_creating_arrays.asp

Activity 4

Aim: Numpy Operations, methods and attributes

Learning outcome: Able to install and different operation in python

Duration: 3.5 hour

List of Hardware/Software requirements:

1. Laptop/Computer with Windows 10/11
2. Internet connection
3. Python

Program / Procedure:

Basic Array Attributes

Armed with our understanding of multidimensional NumPy arrays, we now look at methods for programmatically inspecting an array's attributes (e.g. its dimensionality). It is especially important to understand what an array's "shape" is.

We will use the following array to provide context for our discussion:

```
>>> import numpy as np
>>> example_array = np.array([[[ 0,  1,  2,  3],
...                            [ 4,  5,  6,  7]],
...
...                           [[ 8,  9, 10, 11],
...                            [12, 13, 14, 15]],
...
...                           [[16, 17, 18, 19],
...                            [20, 21, 22, 23]]])
```

According to the preceding discussion, it is a 3-dimensional array structured such that:

- axis-0 discerns which of the 3 sheets to select from.
- axis-1 discerns which of the 2 rows, in any sheet, to select from.
- axis-2 discerns which of the 4 columns, in any sheet and row, to select from.

ndarray.ndim:

The number of axes (dimensions) of the array.

dimensionality of the array

>>> example_array.ndim

3

ndarray.shape:

A tuple of integers indicating the number of elements that are stored along each dimension of the array. For a 2D-array with N rows and M columns, shape will be (N,M). The length of this shape-tuple is therefore equal to the number of dimensions of the array.

shape of the array

>>> example_array.shape

(3, 2, 4)

ndarray.size:

The total number of elements of the array. This is equal to the product of the elements of the array's shape.

size of the array: the number of elements it stores

>>> example_array.size

24

ndarray.dtype:

An object describing the data type of the elements in the array. Recall that NumPy's ND-arrays are *homogeneous*: they can only posses numbers of a uniform data type.

```
# `example_array` contains integers, each of which are stored using 32 bits of memory

>>> example_array.dtype

dtype('int32')
```

ndarray.itemsize:

The size, in bytes (8 bits is 1 byte), of each element of the array. For example, an array of elements of type float64 has itemsize 8 (=64/8), while an array of type complex32 has itemsize 4 (=32/8).

```
# each integer in `example_array` is represented using 4 bytes (32 bits) of memory

>>> example_array.itemsize

4
```

Reference:

- https://www.geeksforgeeks.org/load-csv-data-into-mysql-server-using-php/

Activity 5

Aim: Numpy case studies

Learning outcome: Able to install and different operation in python

Duration: 3.5 hour

List of Hardware/Software requirements:

1. Laptop/Computer with Windows 10/11
2. Internet connection
3. Python

Program / Procedure:

Outline of this article:

The following are the brief contents of what we will be covering in this article:

1- Installing Pandas to your computer
2- Reading a CSV file in Pandas and checking the read file
3- Getting some basic information about the data read in Pandas

As I am planning to continue Python Pandas as a series of articles, it is a good idea to mention the topics we will cover in the upcoming articles:

1- Filtering in Pandas
1- 2-Adding/removing columns/rows and updating them
2- Sorting data, grouping and aggregating in Pandas
3- 4-Cleaning issues in Pandas and examples

We are going to answer the following questions in this article:

1- How can I install Pandas to my computer?
2- How can I load a CSV file as a Pandas DataFrame?
3- How can I explore the basic information about a loaded CSV file(Pandas table)?

So, let's get to work and start exploring!!

We are going to use a CSV file downloaded from Kaggle named as "Are your employees burning out?". It was already in my local drive. This dataset is also available for public use in Kaggle.com. Just go ahead and download it if you would like to do the same exercises with me in this article.

How to install Pandas to your computer?

As main aim of this article is explaining basics of Pandas, I will mention the very basic step of installing Pandas to your computer. Just execute the below command in your terminal within your **virtual environment**. For more detailed instructions on installing Pandas, a Google search may be helpful.

pip install pandas

So let us start coding now, here we go with importing the Pandas library.

#Importing Pandas libraryimport pandas

Now let us read our .csv file from our local drive and and check whether it is loaded. To check the data read from the local drive, I mostly print the data with a head() function to see whether there is a problem in reading. Head() function shows the first 5 rows of the data unless a numerical value is given to the function.

data = pandas.read_csv("/YOUR_LOCAL_PATH_TO_THIS_FILE_ON_YOUR_COMPUTER/train.csv", sep=",")

If a numerical value n is passed to the head() function, first n rows will be displayed. Maximum number of rows without any interruptions in displaying is 60. When 60 is exceeded, Pandas shows the first and last 5 rows within n range.

	Employee ID	Date of Joining	Gender	Company Type	WFH Setup	Designation	Resource Allocation	Mental Fatigue Score	Burn Rate
0	fffe32003000360033003200	2008-09-30	Female	Service	No	2.0	3.0	3.8	0.16
1	fffe3700360033003500	2008-11-30	Male	Service	Yes	1.0	2.0	5.0	0.36
2	fffe3100330032003700	2008-03-10	Female	Product	Yes	2.0	NaN	5.8	0.49
3	fffe3200340038003200	2008-11-03	Male	Service	Yes	1.0	1.0	2.6	0.20
4	fffe3100390034003600	2008-07-24	Female	Service	No	3.0	7.0	6.9	0.52
..	...	...	...	...	...	...	...	...	...
75	fffe3200350037003200	2008-02-04	Female	Service	No	1.0	2.0	4.1	0.27
76	fffe3800390036003200	2008-08-01	Male	Service	No	3.0	7.0	7.4	0.66
77	fffe3400390032003600	2008-05-01	Female	Product	Yes	0.0	1.0	NaN	0.04
78	fffe3300330036003200	2008-05-21	Male	Service	Yes	1.0	2.0	4.4	0.27
79	fffe3200360036003900	2008-11-26	Female	Service	No	2.0	6.0	6.9	0.67

The displayed rows with data.head(80).

	Employee ID	Date of Joining	Gender	Company Type	WFH Setup	Designation	Resource Allocation	Mental Fatigue Score	Burn Rate
0	fffe32003000360033003200	2008-09-30	Female	Service	No	2.0	3.0	3.8	0.16
1	fffe3700360033003500	2008-11-30	Male	Service	Yes	1.0	2.0	5.0	0.36
2	fffe3100330032003700 3900	2008-03-10	Female	Product	Yes	2.0	NaN	5.8	0.49
3	fffe3200340038003200 3900	2008-11-03	Male	Service	Yes	1.0	1.0	2.6	0.20
4	fffe3100390034003100 3600	2008-07-24	Female	Service	No	3.0	7.0	6.9	0.52
5	fffe3300350037003500	2008-11-26	Male	Product	Yes	2.0	4.0	3.6	0.29
6	fffe3300330034003900 3100	2008-01-02	Female	Service	No	3.0	6.0	7.9	0.62
7	fffe3200360032003700 3400	2008-10-31	Female	Service	Yes	2.0	4.0	4.4	0.33
8	fffe3200320030003400 3700	2008-12-27	Female	Service	No	3.0	6.0	NaN	0.56
9	fffe3100360032003000 3200	2008-03-09	Female	Product	No	3.0	6.0	NaN	0.67

The displayed rows with data.head(10).

Tail function can be used similarly to view the values at the bottom of the data table — so called Pandas DataFrame.

print(data.tail(10))

	Employee ID	Date of Joining	Gender	Company Type	WFH Setup Available	Designation	Resource Allocation	Mental Fatigue Score	Burn Rate
22740	fffe3300330038003100 3100	2008-09-05	Female	Product	No	3.0	6.0	7.3	0.55
22741	fffe3100360035003400 3800	2008-01-07	Male	Product	No	2.0	5.0	6.0	NaN
22742	fffe3300320031003900 3000	2008-07-28	Male	Product	No	3.0	5.0	8.1	0.69
22743	fffe3300390030003600	2008-12-15	Female	Product	Yes	1.0	3.0	6.0	0.48
22744	fffe3200350037003200	2008-05-27	Male	Product	No	3.0	7.0	6.2	0.54
22745	fffe3100350037003900 3100	2008-12-30	Female	Service	No	1.0	3.0	NaN	0.41
22746	fffe3300300035003100 3800	2008-01-19	Female	Product	Yes	3.0	6.0	6.7	0.59
22747	fffe390032003000	2008-11-05	Male	Service	Yes	3.0	7.0	NaN	0.72
22748	fffe3300330032003900	2008-01-10	Female	Service	No	2.0	5.0	5.9	0.52
22749	fffe3400350031003800	2008-01-06	Male	Product	No	3.0	6.0	7.8	0.61

The displayed rows with data.tail(10).

From now on, we are going to use DataFrame term instead of Pandas table below. You can think of a DataFrame as a large MS Excel spreadsheet.

For checking the column names of a DataFrame, .columns function is very useful. This is especially useful for DataFrames with too many column names.

print(data.columns)

```
Index(['Employee ID', 'Date of Joining', 'Gender', 'Company Type', 'WFH Setup',
       'Designation', 'Resource Allocation', 'Mental Fatigue Score',
       'Burn Rate'],
      dtype='object')
```

The names of columns displayed after data.columns. This function is useful especially with DataFrames with too many columns.

Below are some basic functions to check the preliminary information about your DataFrame:

.shape function shows the total number of rows and columns of a DataFrame. Our DataFrame has 22750 rows and 9 columns according to this.

print(data.shape)

```
(22750, 9)
```

Number of rows and columns displayed after data.shape.

.info() function shows the main variable types under the relevant columns within the DataFrame.

print(data.info())

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 22750 entries, 0 to 22749
Data columns (total 9 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   Employee ID           22750 non-null  object
 1   Date of Joining       22750 non-null  object
 2   Gender                22750 non-null  object
 3   Company Type          22750 non-null  object
 4   WFH Setup             22750 non-null  object
 5   Designation           22750 non-null  float64
 6   Resource Allocation   21369 non-null  float64
 7   Mental Fatigue Score  20633 non-null  float64
 8   Burn Rate             21626 non-null  float64
dtypes: float64(4), object(5)
memory usage: 1.6+ MB
```

Information displayed about the DataFrame after data.info() function. By default, this function shows the number of non-null entries in each column.

Above, we see that first five columns are probably filled with variables with a string type (or maybe with a Date type); and the remaining four columns have a float variable type. Please note that some other information is also included just above the variables/columns table.

The number of non-null values in each column are also shown.

As a final tool in this article, we can use the describe() function to see the basic statistical information about our columns with numerical values, which are Designation, Resource Allocation, Mental Fatigue Score and Burn Rate in this example.

print(data.describe())

	Designation	Resource Allocation	Mental Fatigue Score	Burn Rate
count	22750.000000	21369.000000	20633.000000	21626.000000
mean	2.178725	4.481398	5.728188	0.452005
std	1.135145	2.047211	1.920839	0.198226
min	0.000000	1.000000	0.000000	0.000000
25%	1.000000	3.000000	4.600000	0.310000
50%	2.000000	4.000000	5.900000	0.450000
75%	3.000000	6.000000	7.100000	0.590000
max	5.000000	10.000000	10.000000	1.000000

The basic statistics shown after data.describe() function

Total number of counts in each column (variable), averages, standard deviations, minimum values, maximum values and 25%, 50%(median) and 75% percentiles are shown after the describe function. With additional parameters passed to describe function, additional information may also be displayed.

Reference:

- https://medium.com/@tansu_61955/python-pandas-fast-forward-with-a-case-study-e44565a9da4b

Activity 6

Aim: Understanding Pandas series and dataframe

Learning outcome: Able to install and different operation in python

Duration: 3.5 hour

List of Hardware/Software requirements:

1. Laptop/Computer with Windows 10/11
2. Internet connection
3. Python

Program / Procedure:

Creating a dataframe from Pandas series

Series is a type of list in pandas which can take integer values, string values, double values and more. But in Pandas Series we return an object in the form of list, having index starting from *0* to *n*, Where *n* is the length of values in series.

Series can only contain single list with index, whereas dataframe can be made of more than one series or we can say that a dataframe is a collection of series that can be used to analyse the data.

Code #1: Creating a simple Series

```
import pandas as pd

import matplotlib.pyplot as pl

author = ['Jitender', 'Purnima', 'Arpit', 'Jyoti']

auth_series = pd.Series(author)

print(auth_series)
```

Output:

0 Jitender

1 Purnima

2 Arpit

3 Jyoti

dtype: object

Let’s check type of Series:

```
import pandas as pd
import matplotlib.pyplot as plt
author = ['Jitender', 'Purnima', 'Arpit', 'Jyoti']
auth_series = pd.Series(author)
print(type(auth_series))
```

Output:

<class 'pandas.core.series.Series'>

Code #2: Creating Dataframe from Series

```
import pandas as pd
import matplotlib.pyplot as plt
author = ['Jitender', 'Purnima', 'Arpit', 'Jyoti']
article = [210, 211, 114, 178]
auth_series = pd.Series(author)
article_series = pd.Series(article)



frame = { 'Author': auth_series, 'Article': article_series }
```

```
result = pd.DataFrame(frame)

print(result)
```

Output:

```
    Author  Article
0  Jitender      210
1   Purnima      211
2     Arpit      114
3     Jyoti      178
```

Explanation:

We are combining two series *Author* and *Article published*. Create a dictionary so that we can combine the metadata for series. Metadata is the data of data that can define the series of values. Pass this dictionary to pandas DataFrame and finally you can see the result as combination of two series i.e for author and number of articles.

Code #3: How to add series externally in dataframe

```
import pandas as pd

import matplotlib.pyplot as plt

author = ['Jitender', 'Purnima', 'Arpit', 'Jyoti']

article = [210, 211, 114, 178]

auth_series = pd.Series(author)

article_series = pd.Series(article)

frame = { 'Author': auth_series, 'Article': article_series }

result = pd.DataFrame(frame)
```

```
age = [21, 21, 24, 23]

result['Age'] = pd.Series(age)

print(result)
```

Output:

```
    Author  Article  Age
0  Jitender     210   21
1   Purnima     211   21
2     Arpit     114   24
3     Jyoti     178   23
```

Explanation:

We have added one more series externally named as *age* of the authors, then directly added this series in the pandas dataframe. Remember one thing if any value is missing then by default it will be converted into NaN value i.e *null* by default.

Code #4: Missing value in dataframe

```
import pandas as pd

import matplotlib.pyplot as plt

author = ['Jitender', 'Purnima', 'Arpit', 'Jyoti']

article = [210, 211, 114, 178]

auth_series = pd.Series(author)

article_series = pd.Series(article)
```

```
frame = { 'Author': auth_series, 'Article': article_series }

result = pd.DataFrame(frame)

age = [21, 21, 23]

result['Age'] = pd.Series(age)

print(result)
```

Output:

```
    Author  Article   Age
0  Jitender     210  21.0
1   Purnima     211  21.0
2     Arpit     114  23.0
3     Jyoti     178   NaN
```

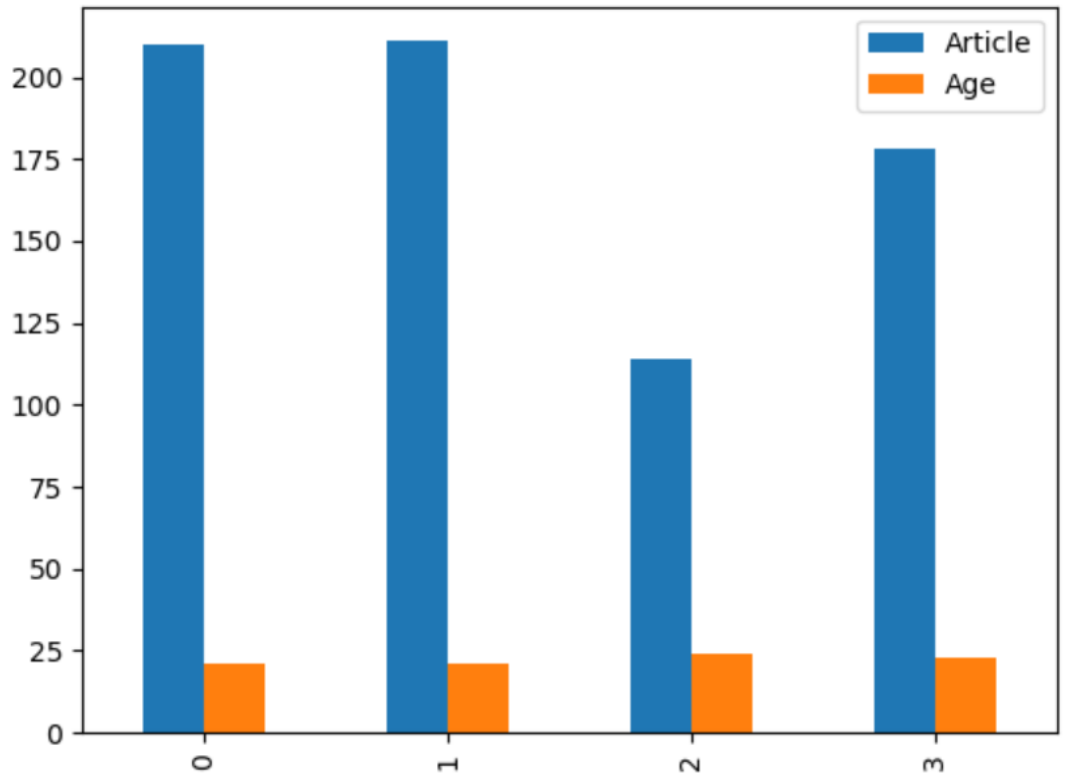
Code #5: Data Plot on graph

Using plot.bar() we have created a bar graph.

```
import pandas as pd

import matplotlib.pyplot as plt

author = ['Jitender', 'Purnima', 'Arpit', 'Jyoti']

article = [210, 211, 114, 178]

auth_series = pd.Series(author)

article_series = pd.Series(article)

frame = { 'Author': auth_series, 'Article': article_series }

result = pd.DataFrame(frame)
```

```
age = [21, 21, 24, 23]

result['Age'] = pd.Series(age)

result.plot.bar()

plt.show()
```

Output:



Reference:

- https://www.geeksforgeeks.org/creating-a-dataframe-from-pandas-series/
- https://media.geeksforgeeks.org/wp-content/uploads/Screenshot-2019-02-02-13.17.36.png

Activity 7

Aim: Pandas ingestion of data from csv, json, html, excel, text files

Learning outcome: Able to install and different operation in python

Duration: 3.5 hour

List of Hardware/Software requirements:

1. Laptop/Computer with Windows 10/11
2. Internet connection
3. Python

Program / Procedure:

Streamlined Data Ingestion with Pandas

Data Ingestion is the process of, transferring data, from varied sources to an approach, where it can be analyzed, archived, or utilized by an establishment. The usual steps, involved in this process, are drawing out data, from its current place, converting the data, and, finally loading it, in a location, for efficient research. Python provides many such tools, and, frameworks for data ingestion. These include Bonobo, Beautiful Soup4, Airflow, Pandas, etc. In this article, we will learn about Data Ingestion with Pandas library.

Data Ingestion with Pandas:

Data Ingestion with Pandas, is the process, of shifting data, from a variety of sources, into the Pandas DataFrame structure. The source of data can be varying file formats such as Comma Separated Data, JSON, HTML webpage table, Excel. In this article, we will learn about, transferring data, from such formats, into the destination, which is a Pandas dataframe object.

Approach:

The basic approach, for transferring any such data, into a dataframe object, is as follows –

- Prepare your source data.
    - Data can be present, on any remote server, or, on a local machine. We need to know, the URL of the file if it's on a remote server. The path of the file, on local machine, is required, if data is present locally.
- Use Pandas 'read_x' method
    - Pandas provide 'read_x' methods, for loading and converting the data, into a Dataframe object.
    - Depending on the data format, use the 'read' method.
- Print data from DataFrame object.
    - Print the dataframe object, to verify, that the conversion was smooth.

File Formats for Ingestion:

In this article, we will be converting, data present in the following files, to dataframe structures –

1. Read data from CSV file
2. Read data from Excel file
3. Read data from JSON file
4. Read data from Clipboard
5. Read data from HTML table from web page
6. Read data from SQLite table

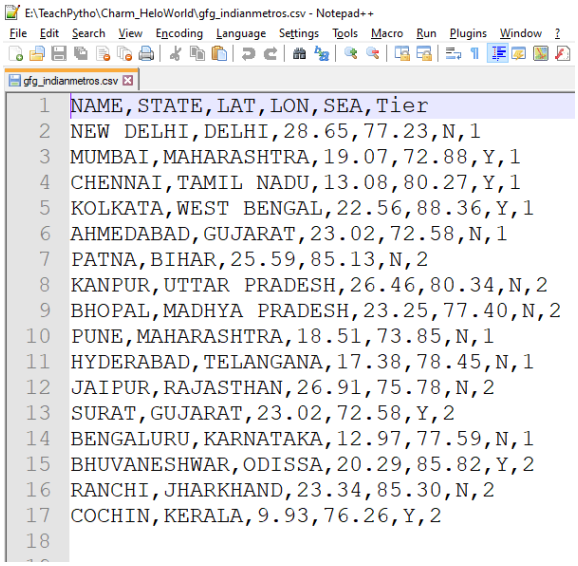
Read data from CSV file

To load, data present in Comma-separated file(CSV), we will follow steps as below:

- Prepare your sample dataset. Here, we have a CSV file, containing information, about Indian Metro cities. It describes if the city is a Tier1 or Tier2 city, their geographical location, state they belong to, and if it is a coastal city.
- Use Pandas method 'read_csv'
    - Method used – *read_csv(file_path)*

- Parameter – String format, containing the path of the file and its name, or, URL when present on the remote server. It reads, the file data, and, converts it, into a valid two-dimensional dataframe object. This method can be used to read data, present in ".csv" as well as ".txt" file formats.

The file contents are as follows:



The contents of "gfg_indianmetros.csv" file

https://media.geeksforgeeks.org/wp-content/uploads/20210617085639/gfgindianmetros.png

The code to get the data in a Pandas Data Frame is:

```
# Import the Pandas library

import pandas

# Load data from Comma separated file

# Use method - read_csv(filepath)

# Parameter - the path/URL of the CSV/TXT file

dfIndianMetros = pandas.read_csv("gfg_indianmetros.csv")

# print the dataframe object

print(dfIndianMetros)
```

Output:

```
# print the dataframe object
print(dfIndianMetros)
```

```
           NAME           STATE    LAT    LON SEA  Tier
0      NEW DELHI           DELHI  28.65  77.23   N     1
1         MUMBAI     MAHARASHTRA  19.07  72.88   Y     1
2        CHENNAI      TAMIL NADU  13.08  80.27   Y     1
3        KOLKATA     WEST BENGAL  22.56  88.36   Y     1
4      AHMEDABAD         GUJARAT  23.02  72.58   N     1
5          PATNA           BIHAR  25.59  85.13   N     2
6         KANPUR   UTTAR PRADESH  26.46  80.34   N     2
7         BHOPAL  MADHYA PRADESH  23.25  77.40   N     2
8           PUNE     MAHARASHTRA  18.51  73.85   N     1
9      HYDERABAD       TELANGANA  17.38  78.45   N     1
10        JAIPUR       RAJASTHAN  26.91  75.78   N     2
11         SURAT         GUJARAT  23.02  72.58   Y     2
12     BENGALURU       KARNATAKA  12.97  77.59   N     1
13  BHUVANESHWAR          ODISSA  20.29  85.82   Y     2
14        RANCHI       JHARKHAND  23.34  85.30   N     2
15        COCHIN          KERALA   9.93  76.26   Y     2
```

The CSV data, in dataframe object

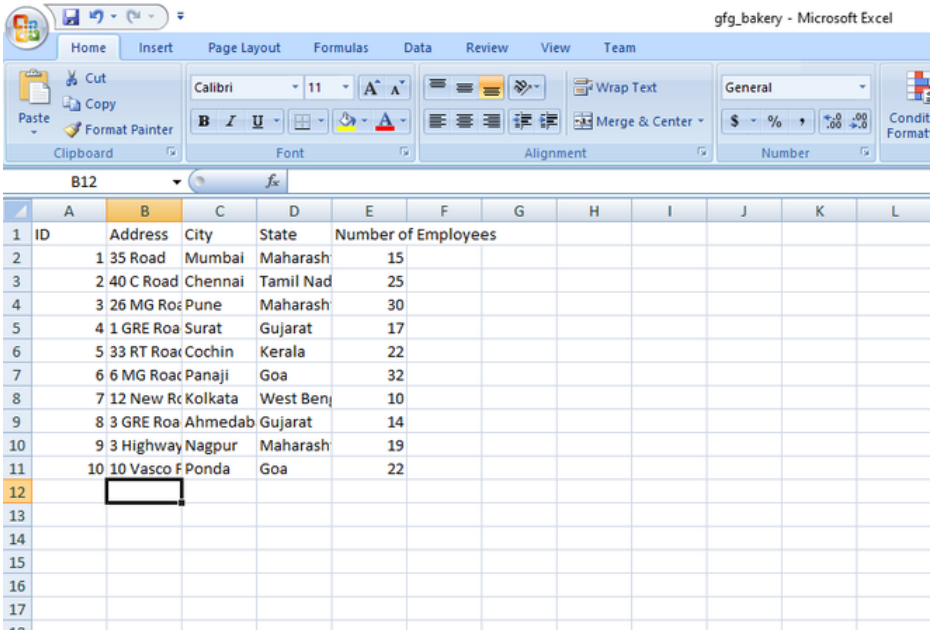
https://media.geeksforgeeks.org/wp-content/uploads/20210617091539/dfcsv.png

Read data from an Excel file

To load data present in an Excel file(.xlsx, .xls) we will follow steps as below-

- Prepare your sample dataset. Here, we have an Excel file, containing information about Bakery and its branches. It describes the number of employees, address of branches of the bakery.
- Use Pandas method ‘read_excel’.
    - Method used – *read_excel(file_path)*
    - Parameter – The method accepts, the path of the file and its name, in string format as a parameter. The file can be on a remote server, or, on a machine locally. It reads the file data, and, converts it, into a valid two-dimensional data frame object. This method, can be used, to read data present in “.xlsx” as well as “.xls” file formats.

The file contents are as follows:



The contents of "gfg_bakery.xlsx" file

https://media.geeksforgeeks.org/wp-content/uploads/20210615103200/gfgbakery.png

The code to get the data in a Pandas DataFrame is:

```
# Import the Pandas library

import pandas

# Load data from an Excel file

# Use method - read_excel(filepath)

# Method parameter - The file location(URL/path) and name

dfBakery = pandas.read_excel("gfg_bakery.xlsx")

# print the dataframe object

print(dfBakery)
```

Output:

```
# print the dataframe object
print(dfBakery)
```

```
   ID        Address       City        State  Number of Employees
0   1         35 Road     Mumbai  Maharashtra                   15
1   2       40 C Road    Chennai   Tamil Nadu                   25
2   3      26 MG Road       Pune  Maharashtra                   30
3   4      1 GRE Road      Surat      Gujarat                   17
4   5      33 RT Road     Cochin       Kerala                   22
5   6       6 MG Road     Panaji          Goa                   32
6   7     12 New Road    Kolkata  West Bengal                   10
7   8      3 GRE Road  Ahmedabad      Gujarat                   14
8   9       3 Highway     Nagpur  Maharashtra                   19
9  10  10 Vasco Road      Ponda          Goa                   22
```

The Excel data, in dataframe object

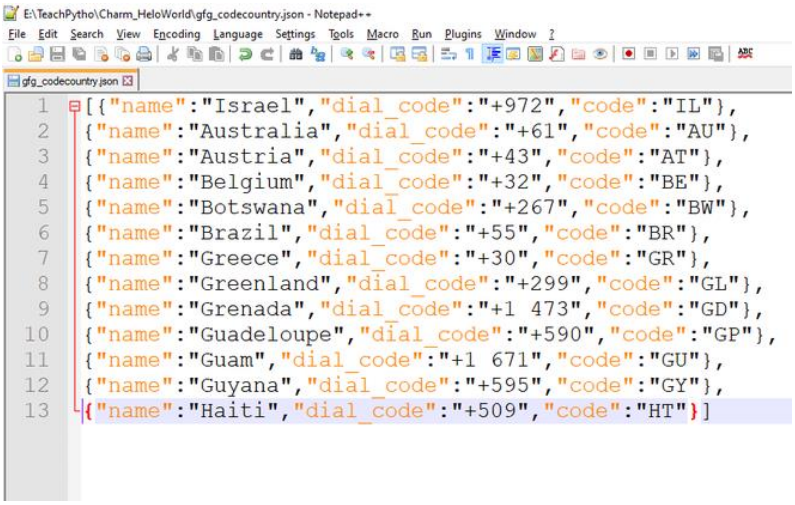
https://media.geeksforgeeks.org/wp-content/uploads/20210616081849/dfxlsx.png

Read data from a JSON file

To load data present in a JavaScript Object Notation file(.json) we will follow steps as below:

- Prepare your sample dataset. Here, we have a JSON file, containing information about Countries and their dial code.
- Use Pandas method 'read_json' .
    - Method used – *read_json(file_path)*
    - Parameter – This method, accepts the path of the file and its name, in string format, as a parameter. It reads the file data, and, converts it, into a valid two-dimensional data frame object.

The file contents are as follows:



The contents of "gfg_codecountry.json" file

https://media.geeksforgeeks.org/wp-content/uploads/20210616090239/gfgcodecountry.png

The code to get the data in a Pandas DataFrame is:

```
# Import the Pandas library

import pandas

# Load data from a JSON file

# Use method - read_json(filepath)

# Method parameter - The file location(URL/path) and name

dfCodeCountry = pandas.read_json("gfg_codecountry.json")

# print the dataframe object

print(dfCodeCountry)
```

Output:

```
# print the dataframe object
print(dfCodeCountry)
```

```
   code dial_code        name
0    IL      +972      Israel
1    AU       +61   Australia
2    AT       +43     Austria
3    BE       +32     Belgium
4    BW      +267    Botswana
5    BR       +55      Brazil
6    GR       +30      Greece
7    GL      +299   Greenland
8    GD    +1 473     Grenada
9    GP      +590  Guadeloupe
10   GU    +1 671        Guam
11   GY      +595      Guyana
12   HT      +509       Haiti
```

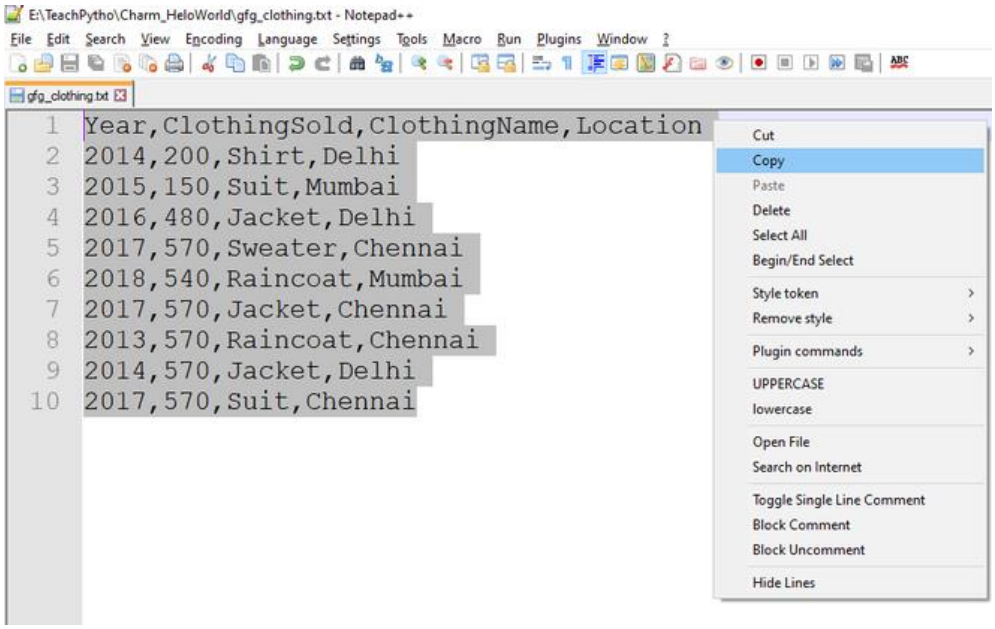
The JSON data, in dataframe objects

Read data from Clipboard

We can also transfer data present in Clipboard to a dataframe object. A clipboard is a part of Random Access Memory(RAM), where copied data is present. Whenever we copy any file, text, image, or any type of data, using the 'Copy' command, it gets stored in the Clipboard. To convert, data present here, follow the steps as mentioned below –

- Select all the contents of the file. The file should be a CSV file. It can be a '.txt' file as well, containing comma-separated values, as shown in the example. Please note, if the file contents are not in a favorable format, then, one can get a Parser Error at runtime.
- Right, Click and say Copy. Now, this data is transferred, to the computer Clipboard.
- Use Pandas method 'read_clipboard' .
    - Method used – read_clipboard
    - Parameter – The method, does not accept any parameter. It reads the latest copied data as present in the clipboard, and, converts it, into a valid two-dimensional dataframe object.

The file contents selected are as follows:



The contents of "gfg_clothing.txt" file

https://media.geeksforgeeks.org/wp-content/uploads/20210616092113/gfgclothing.png

The code to get the data in a Pandas DataFrame is

```
# Import the required library

import pandas

# Copy file contents which are in proper format

# Whatever data you have copied will get transferred to dataframe object

# Method does not accept any parameter

pdCopiedData = pd.read_clipboard()

# Print the data frame object

print(pdCopiedData)
```

Output:

