



# Machine Learning with Python

Shadi M. Alkhatib

Esra'a S. Khataibeh

Dania H. Jarad



---

---

## ACKNOWLEDGMENT

**F**irst and foremost, we would like to thank our ALLAH, this book would not have finished without the strength that ALLAH gave ours.

We would also like to thank the team based on this book.

[Shadimalkhatib@gmail.com](mailto:Shadimalkhatib@gmail.com)

[DaniaHJarad@gmail.com](mailto:DaniaHJarad@gmail.com)

[Esra39765@gmail.com](mailto:Esra39765@gmail.com)

---

---

# Contents

Chapter 1: Introduction

Chapter 2: Python Packages (Numpy and Pandas)

Chapter 3: Data Reprocessing

Chapter 4: Regression

Chapter 5: Simple Linear Regression

Chapter 6: Multiple Linear Regression Intuition

Chapter 7: Polynomial Regression

Chapter 8: Support Vector Regression (SVR)

Chapter 9: Decision Tree Regression

Chapter 10: Random Forest Regression

Chapter 11: Evaluating Regression Models Performance

Chapter 12: Classification

Chapter 13: Logistic Regression

Chapter 14: K-Nearest Neighbors (K-NN)

Chapter 15: Support Vector Machine (SVM)

Chapter 16: Kernel SVM

Chapter 17: Naive Bayes

Chapter 18: Decision Tree Classification

Chapter 19: Random Forest Classification

Chapter 20: Evaluating Classification Models  
Performance

Chapter 21: Clustering

Chapter 22: K-Means Clustering

Chapter 23: Hierarchical Clustering

Chapter 24: Association Rule Learning

Chapter 25: Apriori

Chapter 26: Eclat

Chapter 27: Reinforcement Learning

Chapter 28: Upper Confidence Bound (UCB)

Chapter 29: Thompson Sampling

Chapter 30: Natural Language Processing

Chapter 31: Deep Learning

Chapter 32: Artificial Neural Network (ANN)

Chapter 33: Convolutional Neural Networks (CNN)

Chapter 34: Dimensionality Reduction

---

# CHAPTER 1

---

## INTRODUCTION

### 1.1 Overview

Machine learning is a type of artificial intelligence, which allows software applications that become more accurate in predicting results without explicitly programming them.

The primary focus of machine learning is building algorithms that can receive input data, and use statistical analysis; to predict outputs within an acceptable range.

Machine learning algorithms are categorized into:

**Supervised learning, Unsupervised learning, Reinforcement learning.**

The above explain are illustrated in Figure 1.