```
#Print the dataframe object
print(pdCopiedData)
```

```
  Year,ClothingSold,ClothingName,Location
0                    2014,200,Shirt,Delhi
1                    2015,150,Suit,Mumbai
2                   2016,480,Jacket,Delhi
3                 2017,570,Sweater,Chennai
4                 2018,540,Raincoat,Mumbai
5                  2017,570,Jacket,Chennai
6                2013,570,Raincoat,Chennai
7                    2014,570,Jacket,Delhi
8                    2017,570,Suit,Chennai
```

The clipboard data, in dataframe object

Read data from HTML file

A webpage is usually made of HTML elements. There are different HTML tags such as <head>, <title> , <table>, based on the purpose of data display, on browser. We can transfer, the content between <table> element, present in an HTML webpage, to a Pandas data frame object. Follow the steps as mentioned below –

- Select all the elements present in the <table>, between start and end tags. Assign it, to a Python variable.
- Use Pandas method 'read_html' .
    - Method used – read_html(string within <table> tag)
    - Parameter – The method, accepts string variable, containing the elements present between <table> tag. It reads the elements, traversing through the table, <tr> and <td> tags, and, converts it, into a list object. The first element of the list object is the desired dataframe object.

The HTML webpage used is as follows:

```
<!DOCTYPE html>
<html>
    <head>
        <title>Data Ingestion with Pandas Example</title>
    </head>
<body>
<h2>Welcome To GFG</h2>
<table>
    <thead>
        <tr>
            <th>Date</th>
            <th>Empname</th>
            <th>Year</th>
            <th>Rating</th>
            <th>Region</th>
        </tr>
    </thead>
    <tbody>
        <tr>
            <td>2020-01-01</td>
            <td>Savio</td>
```

```
            <td>2004</td>



             <td>0.5</td>
            <td>South</td>
    </tr>
    <tr>
            <td>2020-01-02</td>
            <td>Rahul</td>
            <td>1998</td>
            <td>1.34</td>
            <td>East</td>
    </tr>
    <tr>
            <td>2020-01-03</td>
            <td>Tina</td>
            <td>1988</td>
            <td>1.00023</td>
            <td>West</td>
    </tr>
    <tr>
            <td>2021-01-03</td>
            <td>Sonia</td>
            <td>2001</td>
            <td>2.23</td>
```

<td>North</td>

</tr>

</tbody>

</table>

</body>

</html>

Write the following code to convert the HTML table content in the Pandas Dataframe object:

```
# Import the Pandas library
import pandas
# Variable containing the elements between <table> tag from webpage
html_string = """
<table>
    <thead>
        <tr>
            <th>Date</th>
            <th>Empname</th>
            <th>Year</th>
            <th>Rating</th>
            <th>Region</th>
        </tr>
    </thead>
 <tbody>
```

```
<tr>
        <td>2020-01-01</td>
        <td>Savio</td>
        <td>2004</td>
        <td>0.5</td>
        <td>South</td>
</tr>
<tr>
        <td>2020-01-02</td>
        <td>Rahul</td>
        <td>1998</td>
        <td>1.34</td>
        <td>East</td>
</tr>
<tr>
        <td>2020-01-03</td>
        <td>Tina</td>
        <td>1988</td>
        <td>1.00023</td>
        <td>West</td>
</tr>
<tr>
        <td>2021-01-03</td>
```

```
            <td>Sonia</td>
            <td>2001</td>
            <td>2.23</td>
            <td>North</td>
        </tr>
        <tr>
            <td>2008-01-03</td>
            <td>Milo</td>
            <td>2008</td>
            <td>3.23</td>
            <td>East</td>
        </tr>
        <tr>
            <td>2006-01-03</td>
            <td>Edward</td>
            <td>2005</td>
            <td>0.43</td>
            <td>West</td>
        </tr>
  </tbody>
</table>"""
  # Pass the string containing html table element
df = pandas.read_html(html_string)
```

```
# Since read_html, returns a list object, extract first element of the list

dfHtml = df[0]

# Print the data frame object

print(dfHtml)
```

Output:

```
#Print the data frame object
print(dfHtml)
```

	Date	Empname	Year	Rating	Region
0	2020-01-01	Savio	2004	0.50000	South
1	2020-01-02	Rahul	1998	1.34000	East
2	2020-01-03	Tina	1988	1.00023	West
3	2021-01-03	Sonia	2001	2.23000	North
4	2008-01-03	Milo	2008	3.23000	East
5	2006-01-03	Edward	2005	0.43000	West

The HTML <table> data, in dataframe object,

Read data from SQL table

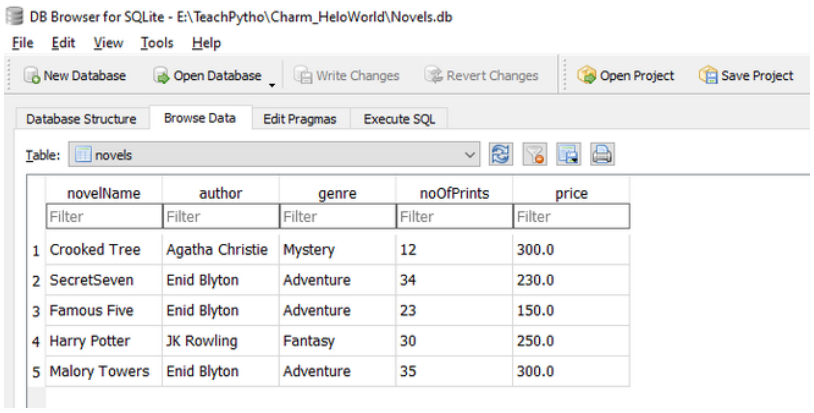
We can convert, data present in database tables, to valid dataframe objects as well. Python allows easy interface, with a variety of databases, such as SQLite, MySQL, MongoDB, etc. SQLite is a lightweight database, which can be embedded in any program. The SQLite database holds all the related SQL tables. We can load, SQLite table data, to a Pandas dataframe object. Follow the steps, as mentioned below –

- Prepare a sample SQLite table using ‘DB Browser for SQLite tool’ or any such tool. These tools allow the effortless creation, edition of database files compatible with SQLite. The database file, has a ‘.db’ file extension. In this example, we
-

have *'Novels.db'* file, containing a table called "novels". This table has information about Novels, such as Novel Name, Price, Genre, etc.

- Here, to connect to the database, we will import the 'sqlite3' module, in our code. The sqlite3 module, is an interface, to connect to the SQLite databases. The sqlite3 library is included in Python, since Python version 2.5. Hence, no separate installation is required. To connect to the database, we will use the SQLite method 'connect', which returns a connection object. The connect method accepts the following parameters:
    - *database_name* – The name of the database in which the table is present. This is a .db extension file. If the file is present, an open connection object is returned. If the file is not present, it is created first and then a connection object is returned.
- Use Pandas method 'read_sql_query'.
    - Method used – *read_sql_query*
    - Parameter – This method accepts the following parameters
        - SQL query – Select query, to fetch the required rows from the table.
        - Connection object – The connection object returned by the 'connect' method.
          The *read_sql_query* method, converts, the resultant rows of the query, to a dataframe object.
- Print the dataframe object using the print method.

The ***Novels.db*** database file looks as follows –



The novels table, as seen, using DB Browser for SQLite tool

Write the following code to convert the Novels table, in Pandas Data frame object:

```
# Import the required libraries
import sqlite3
import pandas
# Prepare a connection object
# Pass the Database name as a parameter
conn = sqlite3.connect("Novels.db")
# Use read_sql_query method
# Pass SELECT query and connection object as parameter
pdSql = pd.read_sql_query("SELECT * FROM novels", conn)
# Print the dataframe object
print(pdSql)
# Close the connection object
conn.close()
```

Output:

```
#Print the dataframe object
print(pdSql)
```

	novelName	author	genre	noOfPrints	price
0	Crooked Tree	Agatha Christie	Mystery	12	300.0
1	SecretSeven	Enid Blyton	Adventure	34	230.0
2	Famous Five	Enid Blyton	Adventure	23	150.0
3	Harry Potter	JK Rowling	Fantasy	30	250.0
4	Malory Towers	Enid Blyton	Adventure	35	300.0

The Novels table data in dataframe object

Reference:

https://www.geeksforgeeks.org/streamlined-data-ingestion-with-pandas/

Activity 8

Aim: Pandas functionalities for Series & Data Frames

Learning outcome: Able to install and different operation in python

Duration: 3.5 hour

List of Hardware/Software requirements:

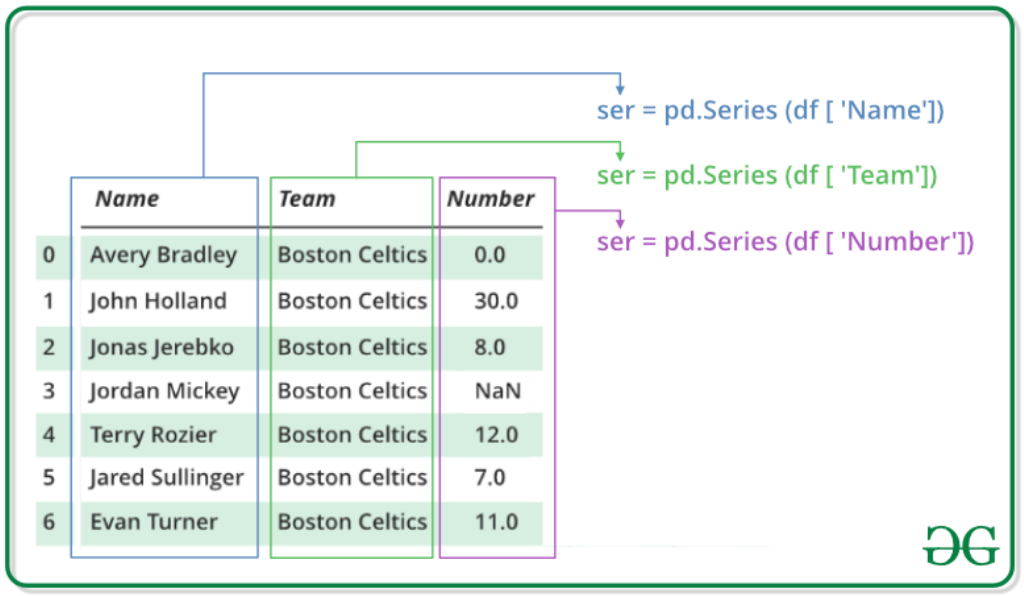
1. Laptop/Computer with Windows 10/11
2. Internet connection
3. Python

Program / Procedure:

Python | Pandas Series

Pandas Series is a one-dimensional labeled array capable of holding data of any type (integer, string, float, python objects, etc.). The axis labels are collectively called *index*. Pandas Series is nothing but a column in an excel sheet.

Labels need not be unique but must be a hashable type. The object supports both integer and label-based indexing and provides a host of methods for performing operations involving the index.



In this article, we are using nba.csv file.

We will get a brief insight on all these basic operations which can be performed on Pandas Series:

- Creating a Series
- Accessing element of Series
- Indexing and Selecting Data in Series
- Binary operation on Series
- Conversion Operation on Series

Creating a Pandas Series

In the real world, a Pandas Series will be created by loading the datasets from existing storage, storage can be SQL Database, CSV file, and Excel file. Pandas Series can be created from the lists, dictionary, and from a scalar value etc. Series can be created in different ways, here are some ways by which we create a series:

Creating a series from array: In order to create a series from array, we have to import a numpy module and have to use array() function.

```
# import pandas as pd
import pandas as pd
```

```
# import numpy as np
import numpy as np

# simple array
data = np.array(['g','e','e','k','s'])
 ser = pd.Series(data)
print(ser)
```

Output:

```
0    g
1    e
2    e
3    k
4    s
dtype: object
```

Creating a series from Lists:

In order to create a series from list, we have to first create a list after that we can create a series from list.

```
import pandas as pd

# a simple list

list = ['g', 'e', 'e', 'k', 's']

# create series form a list

ser = pd.Series(list)

print(ser)
```

Output:

```
0    g
1    e
2    e
3    k
4    s
dtype: object
```

Accessing element of Series

There are two ways through which we can access element of series, they are :

- Accessing Element from Series with Position
- Accessing Element Using Label (index)

Accessing Element from Series with Position: In order to access the series element refers to the index number. Use the index operator [ ] to access an element in a series. The index must be an integer. In order to access multiple elements from a series, we use Slice operation.

Accessing first 5 elements of Series

```
# import pandas and numpy

import pandas as pd

import numpy as np

 # creating simple array

data = np.array(['g','e','e','k','s','f', 'o','r','g','e','e','k','s'])

ser = pd.Series(data)

#retrieve the first element

print(ser[:5])
```

Output:

```
0    g
1    e
2    e
3    k
4    s
dtype: object
```

Accessing Element Using Label (index):
In order to access an element from series, we have to set values by index label. A Series is like a fixed-size dictionary in that you can get and set values by index label.

Accessing a single element using index label

```
# import pandas and numpy

import pandas as pd

import numpy as np

# creating simple array

data = np.array(['g','e','e','k','s','f', 'o','r','g','e','e','k','s'])

ser = pd.Series(data,index=[10,11,12,13,14,15,16,17,18,19,20,21,22])

# accessing a element using index element

print(ser[16])
```

Output:

o

Indexing and Selecting Data in Series

Indexing in pandas means simply selecting particular data from a Series. Indexing could mean selecting all the data, some of the data from particular columns. Indexing can also be known as **Subset Selection**.

Indexing a Series using indexing operator []:
Indexing operator is used to refer to the square brackets following an object. The .loc and .iloc indexers also use the indexing operator to make selections. In this indexing operator to refer to df[ ].

```
# importing pandas module

import pandas as pd

# making data frame

df = pd.read_csv("nba.csv")

ser = pd.Series(df['Name'])

data = ser.head(10)

data
```

Output:

```
0    Avery Bradley
1      Jae Crowder
2     John Holland
3      R.J. Hunter
4    Jonas Jerebko
5     Amir Johnson
6    Jordan Mickey
7     Kelly Olynyk
8     Terry Rozier
9     Marcus Smart
Name: Name, dtype: object
```

Now we access the element of series using index operator [ ].

using indexing operator

data[3:6]

Output:

```
3      R.J. Hunter
4    Jonas Jerebko
5     Amir Johnson
Name: Name, dtype: object
```

Indexing a Series using .loc[ ] :
This function selects data by refering the explicit index . The df.loc indexer selects data in a different way than just the indexing operator. It can select subsets of data.

importing pandas module

import pandas as pd

making data frame

df = pd.read_csv("nba.csv")

ser = pd.Series(df['Name'])

data = ser.head(10)

data

Output:

```
0    Avery Bradley
1      Jae Crowder
2     John Holland
3      R.J. Hunter
4    Jonas Jerebko
5     Amir Johnson
6    Jordan Mickey
7     Kelly Olynyk
8     Terry Rozier
9     Marcus Smart
Name: Name, dtype: object
```

Now we access the element of series using .loc[] function.

using .loc[] function

data.loc[3:6]

Output :

```
3      R.J. Hunter
4    Jonas Jerebko
5     Amir Johnson
6    Jordan Mickey
Name: Name, dtype: object
```

Indexing a Series using .iloc[ ] :

This function allows us to retrieve data by position. In order to do that, we'll need to specify the positions of the data that we want. The df.iloc indexer is very similar to df.loc but only uses integer locations to make its selections.

importing pandas module

import pandas as pd

making data frame

df = pd.read_csv("nba.csv")

ser = pd.Series(df['Name'])

data = ser.head(10)

data

Output:

```
0    Avery Bradley
1      Jae Crowder
2     John Holland
3      R.J. Hunter
4    Jonas Jerebko
5     Amir Johnson
6    Jordan Mickey
7     Kelly Olynyk
8     Terry Rozier
9     Marcus Smart
Name: Name, dtype: object
```

Now we access the element of Series using .iloc[] function.

```
# using .iloc[] function

data.iloc[3:6]
```

Output:

```
3      R.J. Hunter
4    Jonas Jerebko
5     Amir Johnson
Name: Name, dtype: object
```

Binary Operation on Series

We can perform binary operation on series like addition, subtraction and many other operation. In order to perform binary operation on series we have to use some function like .add(),.sub() etc..

Code #1:

```
# importing pandas module

import pandas as pd

# creating a series

data = pd.Series([5, 2, 3,7], index=['a', 'b', 'c', 'd'])

 # creating a series

data1 = pd.Series([1, 6, 4, 9], index=['a', 'b', 'd', 'e'])

print(data, "\n\n", data1)
```

Output:

```
a    5
b    2
c    3
d    7
dtype: int64

 a    1
b    6
d    4
e    9
dtype: int64
```

Now we add two series using .add() function.

```
# adding two series using

# .add

data.add(data1, fill_value=0)
```

Output:

```
a     6.0
b     8.0
c     3.0
d    11.0
e     9.0
dtype: float64
```

Code #2:

```
# importing pandas module

import pandas as pd

# creating a series

data = pd.Series([5, 2, 3,7], index=['a', 'b', 'c', 'd'])

 # creating a series

data1 = pd.Series([1, 6, 4, 9], index=['a', 'b', 'd', 'e'])

print(data, "\n\n", data1)
```

Output:

```
a    5
b    2
c    3
d    7
dtype: int64

 a    1
b    6
d    4
e    9
dtype: int64
```

Now we subtract two series using .sub function.

```
# subtracting two series using

# .sub

data.sub(data1, fill_value=0)
```

Output :

```
a    4.0
b   -4.0
c    3.0
d    3.0
e   -9.0
dtype: float64
```

Conversion Operation on Series

In conversion operation we perform various operation like changing datatype of series, changing a series to list etc. In order to perform conversion operation we have various function which help in conversion like .astype(), .tolist() etc.

Code #1:

```
# Python program using astype

# to convert a datatype of series

# importing pandas module

import pandas as pd

# reading csv file from url


data = pd.read_csv("nba.csv")
```

```
# dropping null value columns to avoid errors

data.dropna(inplace = True)

# storing dtype before converting

before = data.dtypes

# converting dtypes using astype

data["Salary"]= data["Salary"].astype(int)

data["Number"]= data["Number"].astype(str)

# storing dtype after converting

after = data.dtypes

# printing to compare

print("BEFORE CONVERSION\n", before, "\n")

print("AFTER CONVERSION\n", after, "\n")
```

Output:

```
BEFORE CONVERSION
 Name         object
Team         object
Number      float64
Position     object
Age         float64
Height       object
Weight      float64
College      object
Salary      float64
dtype: object

AFTER CONVERSION
 Name         object
Team         object
Number       object
Position     object
Age         float64
Height       object
Weight      float64
College      object
Salary        int32
dtype: object
```

Code #2:

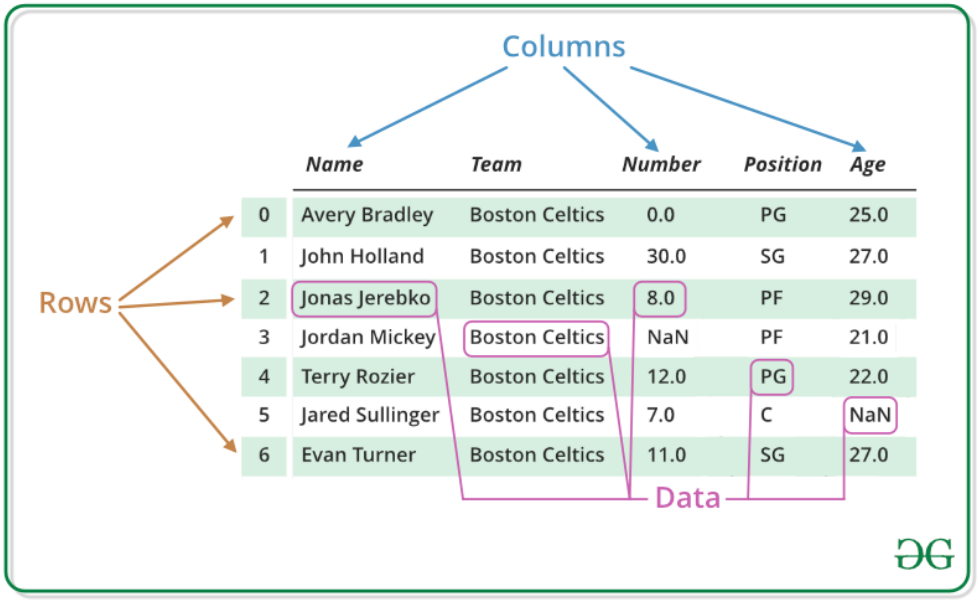
```
# Python program converting

# a series into list

# importing pandas module

import pandas as pd

# importing regex module

import re

# making data frame

data = pd.read_csv("nba.csv")

# removing null values to avoid errors

data.dropna(inplace = True)

# storing dtype before operation

dtype_before = type(data["Salary"])

# converting to list

salary_list = data["Salary"].tolist()

# storing dtype after operation

dtype_after = type(salary_list)

# printing dtype

print("Data type before converting = {}\nData type after converting = {}"

    .format(dtype_before, dtype_after))

# displaying list

salary_list
```

Output:

```
Data type before converting = <class 'pandas.core.series.Series'>
Data type after converting = <class 'list'>

[7730337.0,
 6796117.0,
 1148640.0,
 1170960.0,
 2165160.0,
 1824360.0,
 3431040.0,
 2569260.0,
 6912869.0,
 3425510.0,
 1749840.0,
 2616975.0,
 845059.0,
      .
      .
      .
 9463484.0,
 12000000.0
 15409570.0
 1348440.0,
 981348.0,
 2239800.0,
 2433333.0,
 947276.0]
```

Python | Pandas DataFrame

Pandas DataFrame is two-dimensional size-mutable, potentially heterogeneous tabular data structure with labeled axes (rows and columns). A Data frame is a two-dimensional data structure, i.e., data is aligned in a tabular fashion in rows and columns. Pandas DataFrame consists of three principal components, the **data**, **rows**, and **columns**.



We will get a brief insight on all these basic operation which can be performed on Pandas DataFrame :

- Creating a DataFrame
- Dealing with Rows and Columns
- Indexing and Selecting Data
- Working with Missing Data
- Iterating over rows and columns

Creating a Pandas DataFrame

In the real world, a Pandas DataFrame will be created by loading the datasets from existing storage, storage can be SQL Database, CSV file, and Excel file. Pandas DataFrame can be created from the lists, dictionary, and from a list of dictionary etc. Dataframe can be created in different ways here are some ways by which we create a dataframe:

Creating a dataframe using List: DataFrame can be created using a single list or a list of lists.

```
# import pandas as pd

import pandas as pd

# list of strings

lst = ['Geeks', 'For', 'Geeks', 'is',

        'portal', 'for', 'Geeks']

# Calling DataFrame constructor on list

df = pd.DataFrame(lst)

print(df)
```

Output:

	0
0	Geeks
1	For
2	Geeks
3	is
4	portal
5	for
6	Geeks

Reference:

- https://www.geeksforgeeks.org/python-pandas-series/#:~:text=Pandas%20Series%20is%20a%20one,must%20be%20a%20hashable%20type.

Activity 9

Aim: Grouping, Merging, concatenating, joining, segregation

Learning outcome: Able to install and different operation in python

Duration: 3.5 hour

List of Hardware/Software requirements:

1. Laptop/Computer with Windows 10/11
2. Internet connection
3. Python

Program / Procedure:

Use groupby() function to form groups based on more than one category (i.e. Use more than one column to perform the splitting).

```
# importing pandas as pd

import pandas as pd

# Creating the dataframe

df = pd.read_csv("nba.csv")

# First grouping based on "Team"

# Within each team we are grouping based on "Position"

gkk = df.groupby(['Team', 'Position'])

# Print the first value in each group

gkk.first()
```

Output:

Team	Position	Name	Number	Age	Height	Weight	College	Salary
Atlanta Hawks	C	Al Horford	15.0	30.0	6-10	245.0	Florida	12000000.0
	PF	Kris Humphries	43.0	31.0	6-9	235.0	Minnesota	1000000.0
	PG	Dennis Schroder	17.0	22.0	6-1	172.0	Wake Forest	1763400.0
	SF	Kent Bazemore	24.0	26.0	6-5	201.0	Old Dominion	2000000.0
	SG	Tim Hardaway Jr.	10.0	24.0	6-6	205.0	Michigan	1304520.0
Boston Celtics	C	Kelly Olynyk	41.0	25.0	7-0	238.0	Gonzaga	2165160.0
	PF	Jonas Jerebko	8.0	29.0	6-10	231.0	LSU	5000000.0
	PG	Avery Bradley	0.0	25.0	6-2	180.0	Texas	7730337.0
	SF	Jae Crowder	99.0	25.0	6-6	235.0	Marquette	6796117.0
	SG	John Holland	30.0	27.0	6-5	205.0	Boston University	1148640.0
Brooklyn Nets	C	Brook Lopez	11.0	28.0	7-0	275.0	Stanford	19689000.0
	PF	Chris McCullough	1.0	21.0	6-11	200.0	Syracuse	1140240.0
	PG	Jarrett Jack	2.0	32.0	6-3	200.0	Georgia Tech	6300000.0
	SG	Bojan Bogdanovic	44.0	27.0	6-8	216.0	Oklahoma State	3425510.0
Charlotte Hornets	C	Al Jefferson	25.0	31.0	6-10	289.0	Wisconsin	13500000.0
	PF	Tyler Hansbrough	50.0	30.0	6-9	250.0	North Carolina	947276.0
	PG	Jorge Gutierrez	12.0	27.0	6-3	189.0	California	189455.0
	SF	Michael Kidd-Gilchrist	14.0	22.0	6-7	232.0	Kentucky	6331404.0

groupby() is a very powerful function with a lot of variations. It makes the task of splitting the dataframe over some criteria really easy and efficient.

Pandas DataFrame merge() Method

```
import pandas as pd

data1 = {
  "name": ["Sally", "Mary", "John"],
  "age": [50, 40, 30]
}
data2 = {
  "name": ["Sally", "Peter", "Micky"],
  "age": [77, 44, 22]
}

df1 = pd.DataFrame(data1)
df2 = pd.DataFrame(data2)

newdf = df1.merge(df2, how='right')
```

Output:

```
    name  age
0  Sally   77
1  Peter   44
2  Micky   22
```

Concatenating 2 DataFrames horizontally with axis = 1.

```
# importing the module
import pandas as pd
# creating the DataFrames
df1 = pd.DataFrame({'A': ['A0', 'A1', 'A2', 'A3'],
                    'B': ['B0', 'B1', 'B2', 'B3']})
display('df1:', df1)
df2 = pd.DataFrame({'C': ['C0', 'C1', 'C2', 'C3'],
                    'D': ['D0', 'D1', 'D2', 'D3']})
display('df2:', df2)
# concatenating
display('After concatenating:')
display(pd.concat([df1, df2],
                  axis = 1))
```

Output:

'df1:'

	A	B
0	A0	B0
1	A1	B1
2	A2	B2
3	A3	B3

'df2:'

	C	D
0	C0	D0
1	C1	D1
2	C2	D2
3	C3	D3

'After concatenating:'

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

Joining DataFrame

In order to join dataframe, we use .join() function this function is used for combining the columns of two potentially differently-indexed DataFrames into a single result DataFrame.

```
# importing pandas module

import pandas as pd

# Define a dictionary containing employee data

data1 = {'Name':['Jai', 'Princi', 'Gaurav', 'Anuj'],

        'Age':[27, 24, 22, 32]}
```

```
# Define a dictionary containing employee data
data2 = {'Address':['Allahabad', 'Kannuaj', 'Allahabad', 'Kannuaj'],
        'Qualification':['MCA', 'Phd', 'Bcom', 'B.hons']}
# Convert the dictionary into DataFrame
df = pd.DataFrame(data1,index=['K0', 'K1', 'K2', 'K3'])
# Convert the dictionary into DataFrame
df1 = pd.DataFrame(data2, index=['K0', 'K2', 'K3', 'K4'])
print(df, "\n\n", df1)
```

Now we are use .join() method in order to join dataframes

```
# joining dataframe
res = df.join(df1)
res
```

Output:

Left

	Name	Age
K0	Jai	27
K1	Princi	24
K2	Gaurav	22
K3	Anuj	32

Right

	Address	Qualification
K0	Allahabad	MCA
K2	Kannuaj	Phd
K3	Allahabad	Bcom
K4	Kannuaj	B.hons

	Name	Age	Address	Qualification
K0	Jai	27	Allahabad	MCA
K1	Princi	24	NaN	NaN
K2	Gaurav	22	Kannuaj	Phd
K3	Anuj	32	Allahabad	Bcom

Now we use how = 'outer' in order to get union

```
# getting union

res1 = df.join(df1, how='outer')

res1
```

Output:

	Name	Age	Address	Qualification	Mobile No
0	Jai	27	Nagpur	Msc	97
1	Princi	24	Kanpur	MA	91
2	Gaurav	22	Allahabad	MCA	58
3	Anuj	32	Kannuaj	Phd	76

	Name	Age	Address	Qualification	Salary
2	Gaurav	22	Allahabad	MCA	1000
3	Anuj	32	Kannuaj	Phd	2000
6	Dhiraj	12	Allahabad	Bcom	3000
7	Hitesh	52	Kannuaj	B.hons	4000

	Name	Age	Address	Qualification	Mobile No	Name	Age	Address	Qualification	Salary
2	Gaurav	22	Allahabad	MCA	58	Gaurav	22	Allahabad	MCA	1000
3	Anuj	32	Kannuaj	Phd	76	Anuj	32	Kannuaj	Phd	2000

Reference:

- https://www.geeksforgeeks.org/python-pandas-merging-joining-and-concatenating/

Activity 10

Aim: Python lambda function operations on series or data frames

Learning outcome:

Duration: 3 hours

List of Hardware/Software requirements:

1. Laptop/Computer with Windows OS / Ubuntu
2. Python / Jupyter notebook

Program / Procedure:

Lambda Function

Lambda function contains a single expression.

The Lambda function is a small function that can also use as an anonymous function means it doesn't require any name. The lambda function is useful to solve small problems with less code.

The following syntax is used to apply a lambda function on pandas dataframe:

dataframe.apply(lambda x: x+2)

Applying Lambda Function on a Single Column Using DataFrame.assign() Method

The dataframe.assign() method applies the Lambda function on a single column.

We have applied a lambda function on the column Students Marks. After applying the Lambda function, the student percentages are calculated and stored in a new Percentage column.

The following implementation applies a lambda function on a single column in Pandas dataframe.

```
import pandas as pd
# initialization of list
students_record= [['Samreena',900],['Mehwish',750],['Asif',895],
    ['Mirha',800],['Affan',850],['Raees',950]]
# pandas dataframe creation
dataframe = pd.DataFrame(students_record,columns=['Student Names','Student Marks'])
```

```
# using Lambda function
dataframe1 = dataframe.assign(Percentage = lambda x: (x['Student Marks'] /1000 * 100))
# display dataframe
print(dataframe1)
```

Output/Results snippet:

	Student Names	Student Marks	Percentage
0	Samreena	900	90.0
1	Mehwish	750	75.0
2	Asif	895	89.5
3	Mirha	800	80.0
4	Affan	850	85.0
5	Raees	950	95.0

Program / Procedure:

Applying Lambda Function on Multiple Columns Using DataFrame.assign() Method

We have four columns Student Names, Computer, Math, and Physics. We applied a Lambda function on multiple subjects columns such as Computer, Math, and Physics to calculate the obtained marks stored in the Marks_Obtained column.

```
import pandas as pd
# nested list initialization
values_list = [['Samreena',85, 75, 100], ['Mehwish', 90, 75, 90], ['Asif', 95, 82, 80],
        ['Mirha', 75, 88, 68], ['Affan', 80, 63, 70], ['Raees', 91, 64, 90]]
# pandas dataframe creation
df = pd.DataFrame(values_list, columns=['Student Names','Computer', 'Math', 'Physics'])
# applying Lambda function
dataframe = df.assign(Marks_Obtained=lambda x: (x['Computer'] + x['Math'] + x['Physics']))
# display dataframe
```

print(dataframe)

Output/Results snippet:

	Student Names	Computer	Math	Physics	Marks_Obtained
0	Samreena	85	75	100	260
1	Mehwish	90	75	90	255
2	Asif	95	82	80	257
3	Mirha	75	88	68	231
4	Affan	80	63	70	213
5	Raees	91	64	90	245

Program / Procedure:

Applying Lambda Function on a Single Row Using DataFrame.apply() Method

The dataframe.apply() method applies the Lambda function on a single row.

We applied the lambda function a single row axis=1. Using the lambda function, we incremented each person's Monthly Income by 1000.

```
import pandas as pd
df=pd.DataFrame({
    'ID':[1,2,3,4,5],
    'Names':['Samreena','Asif','Mirha','Affan','Mahwish'],
    'Age':[20,25,15,10,30],
    'Monthly Income':[4000,6000,5000,2000,8000]
})
df['Monthly Income']=df.apply(lambda x: x['Monthly Income']+1000,axis=1)
print(df)
```

Output/Results snippet:

	ID	Names	Age	Monthly Income
0	1	Samreena	20	5000
1	2	Asif	25	7000
2	3	Mirha	15	6000
3	4	Affan	10	3000
4	5	Mahwish	30	9000

Program / Procedure:

Filtering Data by Applying Lambda Function

We can also filter the desired data by applying the Lambda function.

The filter() function takes pandas series and a lambda function. The Lambda function applies to the pandas series that returns the specific results after filtering the given series.

We have applied the lambda function on the Age column and filtered the age of people under 25 years.

```
import pandas as pd
df=pd.DataFrame({
   'ID':[1,2,3,4,5],
   'Names':['Samreena','Asif','Mirha','Affan','Mahwish'],
   'Age':[20,25,15,10,30],
   'Monthly Income':[4000,6000,5000,2000,8000]
})
print(list(filter(lambda x: x<25,df['Age'])))
```

Output/Results snippet:

```
[20, 15, 10]
```

Program / Procedure:

Use the map() Function by Applying Lambda Function

We can use the map() and lambda functions.

The lambda function applies on series to map the series based on the input correspondence. This function is useful to substitute or replace a series with other values.

When we use the map() function, the input size will equal the output size.

```
import pandas as pd
```

```
df=pd.DataFrame({
    'ID':[1,2,3,4,5],
    'Names':['Samreena','Asif','Mirha','Affan','Mahwish'],
    'Age':[20,25,15,10,30],
    'Monthly Income':[4000,6000,5000,2000,8000]
})
df['Monthly Income']=list(map(lambda x: int(x+x*0.5),df['Monthly Income']))
print(df)
```

Output/Results snippet:

	ID	Names	Age	Monthly Income
0	1	Samreena	20	6000
1	2	Asif	25	9000
2	3	Mirha	15	7500
3	4	Affan	10	3000
4	5	Mahwish	30	12000

Program / Procedure:

Use if-else Statement by Applying Lambda Function

We can also apply the conditional statements on pandas dataframes using the lambda function.

We used the conditional statement inside the lambda function. We applied the condition on the Monthly Income column.

If the monthly income is greater and equal to 5000, add Stable inside the Category column; otherwise, add Unstable.

```
import pandas as pd
df=pd.DataFrame({

 'ID':[1,2,3,4,5],
```

```
    'Names':['Samreena','Asif','Mirha','Affan','Mahwish'],
    'Age':[20,25,15,10,30],
    'Monthly Income':[4000,6000,5000,2000,8000]
})
df['Category']=df['Monthly Income'].apply(lambda x: 'Stable' if x>=5000 else 'UnStable')
print(df)
```

Output/Results snippet:

	ID	Names	Age	Monthly Income	Category
0	1	Samreena	20	4000	UnStable
1	2	Asif	25	6000	Stable
2	3	Mirha	15	5000	Stable
3	4	Affan	10	2000	UnStable
4	5	Mahwish	30	8000	Stable

References:

1. https://www.delftstack.com/howto/python-pandas/apply-lambda-functions-to-pandas-dataframe/

Activity 11

Aim: Dealing with missing and noisy data

Learning outcome:

Duration: 3 hours

List of Hardware/Software requirements:

1. Laptop/Computer with Windows OS / Ubuntu
2. Python / Jupyter notebook

Program / Procedure:

Dealing with Missing data

```
import pandas as pd
# Creating the dataframe
df = pd.DataFrame({'Job Position': ['CEO', 'Senior Manager', 'Junior Manager', 'Employee', 'Assistant Staff'], 'Years of Experience':[5, 4, 3, None, 1], 'Salary':[100000,80000,None,40000, 20000]})
# Viewing the contents of the dataframe
df.head()
```

Output/Results snippet:

	Job Position	Years of Experience	Salary
0	CEO	5.0	100000.0
1	Senior Manager	4.0	80000.0
2	Junior Manager	3.0	NaN
3	Employee	NaN	40000.0
4	Assistant Staff	1.0	20000.0

Some of the ways to handle missing data are listed below:

1. Data Removal

Remove the missing data rows (data points) from the dataset. However, when using this technique will decrease the available dataset and in turn result in less robustness of data point if the size of dataset is originally small.

```
# Dropping the 2nd and 3rd index

dropped_df = df.drop([2,3],axis=0)

# Viewing the dataframe

dropped_df
```

Output/Results snippet:

	Job Position	Years of Experience	Salary
0	CEO	5.0	100000.0
1	Senior Manager	4.0	80000.0
4	Assistant Staff	1.0	20000.0

2. Fill missing value through statistical imputation

Fill the missing data by taking the mean or median of the available data points. Generally, the median of the data points is used to fill the missing values as it is not affected heavily by outliers like the mean. Here, we have used the median to fill the missing data.

```
# Filling each column with their mean values

df['Years of Experience'] = df['Years of Experience'].fillna(df['Years of Experience'].mean())

df['Salary'] = df['Salary'].fillna(df['Salary'].mean())

# Viewing the dataframe

df
```

Output/Results snippet:

	Job Position	Years of Experience	Salary
0	CEO	5.00	100000.0
1	Senior Manager	4.00	80000.0
2	Junior Manager	3.00	60000.0
3	Employee	3.25	40000.0
4	Assistant Staff	1.00	20000.0

Program / Procedure:

Dealing with Noisy data

Binning method is used to smoothing data or to handle noisy data. In this method, the data is first sorted and then the sorted values are distributed into a number of buckets or bins. As binning methods consult the neighborhood of values, they perform local smoothing.

There are three approaches to perform smoothing –

Smoothing by bin means: In smoothing by bin means, each value in a bin is replaced by the mean value of the bin.

Smoothing by bin median: In this method each bin value is replaced by its bin median value.

Smoothing by bin boundary: In smoothing by bin boundaries, the minimum and maximum values in a given bin are identified as the bin boundaries. Each bin value is then replaced by the closest boundary value.

Approach:

1. Sort the array of given data set.
2. Divides the range into N intervals, each containing the approximately same number of samples (Equal-depth partitioning).
3. Store mean/ median/ boundaries in each row.

Examples:

Sorted data for price (in dollars): 4, 8, 9, 15, 21, 21, 24, 25, 26, 28, 29, 34

Smoothing by bin means:

- Bin 1: 9, 9, 9, 9

- Bin 2: 23, 23, 23, 23
- Bin 3: 29, 29, 29, 29

Smoothing by bin boundaries:

- Bin 1: 4, 4, 4, 15
- Bin 2: 21, 21, 25, 25
- Bin 3: 26, 26, 26, 34

Smoothing by bin median:

- Bin 1: 9 9, 9, 9
- Bin 2: 24, 24, 24, 24
- Bin 3: 29, 29, 29, 29

Use the following basic syntax to perform data binning on a pandas DataFrame:

```
import pandas as pd
#perform binning with 3 bins
df['new_bin'] = pd.qcut(df['variable_name'], q=3)
```

Example 1

```
import pandas as pd
#create DataFrame
df = pd.DataFrame({'points': [4, 4, 7, 8, 12, 13, 15, 18, 22, 23, 23, 25],
                   'assists': [2, 5, 4, 7, 7, 8, 5, 4, 5, 11, 13, 8],
                   'rebounds': [7, 7, 4, 6, 3, 8, 9, 9, 12, 11, 8, 9]})
#view DataFrame
print(df)
```

Output/Results snippet:

	points	assists	rebounds
0	4	2	7
1	4	5	7
2	7	4	4
3	8	7	6
4	12	7	3
5	13	8	8
6	15	5	9
7	18	4	9
8	22	5	12
9	23	11	11
10	23	13	8
11	25	8	9

Perform Basic Data Binning

The following code shows how to perform data binning on the points variable using the qcut() function with specific break marks:

```
#perform data binning on points variable

df['points_bin'] = pd.qcut(df['points'], q=3)

#view updated DataFrame

print(df)
```

Output/Results snippet:

	points	assists	rebounds	points_bin
0	4	2	7	(3.999, 10.667]
1	4	5	7	(3.999, 10.667]
2	7	4	4	(3.999, 10.667]
3	8	7	6	(3.999, 10.667]
4	12	7	3	(10.667, 19.333]
5	13	8	8	(10.667, 19.333]
6	15	5	9	(10.667, 19.333]
7	18	4	9	(10.667, 19.333]
8	22	5	12	(19.333, 25.0]
9	23	11	11	(19.333, 25.0]
10	23	13	8	(19.333, 25.0]
11	25	8	9	(19.333, 25.0]

Notice that each row of the data frame has been placed in one of three bins based on the value in the points column.

We can use the value_counts() function to find how many rows have been placed in each bin:

```
#count frequency of each bin

df['points_bin'].value_counts()
```

Output/Results snippet:

```
(3.999, 10.667]     4

(10.667, 19.333]    4

(19.333, 25.0]      4

Name: points_bin, dtype: int64
```

References:

1. https://medium.com/@theclickreader/data-preprocessing-in-python-handling-missing-data-b717bcd4a264
2. https://www.geeksforgeeks.org/python-binning-method-for-data-smoothing/
3. https://www.statology.org/data-binning-in-python/

Activity 12

Aim: Finding outliers

Learning outcome:

Duration: 3 hours

List of Hardware/Software requirements:

3. Laptop/Computer with Windows OS / Ubuntu
4. Python / Jupyter notebook

Program / Procedure:

An Outlier is a data-item/object that deviates significantly from the rest of the (so-called normal) objects. They can be caused by measurement or execution errors. The analysis for outlier detection is referred to as outlier mining. There are many ways to detect the outliers, and the removal process is the data frame same as removing a data item from the panda's data frame.

Here pandas data frame is used for a more realistic approach as in real-world project need to detect the outliers arouse during the data analysis step, the same approach can be used on lists and series-type objects.

Dataset:

Dataset used is Boston Housing dataset as it is preloaded in the sklearn library.

```
# Importing

import sklearn

from sklearn.datasets import load_boston

import pandas as pd

import matplotlib.pyplot as plt

# Load the dataset

bos_hou = load_boston()

# Create the dataframe

column_name = bos_hou.feature_names

df_boston = pd.DataFrame(bos_hou.data)
```

df_boston.columns = column_name

df_boston.head()

Output/Results snippet:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4.98
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9.14
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4.03
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2.94
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	5.33

Detecting the outliers

Outliers can be detected using visualization, implementing mathematical formulas on the dataset, or using the statistical approach. All of these are discussed below.

1. Visualization

Example 1: Using Box Plot

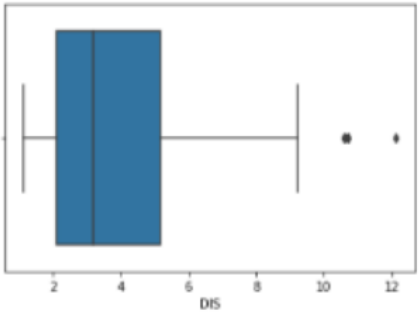
It captures the summary of the data effectively and efficiently with only a simple box and whiskers. Boxplot summarizes sample data using 25th, 50th, and 75th percentiles. One can just get insights(quartiles, median, and outliers) into the dataset by just looking at its boxplot.

Box Plot

import seaborn as sns

sns.boxplot(df_boston['DIS'])

Output/Results snippet:



In the above graph, can clearly see those values above 10 are acting as the outliers.

```
# Position of the Outlier

print(np.where(df_boston['DIS']>10))
```

Output/Results snippet:

```
(array([351, 352, 353, 354, 355]),)
```

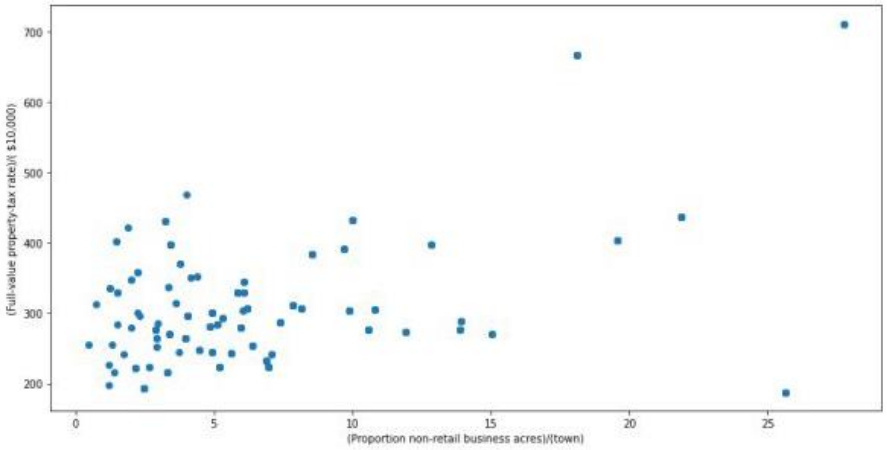
Example 2: Using ScatterPlot.

It is used when you have paired numerical data, or when your dependent variable has multiple values for each reading independent variable, or when trying to determine the relationship between the two variables. In the process of utilizing the scatter plot, one can also use it for outlier detection.

To plot the scatter plot one requires two variables that are somehow related to each other. So here, 'Proportion of non-retail business acres per town' and 'Full-value property-tax rate per $10,000' are used whose column names are "INDUS" and "TAX" respectively.

```
# Scatter plot

fig, ax = plt.subplots(figsize = (18,10))

ax.scatter(df_boston['INDUS'], df_boston['TAX'])

# x-axis label

ax.set_xlabel('(Proportion non-retail business acres)/(town)')

# y-axis label

ax.set_ylabel('(Full-value property-tax rate)/( $10,000)')

plt.show()
```

Output/Results snippet:



Looking at the graph can summarize that most of the data points are in the bottom left corner of the graph but there are few points that are exactly;y opposite that is the top right corner of the graph. Those points in the top right corner can be regarded as Outliers.

Using approximation can say all those data points that are x>20 and y>600 are outliers. The following code can fetch the exact position of all those points that satisfy these conditions.

```
# Position of the Outlier

print(np.where((df_boston['INDUS']>20) & (df_boston['TAX']>600)))
```

Output/Results snippet:

```
(array([488, 489, 490, 491, 492]),)
```

2. Z-score

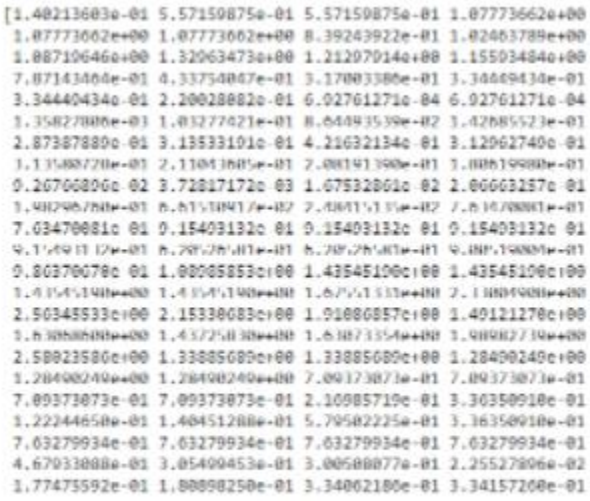
Z- Score is also called a standard score. This value/score helps to understand that how far is the data point from the mean. And after setting up a threshold value one can utilize z score values of data points to define the outliers.

Zscore = (data_point -mean) / std. deviation

Z score

```
from scipy import stats

import numpy as np

z = np.abs(stats.zscore(df_boston['DIS']))

print(z)
```

Output/Results snippet:



part of the list(z)

The above output is just a snapshot of part of the data; the actual length of the list(z) is 506 that is the number of rows. It prints the z-score values of each data item of the column

Now to define an outlier threshold value is chosen which is generally 3.0. As 99.7% of the data points lie between +/- 3 standard deviation (using Gaussian Distribution approach).

```
threshold = 3

# Position of the outlier

print(np.where(z > 3))
```

Output/Results snippet:

```
(array([351, 352, 353, 354, 355]),)
```

3. IQR (Inter Quartile Range)

IQR (Inter Quartile Range) Inter Quartile Range approach to finding the outliers is the most commonly used and most trusted approach used in the research field.

$$IQR = Quartile3 – Quartile1$$

```
# IQR
Q1 = np.percentile(df_boston['DIS'], 25,interpolation = 'midpoint')
Q3 = np.percentile(df_boston['DIS'], 75,interpolation = 'midpoint')
IQR = Q3 - Q1
```

Output/Results snippet:

```
3.0646000000000004
```

To define the outlier base value is defined above and below datasets normal range namely Upper and Lower bounds, define the upper and the lower bound (1.5*IQR value is considered) :

$$upper = Q3 + 1.5*IQR$$

$$lower = Q1 – 1.5*IQR$$

In the above formula as according to statistics, the 0.5 scale-up of IQR (new_IQR = IQR + 0.5*IQR) is taken, to consider all the data between 2.7 standard deviations in the Gaussian Distribution.

```
# Above Upper bound
upper = df_boston['DIS'] >= (Q3+1.5*IQR)
print("Upper bound:",upper)
print(np.where(upper))
```

```
# Below Lower bound
lower = df_boston['DIS'] <= (Q1-1.5*IQR)
print("Lower bound:", lower)
print(np.where(lower))
```

Output/Results snippet:

```
Upper bound: 9.76185
(array([351, 352, 353, 354, 355]),)
Lower bound: -2.496550000000001
(array([], dtype=int64),)
```

Removing the outliers

For removing the outlier, one must follow the same process of removing an entry from the dataset using its exact position in the dataset because in all the above methods of detecting the outliers end result is the list of all those data items that satisfy the outlier definition according to the method used.

dataframe.drop(row_index, inplace = True)

The above code can be used to drop a row from the dataset given the row_indexes to be dropped. Inplace =True is used to tell python to make the required change in the original dataset. row_index can be only one value or list of values or NumPy array but it must be one dimensional.

Example:

df_boston.drop(lists[0],inplace = True)

Detecting the outliers using IQR and removing them.

```
# Importing
import sklearn
from sklearn.datasets import load_boston
import pandas as pd
# Load the dataset
bos_hou = load_boston()
```

```
# Create the dataframe
column_name = bos_hou.feature_names
df_boston = pd.DataFrame(bos_hou.data)
df_boston.columns = column_name
df_boston.head()
''' Detection '''
# IQR
Q1 = np.percentile(df_boston['DIS'], 25,interpolation = 'midpoint')
Q3 = np.percentile(df_boston['DIS'], 75,interpolation = 'midpoint')
IQR = Q3 - Q1
print("Old Shape: ", df_boston.shape)
# Upper bound
upper = np.where(df_boston['DIS'] >= (Q3+1.5*IQR))
# Lower bound
lower = np.where(df_boston['DIS'] <= (Q1-1.5*IQR))
''' Removing the Outliers '''
df_boston.drop(upper[0], inplace = True)
df_boston.drop(lower[0], inplace = True)
print("New Shape: ", df_boston.shape)
```

Output/Results snippet:

```
Old Shape:  (506, 13)
New Shape:  (501, 13)
```

References:

1. https://www.geeksforgeeks.org/detect-and-remove-the-outliers-using-python/

Activity 13

Aim: Visualizing your data through matplotlib under basic charts

Learning outcome:

Duration: 5 hours

List of Hardware/Software requirements:

1. Laptop/Computer with Windows OS / Ubuntu
2. Python / Jupyter notebook

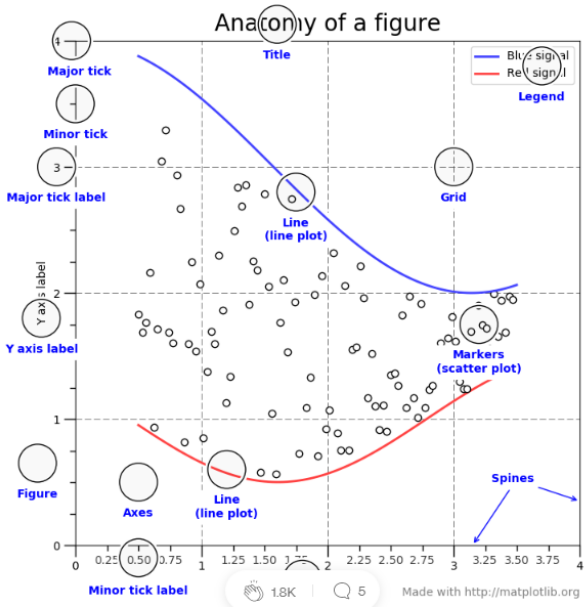
Program / Procedure:

Data Visualization is an important part of business activities as organizations nowadays collect a huge amount of data. Sensors all over the world are collecting climate data, user data through clicks, car data for prediction of steering wheels etc. All of these data collected hold key insights for businesses and visualizations make these insights easy to interpret.

Matplotlib

Matplotlib is a 2-D plotting library that helps in visualizing figures. Matplotlib emulates Matlab like graphs and visualizations. Matlab is not free, is difficult to scale and as a programming language is tedious. So, matplotlib in Python is used as it is a robust, free and easy library for data visualization.

Anatomy of Matplotlib Figure



The figure contains the overall window where plotting happens, contained within the figure are where actual graphs are plotted. Every Axes has an x-axis and y-axis for plotting. And contained within the axes are titles, ticks, labels associated with each axis. An essential figure of matplotlib is that we can more than axes in a figure which helps in building multiple plots, as shown below. In matplotlib, pyplot is used to create figures and change the characteristics of figures.

Things to follow

Plotting of Matplotlib is quite easy. Generally, while plotting they follow the same steps in each and every plot. Matplotlib has a module called pyplot which aids in plotting figure. The Jupyter notebook is used for running the plots. We import matplotlib.pyplot as plt for making it call the package module.

- Importing required libraries and dataset to plot using Pandas pd.read_csv()
- Extracting important parts for plots using conditions on Pandas Dataframes.
- plt.plot()for plotting line chart similarly in place of plot other functions are used for plotting. All plotting functions require data and it is provided in the function through parameters.
- plot.xlabel , plt.ylabel for labeling x and y-axis respectively.
- plt.xticks , plt.yticks for labeling x and y-axis observation tick points respectively.
- plt.legend() for signifying the observation variables.
- plt.title() for setting the title of the plot.
- plot.show() for displaying the plot.

Different types of Matplotlib Plots

Matplotlib supports a variety of plots including line charts, bar charts, histograms, scatter plots, etc.

Line Chart

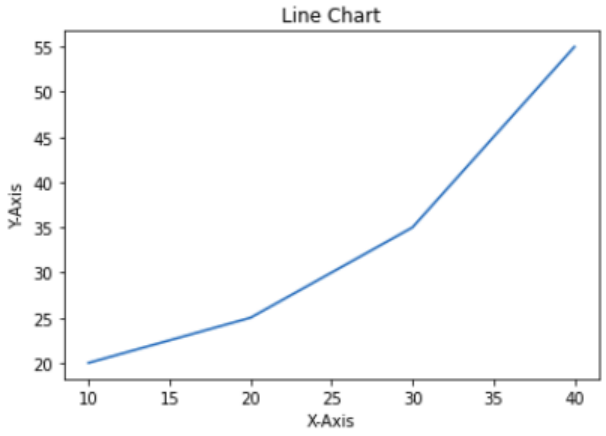
Line chart is one of the basic plots and can be created using the plot() function. It is used to represent a relationship between two data X and Y on a different axis.

Syntax:

matplotlib.pyplot.plot(*args, scalex=True, scaley=True, data=None, **kwargs)

```
import matplotlib.pyplot as plt
# initializing the data
x = [10, 20, 30, 40]
y = [20, 25, 35, 55]
# plotting the data
plt.plot(x, y)
# Adding title to the plot
plt.title("Line Chart")
# Adding label on the y-axis
plt.ylabel('Y-Axis')
# Adding label on the x-axis
plt.xlabel('X-Axis')
plt.show()
```

Output/Results snippet:



Bar Chart

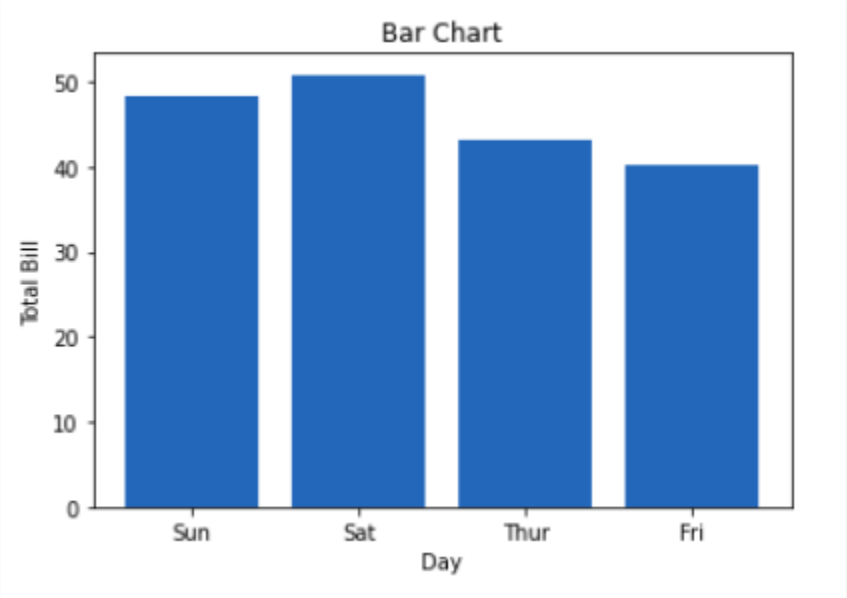
A bar chart is a graph that represents the category of data with rectangular bars with lengths and heights that is proportional to the values which they represent. The bar plots can be plotted horizontally or vertically. A bar chart describes the comparisons between the discrete categories. It can be created using the bar() method.

In the below example, we will use the tips dataset. Tips database is the record of the tip given by the customers in a restaurant for two and a half months in the early 1990s. It contains 6 columns as total_bill, tip, sex, smoker, day, time, size.

```
import matplotlib.pyplot as plt

import pandas as pd

# Reading the tips.csv file

data = pd.read_csv('tips.csv')

# initializing the data

x = data['day']

y = data['total_bill']

# plotting the data

plt.bar(x, y)

# Adding title to the plot


plt.title("Tips Dataset")
```

```
# Adding label on the y-axis
plt.ylabel('Total Bill')
# Adding label on the x-axis
plt.xlabel('Day')
plt.show()
```

Output/Results snippet:



Histogram

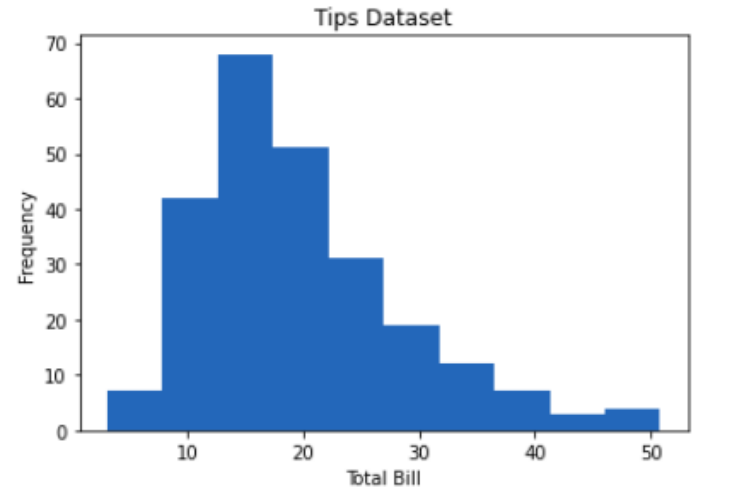
A histogram is basically used to represent data provided in a form of some groups. It is a type of bar plot where the X-axis represents the bin ranges while the Y-axis gives information about frequency. The hist() function is used to compute and create histogram of x.

Syntax:

matplotlib.pyplot.hist(x, bins=None, range=None, density=False, weights=None, cumulative=False, bottom=None, histtype='bar', align='mid', orientation='vertical', rwidth=None, log=False, color=None, label=None, stacked=False, *, data=None, **kwargs)

```
import matplotlib.pyplot as plt
import pandas as pd
# Reading the tips.csv file
data = pd.read_csv('tips.csv')
# initializing the data
x = data['total_bill']
# plotting the data
plt.hist(x)
# Adding title to the plot
plt.title("Tips Dataset")
# Adding label on the y-axis
plt.ylabel('Frequency')
# Adding label on the x-axis
plt.xlabel('Total Bill')
plt.show()
```

Output/Results snippet:



Scatter Plot

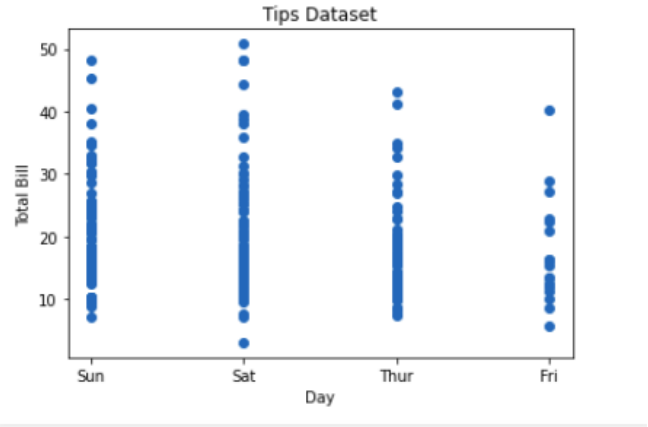
Scatter plots are used to observe relationships between variables. The scatter() method in the matplotlib library is used to draw a scatter plot.

Syntax:

matplotlib.pyplot.scatter(x_axis_data, y_axis_data, s=None, c=None, marker=None, cmap=None, vmin=None, vmax=None, alpha=None, linewidths=None, edgecolors=None

```
import matplotlib.pyplot as plt
import pandas as pd
# Reading the tips.csv file
data = pd.read_csv('tips.csv')
# initializing the data
x = data['day']
y = data['total_bill']
# plotting the data
plt.scatter(x, y)
# Adding title to the plot
plt.title("Tips Dataset")
# Adding label on the y-axis
plt.ylabel('Total Bill')
# Adding label on the x-axis
plt.xlabel('Day')
plt.show()
```

Output/Results snippet:



Pie Chart

Pie chart is a circular chart used to display only one series of data. The area of slices of the pie represents the percentage of the parts of the data. The slices of pie are called wedges. It can be created using the pie() method.

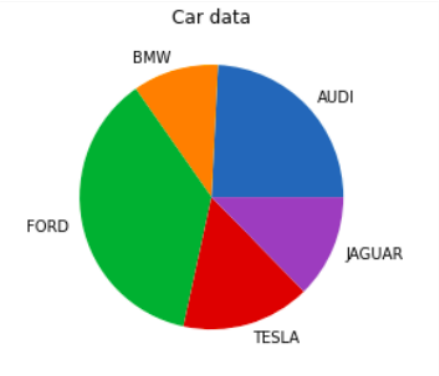
Syntax:

matplotlib.pyplot.pie(data, explode=None, labels=None, colors=None, autopct=None, shadow=False)

```
import matplotlib.pyplot as plt

import pandas as pd

# Reading the tips.csv file

data = pd.read_csv('tips.csv')

# Initializing the data

cars = ['AUDI', 'BMW', 'FORD', 'TESLA', 'JAGUAR',]

data = [23, 10, 35, 15, 12]

# plotting the data

plt.pie(data, labels=cars)

# Adding title to the plot

plt.title("Car data")
```

plt.show()

Output/Results snippet:



Saving a Plot

For saving a plot in a file on storage disk, savefig() method is used. A file can be saved in many formats like .png, .jpg, .pdf, etc.

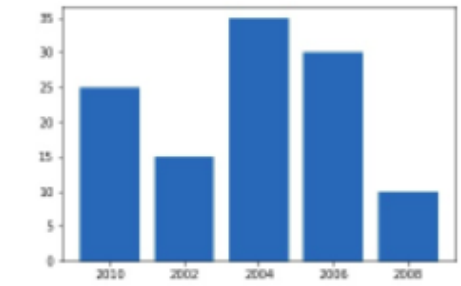
Syntax:

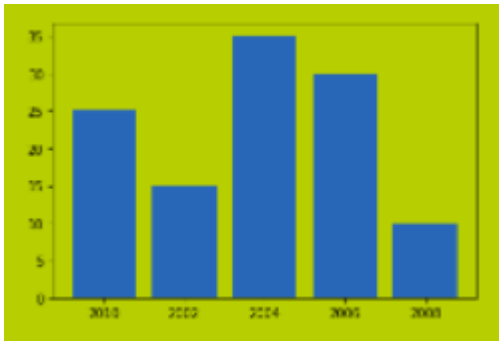
pyplot.savefig(fname, dpi=None, facecolor='w', edgecolor='w', orientation='portrait', papertype=None, format=None, transparent=False, bbox_inches=None, pad_inches=0.1, frameon=None, metadata=None)

```
import matplotlib.pyplot as plt

# Creating data

year = ['2010', '2002', '2004', '2006', '2008']

production = [25, 15, 35, 30, 10]

# Plotting barchart

plt.bar(year, production)

# Saving the figure.

plt.savefig("output.jpg")

# Saving figure by changing parameter values

plt.savefig("output1", facecolor='y', bbox_inches="tight",
```

pad_inches=0.3, transparent=True)

Output/Results snippet:





References:

1. https://towardsdatascience.com/data-visualization-using-matplotlib-16f1aae5ce70
2. https://www.geeksforgeeks.org/data-visualization-using-matplotlib/

Activity 14

Aim: Labels, legends and axes

Learning outcome:

Duration: 3 hours

List of Hardware/Software requirements:

1. Laptop/Computer with Windows OS / Ubuntu
2. Python / Jupyter notebook

Program / Procedure:

Adding Title

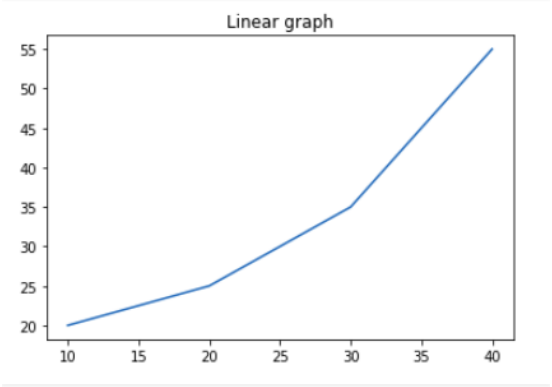
The title() method in matplotlib module is used to specify the title of the visualization depicted and displays the title using various attributes.

Syntax:

matplotlib.pyplot.title(label, fontdict=None, loc='center', pad=None, **kwargs)

```
import matplotlib.pyplot as plt
# initializing the data
x = [10, 20, 30, 40]
y = [20, 25, 35, 55]
# plotting the data
plt.plot(x, y)
# Adding title to the plot
plt.title("Linear graph")
plt.show()
```

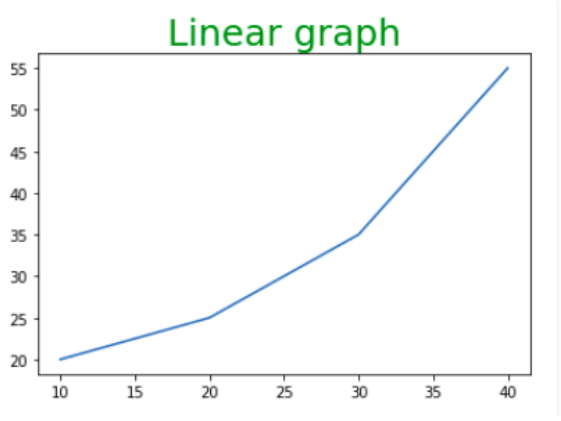
Output/Results snippet:



We can also change the appearance of the title by using the parameters of this function.