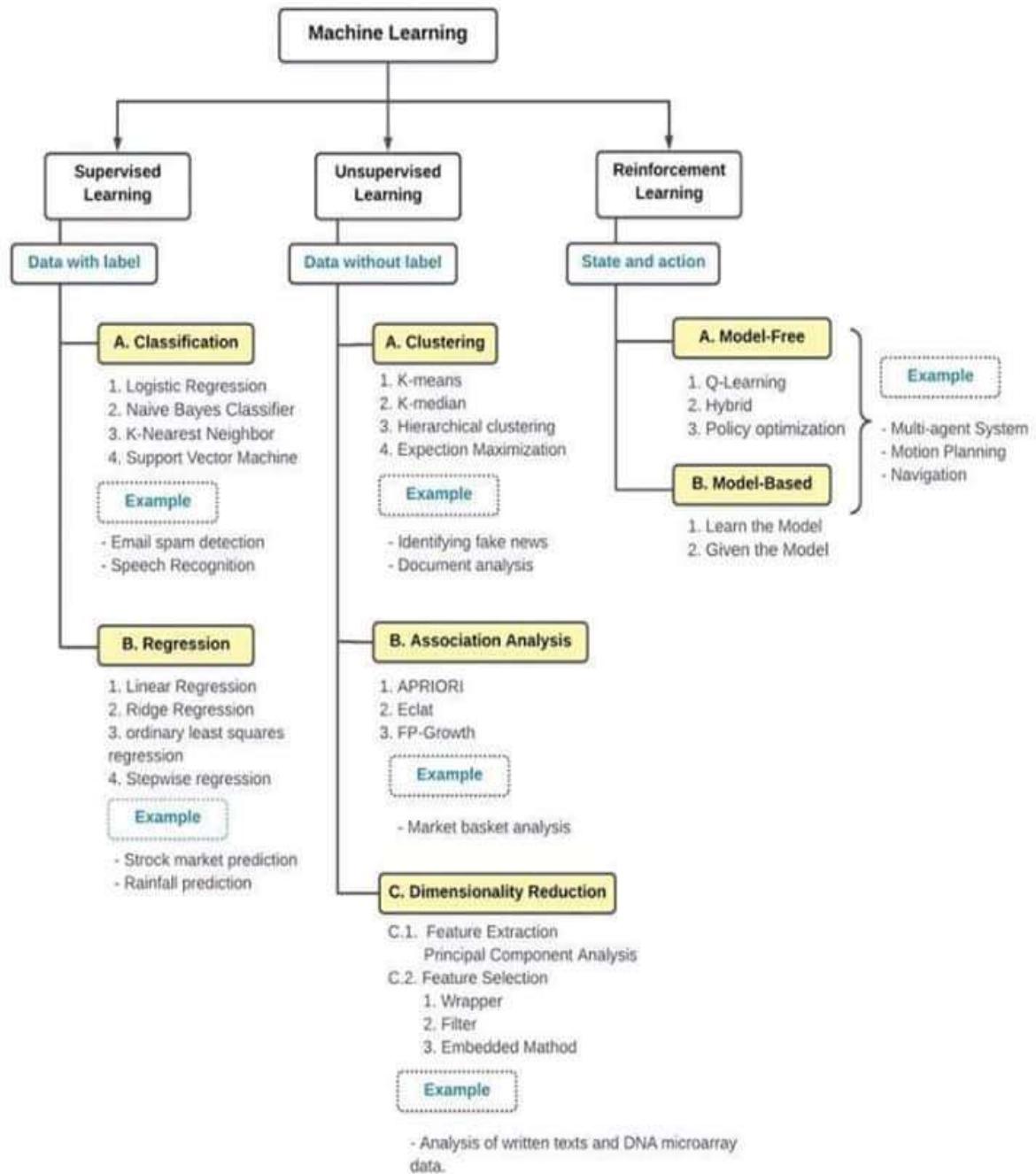


Figure 1.1

## 1.2 Machine learning applications

There are several applications that depend on the machine language

### 1. Virtual Reality (VR)

Computer simulation of environments that can be simulated physically in some places in the real world, where the glasses are worn and direct directions are determined.



Figure 1.2

## 2. Robot dog

Certain algorithms are used that determine the path with great accuracy.



Figure 1.3

## 3. Phone control panel

When an audio clip recording begins, the speech is somewhat literal



Figure 1.4

#### 4. Data science

Especially in the field of medicine where it is known whether the body is healthy or suffering from chronic X-ray diseases.

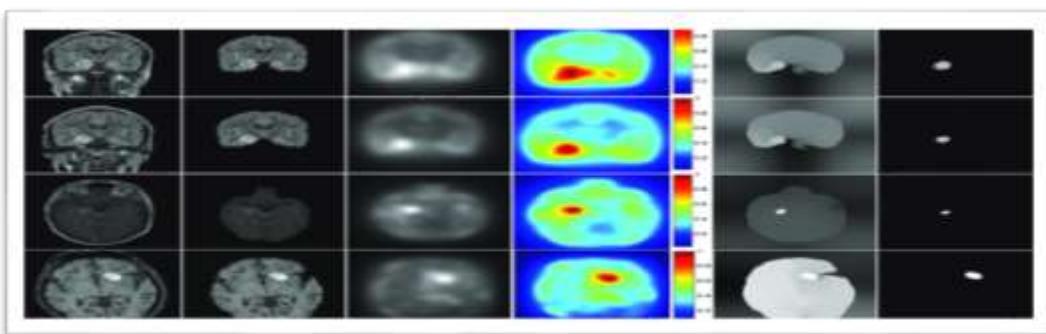


Figure 1.5

## 1.3 Machine learning is the future

The machine language is data dependent. The data over, the more learning possibilities.