```
import matplotlib.pyplot as plt
# Initializing the data
x = [10, 20, 30, 40]
y = [20, 25, 35, 55]
# plotting the data
plt.plot(x, y)
# Adding title to the plot
plt.title("Linear graph", fontsize=25, color="green")
plt.show()
```

Output/Results snippet:



Adding X Label and Y Label

In layman's terms, the X label and the Y label are the titles given to X-axis and Y-axis respectively. These can be added to the graph by using the xlabel() and ylabel() methods.

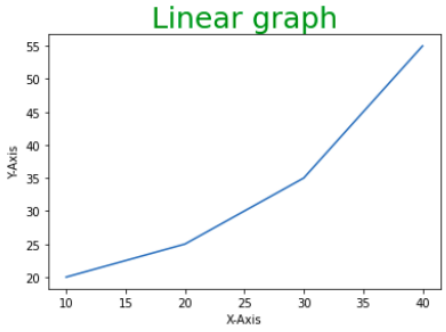
Syntax:

matplotlib.pyplot.xlabel(xlabel, fontdict=None, labelpad=None, **kwargs)

matplotlib.pyplot.ylabel(ylabel, fontdict=None, labelpad=None, **kwargs)

```
import matplotlib.pyplot as plt
# initializing the data
x = [10, 20, 30, 40]
y = [20, 25, 35, 55]
# plotting the data
plt.plot(x, y)
# Adding title to the plot
plt.title("Linear graph", fontsize=25, color="green")
# Adding label on the y-axis
plt.ylabel('Y-Axis')
# Adding label on the x-axis
plt.xlabel('X-Axis')
plt.show()
```

Output/Results snippet:



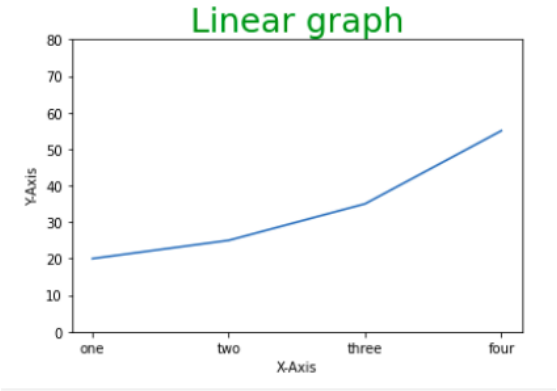
Setting Limits and Tick labels

You might have seen that Matplotlib automatically sets the values and the markers(points) of the X and Y axis, however, it is possible to set the limit and markers manually. xlim() and ylim() functions are used to set the limits of the X-axis and Y-axis respectively. Similarly, xticks() and yticks() functions are used to set tick labels.

```
import matplotlib.pyplot as plt
```

```
# initializing the data
x = [10, 20, 30, 40]
y = [20, 25, 35, 55]
# plotting the data
plt.plot(x, y)
# Adding title to the plot
plt.title("Linear graph", fontsize=25, color="green")
# Adding label on the y-axis
plt.ylabel('Y-Axis')
# Adding label on the x-axis
plt.xlabel('X-Axis')
# Setting the limit of y-axis
plt.ylim(0, 80)
# setting the labels of x-axis
plt.xticks(x, labels=["one", "two", "three", "four"])
plt.show()
```

Output/Results snippet:



Adding Legends

A legend is an area describing the elements of the graph. In simple terms, it reflects the data displayed in the graph's Y-axis. It generally appears as the box containing a small sample of each color on the graph and a small description of what this data means.

The attribute bbox_to_anchor=(x, y) of legend() function is used to specify the coordinates of the legend, and the attribute ncol represents the number of columns that the legend has. Its default value is 1.

Syntax:

matplotlib.pyplot.legend(["name1", "name2"], bbox_to_anchor=(x, y), ncol=1)

```
import matplotlib.pyplot as plt
# initializing the data
x = [10, 20, 30, 40]
y = [20, 25, 35, 55]
# plotting the data
plt.plot(x, y)
# Adding title to the plot
plt.title("Linear graph", fontsize=25, color="green")
# Adding label on the y-axis

plt.ylabel('Y-Axis')
```

```
# Adding label on the x-axis
plt.xlabel('X-Axis')
# Setting the limit of y-axis
plt.ylim(0, 80)
# setting the labels of x-axis
plt.xticks(x, labels=["one", "two", "three", "four"])
# Adding legends
plt.legend(["GFG"])
plt.show()
```

Output/Results snippet:

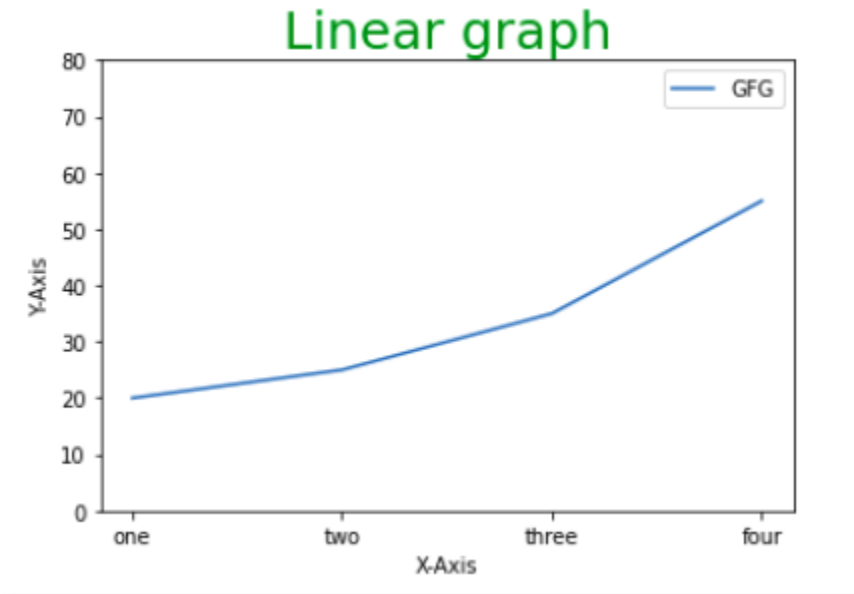


Figure class

Consider the figure class as the overall window or page on which everything is drawn. It is a top-level container that contains one or more axes. A figure can be created using the figure() method.

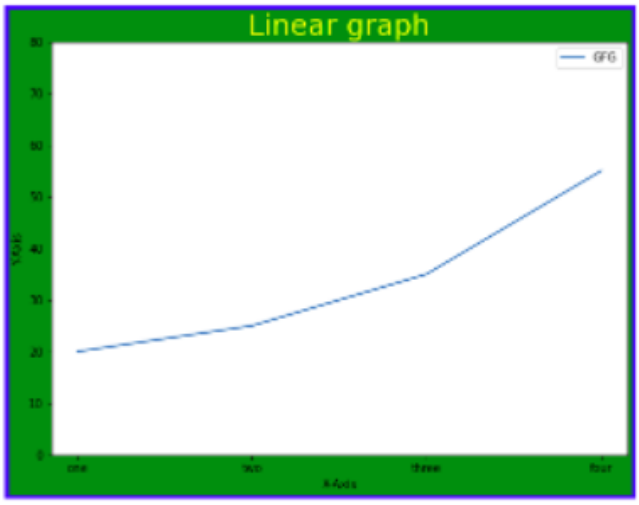
Syntax:

class matplotlib.figure.Figure(figsize=None, dpi=None, facecolor=None, edgecolor=None, linewidth=0.0, frameon=None, subplotpars=None, tight_layout=None, constrained_layout=None)

```
# Python program to show pyplot module

import matplotlib.pyplot as plt

from matplotlib.figure import Figure

# initializing the data

x = [10, 20, 30, 40]

y = [20, 25, 35, 55]

# Creating a new figure with width = 7 inches and height = 5 inches with face color as

# green, edgecolor as red and the line width of the edge as 7

fig = plt.figure(figsize =(7, 5), facecolor='g', edgecolor='b', linewidth=7)

# Creating a new axes for the figure

ax = fig.add_axes([1, 1, 1, 1])

# Adding the data to be plotted

ax.plot(x, y)

# Adding title to the plot

plt.title("Linear graph", fontsize=25, color="yellow")

# Adding label on the y-axis

plt.ylabel('Y-Axis')

# Adding label on the x-axis

plt.xlabel('X-Axis')

# Setting the limit of y-axis

plt.ylim(0, 80)

# setting the labels of x-axis


plt.xticks(x, labels=["one", "two", "three", "four"])
```

```
# Adding legends
plt.legend(["GFG"])
plt.show()
```

Output/Results snippet:



Axes Class

Axes class is the most basic and flexible unit for creating sub-plots. A given figure may contain many axes, but a given axes can only be present in one figure. The axes() function creates the axes object.

Syntax:

axes([left, bottom, width, height])

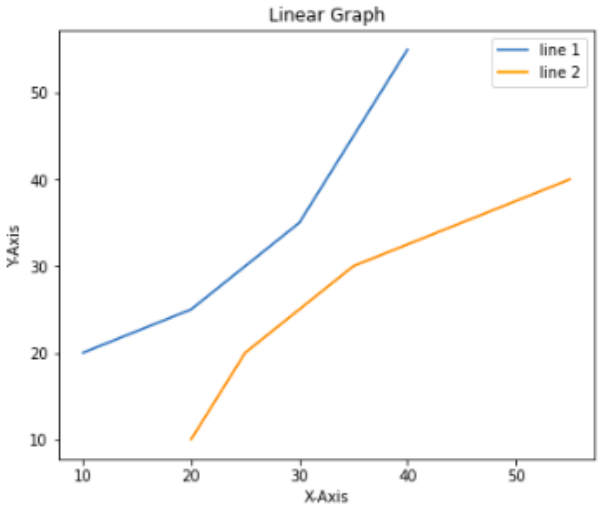
Just like pyplot class, axes class also provides methods for adding titles, legends, limits, labels, etc.

- Adding Title – ax.set_title()
- Adding X Label and Y label – ax.set_xlabel(), ax.set_ylabel()
- Setting Limits – ax.set_xlim(), ax.set_ylim()

- Tick labels – ax.set_xticklabels(), ax.set_yticklabels()
- Adding Legends – ax.legend()

```
# Python program to show pyplot module
import matplotlib.pyplot as plt
from matplotlib.figure import Figure
# initializing the data
x = [10, 20, 30, 40]
y = [20, 25, 35, 55]
fig = plt.figure(figsize = (5, 4))
# Adding the axes to the figure
ax = fig.add_axes([1, 1, 1, 1])
# plotting 1st dataset to the figure
ax1 = ax.plot(x, y)
# plotting 2nd dataset to the figure
ax2 = ax.plot(y, x)
# Setting Title
ax.set_title("Linear Graph")
# Setting Label
ax.set_xlabel("X-Axis")
ax.set_ylabel("Y-Axis")
# Adding Legend
ax.legend(labels = ('line 1', 'line 2'))
plt.show()
```

Output/Results snippet:



References:

1. https://www.geeksforgeeks.org/data-visualization-using-matplotlib/

Activity 15

Aim: Subplotting, grid, and 3D plots

Learning outcome:

Duration: 3 hours

List of Hardware/Software requirements:

1. Laptop/Computer with Windows OS / Ubuntu
2. Python / Jupyter notebook

Program/Procedure:

Multiple Plots

We have learned about the basic components of a graph that can be added so that it can convey more information. One method can be by calling the plot function again and again with a different set of values as shown in the above example. Now let's see how to plot multiple graphs using some functions and also how to plot subplots.

Method 1: Using the add_axes() method

The add_axes() method is used to add axes to the figure. This is a method of figure class

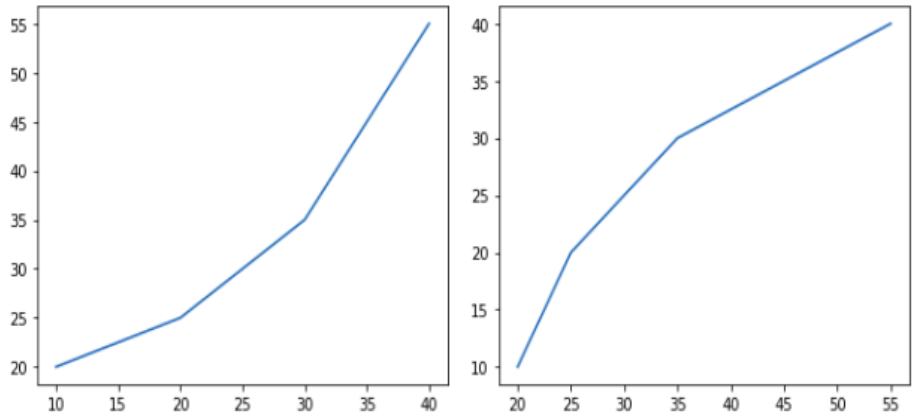
Syntax:

add_axes(self, *args, **kwargs)

```
# Python program to show pyplot module
import matplotlib.pyplot as plt
from matplotlib.figure import Figure
# initializing the data
x = [10, 20, 30, 40]
y = [20, 25, 35, 55]
# Creating a new figure with width = 5 inches and height = 4 inches
```

```
fig = plt.figure(figsize =(5, 4))
# Creating first axes for the figure
ax1 = fig.add_axes([0.1, 0.1, 0.8, 0.8])
# Creating second axes for the figure
ax2 = fig.add_axes([1, 0.1, 0.8, 0.8])
# Adding the data to be plotted
ax1.plot(x, y)
ax2.plot(y, x)
plt.show()
```

Output/Results snippet:



Method 2: Using subplot() method.

This method adds another plot at the specified grid position in the current figure.

Syntax:

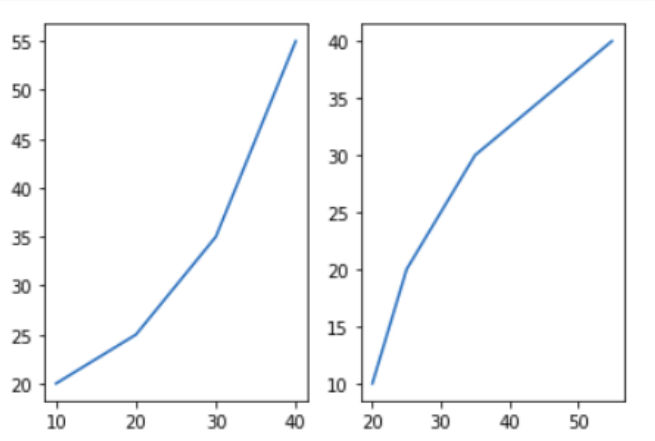
subplot(nrows, ncols, index, **kwargs)

subplot(pos, **kwargs)

subplot(ax)

```
import matplotlib.pyplot as plt
# initializing the data
x = [10, 20, 30, 40]
y = [20, 25, 35, 55]
# Creating figure object
plt.figure()
# addind first subplot
plt.subplot(121)
plt.plot(x, y)
# addding second subplot
plt.subplot(122)
plt.plot(y, x)
```

Output/Results snippet:



Method 3: Using subplots() method

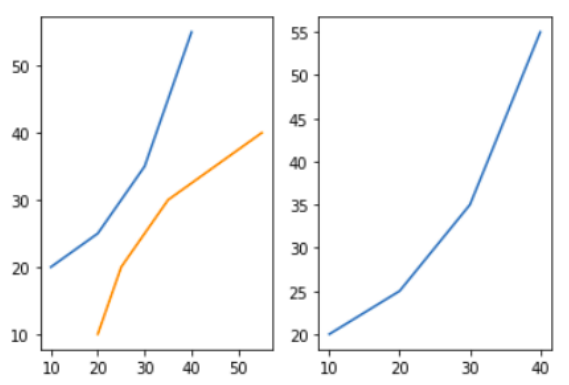
This function is used to create figures and multiple subplots at the same time.

Syntax:

matplotlib.pyplot.subplots(nrows=1, ncols=1, sharex=False, sharey=False, squeeze=True, subplot_kw=None, gridspec_kw=None, **fig_kw)

```
import matplotlib.pyplot as plt
# initializing the data
x = [10, 20, 30, 40]
y = [20, 25, 35, 55]
# Creating the figure and subplots according the argument passed
fig, axes = plt.subplots(1, 2)
# plotting the data in the 1st subplot
axes[0].plot(x, y)
# plotting the data in the 1st subplot only
axes[0].plot(y, x)
# plotting the data in the 2nd subplot only
axes[1].plot(x, y)
```

Output/Results snippet:



Method 4: Using subplot2grid() method

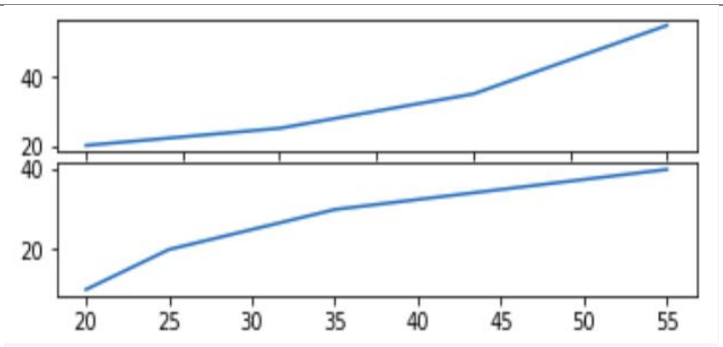
This function creates axes object at a specified location inside a grid and also helps in spanning the axes object across multiple rows or columns. In simpler words, this function is used to create multiple charts within the same figure.

Syntax:

Plt.subplot2grid(shape, location, rowspan, colspan)

```
import matplotlib.pyplot as plt
# initializing the data
x = [10, 20, 30, 40]
y = [20, 25, 35, 55]
# adding the subplots
axes1 = plt.subplot2grid (
(7, 1), (0, 0), rowspan = 2, colspan = 1)
axes2 = plt.subplot2grid (
(7, 1), (2, 0), rowspan = 2, colspan = 1)
# plotting the data
axes1.plot(x, y)
axes2.plot(y, x)
```

Output/Results snippet:



Grids in Matplotlib

Grids are made up of intersecting straight (vertical, horizontal and angular) or curved lines used to structure our content. Matplotlib helps us to draw plain graphs but it sometimes necessary to use grids for better understanding and get a reference for our data points. Thus, Matplotlib provides a grid() for easy creation of gridlines with tonnes of customization.

matplotlib.pyplot.grid()

Syntax:

matplotlib.pyplot.grid(b=None, which='major', axis='both', **kwargs)

Program:

Parameters: -

b: bool value to specify whether to show grid-lines. Default is True

which: The grid lines to apply changes. Values: {'major', 'minor', 'both'}

axis: The axis to apply changes on. Values: {'both', 'x', 'y'}

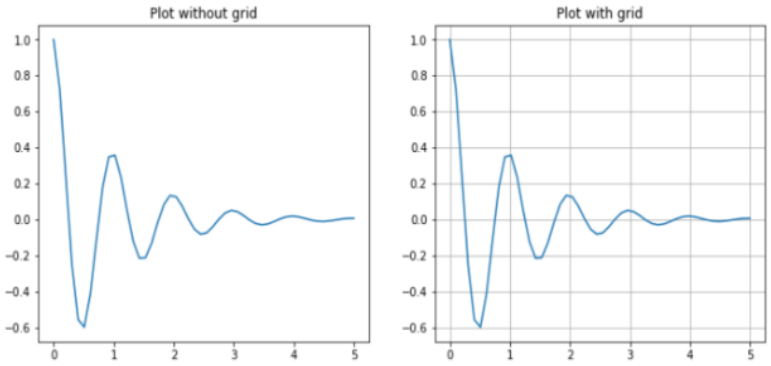
****kwargs:** Optional line properties

Returns: This function doesn't return anything.

The grid() sets the visibility of grids by specifying a boolean value (True/False). We can also choose to display minor or major ticks or both. Also, color, linewidth and linestyle can be changed as additional parameters.

```
# Implementation of matplotlib function

import matplotlib.pyplot as plt

import numpy as np

# dummy data

x1 = np.linspace(0.0, 5.0)

y1 = np.cos(2 * np.pi * x1) * np.exp(-x1)

# creates two subplots


fig, (ax1, ax2) = plt.subplots(1, 2, figsize = (12, 5))

# Plot without grid

ax1.plot(x1, y1)

ax1.set_title('Plot without grid')

# plot with grid

ax2.plot(x1, y1)

ax2.set_title("Plot with grid")

# draw gridlines

ax2.grid(True)

plt.show()
```

Output/Results snippet:

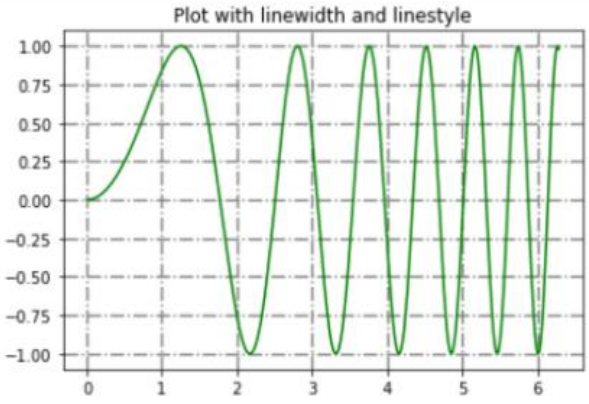


Now let’s draw gridlines using extra line properties such as color, linestyle and linewidth.

Program:

```
# Implementation of matplotlib function
import matplotlib.pyplot as plt
import numpy as np
# dummy data
x = np.linspace(0, 2 * np.pi, 400)

y = np.sin(x ** 2)
# set graph color
plt.plot(x, y, 'green')
# to set title
plt.title("Plot with linewidth and linestyle")
# draws gridlines of grey color using given linewidth and linestyle
plt.grid(True, color = "grey", linewidth = "1.4", linestyle = "-.")
plt.show()
```

Output/Results snippet:

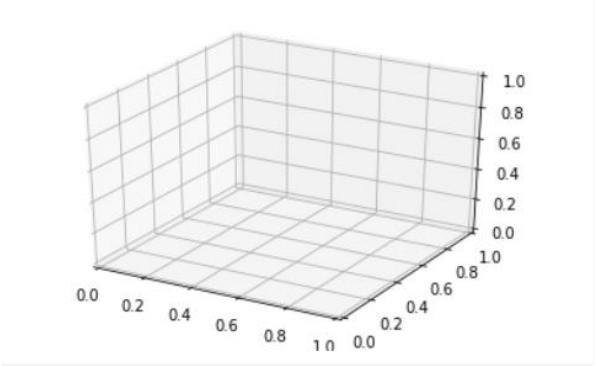


3D – Plotting

Program:

```
import numpy as np
import matplotlib.pyplot as plt
fig = plt.figure()
ax = plt.axes(projection ='3d')
```

Output/Results snippet:



Plotting 3-D Lines and Points

Graph with lines and point are the simplest 3-dimensional graph. ax.plot3d and ax.scatter are the function to plot line and point graph respectively.

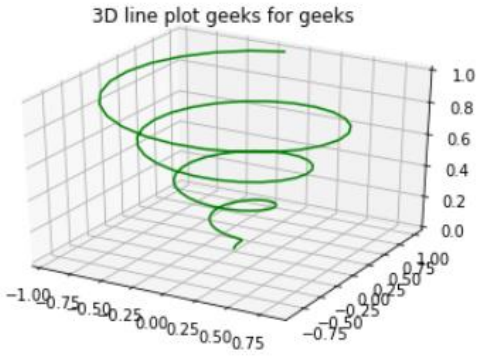
3-dimensional line graph

Program:

```
# importing mplot3d toolkits, numpy and matplotlib
from mpl_toolkits import mplot3d
import numpy as np
import matplotlib.pyplot as plt
fig = plt.figure()
# syntax for 3-D projection
ax = plt.axes(projection ='3d')
# defining all 3 axes
z = np.linspace(0, 1, 100)
x = z * np.sin(25 * z)

y = z * np.cos(25 * z)
# plotting
ax.plot3D(x, y, z, 'green')
ax.set_title('3D line plot geeks for geeks')
plt.show()
```

Output/Results snippet:

```
Text(0.5, 0.92, '3D line plot geeks for geeks')
```



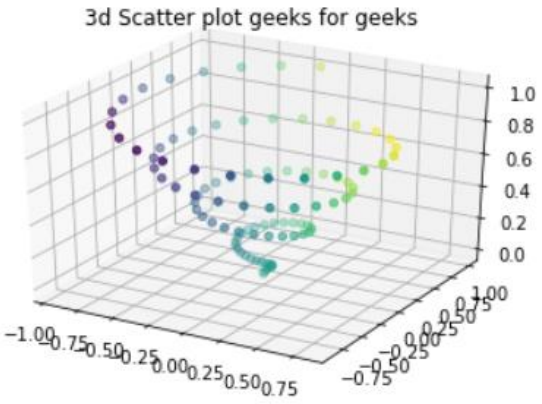
3-dimensional scattered graph

Program:

```
# importing mplot3d toolkits
from mpl_toolkits import mplot3d
import numpy as np
import matplotlib.pyplot as plt
fig = plt.figure()
# syntax for 3-D projection
ax = plt.axes(projection ='3d')
# defining axes
z = np.linspace(0, 1, 100)

x = z * np.sin(25 * z)
y = z * np.cos(25 * z)
c = x + y
ax.scatter(x, y, z, c = c)
# syntax for plotting
ax.set_title('3d Scatter plot geeks for geeks')
```

plt.show()

Output/Results snippet:



References:

1. https://www.geeksforgeeks.org/data-visualization-using-matplotlib/
2. https://www.geeksforgeeks.org/grids-in-matplotlib/
3. https://www.geeksforgeeks.org/three-dimensional-plotting-in-python-using-matplotlib/

Activity 16

Aim: Plot formatting- custom attribute values

Learning outcome:

Duration: 4 hours

List of Hardware/Software requirements:

3. Laptop/Computer with Windows OS / Ubuntu
4. Python / Jupyter notebook

Program/Procedure:

Line Chart

We may use the following properties –

- color: Changing the color of the line
- linewidth: Cutomizing the width of the line
- marker: For changing the style of actual plotted point
- markersize: For changing the size of the markers
- linestyle: For defining the style of the plotted line

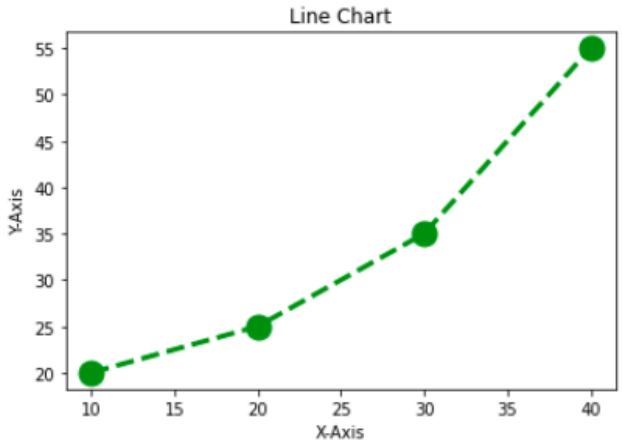
Different Linestyle available

Character	Definition
–	Solid line
—	Dashed line
-.	dash-dot line
:	Dotted line
.	Point marker
o	Circle marker
,	Pixel marker
v	triangle_down marker
^	triangle_up marker

<	triangle_left marker
>	triangle_right marker
1	tri_down marker
2	tri_up marker
3	tri_left marker
4	tri_right marker
s	square marker
p	pentagon marker
*****	star marker
h	hexagon1 marker
H	hexagon2 marker
+	Plus marker
x	X marker
D	Diamond marker
d	thin_diamond marker
|	vline marker
_	hline marker

```
import matplotlib.pyplot as plt

# initializing the data

x = [10, 20, 30, 40]

y = [20, 25, 35, 55]

# plotting the data

plt.plot(x, y, color='green', linewidth=3, marker='o', markersize=15, linestyle='--')

# Adding title to the plot

plt.title("Line Chart")

# Adding label on the y-axis

plt.ylabel('Y-Axis')

# Adding label on the x-axis

plt.xlabel('X-Axis')

plt.show()
```

Output/Results snippet:



Bar Chart

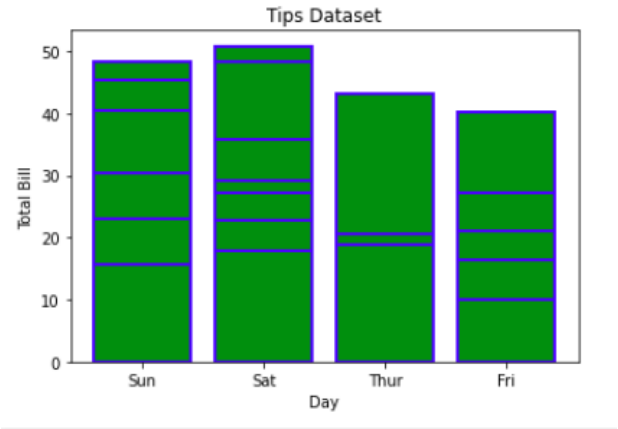
Customization that is available for the Bar Chart –

- color: For the bar faces
- edgecolor: Color of edges of the bar
- linewidth: Width of the bar edges
- width: Width of the bar

```
import matplotlib.pyplot as plt
import pandas as pd
# Reading the tips.csv file
data = pd.read_csv('tips.csv')
# initializing the data
x = data['day']
y = data['total_bill']
# plotting the data
plt.bar(x, y, color='green', edgecolor='blue', linewidth=2)
```

```
# Adding title to the plot
plt.title("Tips Dataset")
# Adding label on the y-axis
plt.ylabel('Total Bill')
# Adding label on the x-axis
plt.xlabel('Day')
plt.show()
```

Output/Results snippet:



Note: The lines in between the bars refer to the different values in the Y-axis of the particular value of the X-axis.

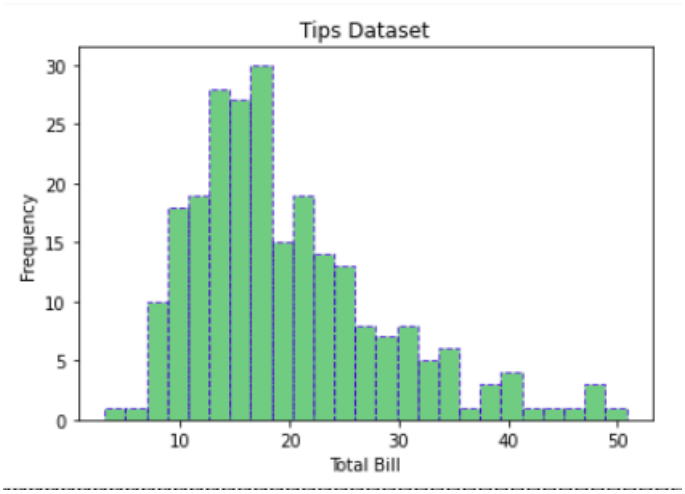
Histogram

Customization that is available for the Histogram –

- bins: Number of equal-width bins
- color: For changing the face color
- edgecolor: Color of the edges
- linestyle: For the edgelines
- alpha: blending value, between 0 (transparent) and 1 (opaque)

```
import matplotlib.pyplot as plt
import pandas as pd
# Reading the tips.csv file
data = pd.read_csv('tips.csv')
# initializing the data
x = data['total_bill']
# plotting the data
plt.hist(x, bins=25, color='green', edgecolor='blue', linestyle='--', alpha=0.5)
# Adding title to the plot
plt.title("Tips Dataset")
# Adding label on the y-axis
plt.ylabel('Frequency')
# Adding label on the x-axis
plt.xlabel('Total Bill')
plt.show()
```

Output/Results snippet:



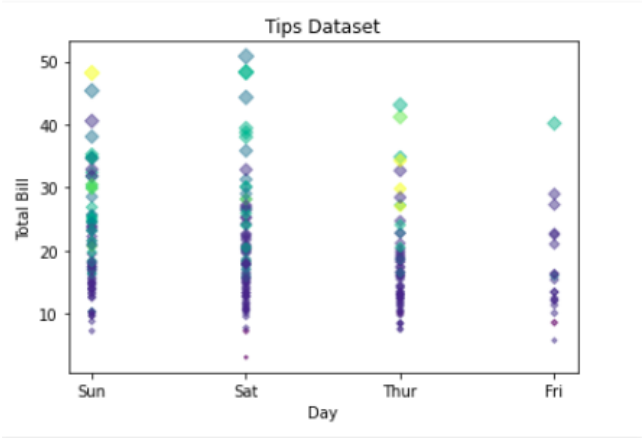
Scatter Plot

Customizations that are available for the scatter plot are –

- s: marker size (can be scalar or array of size equal to size of x or y)
- c: color of sequence of colors for markers
- marker: marker style
- linewidths: width of marker border
- edgecolor: marker border color
- alpha: blending value, between 0 (transparent) and 1 (opaque)

```
import matplotlib.pyplot as plt
import pandas as pd
# Reading the tips.csv file
data = pd.read_csv('tips.csv')
# initializing the data
x = data['day']
y = data['total_bill']
# plotting the data
plt.scatter(x, y, c=data['size'], s=data['total_bill'], marker='D', alpha=0.5)
# Adding title to the plot
plt.title("Tips Dataset")
# Adding label on the y-axis
plt.ylabel('Total Bill')
# Adding label on the x-axis
plt.xlabel('Day')
plt.show()
```

Output/Results snippet:



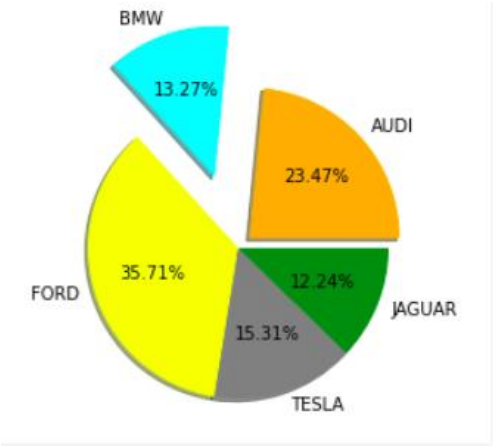
Pie Chart

Customizations that are available for the Pie chart are –

- explode: Moving the wedges of the plot
- autopct: Label the wedge with their numerical value.
- color: Attribute is used to provide color to the wedges.
- shadow: Used to create shadow of wedge.

```
import matplotlib.pyplot as plt
import pandas as pd
# Reading the tips.csv file
data = pd.read_csv('tips.csv')
# initializing the data
cars = ['AUDI', 'BMW', 'FORD', 'TESLA', 'JAGUAR',]
data = [23, 13, 35, 15, 12]
explode = [0.1, 0.5, 0, 0, 0]
colors = ("orange", "cyan", "yellow", "grey", "green",)
# plotting the data
plt.pie(data, labels=cars, explode=explode, autopct='%1.2f%%', colors=colors, shadow=True)
plt.show()
```

Output/Results snippet:



References:

1. https://www.geeksforgeeks.org/data-visualization-using-matplotlib/

Activity 17

Aim: Advanced charts in seaborn- countplot(), jointplot(), boxplot(), heatmap(), regression plot, etc

Learning outcome:

Duration: 6 hours

List of Hardware/Software requirements:

1. Laptop/Computer with Windows OS / Ubuntu
2. Python / Jupyter notebook

Program/Procedure:

Seaborn is a library mostly used for statistical plotting in Python. It is built on top of Matplotlib and provides beautiful default styles and color palettes to make statistical plots more attractive.

Seaborn can be installed using the pip. Type the below command in the terminal.

pip install seaborn

Plotting categorical scatter plots with Seaborn

Stripplot

 # Python program to illustrate

 # Plotting categorical scatter

 # plots with Seaborn

 # importing the required module

import matplotlib.pyplot as plt

import seaborn as sns

 # x axis values

x =['sun', 'mon', 'fri', 'sat', 'tue', 'wed', 'thu']

 # y axis values

y =[5, 6.7, 4, 6, 2, 4.9, 1.8]

 # plotting strip plot with seaborn

```
ax = sns.stripplot(x, y);
```

 # giving labels to x-axis and y-axis

```
ax.set(xlabel ='Days', ylabel ='Amount_spend')
```

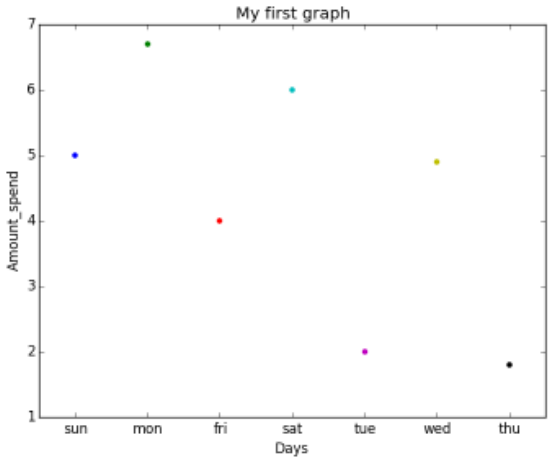
 # giving title to the plot

```
plt.title('My first graph');
```

 # function to show plot

```
plt.show()
```

Output/Results snippet:



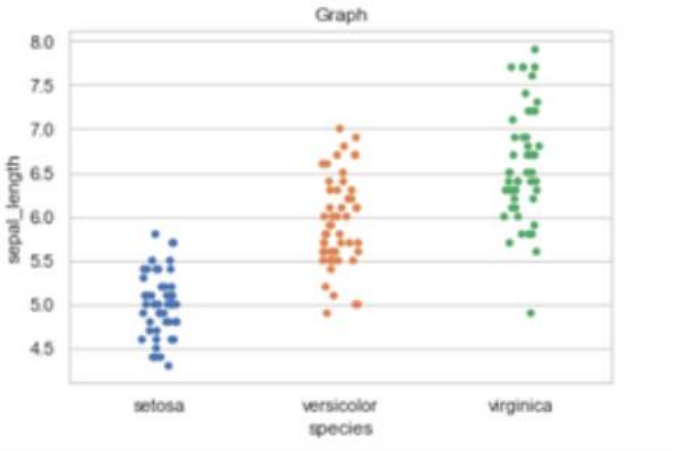
This is the one kind of scatter plot of categorical data with the help of seaborn.

- Categorical data is represented on the x-axis and values correspond to them represented through the y-axis.
- .striplot() function is used to define the type of the plot and to plot them on canvas using.
- .set() function is used to set labels of x-axis and y-axis.
- .title() function is used to give a title to the graph.
- To view plot we use .show() function.

Stripplot using inbuilt data-set given in seaborn:

```
# Python program to illustrate Stripplot using inbuilt data-set given in seaborn
# importing the required module
import matplotlib.pyplot as plt
import seaborn as sns
# use to set style of background of plot
sns.set(style="whitegrid")
# loading data-set
iris = sns.load_dataset('iris')
# plotting strip plot with seaborn deciding the attributes of dataset on which plot should be made
ax = sns.stripplot(x='species', y='sepal_length', data=iris)
# giving title to the plot
plt.title('Graph')
# function to show plot
plt.show()
```

Output/Results snippet:



Explanation:

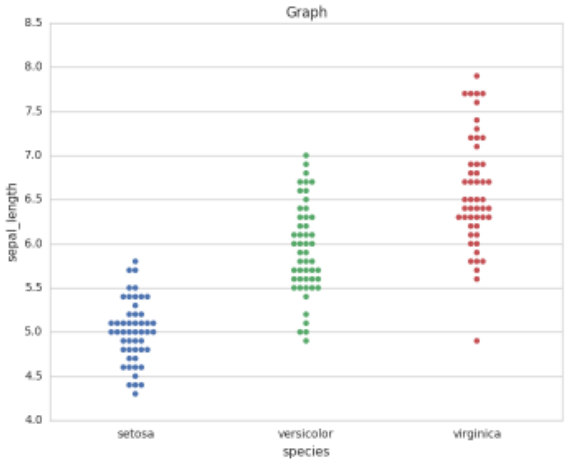
- iris is the dataset already present in seaborn module for use.
- We use .load_dataset() function in order to load the data.We can also load any other file by giving the path and name of the file in the argument.

- .set(style=”whitegrid”) function here is also use to define the background of plot. We can use “darkgrid”
- instead of whitegrid if we want the dark-colored background.
- In .stripplot() function we have to define which attribute of the dataset to be on the x-axis and which attribute of the dataset should on y-axis.data = iris means attributes which we define earlier should be taken from the given data.
- We can also draw this plot with matplotlib but the problem with matplotlib is its default parameters. The reason why Seaborn is so great with DataFrames is, for example, labels from DataFrames are automatically propagated to plots or other data structures as you see in the above figure column name species comes on the x-axis and column name stepal_length comes on the y-axis, that is not possible with matplotlib. We have to explicitly define the labels of the x-axis and y-axis.

Swarmplot

```
# Python program to illustrate plotting using Swarmplot

# importing the required module

import matplotlib.pyplot as plt

import seaborn as sns

# use to set style of background of plot

sns.set(style="whitegrid")

# loading data-set

iris = sns.load_dataset('iris')

# plotting strip plot with seaborn deciding the attributes of dataset on which plot should be made

ax = sns.swarmplot(x='species', y='sepal_length', data=iris)

# giving title to the plot

plt.title('Graph')

# function to show plot

plt.show()
```

Output/Results snippet:



Explanation:

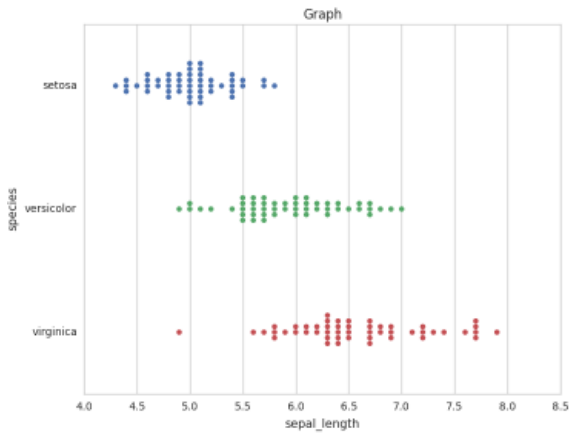
This is very much similar to stripplot but the only difference is that it does not allow overlapping of markers. It causes jittering in the markers of the plot so that graph can easily be read without information loss as seen in the above plot.

- We use .swarmplot() function to plot swarn plot.
- Another difference that we can notice in Seaborn and Matplotlib is that working with DataFrames doesn't go quite as smoothly with Matplotlib, which can be annoying if we doing exploratory analysis with Pandas. And that's exactly what Seaborn does easily, the plotting functions operate on DataFrames and arrays that contain a whole dataset.

```
# importing the required module

import matplotlib.pyplot as plt

import seaborn as sns

# use to set style of background of plot

sns.set(style="whitegrid")

# loading data-set

iris = sns.load_dataset('iris')

# plotting strip plot with seaborn deciding the attributes of dataset on which plot should be made

ax = sns.swarmplot(x='sepal_length', y='species', data=iris)
```

```
# giving title to the plot
plt.title('Graph')
# function to show plot
plt.show()
```

Output/Results snippet:



Barplot

A barplot is basically used to aggregate the categorical data according to some methods and by default it's the mean. It can also be understood as a visualization of the group by action. To use this plot we choose a categorical column for the x-axis and a numerical column for the y-axis, and we see that it creates a plot taking a mean per categorical column.

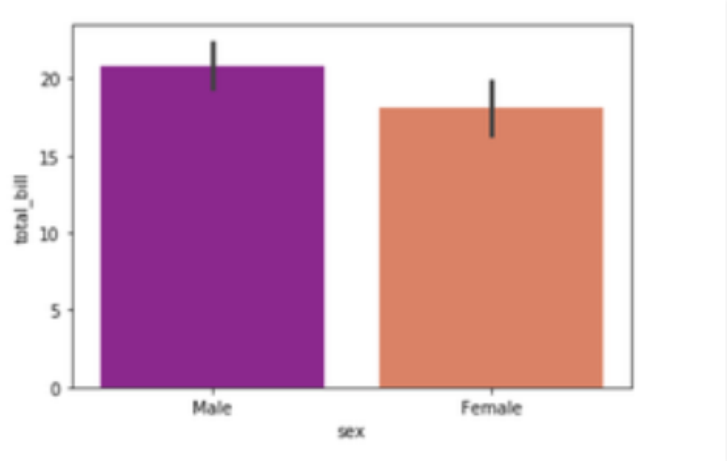
Syntax:

barplot([x, y, hue, data, order, hue_order, …])

```
# import the seaborn library
import seaborn as sns
# reading the dataset
df = sns.load_dataset('tips')
# change the estimator from mean to standard deviation
```

sns.barplot(x ='sex', y ='total_bill', data = df, palette ='plasma')

Output/Results snippet:



Explanation:

Looking at the plot we can say that the average total_bill for the male is more than compared to the female.

- Palette is used to set the color of the plot
- The estimator is used as a statistical function for estimation within each categorical bin.

Countplot

A countplot basically counts the categories and returns a count of their occurrences. It is one of the simplest plots provided by the seaborn library.

Syntax:

countplot([x, y, hue, data, order, …])

import the seaborn library

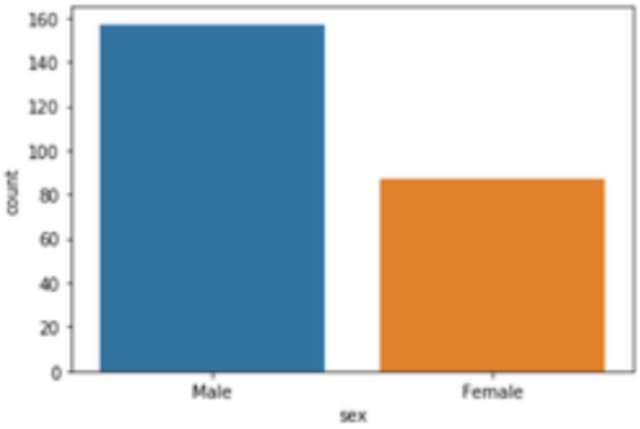
import seaborn as sns

reading the dataset

df = sns.load_dataset('tips')

sns.countplot(x ='sex', data = df)

Output/Results snippet:



Explanation:

Looking at the plot we can say that the number of males is more than the number of females in the dataset. As it only returns the count based on a categorical column, we need to specify only the x parameter.

Boxplot

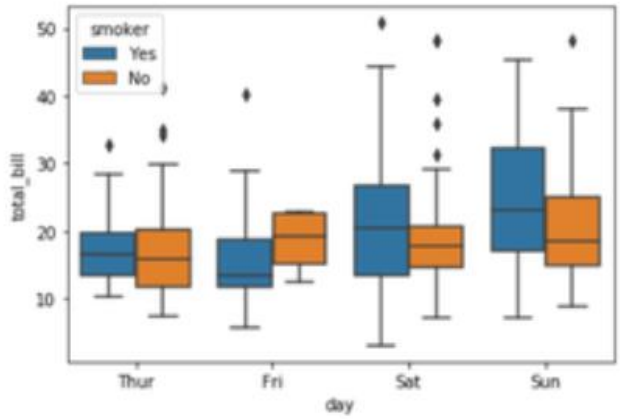
Box Plot is the visual representation of the depicting groups of numerical data through their quartiles. Boxplot is also used to detect the outlier in the data set.

Syntax:

boxplot([x, y, hue, data, order, hue_order, …])

```
# import the seaborn library

import seaborn as sns

# reading the dataset

df = sns.load_dataset('tips')

sns.boxplot(x='day', y='total_bill', data=df, hue='smoker')
```

Output/Results snippet:



Explanation:

x takes the categorical column and y is a numerical column. Hence, we can see the total bill spent each day." hue" parameter is used to further add a categorical separation. By looking at the plot we can say that the people who do not smoke had a higher bill on Friday as compared to the people who smoked.

Violinplot

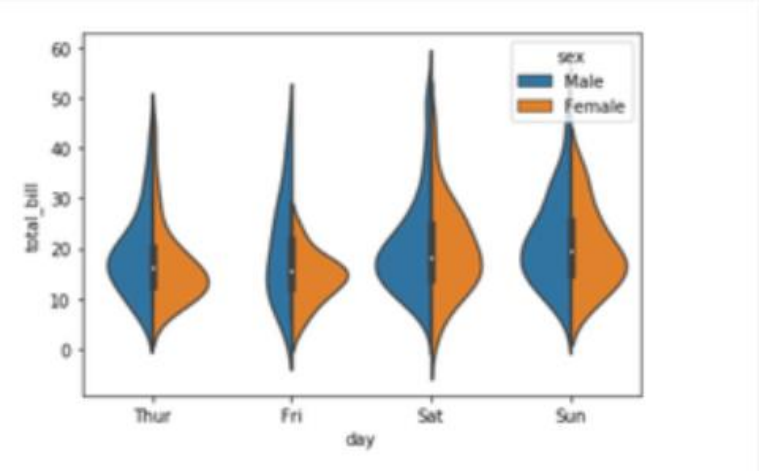
It is similar to the boxplot except that it provides a higher, more advanced visualization and uses the kernel density estimation to give a better description about the data distribution.

Syntax:

violinplot([x, y, hue, data, order, …])

```
# import the seaborn library
import seaborn as sns
# reading the dataset
df = sns.load_dataset('tips')
sns.violinplot(x='day', y='total_bill', data=df, hue='sex', split=True)
```

Output/Results snippet:



Explanation:

- hue is used to separate the data further using the sex category
- setting split=True will draw half of a violin for each level. This can make it easier to directly compare the distributions.

Stripplot

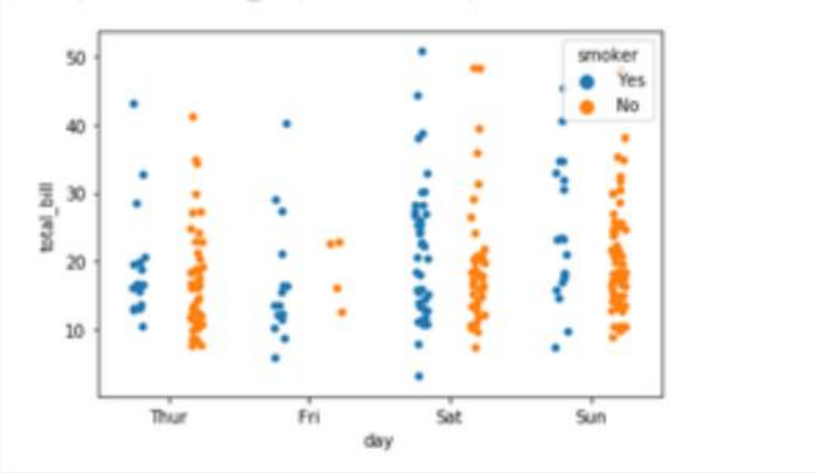
It basically creates a scatter plot based on the category.

Syntax:

stripplot([x, y, hue, data, order, …])

```
# import the seaborn library
import seaborn as sns
# reading the dataset
df = sns.load_dataset('tips')
sns.stripplot(x='day', y='total_bill', data=df, jitter=True, hue='smoker', dodge=True)
```

Output/Results snippet:



Explanation:

- One problem with strip plot is that you can't really tell which points are stacked on top of each other and hence we use the jitter parameter to add some random noise.
- jitter parameter is used to add an amount of jitter (only along the categorical axis) which can be useful when you have many points and they overlap so that it is easier to see the distribution.
- hue is used to provide an additional categorical separation
- setting split=True is used to draw separate strip plots based on the category specified by the hue parameter.

Heatmap

Heatmap is defined as a graphical representation of data using colors to visualize the value of the matrix. In this, to represent more common values or higher activities brighter colors basically reddish colors are used and to represent less common or activity values, darker colors are preferred. Heatmap is also defined by the name of the shading matrix. Heatmaps in Seaborn can be plotted by using the seaborn.heatmap() function.

seaborn.heatmap()

Syntax:

seaborn.heatmap(data, *, vmin=None, vmax=None, cmap=None, center=None, annot_kws=None, linewidths=0, linecolor='white', cbar=True, **kwargs)

Important Parameters:

- data: 2D dataset that can be coerced into an ndarray.
- vmin, vmax: Values to anchor the colormap, otherwise they are inferred from the data and other keyword arguments.
- cmap: The mapping from data values to color space.
- center: The value at which to center the colormap when plotting divergent data.
- annot: If True, write the data value in each cell.
- fmt: String formatting code to use when adding annotations.
- linewidths: Width of the lines that will divide each cell.
- linecolor: Color of the lines that will divide each cell.
- cbar: Whether to draw a colorbar.

All the parameters except data are optional.

Returns: An object of type matplotlib.axes._subplots.AxesSubplot

Basic Heatmap

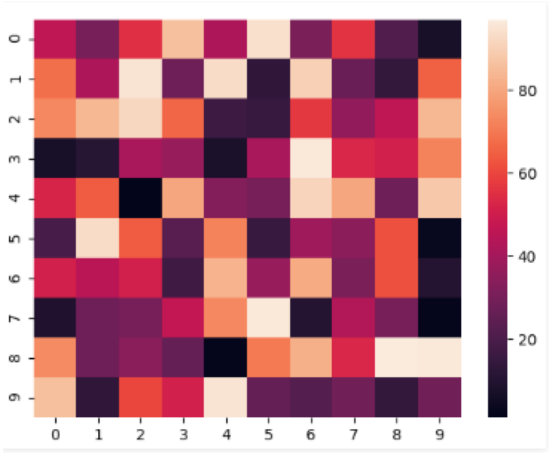
Making a heatmap with the default parameters. We will be creating a 10×10 2-D data using the randint() function of the NumPy module.

```
# importing the modules
import numpy as np
import seaborn as sn
import matplotlib.pyplot as plt
# generating 2-D 10x10 matrix of random numbers from 1 to 100
data = np.random.randint(low = 1, high = 100, size = (10, 10))
print("The data to be plotted:\n")
print(data)
# plotting the heatmap
hm = sn.heatmap(data = data)
```

```
# displaying the plotted heatmap
plt.show()
```

Output/Results snippet:

```
The data to be plotted:

[[46 30 55 86 42 94 31 56 21  7]
[68 42 95 28 93 13 90 27 14 65]
[73 84 92 66 16 15 57 36 46 84]
[ 7 11 41 37  8 41 96 53 51 72]
[52 64  1 80 33 30 91 80 28 88]
[19 93 64 23 72 15 39 35 62  3]
[51 45 51 17 83 37 81 31 62 10]
[ 9 28 30 47 73 96 10 43 30  2]
[74 28 34 26  2 70 82 53 97 96]
[86 13 60 51 95 26 22 29 14 29]]
```



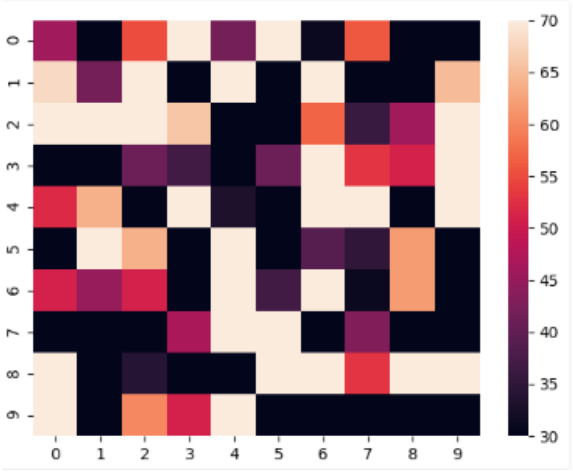
Anchoring the colormap

If we set the vmin value to 30 and the vmax value to 70, then only the cells with values between 30 and 70 will be displayed. This is called anchoring the colormap.

```
# importing the modules
import numpy as np
import seaborn as sn
```

```
import matplotlib.pyplot as plt
# generating 2-D 10x10 matrix of random numbers from 1 to 100
data = np.random.randint(low=1, high=100, size=(10, 10))
# setting the parameter values
vmin = 30
vmax = 70
# plotting the heatmap
hm = sn.heatmap(data=data, vmin=vmin, vmax=vmax)
# displaying the plotted heatmap
plt.show()
```

Output/Results snippet:



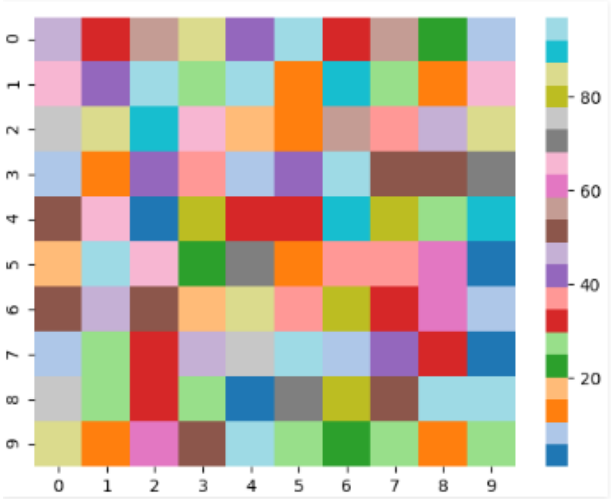
Choosing the colormap

In this, we will be looking at the cmap parameter. Matplotlib provides us with multiple colormaps, you can look at all of them here. In our example, we'll be using tab20.

```
# importing the modules
import numpy as np
```

```
import seaborn as sn
import matplotlib.pyplot as plt
# generating 2-D 10x10 matrix of random numbers from 1 to 100
data = np.random.randint(low=1, high=100, size=(10, 10))
# setting the parameter values
cmap = "tab20"
# plotting the heatmap
hm = sn.heatmap(data=data, cmap=cmap)
# displaying the plotted heatmap
plt.show()
```

Output/Results snippet:



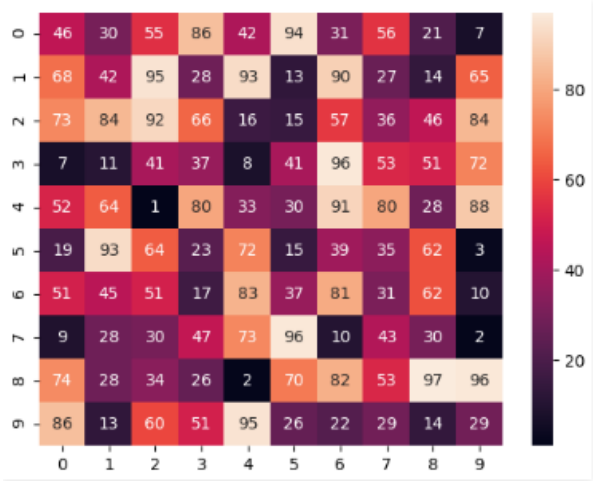
Displaying the cell values

If we want to display the value of the cells, then we pass the parameter annot as True. fmt is used to select the datatype of the contents of the cells displayed.

```
# importing the modules
import numpy as np
```

```
import seaborn as sn
import matplotlib.pyplot as plt
# generating 2-D 10x10 matrix of random numbers from 1 to 100
data = np.random.randint(low=1, high=100, size=(10, 10))
# setting the parameter values
annot = True
# plotting the heatmap
hm = sn.heatmap(data=data, annot=annot)
# displaying the plotted heatmap
plt.show()
```

Output/Results snippet:



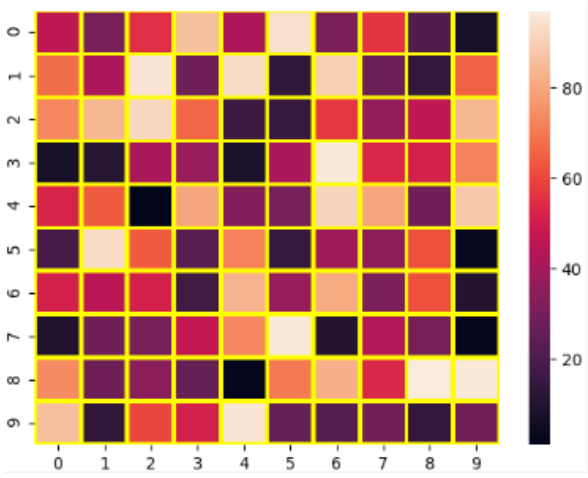
Customizing the separating line

We can change the thickness and the color of the lines separating the cells using the linewidths and linecolor parameters respectively.

```
# importing the modules
import numpy as np
```

```
import seaborn as sn
import matplotlib.pyplot as plt
# generating 2-D 10x10 matrix of random numbers from 1 to 100
data = np.random.randint(low=1, high=100, size=(10, 10))
# setting the parameter values
linewidths = 2
linecolor = "yellow"
# plotting the heatmap
hm = sn.heatmap(data=data, linewidths=linewidths, linecolor=linecolor)
# displaying the plotted heatmap
plt.show()
```

Output/Results snippet:



Regression Plot

The regression plots in seaborn are primarily intended to add a visual guide that helps to emphasize patterns in a dataset during exploratory data analyses. Regression plots as the name suggests creates a regression line between 2 parameters and helps to visualize their linear relationships.

Load the dataset

```
# import the library

import seaborn as sns

# load the dataset

dataset = sns.load_dataset('tips')

# the first five entries of the dataset

dataset.head()
```

Output/Results snippet:

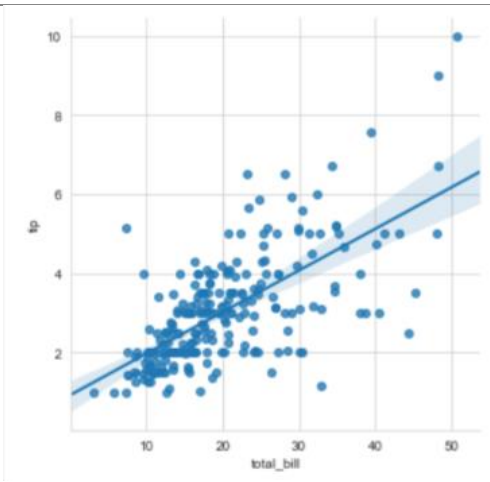
	total_bill	tip	sex	smoker	day	time	size
0	16.99	1.01	Female	No	Sun	Dinner	2
1	10.34	1.66	Male	No	Sun	Dinner	3
2	21.01	3.50	Male	No	Sun	Dinner	3
3	23.68	3.31	Male	No	Sun	Dinner	2
4	24.59	3.61	Female	No	Sun	Dinner	4

Regression plots in seaborn can be easily implemented with the help of the lmplot() function. lmplot() can be understood as a function that basically creates a linear model plot. lmplot() makes a very simple linear regression plot.It creates a scatter plot with a linear fit on top of it.

Simple linear plot

```
sns.set_style('whitegrid')

sns.lmplot(x ='total_bill', y ='tip', data = dataset)
```

Output/Results snippet:



Explanation

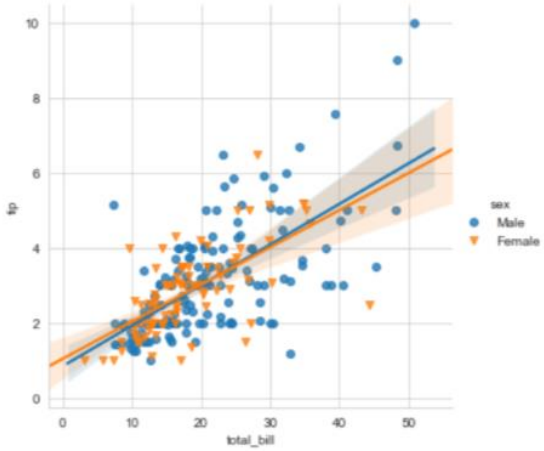
x and y parameters are specified to provide values for the x and y axes. sns.set_style() is used to have a grid in the background instead of a default white background. The data parameter is used to specify the source of information for drawing the plots.

Linear plot with additional parameters

sns.set_style('whitegrid')

sns.lmplot(x ='total_bill', y ='tip', data = dataset, hue ='sex', markers =['o', 'v'])

Output/Results snippet:



Explanation

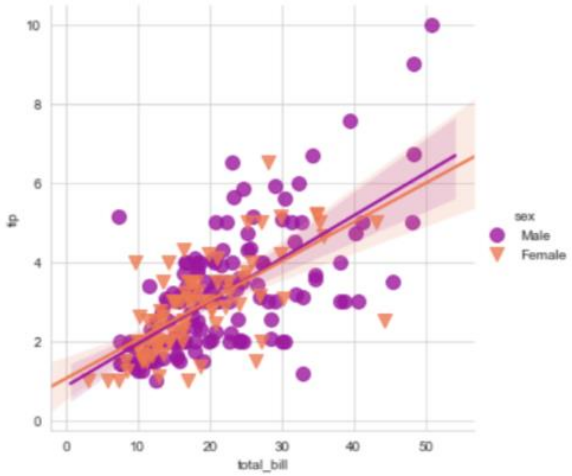
In order to have a better analysis capability using these plots, we can specify hue to have a categorical separation in our plot as well as use markers that come from the matplotlib marker symbols. Since we have two separate categories we need to pass in a list of symbols while specifying the marker.

Setting the size and color of the plot