In the period 2000-2005, data increased significantly, as there is a huge amount of data, for example, Google and Facebook have **data scientists** in huge quantities all over the world where they are called **big data**.

The figure shows the actual time over which the data scientists increased, as the results of the increase are expected to be in 2020

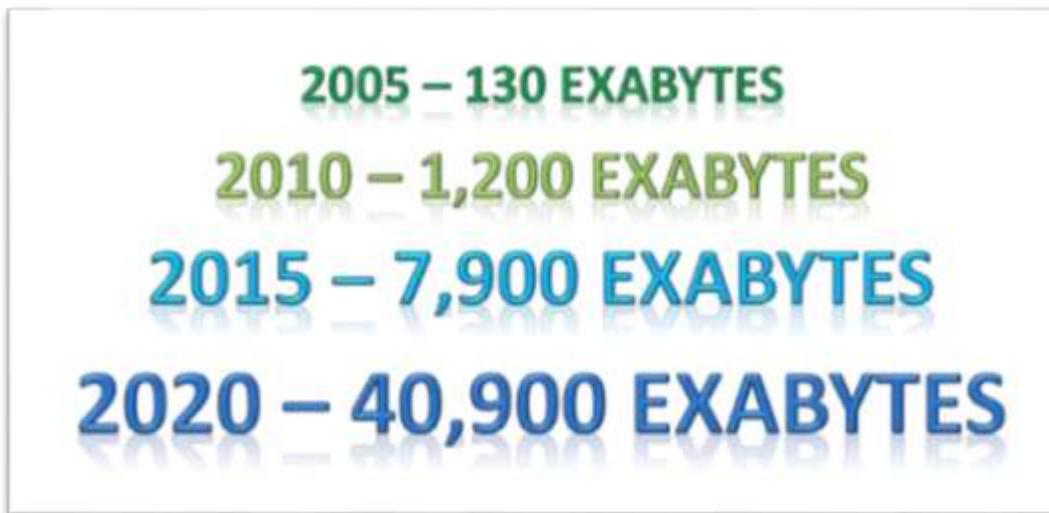


Figure 1.6

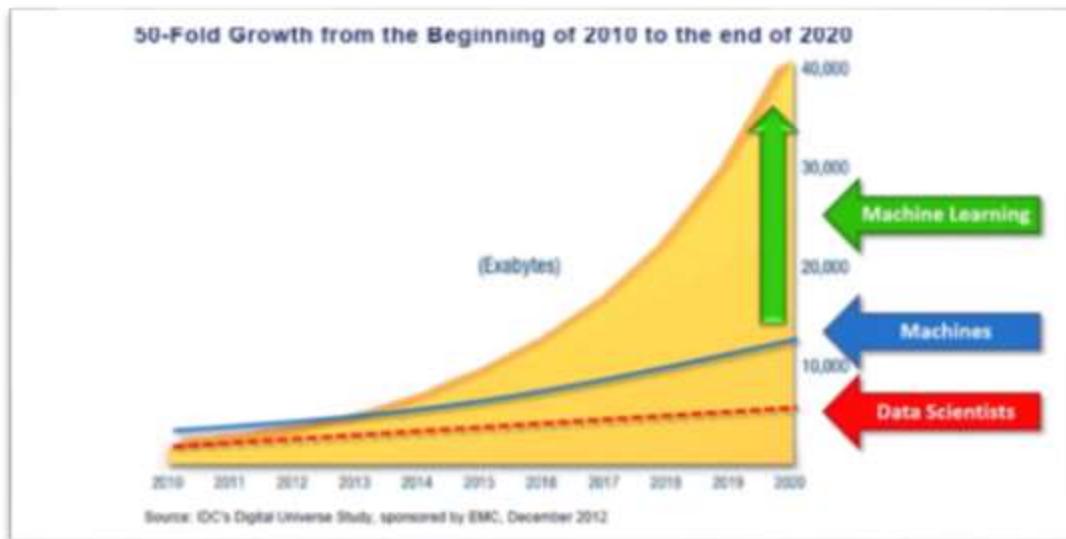


Figure 1.7

Nombre	Simbolo	Potencias binarias y valores decimales
byte	b	$2^0 = 1$
Kbyte	KB	$2^{10} = 1\,024$
Megabyte	MB	$2^{20} = 1\,048\,576$
Gigabyte	GB	$2^{30} = 1\,073\,741\,824$
Terabyte	TB	$2^{40} = 1\,099\,511\,627\,776$
Petabyte	PB	$2^{50} = 1\,125\,899\,906\,842\,624$
Exabyte	EB	$2^{60} = 1\,152\,921\,504\,606\,846\,976$
Zettabyte	ZB	$2^{70} = 1\,180\,591\,620\,717\,411\,303\,424$
Yottabyte	YB	$2^{80} = 1\,208\,925\,819\,614\,629\,174\,706\,176$

Figure 1.8: Memory units of measurement

## 1.4 Install Anaconda and Python

To prepare the suitable work environment, we install **Anaconda**, which is a set of packages such as **Spyder**, **Jupiter** and other editors.

### 1.4.1 Go to the Anaconda Website and choose either

[https://www.anaconda.com/products/individual?fbclid=IwAR0MnpEcli-nYMjQujriiFA\\_v\\_Kazx3-fjBSPTnHMaUszp10mf5hRcdVTIQ](https://www.anaconda.com/products/individual?fbclid=IwAR0MnpEcli-nYMjQujriiFA_v_Kazx3-fjBSPTnHMaUszp10mf5hRcdVTIQ)