```
sns.set_style('whitegrid')
```

```
sns.lmplot(x ='total_bill', y ='tip', data = dataset, hue ='sex', markers =['o', 'v'], scatter_kws ={'s':100}, palette ='plasma')
```

Output/Results snippet:



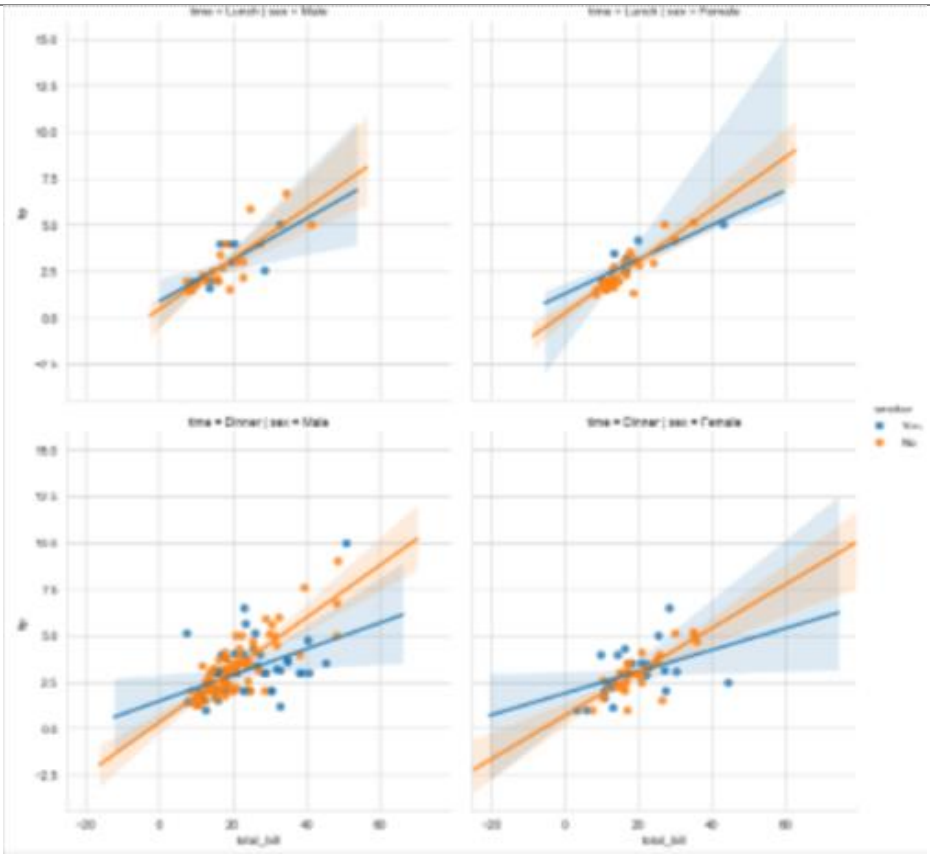
Explanation

In this example what seaborn is doing is that its calling the matplotlib parameters indirectly to affect the scatter plots. We specify a parameter called scatter_kws. We must note that the scatter_kws parameter changes the size of only the scatter plots and not the regression lines. The regression lines remain untouched. We also use the palette parameter to change the color of the plot.

Displaying multiple plots

```
sns.lmplot(x ='total_bill', y ='tip', data = dataset, col ='sex', row ='time', hue ='smoker')
```

Output/Results snippet:



References:

1. https://www.geeksforgeeks.org/plotting-graph-using-seaborn-python/?ref=lbp
2. https://www.geeksforgeeks.org/seaborn-regression-plots/

Learning Outcome

After completing this module, the student should be able to learn scikit-learn library.

To meet the learning outcome, a student has to complete the following activities

1. Installing sklearn library (2 hrs)
2. Simple linear regression using excel (3 hrs)
3. OLS in sklearn (3 hrs)
4. Train-test-split of data in sklearn (3 hrs)
5. Methods of linear regression- fit(), predict(), coeff_, intercept_, score() (3 hrs)

Activity 1

Aim: Installing sklearn library.

Learning outcome: Able to install scikit-learn library.

Duration: 2 hours

List of Hardware/Software requirements:

1. Laptop/Computer with Windows 10 / Linux OS - Ubuntu 18.04 LTS
2. Jupyter notebook / Google colab
3. Python 3 and above

Code/Program/Procedure (with comments):

Operating System - Windows

1. Scikit-learn requires Python 3.6+. To check which version of Python you have installed, run the following command:

python3 --version

The output should be similar to:

Python 3.8.2

2. If you have a valid Python version you can run the following command to download and install a pre-built binary of scikit-learn:

pip install scikit-learn

The following dependencies will be automatically installed along with scikit-learn:

NumPy 1.13.3+

SciPy 0.19.1+

Joblib 0.11+

threadpoolctl 2.0.0+

Alternatively, if you already have scikit-learn and/or any of its dependencies are already installed, they can be updated as part of the installation by running the following command:

pip install -U scikit-learn

Operating system – Ubuntu

$ sudo apt-get install python3-sklearn python3-sklearn-lib python3-sklearn-doc

You can verify your Scikit-learn installation with the following command:

python -m pip show scikit-learn

Output/Results snippet:

```
C:\>python3 -m pip show scikit-learn
Name: scikit-learn
Version: 0.24.0
Summary: A set of python modules for machine learning and data mining
Home-page: http://scikit-learn.org
Author: None
Author-email: None
License: new BSD
Location: c:\python38\lib\site-packages
Requires: numpy, scipy, threadpoolctl, joblib
Required-by:
```

References:

- https://www.geeksforgeeks.org/how-to-install-scikit-learn-on-linux/
- https://scikit-learn.org/stable/install.html
- https://www.activestate.com/resources/quick-reads/how-to-install-scikit-learn/

Activity 2

Aim: Simple linear regression using excel.

Learning outcome: Able to learn how linear regression can be done with or without excel specific tool.

Duration: 3 hours

List of Hardware/Software requirements:

1. Laptop/Computer with Windows 10 / Linux OS - Ubuntu 18.04 LTS
2. Microsoft Excel 2013 and above

Code/Program/Procedure (with comments):

Linear regression is a statistical technique/method used to study the relationship between two continuous quantitative variables.

A linear regression line has an equation of the kind: **Y= a + bX;**

Where:

X is the explanatory variable,

Y is the dependent variable,

b is the slope of the line,

a is the y-intercept (i.e., the value of y when x=0).

Method #1 – Scatter Chart with a Trendline

Let us say we have a dataset of some individuals with their age, bio-mass index (BMI), and the amount spent by them on medical expenses in a month. Now with an insight into the individuals' characteristics like age and BMI, we wish to find how these variables affect the medical expenses, and hence use these to carry out regression and estimate/predict the average medical expenses for some specific individuals. Let us first see how only age affects medical expenses. Let us see the dataset:

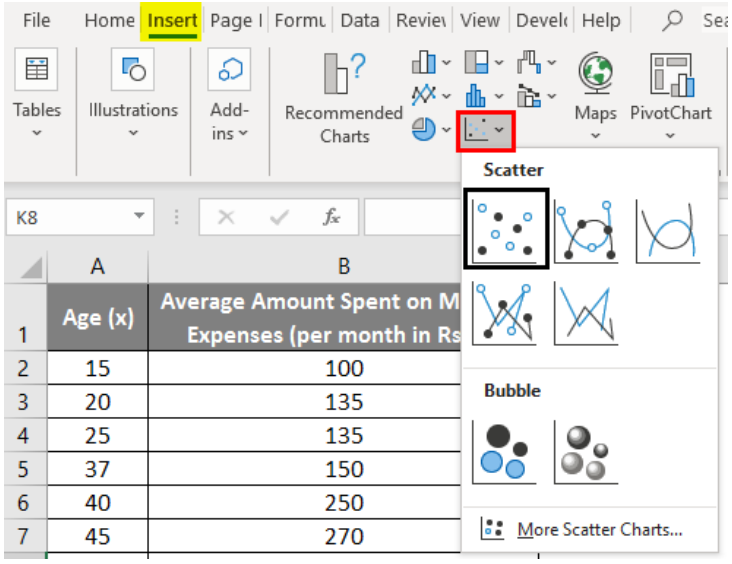
	A	B
1	Age (x)	Average Amount Spent on Medical Expenses (per month in Rs) (y)
2	15	100
3	20	135
4	25	135
5	37	150
6	40	250
7	45	270
8	48	290
9	50	360
10	55	375
11	61	400
12	64	500
13	67	1000
14	70	1500
15		

Amount on medical expenses= b*age + a

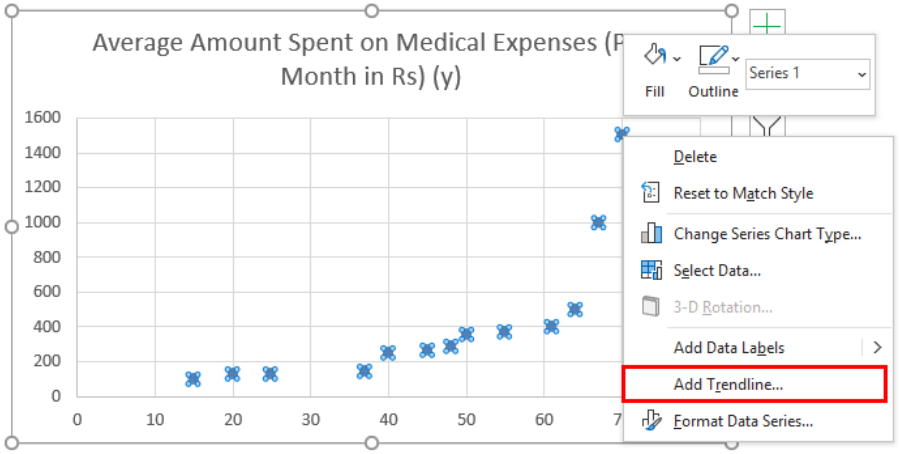
- Select the two columns of the dataset (x and y), including headers.

	A	B
1	Age (x)	Average Amount Spent on Medical Expenses (per month in Rs) (y)
2	15	100
3	20	135
4	25	135
5	37	150
6	40	250
7	45	270
8	48	290
9	50	360
10	55	375
11	61	400
12	64	500
13	67	1000
14	70	1500
15		

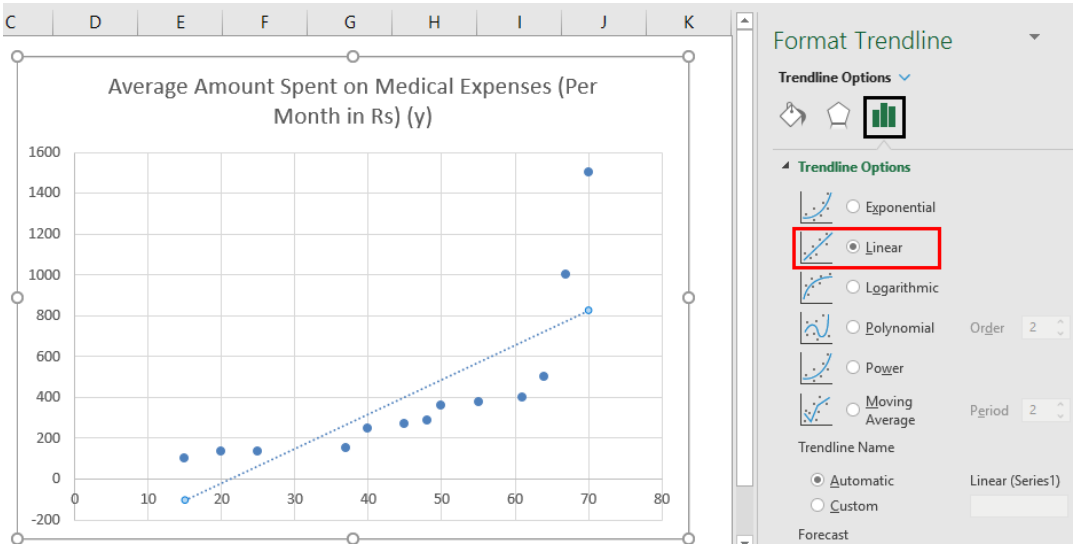
- Click on ‘Insert’ and expand the dropdown for ‘Scatter Chart’ and select ‘Scatter’ thumbnail (first one)



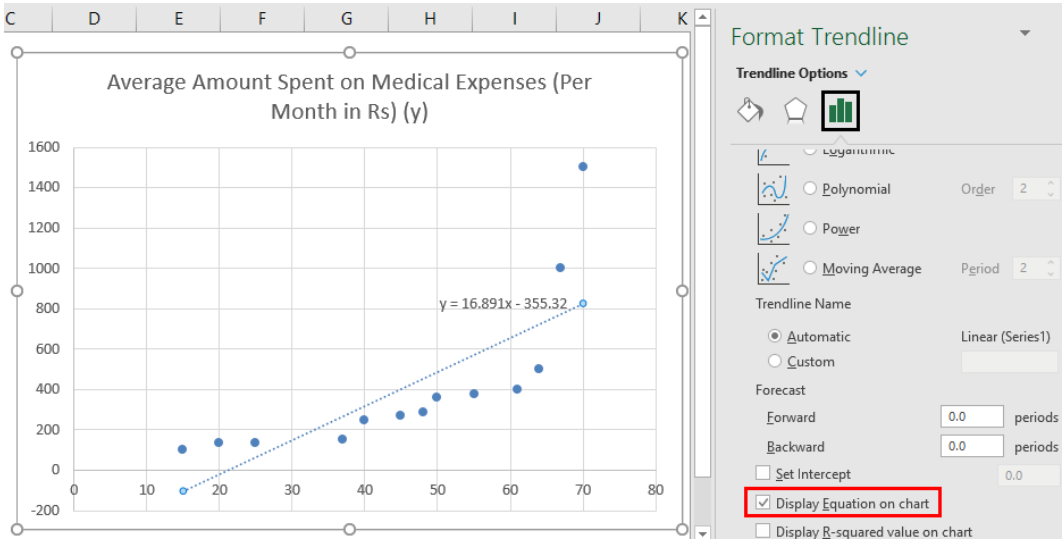
- Now a scatter plot will appear, and we would draw the regression line on this. To do this, right-click on any data point and select ‘Add Trendline.’



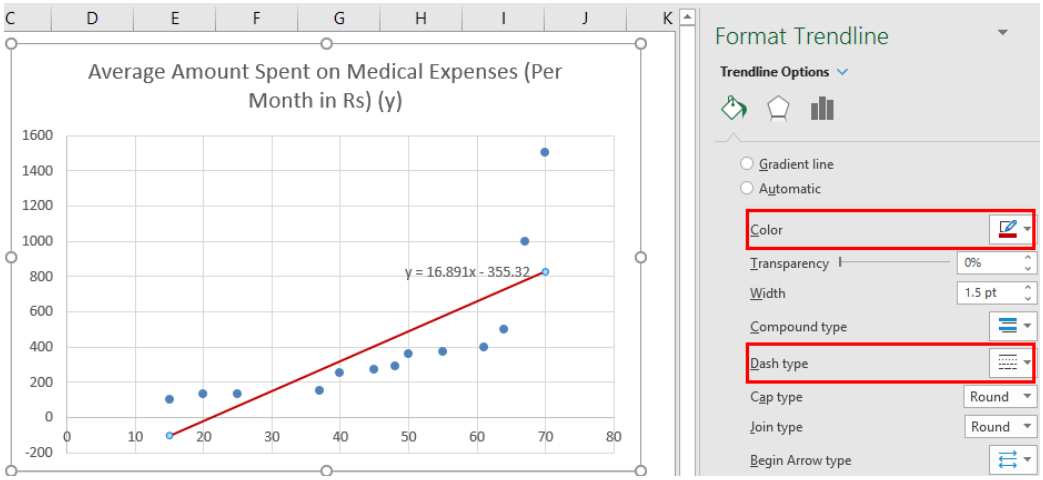
- Now in the ‘Format Trendline’ pane on the right, select ‘Linear Trendline’ and ‘Display Equation on Chart’.



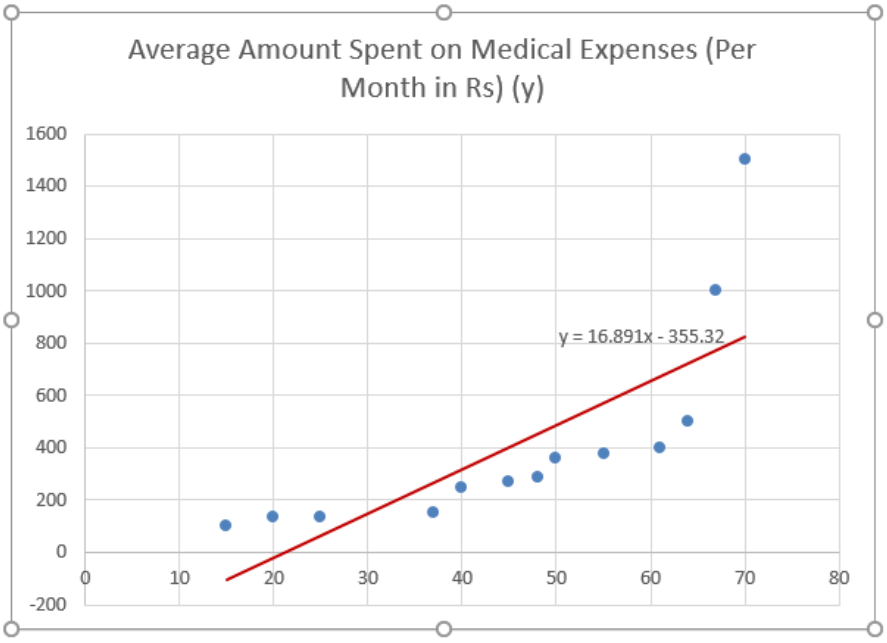
- Select 'Display Equation on Chart'.



- We can improvise the chart as per our requirements, like adding axes titles, changing the scale, color and line type.



Output/Results snippet:

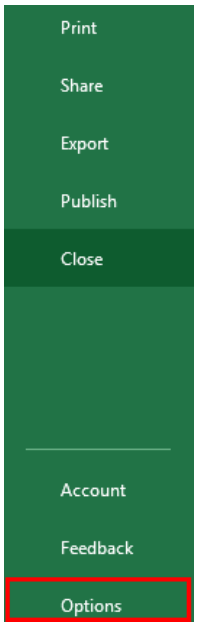


Method #2 – Analysis ToolPak Add-In Method

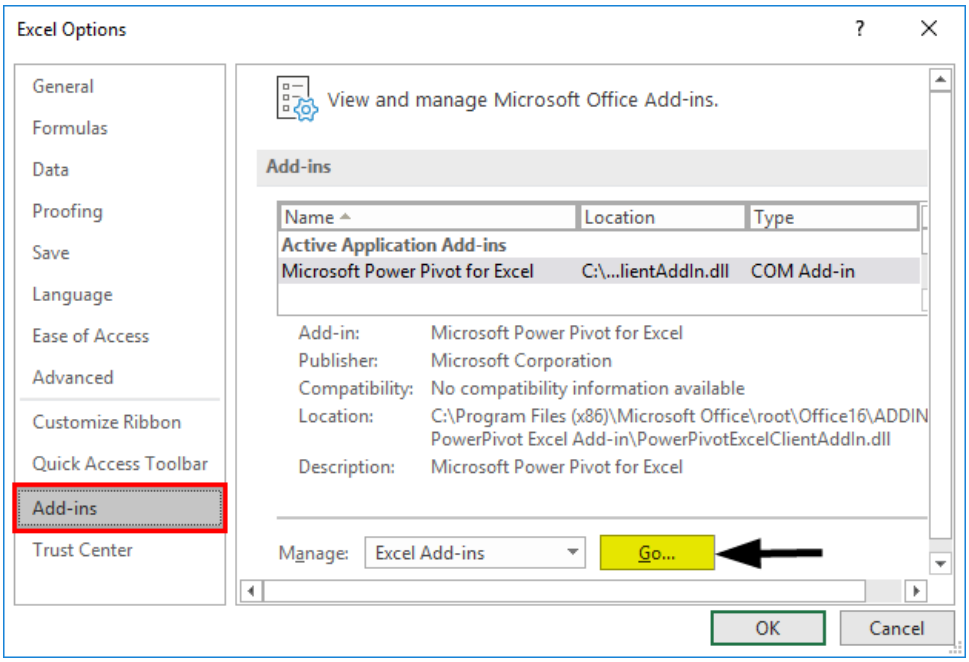
Analysis ToolPak is sometimes not enabled by default, and we need to do it manually. To do so:

	A	B
1	Age (x)	Average Amount Spent on Medical Expenses (Per Month in Rs) (y)
2	15	100
3	20	135
4	25	135
5	37	150
6	40	250
7	45	270
8	48	290
9	50	360
10	55	375
11	61	400
12	64	500
13	67	1000
14	70	1500
15		

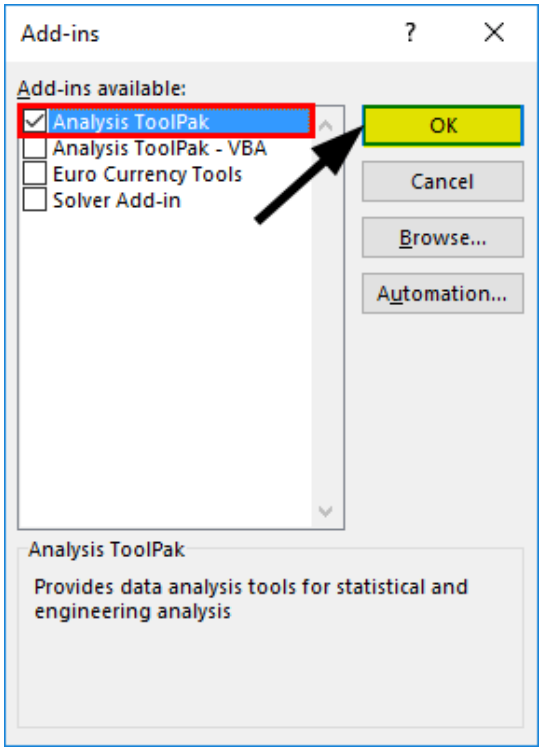
- Click on the ‘File’ menu, after that, click on ‘Options’.



- Select ‘Excel Add-Ins’ in the ‘Manage’ box, and click on ‘Go.’

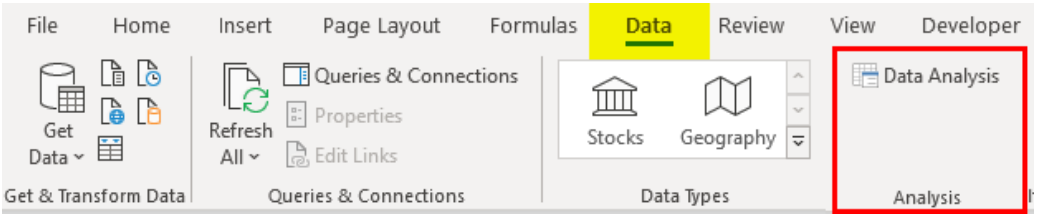


- Select ‘Analysis ToolPak’ -> ‘OK’

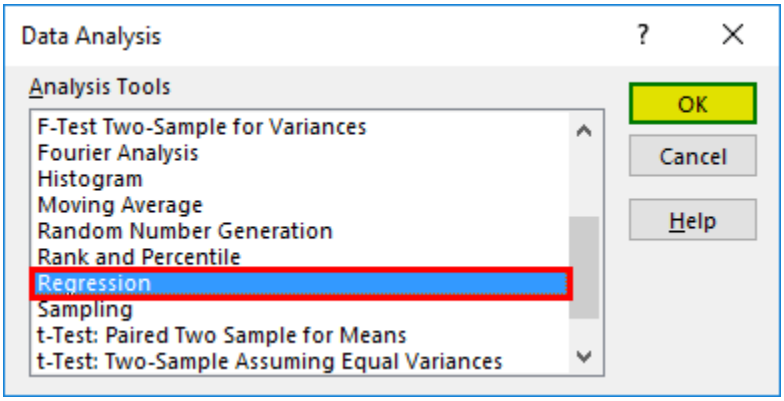


This will add ‘Data Analysis’ tools to the ‘Data’ tab. Now we run the regression analysis:

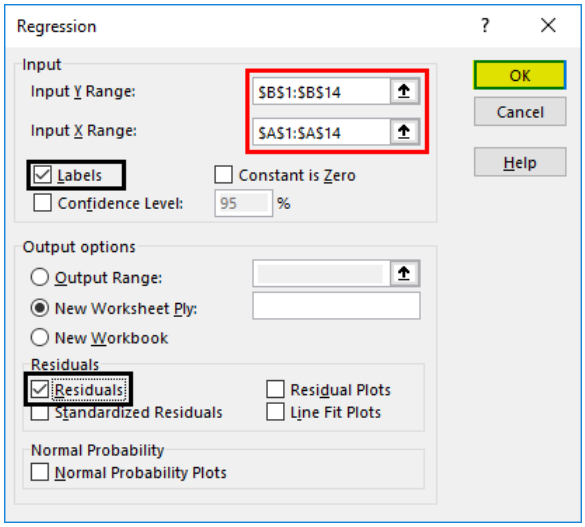
- Click on ‘Data Analysis’ in the ‘Data’ tab



- Select ‘Regression’ -> ‘OK’



- A regression dialog box will appear. Select the Input Y range and Input X range (medical expenses and age, respectively). In the case of multiple linear regression, we can select more columns of independent variables (like if we wish to see the impact of BMI as well on medical expenses).
- Check the ‘Labels’ box to include headers.
- Choose the desired ‘output’ option.
- Select the ‘residuals’ checkbox and click ‘OK.



Output/Results snippet:

- **Regression Statistics** tells how well the regression equation fits the data:

	A	B
1	SUMMARY OUTPUT	
2		
3	Regression Statistics	
4	Multiple R	0.75741
5	R Square	0.57367
6	Adjusted R Square	0.53491
7	Standard Error	272.88
8	Observations	13
9		

- **Multiple R** is the correlation coefficient that measures the strength of a linear relationship between two variables. It lies between -1 and 1, and its absolute value depicts the relationship strength with a large value indicating a stronger relationship, a low value indicating negative and zero value indicating no relationship.
- **R Square** is the Coefficient of Determination used as an indicator of goodness of fit. It lies between 0 and 1, with a value close to 1 indicating that the model is a good fit. In this case, 0.57=57% of y-values are explained by the x-values.
- **Adjusted R Square** is R Square adjusted for a number of predictors in the case of multiple linear regression.
- **Standard Error** depicts the precision of regression analysis.
- **Observations** depict the number of model observations.
- **Anova** tells the level of variability within the regression model.

10	ANOVA					
11		*df*	*SS*	*MS*	*F*	*Significance F*
12	Regression	1	1102174	1102173.8	14.801518	0.00271224
13	Residual	11	819099.3	74463.569		
14	Total	12	1921273			

- **Coefficients** are the most important part used to build regression equation.

	Coefficients	Standard Error	t Stat	P-value	Lower 95%	Upper 95%	Lower 95.0%	Upper 95.0%
Intercept	-355.3228922	215.361787	-1.649888296	0.1271999	-829.33099	118.685206	-829.33099	118.685206
Age (x)	16.89145326	4.3904991	3.847274049	0.0027122	7.228029892	26.5548766	7.22802989	26.5548766

So, our regression equation would be: y= 16.891 x – 355.32. This is the same as that done by method 1 (scatter chart with a trendline).

Now, if we wish to predict average medical expenses when age is 72:

So, y= 16.891 * 72 -355.32 = 860.832

So, this way, we can predict values of y for any other values of x.

- **Residuals** indicate the difference between actual and predicted values.

RESIDUAL OUTPUT

Observation	*Predicted Average Amount Spent on Medical Expenses (Per Month in Rs) (y)*	*Residuals*
1	-101.9510932	201.951093
2	-17.49382691	152.493827
3	66.9634394	68.0365606
4	269.6608786	-119.660879
5	320.3352384	-70.3352384
6	404.7925047	-134.792505
7	455.4668645	-165.466864
8	489.249771	-129.249771
9	573.7070373	-198.707037
10	675.0557569	-275.055757
11	725.7301167	-225.730117
12	776.4044765	223.595524
13	827.0788363	672.921164

References:

- https://www.educba.com/linear-regression-in-excel/
- https://www.statology.org/simple-linear-regression-excel/

Activity 3

Aim: OLS in sklearn

Learning outcome: Able to learn how linear regression can be done with or without excel specific tool.

Duration: 3 hours

List of Hardware/Software requirements:

1. Laptop/Computer with Windows 10 / Linux OS - Ubuntu 18.04 LTS
2. Jupyter notebook / Google colab
3. Python 3 and above

Code/Program/Procedure (with comments):

Ordinary Least Squares is a method for finding the linear combination of features that best fits the observed outcome in the following sense.

If the vector of outcomes to be predicted is y, and the explanatory variables form the matrix X, then OLS will find the vector β solving

$$\min_{\beta} |\hat{y} - y|_2^2,$$

where $\hat{y} = X\beta$ is the linear prediction.

In sklearn, this is done using **sklearn.linear_model.LinearRegression**

Application Context

OLS should only be applied to regression problems; it is generally unsuitable for classification problems: Contrast

Is an email spam? (Classification)

What is the linear relationship between upvotes depend on the length of answer? (Regression)

Discovering the Data

```
import pandas as pd

dataset_url = 'https://sealevel-
nexus.jpl.nasa.gov/data/ice_shelf_dh_mean_v1/ice_shelf_dh_mean_v1_height.csv'

dataset = pd.read_csv(dataset_url)

dataset.head()

#Let's create x and y vectors.

import numpy as np

# Read CSV into table and get (x, y) pairs.

N = dataset.shape[0] # size of input samples

x = np.array(dataset['Year']).reshape([N, 1])

y = np.array(dataset['All Antarctica']).reshape([N, 1])

points = np.hstack([x, y])
```

Creating the Model - Least Squares Estimation

Solve the Least Squares Regression by Hand

```
# Calculate power series sums.

x0 = np.sum(x**0)

x1 = np.sum(x**1)

x2 = np.sum(x**2)

x3 = np.sum(x**3)

x4 = np.sum(x**4)

yx0 = np.sum(y * x**0)

yx1 = np.sum(y * x**1)

yx2 = np.sum(y * x**2)
```

Create 3rd order model matrices.

A = [[x0, x1, x2], [x1, x2, x3], [x2, x3, x4]]

B = [[yx0], [yx1], [yx2]]

Obtain Model Coefficients

import numpy.linalg as lin

M = np.matmul(lin.inv(A), B)

The degree-two polynomial coefficients are found as below.

[[-5.48765643e+03],

[5.49213398e+00],

[-1.37413749e-03]]

Simulate the Estimated Curve

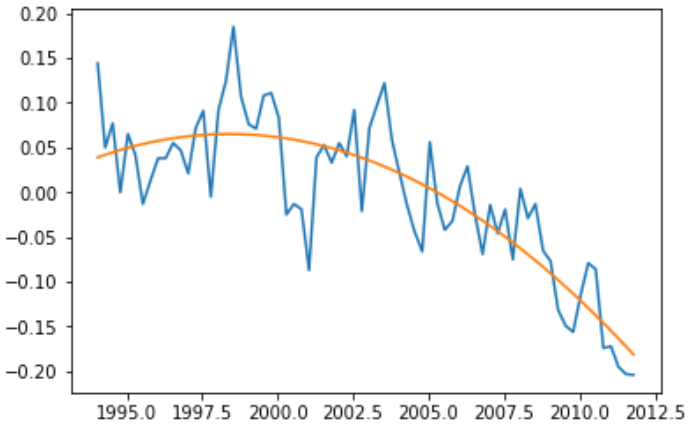
#To visualize the result, we can create y_estimated

import matplotlib.pyplot as plt

y_estimated = x**0 * M[0] + x**1 * M[1] + x**2 * M[2]

plt.plot(x, y, x, y_estimated)

plt.show()



Prediction of Future Values

y2020 = 2020**0 * M[0] + 2020**1 * M[1] + 2020**2 * M[2]

RMS Error

#To see the overall performance of the fit, we can simply take root-mean-square of the error.

rmse = (np.sum((yest - y) **2) / len(y)) ** 0.5

Output/Results snippet:

The result is 0.047179935281228005.

References:

- http://www.atakansarioglu.com/machine-learning-example-generalized-least-squares-sklearn-scikit-python-hands-on/#solve_the_least_squares_regression_by_hand
- https://www.datarobot.com/blog/ordinary-least-squares-in-python/

Activity 4

Aim: Train-test-split of data in sklearn

Learning outcome: Able to learn how to Train-test-split of data in sklearn.

Duration: 3 hours

List of Hardware/Software requirements:

1. Laptop/Computer with Windows 10 / Linux OS - Ubuntu 18.04 LTS
2. Jupyter notebook / Google colab
3. Python 3 and above

Code/Program/Procedure (with comments):

Configuring Test Train Split

Before splitting the data, you need to know how to configure the train test split percentage.

In most cases, the common split percentages are

Train: 80%, Test: 20%

Train: 67%, Test: 33%

Train: 50%, Test: 50%

Loading The Dataset

```
import numpy as np
from sklearn.datasets import load_iris
# the iris dataset which has four features Sepal_length, Sepal_width, Petal_length, and Petal_Width
iris = load_iris()
x = iris.data
y = iris.target
```

Train Test Split Using Sklearn Library

You can split the dataset into train and test set using the train_test_split() method of the sklearn library. It accepts one mandatory parameter.

–*Input Dataset* – It is a sequence of array-like objects of the same size. Allowed inputs are lists, NumPy arrays, scipy-sparse matrices, or pandas data frames.

The Input dataset passed as X and y along with the test_size = 0.4. It means the data will be split into 60% for training and 40% for testing.

```
from collections import Counter

from sklearn.model_selection import train_test_split

#Split dataset into train and test

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.4)

print(Counter(y_train))

print(Counter(y_test))
```

Output/Results snippet:

```
Counter({0: 34, 1: 25, 2: 31})

Counter({0: 16, 1: 25, 2: 19})
```

The train set contains, 34 number of 0 labels, 25 number of 1 labels, and 31 number of 2 labels.

Train Test Split with Groups

You can do a train test split with groups using the GroupShuffleSplit() method from the sklearn library.

```
from sklearn.datasets import load_iris

from sklearn.model_selection import GroupShuffleSplit

import pandas as pd

data = load_iris()

df = pd.DataFrame(data.data, columns=data.feature_names)

df["target"] = data.target
```

```
train_idx, test_idx = next(GroupShuffleSplit(test_size=.20, n_splits=2, random_state = 7).split(df, groups=df['target']))

train = df.iloc[train_idx]

test = df.iloc[test_idx]

#To display the training set

train.groupby(['target']).count()
```

Output/Results snippet:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
target				
0	50	50	50	50
1	50	50	50	50

```
#To print the test dataset count.

test.groupby(['target']).count()
```

Output/Results snippet:

Dataframe will look like

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
target				
2	50	50	50	50

References:

- https://www.stackvidhya.com/train-test-split-using-sklearn-in-python/
- https://stackabuse.com/scikit-learns-traintestsplit-training-testing-and-validation-sets/

Activity 5

Aim: Methods of linear regression- fit(), predict(), coeff_, intercept_, score()

Learning outcome: Able to learn different methods of linear regression- fit(), predict(), coeff_, intercept_, score()

Duration: 3 hours

List of Hardware/Software requirements:

1. Laptop/Computer with Windows 10 / Linux OS - Ubuntu 18.04 LTS
2. Jupyter notebook / Google colab
3. Python 3 and above

Code/Program/Procedure (with comments):

Step 1: Import packages and classes

```
import numpy as np
from sklearn.linear_model import LinearRegression
```

Step 2: Provide data

```
#The inputs (regressors, 𝑥) and output (predictor, 𝑦) should be arrays (the instances of the class numpy.ndarray)
x = np.array([5, 15, 25, 35, 45, 55]).reshape((-1, 1))
y = np.array([5, 20, 14, 32, 22, 38])

>>> print(x)
```

Output/Results snippet:

```
[[ 5]
 [15]
 [25]
 [35]
 [45]
 [55]]
```

```
>>> print(y)
```

Output/Results snippet:

```
[ 5 20 14 32 22 38]
```

Step 3: Create a model and fit it

```
model = LinearRegression()

model.fit(x, y)

model = LinearRegression().fit(x, y)

>>> r_sq = model.score(x, y)
>>> print('coefficient of determination:', r_sq)
```

Output/Results snippet:

```
coefficient of determination: 0.715875613747954

>>> print('intercept:', model.intercept_)
```

Output/Results snippet:

```
intercept: 5.633333333333329

>>> print('slope:', model.coef_)
```

Output/Results snippet:

```
slope: [0.54]

>>> new_model = LinearRegression().fit(x, y.reshape((-1, 1)))
>>> print('intercept:', new_model.intercept_)
```

Output/Results snippet:

```
intercept: [5.63333333]

>>> print('slope:', new_model.coef_)
```

Output/Results snippet:

slope: [[0.54]]

Step 5: Predict response

```
>>> y_pred = model.predict(x)
>>> print('predicted response:', y_pred, sep='\n')
```

Output/Results snippet:

```
predicted response:
[ 8.33333333 13.73333333 19.13333333 24.53333333 29.93333333 35.33333333]
```

```
>>> y_pred = model.intercept_ + model.coef_ * x
>>> print('predicted response:', y_pred, sep='\n')
```

Output/Results snippet:

```
predicted response:
[[ 8.33333333]
 [13.73333333]
 [19.13333333]
 [24.53333333]
 [29.93333333]
 [35.33333333]]
```

```
>>> x_new = np.arange(5).reshape((-1, 1))
>>> print(x_new)
```

Output/Results snippet:

```
[[0]
 [1]
 [2]
 [3]
 [4]]
```

```
>>> y_new = model.predict(x_new)
>>> print(y_new)
```

Output/Results snippet:

[5.63333333 6.17333333 6.71333333 7.25333333 7.79333333]

Learning outcome- able to implement Logistic Regression and Flask app

After achieving this learning outcome, a student will be able to implement Logistic Regression and Flask app. In order to achieve this learning outcome, a student has to complete the following:

Activities:

1. Implementing logistic regression for binary and multi-class classification (5 hours)
2. Sigmoid function in Logistic regressions (5 hours)
3. Predicting probability of classification models (3 hours)
4. Charting confusion matrix (3 hours)
5. Integration of analytics with django/Flask app (10 hours)

Activity 1

Aim: Implementing logistic regression for binary and multi-class classification

Learning outcome: Able to implement logistic regression for binary and multi-class classification.

Duration: 5 hours

List of Hardware/Software requirements:

1. Anaconda
2. Windows/Linux

Code/Program/Procedure (with comments):

Import Packages, Functions, and Classes

```
import matplotlib.pyplot as plt
import numpy as np
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report
```

Get Data

```
x = np.arange(10).reshape(-1, 1)
y = np.array([0, 0, 0, 0, 1, 1, 1, 1, 1, 1])
x
y
```

Create a Model and Train

```
model = LogisticRegression(solver='liblinear', random_state=0)
model.fit(x, y)
```

Classification model defined

```
model.classes_
```

Evaluate the Model

model.predict_proba(x)

model.predict(x)

model.score(x, y)

Output/Results snippet:

```
array([0, 0, 0, 1, 1, 1, 1, 1, 1, 1])
```

```
0.9
```

Program 2:

Import Packages, Functions, and Classes

```
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
```

Load the Data set

```
data = sns.load_dataset("iris")
data.head()
```

Prepare the training set

```
# x = feature values, all the columns except the last column
x = data.iloc[:, :-1]
# y = target values, last column of the data frame
y = data.iloc[:, -1]
```

Split the data into 80% training and 20% testing

```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=42)
```

Train the model

```
model = LogisticRegression()
model.fit(x_train, y_train)
```

Evaluate the model

```
predictions = model.predict(x_test)

print(predictions)# printing predictions

print () # Printing new line

print (classification_report(y_test, predictions) )

print (accuracy_score(y_test, predictions))
```

Output/Results snippet:

```
Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG,
LogisticRegression()

['versicolor' 'setosa' 'virginica' 'versicolor' 'versicolor' 'setosa'
 'versicolor' 'virginica' 'versicolor' 'versicolor' 'virginica' 'setosa'
 'setosa' 'setosa' 'setosa' 'versicolor' 'virginica' 'versicolor'
 'versicolor' 'virginica' 'setosa' 'virginica' 'setosa' 'virginica'
 'virginica' 'virginica' 'virginica' 'virginica' 'setosa' 'setosa']

              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        10
  versicolor       1.00      1.00      1.00         9
   virginica       1.00      1.00      1.00        11

    accuracy                           1.00        30
   macro avg       1.00      1.00      1.00        30
weighted avg       1.00      1.00      1.00        30

1.0
```

References:

- https://www.w3schools.com/https://www.datacamp.com/community/tutorials/understanding-logistic-regression-python
- https://realpython.com/logistic-regression-python/
- https://randerson112358.medium.com/python-logistic-regression-program-5e1b32f964db

Activity 2

Aim: Implementing sigmoid function in logistic regression

Learning outcome: Able to implement sigmoid function in logistic regression.

Duration: 5 hours

List of Hardware/Software requirements:

1. Anaconda
2. Windows/Linux

Code/Program/Procedure (with comments):

Import Packages, Functions, and Classes

```
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
from scipy import optimize as op
```

Load the Data set

```
data_iris = sns.load_dataset("iris")
data_iris.head()
```

Data setup

```
import numpy as np
species = ['setosa', 'versicolor', 'virginica']
# Number of examples
m = data_iris.shape[0]
# Features
n = 4
# Number of classes
```

```
k = 3
X = np.ones((m,n + 1))
y = np.array((m,1))
X[:,1] = data_iris['petal_length'].values
X[:,2] = data_iris['petal_width'].values
X[:,3] = data_iris['sepal_length'].values
X[:,4] = data_iris['sepal_width'].values

# Labels
y = data_iris['species'].values

# Mean normalization
for j in range(n):
    X[:, j] = (X[:, j] - X[:,j].mean())
**Split the data into 80% training and 20% testing**
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=11)
x = data_iris.drop(['species'], axis=1)
y = data_iris['species']
**Define functions**
# Sigmoid function
def sigmoid(z):
    return 1.0 / (1 + np.exp(-z))

# Regularized cost function
def reglrCostFunction(theta, X, y, lambda_s = 0.1):
    m = len(y)
    h = sigmoid(X.dot(theta))
```

```
    J = (1 / m) * (-y.T.dot(np.log(h)) - (1 - y).T.dot(np.log(1 - h)))
    reg = (lambda_s/(2 * m)) * np.sum(theta**2)
    J = J + reg
     return J

# Regularized gradient function
def reglrGradient(theta, X, y, lambda_s = 0.1):
    m, n = X.shape
    theta = theta.reshape((n, 1))
    y = y.reshape((m, 1))
    h = sigmoid(X.dot(theta))
    reg = lambda_s * theta /m
    gd = ((1 / m) * X.T.dot(h - y))
    gd = gd + reg
    return gd

def logisticRegression(X, y, theta):
    result = op.minimize(fun = reglrCostFunction, x0 = theta, args = (X, y),
                         method = 'TNC', jac = reglrGradient)
    return result.x
**Training the model**
all_theta = np.zeros((k, n + 1))
# One vs all
i = 0
for flower in species:
    tmp_y = np.array(y_train == flower, dtype = int)
    optTheta = logisticRegression(X_train, tmp_y, np.zeros((n + 1,1)))
```

```
    all_theta[i] = optTheta
    i += 1
**Evaluate the model**
Prob = sigmoid(X_test.dot(all_theta.T)) # probability for each flower
pred = [species[np.argmax(Prob[i, :])]
 for i in range(X_test.shape[0])]
print(" Test Accuracy ", accuracy_score(y_test, pred) * 100 , '%')
```

Output/Results snippet:

```
Test Accuracy  96.66666666666667 %
```

References:

- https://www.pluralsight.com/guides/designing-a-machine-learning-model
- https://realpython.com/logistic-regression-python/
- https://randerson112358.medium.com/python-logistic-regression-program-5e1b32f964db

Activity 3

Aim: Write a code for predicting probability of classification models

Learning outcome: Able to implement predicting probability of classification models.

Duration: 3 hours

List of Hardware/Software requirements:

1. Anaconda
2. Windows/Linux

Code/Program/Procedure (with comments):

```
from sklearn.datasets import load_breast_cancer
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn import metrics
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt
import seaborn as sns
sns.set_style('darkgrid')
# Choose a binary classification problem
data = load_breast_cancer()
# Develop predictors X and target y dataframes
X = pd.DataFrame(data['data'], columns=data['feature_names'])
y = abs(pd.Series(data['target'])-1)
# Split data into train and test set in 80:20 ratio
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.2, random_state=1)

 # Build a RF model with default parameters
```

```
model = LogisticRegression(random_state=1)
model.fit(X_train, y_train)
preds = model.predict(X_test)
accuracy = metrics.accuracy_score(y_test, preds)
accuracy
```

Output/Results snippet:

```
0.9473684210526315
```

References:

- https://analyticsindiamag.com/evaluation-metrics-in-ml-ai-for-classification-problems-wpython-code/

Activity 4

Aim: Write a code for charting confusion matrix

Learning outcome: Able to chart confusion matrix.

Duration: 3 hours

List of Hardware/Software requirements:

1. Anaconda
2. Windows/Linux

Code/Program/Procedure (with comments):

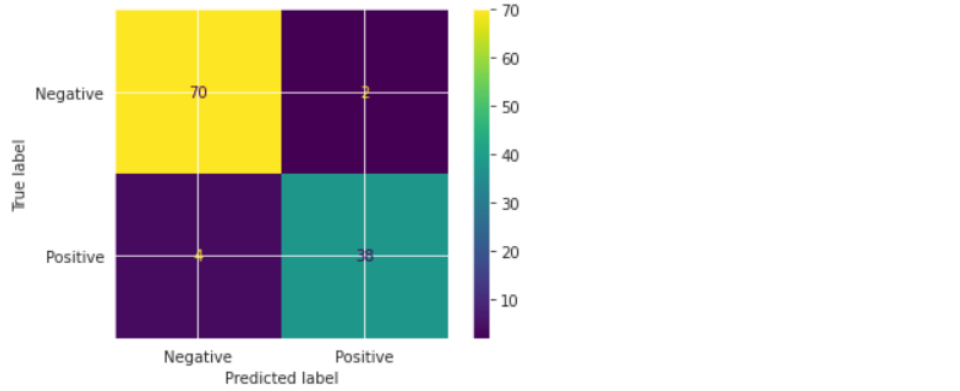
```
from sklearn.datasets import load_breast_cancer
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn import metrics
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt
import seaborn as sns
sns.set_style('darkgrid')
# Choose a binary classification problem
data = load_breast_cancer()
 # Develop predictors X and target y dataframes
X = pd.DataFrame(data['data'], columns=data['feature_names'])
y = abs(pd.Series(data['target'])-1)
 # Split data into train and test set in 80:20 ratio
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.2, random_state=1)
 # Build a RF model with default parameters
```

```
model = LogisticRegression(random_state=1)
model.fit(X_train, y_train)
preds = model.predict(X_test)
accuracy = metrics.accuracy_score(y_test, preds)
accuracy
metrics.plot_confusion_matrix(model, X_test, y_test, display_labels=['Negative', 'Positive'])
precision_positive = metrics.precision_score(y_test, preds, pos_label=1)
precision_negative = metrics.precision_score(y_test, preds, pos_label=0)
precision_positive, precision_negative
recall_sensitivity = metrics.recall_score(y_test, preds, pos_label=1)
recall_specificity = metrics.recall_score(y_test, preds, pos_label=0)
recall_sensitivity, recall_specificity
f1_positive = metrics.f1_score(y_test, preds, pos_label=1)
f1_negative = metrics.f1_score(y_test, preds, pos_label=0)
f1_positive, f1_negative
```

Output/Results snippet:

```
0.9473684210526315
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7f30391d0e50>
```



```
(0.95, 0.9459459459459459)

(0.9047619047619048, 0.9722222222222222)

(0.9268292682926829, 0.9589041095890412)
```

References:

- https://analyticsindiamag.com/evaluation-metrics-in-ml-ai-for-classification-problems-wpython-code/

Activity 5

Aim: Write a code for integration of analytics with Flask app

Learning outcome: Able to integration of analytics with Flask app.

Duration: 10 hours

List of Hardware/Software requirements:

1. Anaconda
2. Windows/Linux

Code/Program/Procedure (with comments):

Classification Model (model.py)

```
import pandas as pd
import numpy as np
import seaborn as sns
from sklearn.linear_model import LogisticRegression
data = sns.load_dataset("iris")
data.head()
variety_mappings = {0: 'Setosa', 1: 'Versicolor', 2: 'Virginica'}
data = data.replace(['Setosa', 'Versicolor' , 'Virginica'],[0, 1, 2])
X = data.iloc[:, 0:-1]
y = data.iloc[:, -1]
logreg = LogisticRegression()
logreg.fit(X, y)
def classify(a, b, c, d):
    arr = np.array([a, b, c, d])
    arr = arr.astype(np.float64)
    query = arr.reshape(1, -1)
```

```
    prediction = variety_mappings[logreg.predict(query)[0]]
    return prediction
```

HTML webpage (home.html)

```
<!DOCTYPE html>
<html>
<head>
        <meta charset="utf-8">
        <meta name="viewport" content="width=device-width, initial-scale=1.0">
        <title>Flower Variety</title>
        <link rel="stylesheet"
 href="https://cdnjs.cloudflare.com/ajax/libs/bulma/0.9.0/css/bulma.min.css">
        <link href="https://fonts.googleapis.com/icon?family=Material+Icons"
rel="stylesheet">
    <style>
        html{
            overflow: hidden;
        }

        body{
            position: absolute;
            width: 100%;
            height: 100%;
            margin: 0;
            padding: 0;
        }
```

```
        #login-form-container{
            position: absolute;
            width: 100%;
            height: 100%;
            display: flex;
            align-items: center;
            justify-content: center;
        }
    </style>
</head>
<body>
    <div id="login-form-container">
        <form action="classify" method="GET">
            <div class="card" style="width: 400px">
            <div class="card-content">
                <div class="media">
                <div class="is-size-4 has-text-centered">Flower Variety Classification</div>
                </div>
                <div class="content">


                <div class="field">
                    <p class="control">
                    Sepal Length: <input class="input" type="number" value='0.00' step='0.01'
name="slen" id="slen">
                    </p>
                </div>
```

```
            <div class="field">
              <p class="control">
              Sepal Width: <input class="input" type="number" value='0.00' step='0.01'
name="swid" id="swid">
              </p>
            </div>

            <div class="field">
              <p class="control">
              Petal Length: <input class="input" type="number" value='0.00' step='0.01'
name="plen" id="plen">
              </p>
            </div>

            <div class="field">
              <p class="control">
              Petal Width: <input class="input" type="number" value='0.00' step='0.01'
name="pwid" id="pwid">
              </p>
            </div>

            <div class="field">
              <button class="button is-fullwidth is-rounded is-success">Submit</button>
            </div>
            </div>
          </div>
        </form>
```

```
    </div>
</body>
</html>
```

output.html

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <meta name="viewport" content="width=device-width, initial-scale=1.0">
        <title>Flower Variety</title>
        <link rel="stylesheet"
href="https://cdnjs.cloudflare.com/ajax/libs/bulma/0.9.0/css/bulma.min.css">
        <link href="https://fonts.googleapis.com/icon?family=Material+Icons" rel="stylesheet">
        <style>
            html{
                overflow: hidden;
            }

            body{
                position: absolute;
                width: 100%;
                height: 100%;
                margin: 0;
                padding: 0;
            }
```

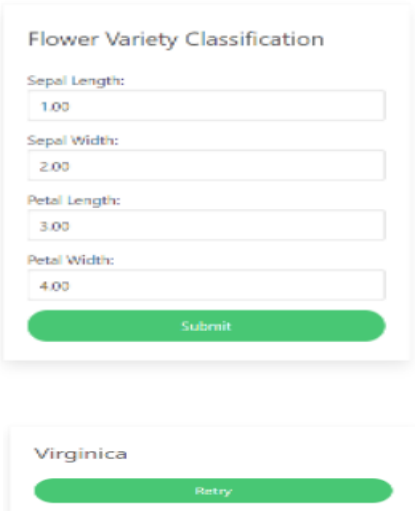
```
        #login-form-container{
            position: absolute;
            width: 100%;
            height: 100%;
            display: flex;
            align-items: center;
            justify-content: center;
        }
    </style>
</head>
<body>
    <div id="login-form-container">
        <div class="card" style="width: 400px">
            <div class="card-content">
                <div class="media">
                    <div class="is-size-4 has-text-centered">{{ variety }}</div>
                </div>
                <form action="home">
                    <div class="field">
                        <button class="button is-fullwidth is-rounded is-success">Retry</button>
                    </div>
                </form>
            </div>
        </div>
    </div>
```

```
    </body>
</html>
```

Flask Framework (server.py)

```
import model
from flask import Flask, request, render_template,jsonify
app = Flask(__name__,template_folder="templates")
# Default route set as 'home'
@app.route('/home')
def home():
    return render_template('home.html') # Render home.html
# Route 'classify' accepts GET request
@app.route('/classify',methods=['POST','GET'])
def classify_type():
    try:
        sepal_len = request.args.get('slen')
        sepal_wid = request.args.get('swid')
        petal_len = request.args.get('plen')
        petal_wid = request.args.get('pwid')
        variety = model.classify(sepal_len, sepal_wid, petal_len, petal_wid)
        return render_template('output.html', variety=variety)
    except:
        return 'Error'
if(__name__=='__main__'):
                    app.run(debug=True)
```

Output/Results snippet:



References:

- https://www.section.io/engineering-education/deploying-machine-learning-models-using-flask/

Learning Outcome

After completing this module, the student should be **able to understand introduction to business.**

To meet the learning outcome, a student has to complete the following activities

1. Creating linear regression model in python
2. Evaluating linear regression model
3. Performing minmax scaling and standard scaling
4. Implementing KNN in python using sklearn
5. Evaluation of KNN model in python, and visualizing results
6. Evaluating model using AUC, ROC curve

Activity 1

Aim: Creating Linear Regression model in Python

Learning outcome: Able to understand basic computer network technology.

Duration: 5 hours

List of Hardware/Software requirements:

1. Anaconda
2. Windows/Linux

Code/Program/Procedure (with comments):

The steps involved are:

1. Importing the dataset.

The first and foremost thing we need to do is import the dataset. We have various websites which have these datasets to be used by anyone.

```
!wget 'https://archive.org/download/ages-and-heights/AgesAndHeights.pkl'
```

This single line of code helps us fetch the data used for the tutorial from the URL directly.

2. Visualising the Data

In this step after importing the data and mounting it with Colab let's have an overview of the dataset by importing a Module called pandas. Since the dataset we have has an extension of .pkl we just view it by the function available in the pandas library.

```
import pandas as pd

raw_data = pd.read_pickle('AgesAndHeights.pkl')

raw_data
```

We import the library to read the dataset and store it in a variable called raw_data. We then display the content of raw_data which is in a tabulated format.

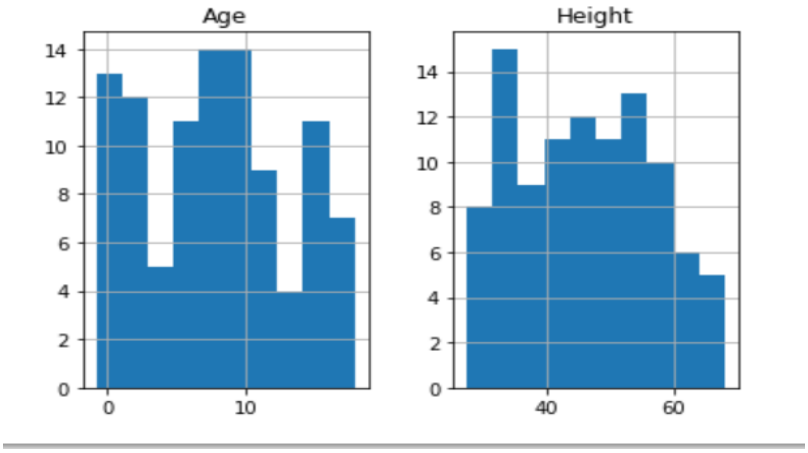
	Age	Height
0	14.767874	59.627484
1	3.107671	36.146453
2	7.266917	46.912878
3	1.815180	29.125660
4	16.753758	68.170414
...	...	...
95	7.323712	46.857505
96	5.591509	39.339990
97	2.625606	32.918925
98	5.519293	40.704154
99	13.117413	55.177407

100 rows × 2 columns

We can see the data which we have and it contains only 2 columns namely, Age (in years) and Height (in inches) and 100 rows which is actually the representation of a person.

```
raw_data.hist()
```

This single line of code has a great impact on the way we look at the dataset. We only had a numerical view of the dataset but we can now run this cell to get a histogram view of the dataset which is very helpful. It represents the data present in the individual columns as individual graphs.



The Y-axis in both the plots refers to frequency and X-axis represents Age and Height respectively.

3. Data Cleaning

We have to build the model using valid datasets and clean the unaccountable Data. In the above image, we can know that there are a few entries that have an age less than zero which is meaningless. Hence, we need to clean those data to get better accuracy.

```
cleaned_data = raw_data[raw_data['Age'] > 0]

cleaned_data
```

I use variable cleaned_data to store the valid age values and display them to the user.

	Age	Height
0	14.767874	59.627484
1	3.107671	36.146453
2	7.266917	46.912878
3	1.815180	29.125660
4	16.753758	68.170414
...	...	...
95	7.323712	46.857505
96	5.591509	39.339990
97	2.625606	32.918925
98	5.519293	40.704154
99	13.117413	55.177407

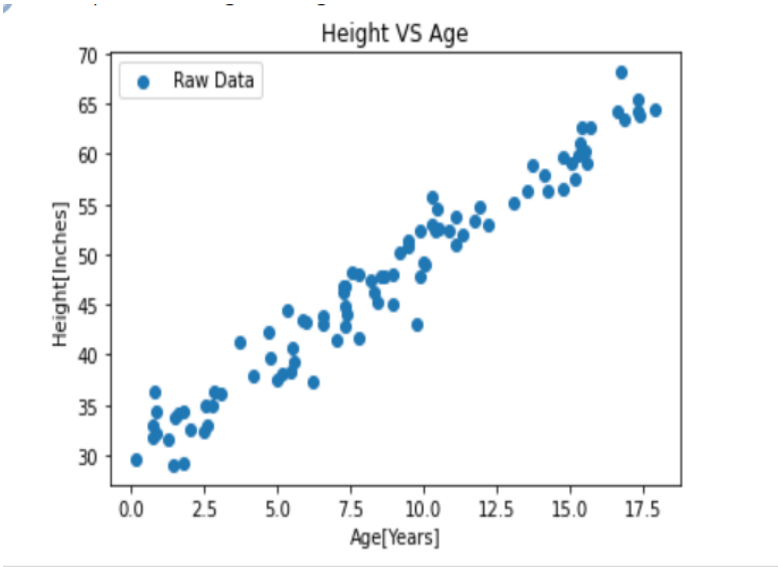
93 rows × 2 columns

Initially, we had 100 rows but after performing Data Cleaning it's pretty clear that there are seven rows which we had age < 0 and we have removed them. As a professional, we aren't supposed to delete the data as we are reducing the data and thereby accuracy of our model gets reduced. To keep it simple I have just removed them.

Visualize the Cleaned Data: I have now used the cleaned data and visualized it in the form of a graph.

```
import matplotlib.pyplot as plt

ages = cleaned_data['Age']

heights = cleaned_data['Height']

plt.scatter(ages,heights, label='Raw Data')

plt.title('Height VS Age')

plt.xlabel('Age[Years]')

plt.ylabel('Height[Inches]')

plt.legend()
```

To plot graphs in python I import matplotlib.pyplot library. I represent Age on X-axis and Height on Y-axis. The points in the plot refer to the Raw data.



3. Build the Model and Train it

This is where the ML Algorithm i.e. Simple Linear Regression comes into play.

```
def y_hat(age, params):
    alpha = params['alpha']
    beta = params['beta']
    return alpha + beta * age
age = int(input('Enter age: '))
y_hat(age, parameters)
```

```
Enter age: 6
64
```

I used a dictionary named parameters which has alpha and beta as key with 40 and 4 as values respectively. I have also defined a function y_hat which takes age, and params as parameters. This function uses the basic straight-line equation and returns y i.e. height as in our case. If we pass the required parameters and run the function, we find that the height we get for the age as input is not matched. Hence, we use the function mentioned below to rain the model.

```
def learn_parameters(data, params):
    x, y = data['Age'], data['Height']
    x_bar, y_bar = x.mean(), y.mean()
    x, y = x.to_numpy(), y.to_numpy()
    beta = sum( ((x-x_bar) * (y-y_bar)) / sum( (x-x_bar)**2))
    alpha = y_bar - beta * x_bar
    params['alpha'] = alpha
    params['beta'] = beta
```

```
new_parameter = {'alpha' : -2, 'beta' : 1000}
learn_parameters(cleaned_data, new_parameter)
new_parameter
```

This is where we use a method to find the correct alpha and beta. The function learn_parameters takes cleaned_data and a dummy dictionary new_parameter which can have any value for alpha and beta. So, when we pass them as arguments to parameters and function runs and we can get the correct value of alpha and beta which is found to close to 30 and 2 respectively and replace the old values with the new ones.

```
new_parameter = {'alpha' : -2, 'beta' : 1000}
learn_parameters(cleaned_data, new_parameter)
new_parameter
```

```
{'alpha': 29.961857614615834, 'beta': 2.0014168989106302}
```

We have accurately found the values of alpha and beta, and our next goal is to train the data. But let me the untrained predicted values to what extent they are accurate.

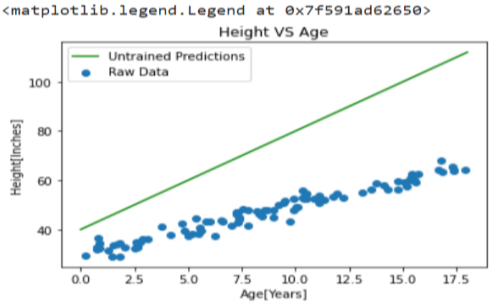
```
spaced_ages = list(range(19))
spaced_untrained_predictions = [y_hat(x, parameters) for x in spaced_ages]
print(spaced_untrained_predictions)
```

```
[40, 44, 48, 52, 56, 60, 64, 68, 72, 76, 80, 84, 88, 92, 96, 100, 104, 108, 112]
```

I use a list named spaces_ages that has values from 0 to 18 (end – 1). Then another list named spaced_untrained_predictions that has the predicted values for the height uses the y_hat function defined earlier to predict it. These values are plotted in a graph and visualized.

```
ages = cleaned_data['Age']
heights = cleaned_data['Height']
plt.scatter(ages,heights, label='Raw Data')
plt.plot(spaced_ages, spaced_untrained_predictions, label = 'Untrained Predictions', color = 'green')
plt.title('Height VS Age')
plt.xlabel('Age[Years]')
plt.ylabel('Height[Inches]')
plt.legend()
plt.show()
```

```
<matplotlib.legend.Legend at 0x7f591ad62650>
```



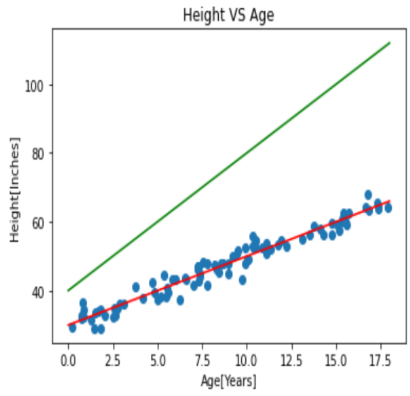
The green line shows that the spaced_untrained_predictions have largely deviated from the actual values and the accuracy is very poor. Hence, accuracy needs to be increased for which we need to train the data.

Plotting Spaced Trained Predicted Values

```
[ ] spaced_trained_predictions = [y_hat(x, new_parameter) for x in spaced_ages]
    print('Trained Predicted Values: ',spaced_trained_predictions)
    plt.scatter(ages,heights, label='Raw Data')
    plt.plot(spaced_ages, spaced_untrained_predictions, label = 'Untrained Predictions', color = 'green')
    plt.plot(spaced_ages, spaced_trained_predictions, label = 'Trained Predictions', color = 'red')
    plt.title('Height VS Age')
    plt.xlabel('Age[Years]')
    plt.ylabel('Height[Inches]')
    plt.legend()
    plt.show()
```

So instead of using parameters, we use new_parameters as it contains the accurate value of alpha and beta and stores it in a list named spaced_trained_predictions. So, when we plot a graph for this, we can see a visible difference and the accuracy has increased a lot. Therefore, we have successfully built and trained the model. Proof for that is the values of spaced_trained_predictions and the graph.

```
Trained Predicted Values:  [29.961857614615834, 31.963274513526464, 33.96469141243709, 35.966108311347725, 37.96752521025836, 39.968942109168985
Text(0, 0.5, 'Height[Inches]')
```



The Greenline refers to the values of spaced_untrained_predictions and Redline refers to the values of spaced_trained_predictions.

4. Make Predictions on Unseen Data

With the help of this trained model, we can now make accurate predictions.

```
new_age = int(input('Enter age to predict height: '))
y_hat(new_age, new_parameter)
```

```
Enter age to predict height: 40
110.01853357104105
```

So, we can see for any given age we find the possible height in inches. Finally, we have successfully trained the model and with utmost accuracy.

Activity 2

Aim: Evaluating Linear Regression model.

Learning outcome: Able to understand basic computer network technology.

Duration: 5 hours

List of Hardware/Software requirements:

1. Anaconda
2. Windows/Linux

Code/Program/Procedure (with comments):

Linear Regression with Python Scikit Learn

In this section we will see how the Python Scikit-Learn library for machine learning can be used to implement regression functions. We will start with simple linear regression involving two variables and then we will move towards linear regression involving multiple variables.

Simple Linear Regression

In this regression task we will predict the percentage of marks that a student is expected to score based upon the number of hours they studied. This is a simple linear regression task as it involves just two variables.

Importing Libraries

To import necessary libraries for this task, execute the following import statements:

```
import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

%matplotlib inline
```

Dataset

The dataset being used for this example has been made publicly available and can be downloaded from this link:

https://drive.google.com/open?id=1oakZCv7g3mlmCSdv9J8kdSaqO5_6dIOw

Note: This example was executed on a Windows based machine and the dataset was stored in "D:\datasets" folder. You can download the file in a different location as long as you change the dataset path accordingly.

The following command imports the CSV dataset using pandas:

```
dataset = pd.read_csv('D:\Datasets\student_scores.csv')
```

Now let's explore our dataset a bit. To do so, execute the following script:

```
dataset.shape
```

After doing this, you should see the following printed out:

```
(25, 2)
```

This means that our dataset has 25 rows and 2 columns. Let's take a look at what our dataset actually looks like. To do this, use the head() method:

```
dataset.head()
```

The above method retrieves the first 5 records from our dataset, which will look like this:

	Hours	Scores
0	2.5	21
1	5.1	47
2	3.2	27
3	8.5	75
4	3.5	30

To see statistical details of the dataset, we can use describe():

```
dataset.describe()
```

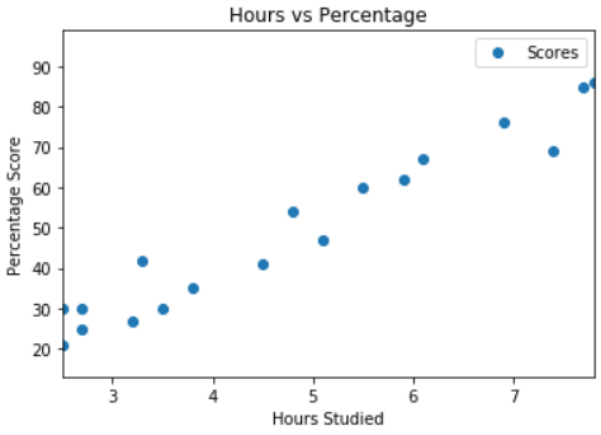
	Hours	Scores
count	25.000000	25.000000
mean	5.012000	51.480000
std	2.525094	25.286887
min	1.100000	17.000000
25%	2.700000	30.000000
50%	4.800000	47.000000
75%	7.400000	75.000000
max	9.200000	95.000000

And finally, let's plot our data points on 2-D graph to eyeball our dataset and see if we can manually find any relationship between the data. We can create the plot with the following script:

```
dataset.plot(x='Hours', y='Scores', style='o')

plt.title('Hours vs Percentage')

plt.xlabel('Hours Studied')

plt.ylabel('Percentage Score')

plt.show()
```

In the script above, we use plot() function of the pandas dataframe and pass it the column names for x coordinate and y coordinate, which are "Hours" and "Scores" respectively.