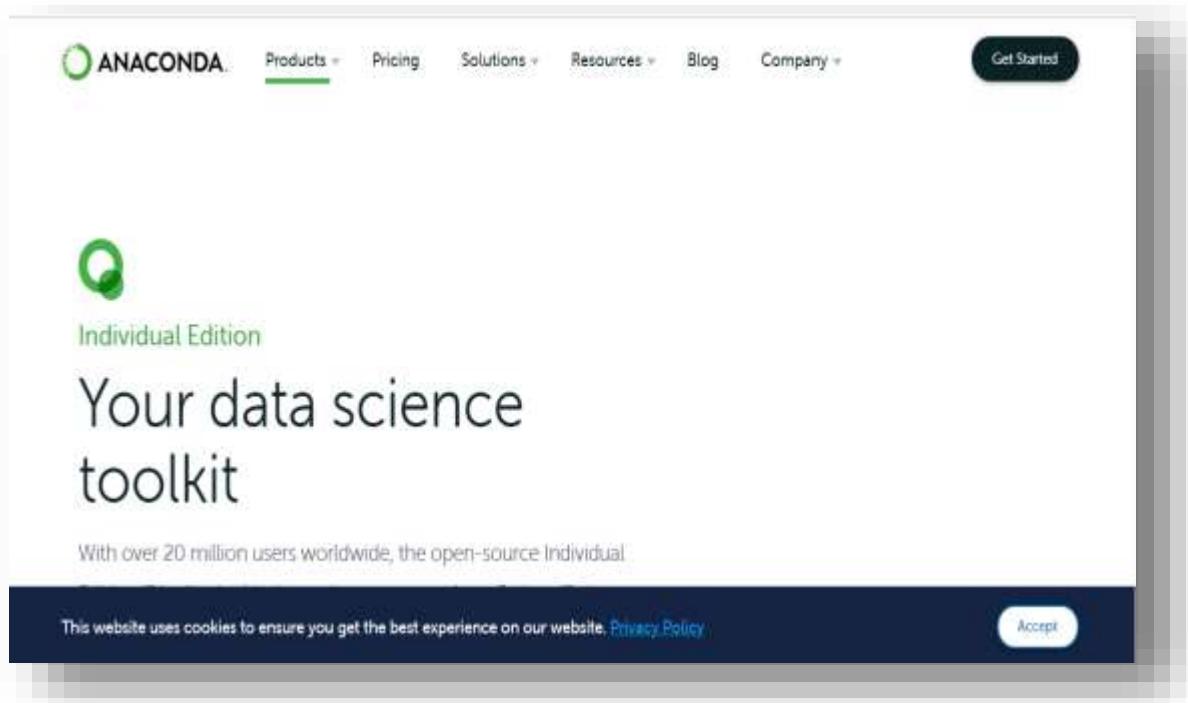


Figure 1.9

## 1.4.2 Choose the operating system for your device and Version



Figure 1.10

## 1.4.3 Locate your download.



Figure 1.11

#### 1.4.4 Click Next.

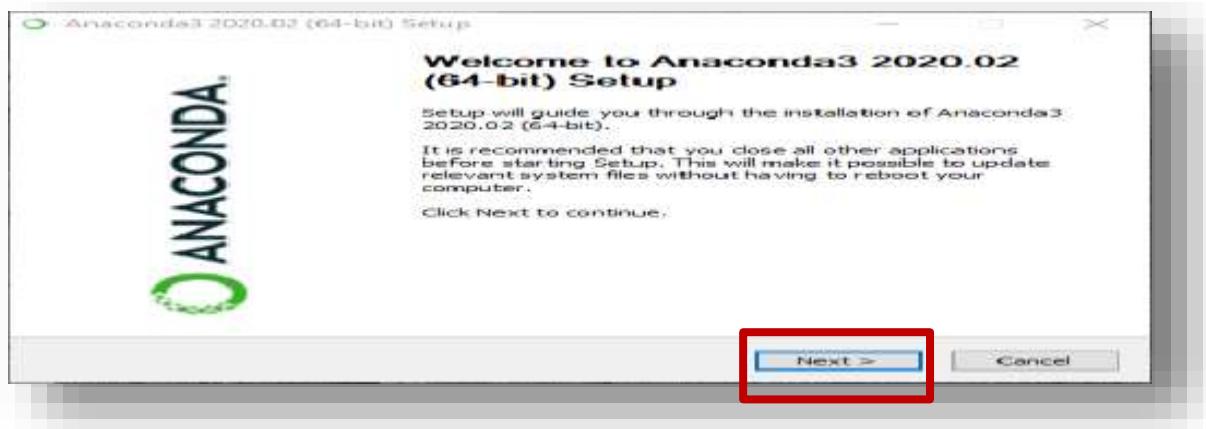


Figure 1.12

#### 1.4.5 Read the licensing terms and click "I Agree"

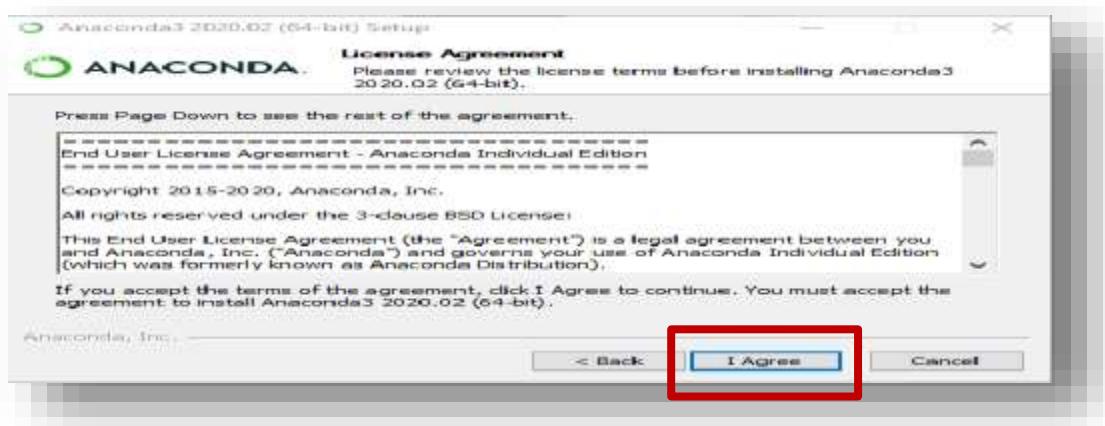


Figure 1.13

#### 1.4.6 Select an install for "Just Me" unless you're installing for all users (which requires Windows Administrator privileges) and click Next

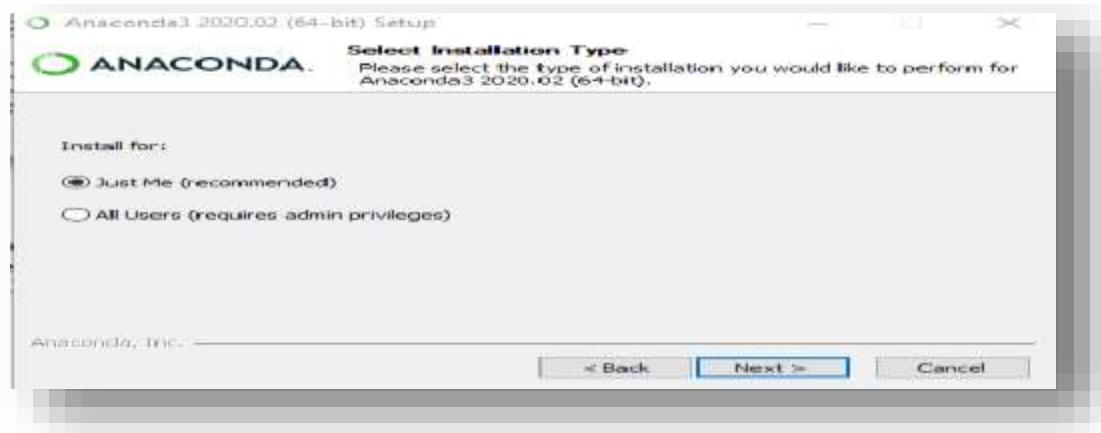


Figure 1.14

1.4.7 Select a destination folder to install Anaconda and click the Next button.

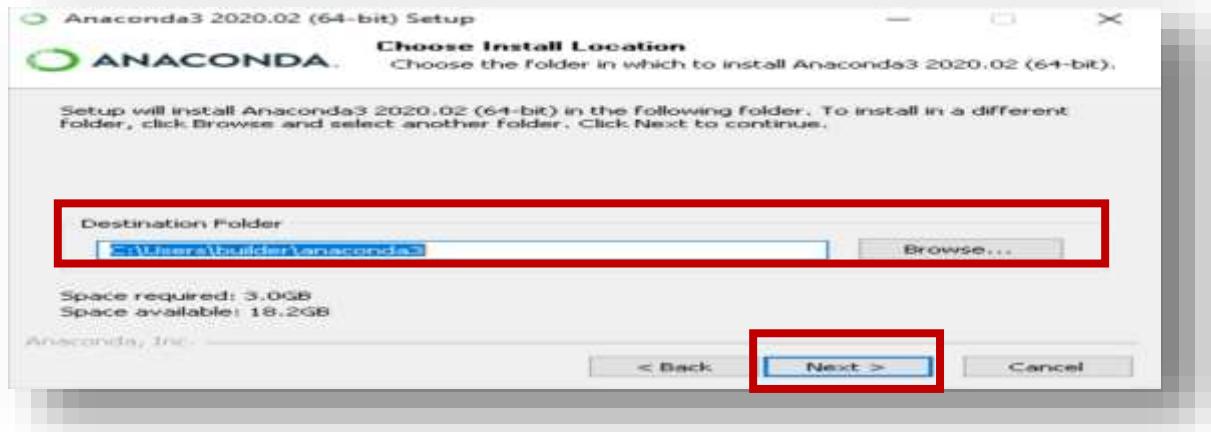


Figure 1.15

1.4.8 Choose whether to add Anaconda to your PATH environment variable. We recommend not adding Anaconda to the PATH environment variable, since this can interfere with other software. Instead, use Anaconda

software by opening Anaconda Navigator or the Anaconda Prompt from the Start Menu.

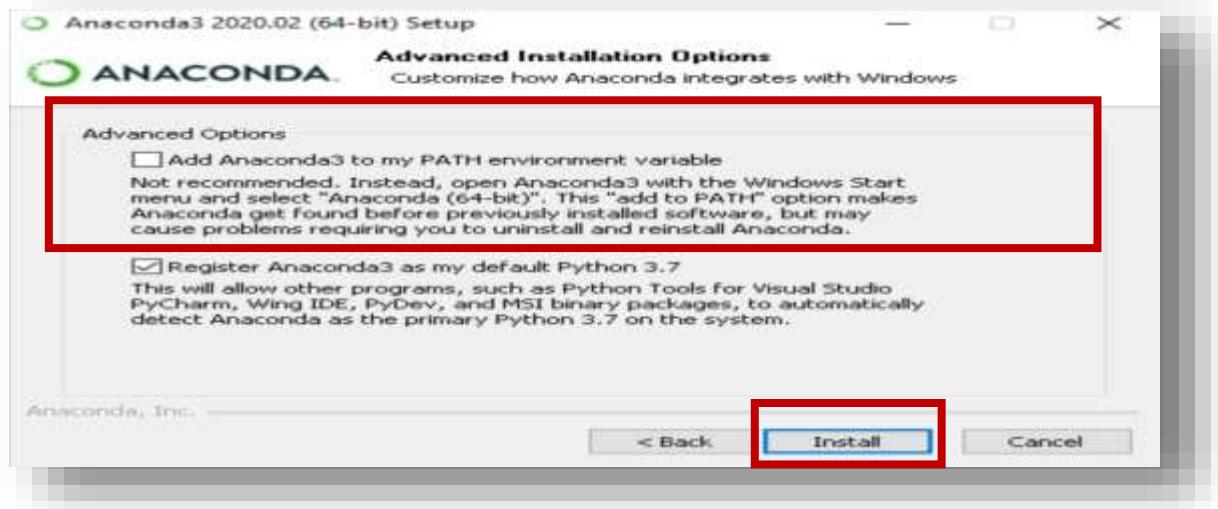


Figure 1.16

1.4.9 Click the Install button. If you want to watch the packages Anaconda is installing, click Show Details.

1.4.10 Click the Next button.

1.4.11 Optional: To install PyCharm for Anaconda, click on the link to <https://www.anaconda.com/pycharm>. Or to install Anaconda without PyCharm, click the Next button.



Figure 1.17

1.4.12 After a successful installation you will see the “Thanks for installing Anaconda” dialog box:

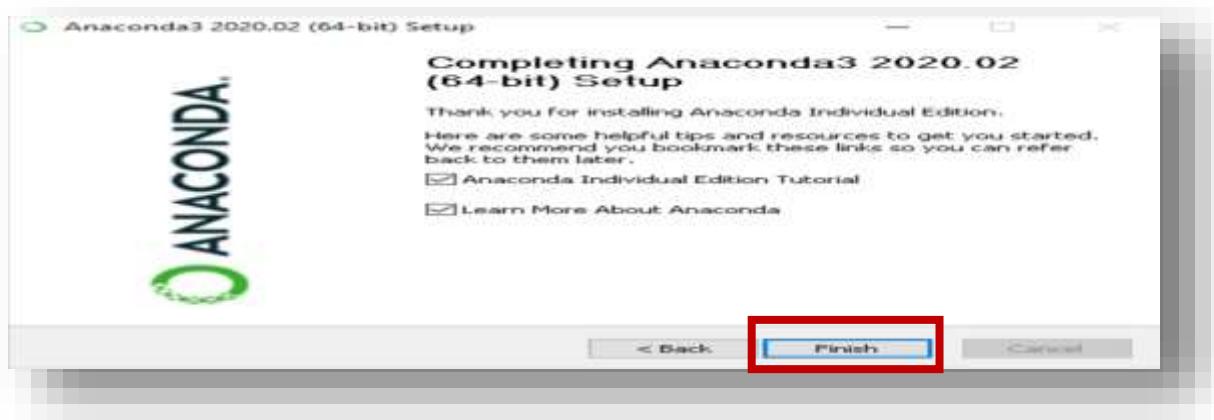


Figure 1.18

### 1.4.13 Welcome to the Anaconda

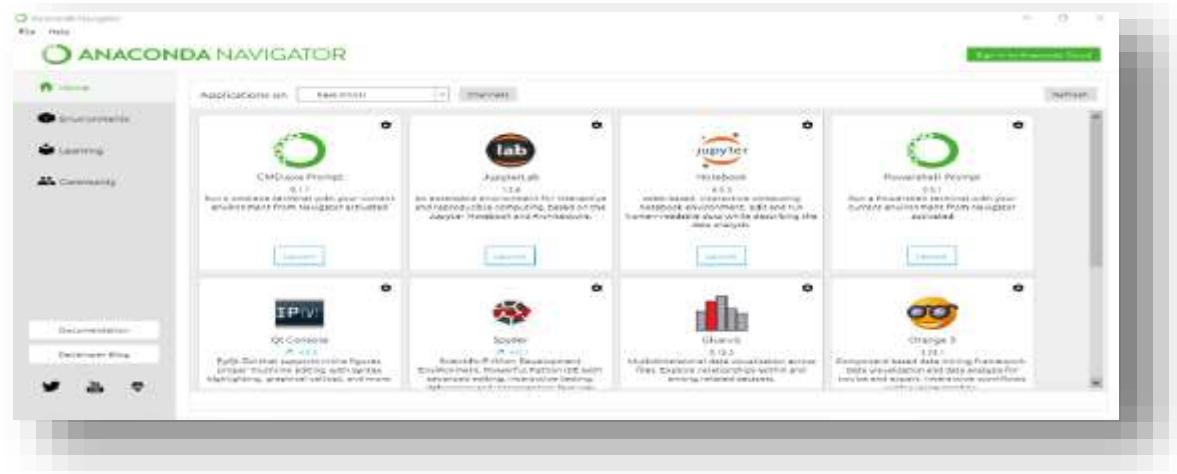


Figure 1.19

## 1.5 Jupyter Notebooks

Jupyter Notebook is an open source web application that allows creating and sharing documents containing live programming symbols, mathematical equations, illustrations and texts.

Previous components that are used for a specific hypothesis such as data analysis, statistical operations, and machine education. Even software exercises.

## 1. What do we mean by live software code?

The main part of Jupyter Notebook is a webpage that arranges and displays its contents the way you like, with the ability to include a code that you can implement inside the page and get the result of the implementation directly. And if something goes wrong, you can fix it and implement it again

## 2. What does this have to do with Python?



Figure 1.20

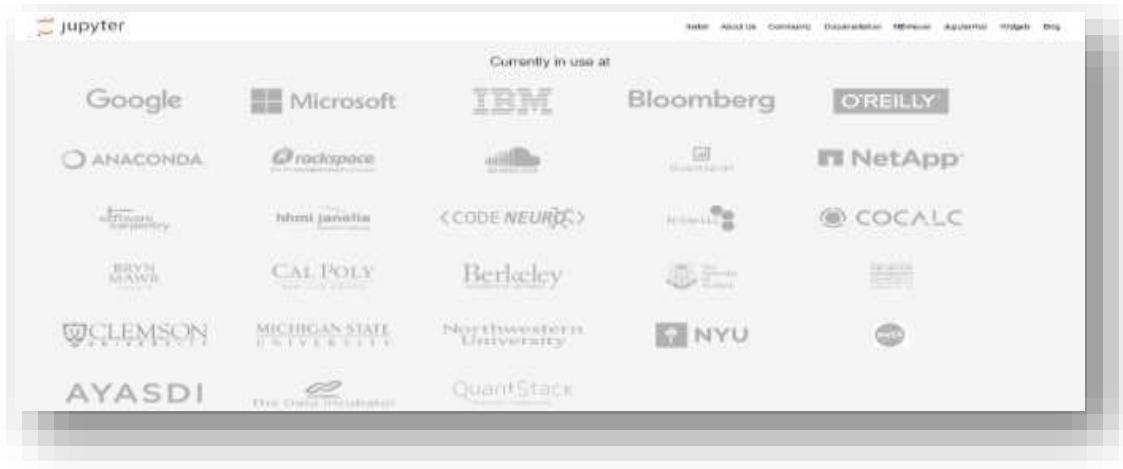


Figure 1.21

The Jupyter tool is mainly based on any Python, which is an interactive version of the Python command interpreter, so the code that you can include on the page, run it, and work with it is the Python code.

### 3. Is that limited to the Python programming language only?

No, according to the Habiter official website, you can guarantee and implement software codes for more than 40 programming languages, including data-processing programming languages such as R - Scala.

To activate