The resulting plot will look like this:



From the graph above, we can clearly see that there is a positive linear relation between the number of hours studied and percentage of score.

Preparing the Data

Now we have an idea about statistical details of our data. The next step is to divide the data into "attributes" and "labels". Attributes are the independent variables while labels are dependent variables whose values are to be predicted. In our dataset we only have two columns. We want to predict the percentage score depending upon the hours studied. Therefore our attribute set will consist of the "Hours" column, and the label will be the "Score" column. To extract the attributes and labels, execute the following script:

```
X = dataset.iloc[:, :-1].values

y = dataset.iloc[:, 1].values
```

The attributes are stored in the X variable. We specified "-1" as the range for columns since we wanted our attribute set to contain all the columns except the last one, which is "Scores". Similarly the y variable contains the labels. We specified 1 for the label column since the index for "Scores" column is 1. Remember, the column indexes start with 0, with 1 being the second column. In the next section, we will see a better way to specify columns for attributes and labels.

Now that we have our attributes and labels, the next step is to split this data into training and test sets. We'll do this by using Scikit-Learn's built-in train_test_split() method:

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)
```

The above script splits 80% of the data to training set while 20% of the data to test set. The test_size variable is where we actually specify the proportion of test set.

Training the Algorithm

We have split our data into training and testing sets, and now is finally the time to train our algorithm. Execute following command:

```
from sklearn.linear_model import LinearRegression

regressor = LinearRegression()

regressor.fit(X_train, y_train)
```

With Scikit-Learn it is extremely straight forward to implement linear regression models, as all you really need to do is import the LinearRegression class, instantiate it, and call the fit() method along with our training data. This is about as simple as it gets when using a machine learning library to train on your data.

In the theory section we said that linear regression model basically finds the best value for the intercept and slope, which results in a line that best fits the data. To see the value of the intercept and slop calculated by the linear regression algorithm for our dataset, execute the following code.

To retrieve the intercept:

```
print(regressor.intercept_)
```

The resulting value you see should be approximately 2.01816004143.

For retrieving the slope (coefficient of x):

```
print(regressor.coef_)
```

The result should be approximately 9.91065648.

This means that for every one unit of change in hours studied, the change in the score is about 9.91%. Or in simpler words, if a student studies one hour more than they previously studied for an exam, they can expect to achieve an increase of 9.91% in the score achieved by the student previously.

Making Predictions

Now that we have trained our algorithm, it's time to make some predictions. To do so, we will use our test data and see how accurately our algorithm predicts the percentage score. To make pre-dictions on the test data, execute the following script:

```
y_pred = regressor.predict(X_test)
```

The y_pred is a numpy array that contains all the predicted values for the input values in the X_test series.

To compare the actual output values for X_test with the predicted values, execute the following script:

```
df = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred})

df
```

The output looks like this:

	Actual	Predicted
0	20	16.884145
1	27	33.732261
2	69	75.357018
3	30	26.794801
4	62	60.491033

Though our model is not very precise, the predicted percentages are close to the actual ones.

Note: The values in the columns above may be different in your case because the train_test_split function randomly splits data into train and test sets, and your splits are likely different from the one shown in this article.

Evaluating the Algorithm

The final step is to evaluate the performance of algorithm. This step is particularly important to compare how well different algorithms perform on a particular dataset. For regression algorithms, three evaluation metrics are commonly used:

1. Mean Absolute Error (MAE) is the mean of the absolute value of the errors. It is calculated as:

$$\frac{1}{n}\sum_{i=1}^{n}|Actual - Predicted|$$

2. Mean Squared Error (MSE) is the mean of the squared errors and is calculated as:

$$\frac{1}{n}\sum_{i=1}^{n}|Actual - Predicted|^2$$

3. Root Mean Squared Error (RMSE) is the square root of the mean of the squared errors:

$$\sqrt{\frac{1}{n}\sum_{i=1}^{n}|Actual - Predicted|^2}$$

we don't have to perform these calculations manually. The Scikit-Learn library comes with pre-built functions that can be used to find out these values for us.

Let's find the values for these metrics using our test data. Execute the following code:

```
from sklearn import metrics

print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, y_pred))

print('Mean Squared Error:', metrics.mean_squared_error(y_test, y_pred))

print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
```

Output/Results snippet:

The output will look similar to this (but probably slightly different):

```
Mean Absolute Error: 4.183859899

Mean Squared Error: 21.5987693072

Root Mean Squared Error: 4.6474476121
```

You can see that the value of root mean squared error is 4.64, which is less than 10% of the mean value of the percentages of all the students i.e. 51.48. This means that our algorithm did a decent job.

Activity 3

Aim: Performing minmax scaling and standard scaling

Learning outcome: Able to understand basic computer network technology.

Duration: 5 hours

List of Hardware/Software requirements:

1. Anaconda
2. Windows/Linux

Code/Program/Procedure (with comments):

StandardScaler follows Standard Normal Distribution (SND). Therefore, it makes mean = 0 and scales the data to unit variance.

MinMaxScaler scales all the data features in the range [0, 1] or else in the range [-1, 1] if there are negative values in the dataset. This scaling compresses all the inliers in the narrow range [0, 0.005].

In the presence of outliers, StandardScaler does not guarantee balanced feature scales, due to the influence of the outliers while computing the empirical mean and standard deviation. This leads to the shrinkage in the range of the feature values.

By using RobustScaler(), we can remove the outliers and then use either StandardScaler or MinMaxScaler for preprocessing the dataset.

How RobustScaler works:

class

sklearn.preprocessing.RobustScaler(

with_centering=True, with_scaling=True,

quantile_range=(25.0, 75.0),

copy=True,

)

It scales features using statistics that are robust to outliers. This method removes the median and scales the data in the range between 1st quartile and 3rd quartile. i.e., in between 25th quantile and 75th quantile range. This range is also called an Interquartile range.

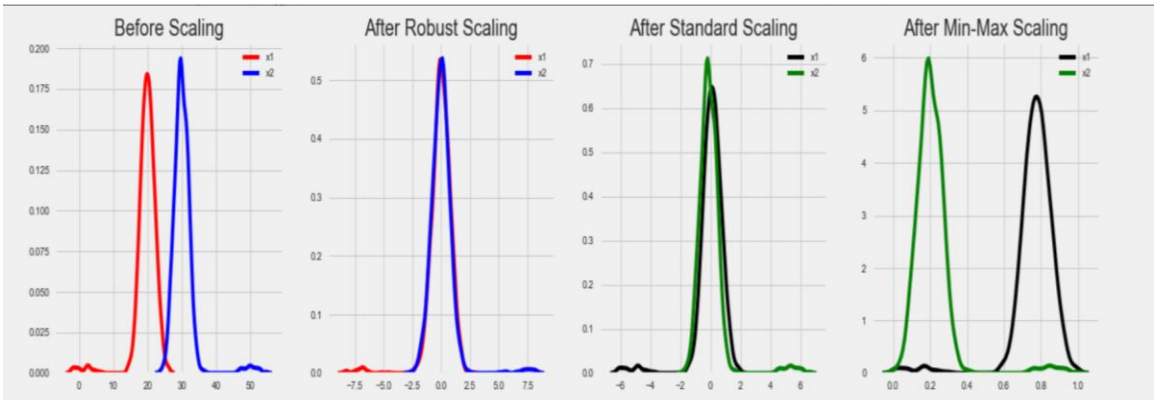
The median and the interquartile range are then stored so that it could be used upon future data using the transform method. If outliers are present in the dataset, then the median and the interquartile range provide better results and outperform the sample mean and variance.

Code: comparison between StandardScaler and MinMaxScaler.

```
# Importing libraries

import pandas as pd

import numpy as np

from sklearn import preprocessing

import matplotlib

import matplotlib.pyplot as plt

import seaborn as sns % matplotlib inline

matplotlib.style.use('fivethirtyeight')

# data

x = pd.DataFrame({

        # Distribution with lower outliers

        'x1': np.concatenate([np.random.normal(20, 2, 1000), np.random.normal(1, 2, 25)]),

        # Distribution with higher outliers

        'x2': np.concatenate([np.random.normal(30, 2, 1000), np.random.normal(50, 2, 25)]),

})

np.random.normal

scaler = preprocessing.RobustScaler()
```

```
robust_df = scaler.fit_transform(x)
robust_df = pd.DataFrame(robust_df, columns =['x1', 'x2'])
scaler = preprocessing.StandardScaler()
standard_df = scaler.fit_transform(x)
standard_df = pd.DataFrame(standard_df, columns =['x1', 'x2'])
scaler = preprocessing.MinMaxScaler()
minmax_df = scaler.fit_transform(x)
minmax_df = pd.DataFrame(minmax_df, columns =['x1', 'x2'])
fig, (ax1, ax2, ax3, ax4) = plt.subplots(ncols = 4, figsize =(20, 5))
ax1.set_title('Before Scaling')
sns.kdeplot(x['x1'], ax = ax1, color ='r')
sns.kdeplot(x['x2'], ax = ax1, color ='b')
ax2.set_title('After Robust Scaling')
sns.kdeplot(robust_df['x1'], ax = ax2, color ='red')
sns.kdeplot(robust_df['x2'], ax = ax2, color ='blue')
ax3.set_title('After Standard Scaling')
sns.kdeplot(standard_df['x1'], ax = ax3, color ='black')
sns.kdeplot(standard_df['x2'], ax = ax3, color ='g')
ax4.set_title('After Min-Max Scaling')
sns.kdeplot(minmax_df['x1'], ax = ax4, color ='black')
sns.kdeplot(minmax_df['x2'], ax = ax4, color ='g')
plt.show()
```

Output/Results snippet:



Activity 4

Aim: Implementing KNN in python using sklearn

Learning outcome: Able to understand basic computer network technology.

Duration: 5 hour

List of Hardware/Software requirements:

1. Anaconda
2. Windows/Linux

Code/Program/Procedure (with comments):

Python's Scikit-Learn library can be used to implement the KNN algorithm in less than 20 lines of code. The download and installation instructions for Scikit learn library are available at here.

Note: The code provided in this tutorial has been executed and tested with Python Jupyter notebook.

The Dataset

We are going to use the famous iris data set for our KNN example. The dataset consists of four attributes: sepal-width, sepal-length, petal-width and petal-length. These are the attributes of specific types of iris plant. The task is to predict the class to which these plants belong. There are three classes in the dataset: Iris-setosa, Iris-versicolor and Iris-virginica. Further details of the dataset are available here.

Importing Libraries

```
import numpy as np

import matplotlib.pyplot as plt

import pandas as pd
```

Importing the Dataset

To import the dataset and load it into our pandas dataframe, execute the following code:

```
url = "https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data"


# Assign colum names to the dataset

names = ['sepal-length', 'sepal-width', 'petal-length', 'petal-width', 'Class']


# Read dataset to pandas dataframe

dataset = pd.read_csv(url, names=names)
```

To see what the dataset actually looks like, execute the following command:

```
dataset.head()
```

Executing the above script will display the first five rows of our dataset as shown below:

	sepal-length	sepal-width	petal-length	petal-width	Class
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa

Preprocessing

The next step is to split our dataset into its attributes and labels. To do so, use the following code:

```
X = dataset.iloc[:, :-1].values

y = dataset.iloc[:, 4].values
```

The X variable contains the first four columns of the dataset (i.e. attributes) while y contains the labels.

Train Test Split

To avoid over-fitting, we will divide our dataset into training and test splits, which gives us a better idea as to how our algorithm performed during the testing phase. This way our algorithm is tested on un-seen data, as it would be in a production application.

To create training and test splits, execute the following script:

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20)
```

The above script splits the dataset into 80% train data and 20% test data. This means that out of total 150 records, the training set will contain 120 records and the test set contains 30 of those records.

Feature Scaling

Before making any actual predictions, it is always a good practice to scale the features so that all of them can be uniformly evaluated.

The gradient descent algorithm (which is used in neural network training and other machine learning algorithms) also converges faster with normalized features.

The following script performs feature scaling:

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

scaler.fit(X_train)


X_train = scaler.transform(X_train)

X_test = scaler.transform(X_test)
```

Training and Predictions

It is extremely straight forward to train the KNN algorithm and make predictions with it, especially when using Scikit-Learn.

```
from sklearn.neighbors import KNeighborsClassifier

classifier = KNeighborsClassifier(n_neighbors=5)

classifier.fit(X_train, y_train)
```

The first step is to import the KNeighborsClassifier class from the sklearn.neighbors library. In the second line, this class is initialized with one parameter, i.e. n_neigbours. This is basically the value for the K. There is no ideal value for K and it is selected after testing and evaluation, however to start out, 5 seems to be the most commonly used value for KNN algorithm.

The final step is to make predictions on our test data. To do so, execute the following script:

```
y_pred = classifier.predict(X_test)
```

Evaluating the Algorithm

For evaluating an algorithm, confusion matrix, precision, recall and f1 score are the most commonly used metrics. The confusion_matrix and classification_report methods of the sklearn.metrics can be used to calculate these metrics. Take a look at the following script:

```
from sklearn.metrics import classification_report, confusion_matrix

print(confusion_matrix(y_test, y_pred))

print(classification_report(y_test, y_pred))
```

The **output** of the above script looks like this:

```
[[11  0  0]
 [ 0 13  0]
 [ 0  1  6]]
                precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00        11
Iris-versicolor       1.00      1.00      1.00        13
 Iris-virginica       1.00      1.00      1.00         6

    avg / total       1.00      1.00      1.00        30
```

The results show that our KNN algorithm was able to classify all the 30 records in the test set with 100% accuracy, which is excellent. Although the algorithm performed very well with this dataset, don't expect the same results with all applications. As noted earlier, KNN doesn't always perform as well with high-dimensionality or categorical features.

Comparing Error Rate with the K Value

In the training and prediction section we said that there is no way to know beforehand which value of K that yields the best results in the first go. We randomly chose 5 as the K value and it just happen to result in 100% accuracy.

One way to help you find the best value of K is to plot the graph of K value and the corresponding error rate for the dataset.

In this section, we will plot the mean error for the predicted values of test set for all the K values between 1 and 40.

To do so, let's first calculate the mean of error for all the predicted values where K ranges from 1 and 40. Execute the following script:

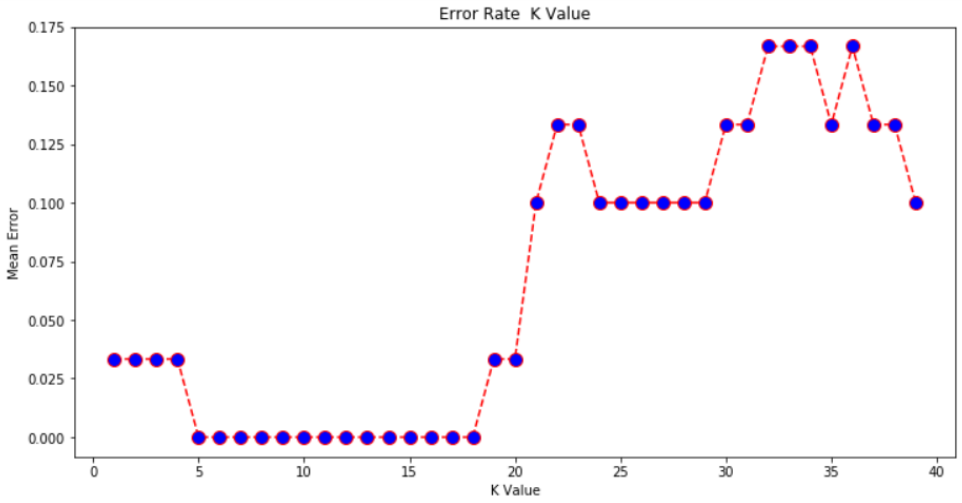
```
error = []


# Calculating error for K values between 1 and 40

for i in range(1, 40):

    knn = KNeighborsClassifier(n_neighbors=i)

    knn.fit(X_train, y_train)

    pred_i = knn.predict(X_test)

    error.append(np.mean(pred_i != y_test))
```

The above script executes a loop from 1 to 40. In each iteration the mean error for predicted values of test set is calculated and the result is appended to the error list.

The next step is to plot the error values against K values. Execute the following script to create the plot:

```
plt.figure(figsize=(12, 6))

plt.plot(range(1, 40), error, color='red', linestyle='dashed', marker='o',

        markerfacecolor='blue', markersize=10)

plt.title('Error Rate K Value')

plt.xlabel('K Value')

plt.ylabel('Mean Error')
```

Output/Results snippet:



From the output we can see that the mean error is zero when the value of the K is between 5 and 18. I would advise you to play around with the value of K to see how it impacts the accuracy of the predictions.

Activity 5

Aim: Evaluation of KNN model in python, and visualizing result.

Learning outcome: Able to understand basic computer network technology.

Duration: 5 hour

List of Hardware/Software requirements:

1. Anaconda
2. Windows/Linux

Code/Program/Procedure (with comments):

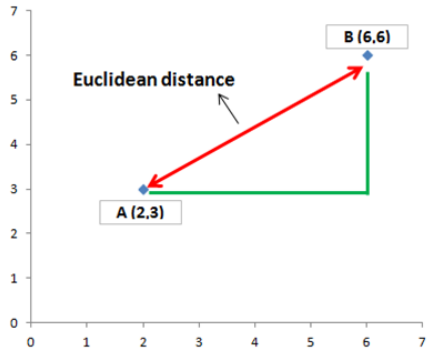
K-nearest neighbors (kNN) is a supervised machine learning algorithm that can be used to solve both classification and regression tasks. I see kNN as an algorithm that comes from real life. People tend to be effected by the people around them. Our behaviour is guided by the friends we grew up with. Our parents also shape our personality in some ways. If you grow up with people who love sports, it is higly likely that you will end up loving sports. There are ofcourse exceptions. kNN works similarly.

The value of a data point is determined by the data points around it.

- If you have one very close friend and spend most of your time with him/her, you will end up sharing similar interests and enjoying same things. That is kNN with k=1.
- If you always hang out with a group of 5, each one in the group has an effect on your behavior and you will end up being the average of 5. That is kNN with k=5.

kNN classifier determines the class of a data point by majority voting principle. If k is set to 5, the classes of 5 closest points are checked. Prediction is done according to the majority class. Similarly, kNN regression takes the mean value of 5 closest points.

We observe people who are close but how data points are determined to be close? The distance between data points is measured. There are many methods to measure the distance. Euclidean distance (minkowski distance with p=2) is one of most commonly used distance measurement. The figure below shows how to calculate euclidean distance between two points in a 2-dimensional space. It is calculated using the square of the difference between x and y coordinates of the points.



$$\textbf{\textit{Euclidean distance}}\ (a, b) = \sqrt{(a_1 - b_1)^2 + (a_2 - b_2)^2}$$

In the case above, euclidean distance is the square root of (16 + 9) which is 5. Euclidean distance in two dimensions remind us the famous pythagorean theorem.

It seems very simple for two points in 2-dimensional space. Each dimension represents a feauture in the dataset. We typically have many samples with many features. To be able to explain the concept clearly, I will go over an example in 2-dimensional space.

Let's start with importing libraries:

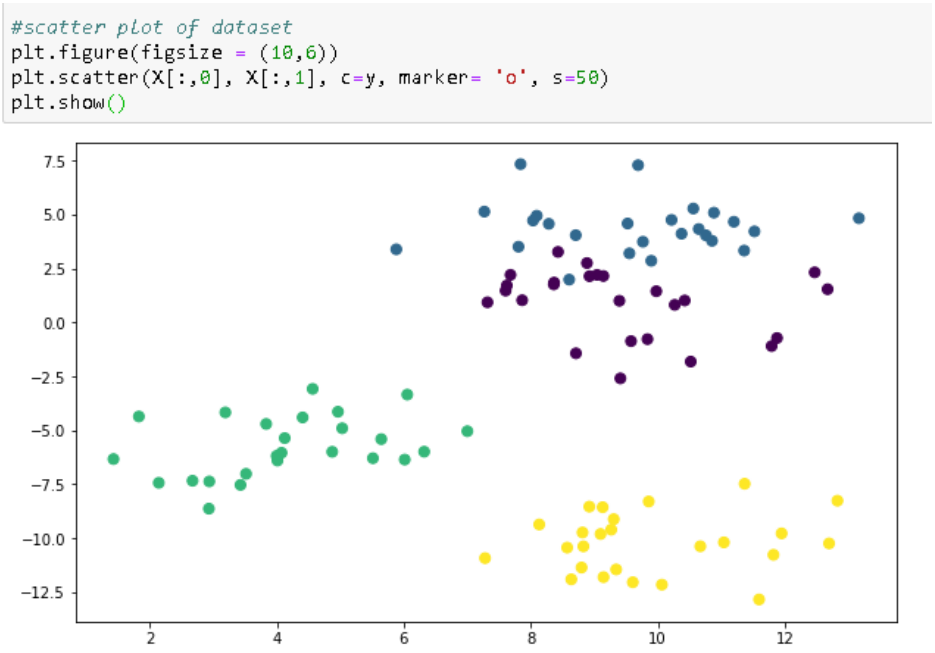
```
import numpy as np
import pandas as pd

import matplotlib.pyplot as plt #data visualization

from sklearn.datasets import make_blobs #synthetic dataset
from sklearn.neighbors import KNeighborsClassifier #kNN classifier
from sklearn.model_selection import train_test_split #train and test sets
```

Scikit-learn provides many useful functions to create synthetic datasets which are very helpful for practicing machine learning algorithms. I will use make_blobs function.

```
#create synthetic dataset
X, y = make_blobs(n_samples = 100, n_features = 2, centers = 4,
                        cluster_std = 1.5, random_state = 4)
```

This code creates a dataset with 100 samples divided into 4 classes and the number of features is 2. Number of samples, features and classes can easily be adjusted using related parameters. We can also adjust how much each cluster (or class) is spread. Let's visualize this synthetic data set:

```
#scatter plot of dataset
plt.figure(figsize = (10,6))
plt.scatter(X[:,0], X[:,1], c=y, marker= 'o', s=50)
plt.show()
```



For any supervised machine learning algorithm, it is very important to divide dataset into train and test sets. We first train the model and test it using different parts of dataset. If this separation is not done, we basically test the model with some data it already knows. We can easily do this separation using train_test_split function.

```
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state = 0)
```

We can specify how much of the original data is used for train or test sets using train_size or test_size parameters, respectively. Default separation is 75% for train set and 25% for test set.

Then we create a kNN classifier object. To show the difference between the importance of k value, I create two classifiers with k values 1 and 5. Then these models are trained using train set. n_neighbors parameter is used to select k value. Default value is 5 so it does not have to be explicitly written.

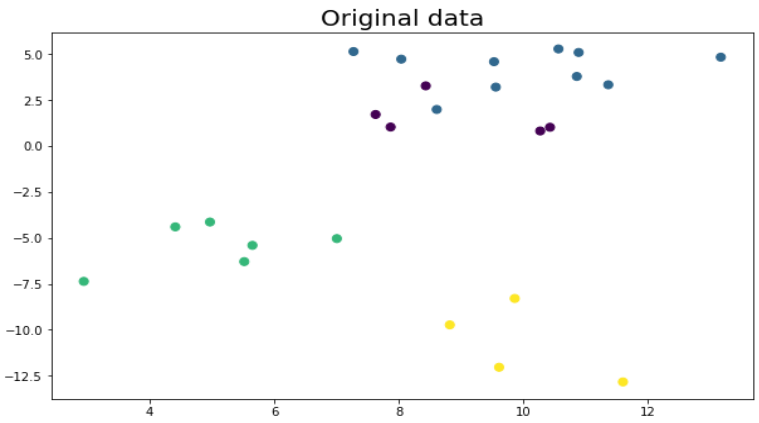
```
# In[72]:


knn5 = KNeighborsClassifier() #k=5
knn1 = KNeighborsClassifier(n_neighbors=1) #k=1


# In[73]:


knn5.fit(X_train, y_train)
knn1.fit(X_train, y_train)
```

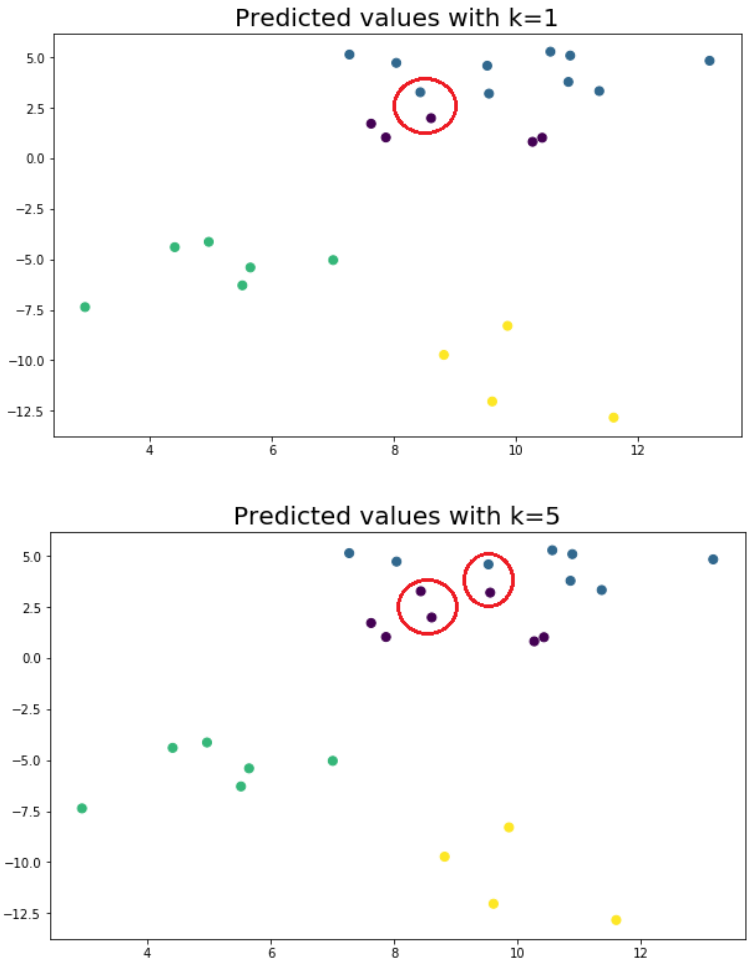
Then we predict the target values in the test set and compare with actual values.

```
y_pred_5 = knn5.predict(X_test)
y_pred_1 = knn1.predict(X_test)
```

In order to see the effect of k values, let's visualize test set and predicted values with k=5 and k=1.

Output/Results snippet:





The result seems to be very similar because we used a substantially small dataset. However, even on small datasets, different k values predict some points differently.

Activity 6

Aim: Evaluating model using AUC, ROC curve.

Learning outcome: Able to understand basic computer network technology.

Duration: 5 hour

List of Hardware/Software requirements:

1. Anaconda
2. Windows/Linux

Code/Program/Procedure (with comments):

Using ROC and AUC in Python

You'll use the White wine quality dataset for the practical part. Here's how to load it with Python:

The first couple of rows look like this:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
0	7.0	0.27	0.36	20.7	0.045	45.0	170.0	1.0010	3.00	0.45	8.8	6
1	6.3	0.30	0.34	1.6	0.049	14.0	132.0	0.9940	3.30	0.49	9.5	6
2	8.1	0.28	0.40	6.9	0.050	30.0	97.0	0.9951	3.26	0.44	10.1	6
3	7.2	0.23	0.32	8.5	0.058	47.0	186.0	0.9956	3.19	0.40	9.9	6
4	7.2	0.23	0.32	8.5	0.058	47.0	186.0	0.9956	3.19	0.40	9.9	6

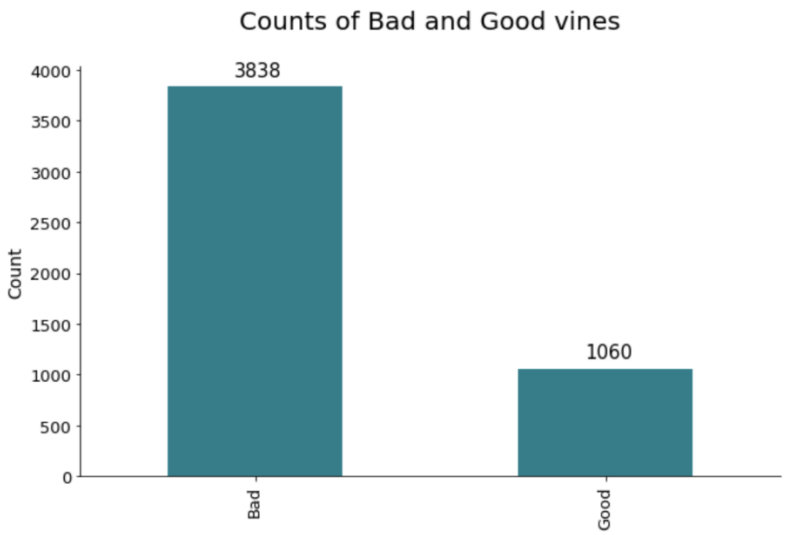
Initially, this is not a binary classification dataset, but you can convert it to one. Let's say the wine is Good if the quality is 7 or above, and Bad otherwise:

```
df['quality'] = ['Good' if quality >= 7 else 'Bad' for quality in df['quality']]
```

There's your binary classification dataset. Let's visualize the counts of good and bad wines next. Here's the code:

```
ax = df['quality'].value_counts().plot(kind='bar', figsize=(10, 6), fontsize=13, color='#087E8B')
ax.set_title('Counts of Bad and Good vines', size=20, pad=30)
ax.set_ylabel('Count', fontsize=14)

for i in ax.patches:
    ax.text(i.get_x() + 0.19, i.get_height() + 100, str(round(i.get_height(), 2)), fontsize=15)
```

And here's the chart:



And there's nothing more to do with regards to preparation. You can make a train/test split next:

```
from sklearn.model_selection import train_test_split
X = df.drop('quality', axis=1)
y = df['quality']

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.25, random_state=42
)
```

Great! The snippet below shows you how to train logistic regression, decision tree, random forests, and extreme gradient boosting models. It also shows you how to grab probabilities for the positive class. It will come in handy later:

```
from sklearn.linear_model import LogisticRegression

from sklearn.tree import DecisionTreeClassifier

from sklearn.ensemble import RandomForestClassifier

from xgboost import XGBClassifier

model_lr = LogisticRegression().fit(X_train, y_train)

probs_lr = model_lr.predict_proba(X_test)[:, 1]

model_dt = DecisionTreeClassifier().fit(X_train, y_train)

probs_dt = model_dt.predict_proba(X_test)[:, 1]

model_rf = RandomForestClassifier().fit(X_train, y_train)

probs_rf = model_rf.predict_proba(X_test)[:, 1]

model_xg = XGBClassifier().fit(X_train, y_train)

probs_xg = model_xg.predict_proba(X_test)[:, 1]
```

You can visualize the ROC curves and calculate the AUC now. The only requirement is to remap the Good and Bad class names to 1 and 0, respectively.

The following code snippet visualizes the ROC curve for the four trained models and shows their AUC score on the legend:

```
from sklearn.metrics import roc_auc_score, roc_curve

y_test_int = y_test.replace({'Good': 1, 'Bad': 0})

auc_lr = roc_auc_score(y_test_int, probs_lr)

fpr_lr, tpr_lr, thresholds_lr = roc_curve(y_test_int, probs_lr)

auc_dt = roc_auc_score(y_test_int, probs_dt)

fpr_dt, tpr_dt, thresholds_dt = roc_curve(y_test_int, probs_dt)
```

```
auc_rf = roc_auc_score(y_test_int, probs_rf)
fpr_rf, tpr_rf, thresholds_rf = roc_curve(y_test_int, probs_rf)
auc_xg = roc_auc_score(y_test_int, probs_xg)
fpr_xg, tpr_xg, thresholds_xg = roc_curve(y_test_int, probs_xg)
plt.figure(figsize=(12, 7))
plt.plot(fpr_lr, tpr_lr, label=f'AUC (Logistic Regression) = {auc_lr:.2f}')
plt.plot(fpr_dt, tpr_dt, label=f'AUC (Decision Tree) = {auc_dt:.2f}')
plt.plot(fpr_rf, tpr_rf, label=f'AUC (Random Forests) = {auc_rf:.2f}')
plt.plot(fpr_xg, tpr_xg, label=f'AUC (XGBoost) = {auc_xg:.2f}')
plt.plot([0, 1], [0, 1], color='blue', linestyle='--', label='Baseline')
plt.title('ROC Curve', size=20)
plt.xlabel('False Positive Rate', size=14)
plt.ylabel('True Positive Rate', size=14)
plt.legend();
```

Output/Results snippet:

Here's the corresponding visualization: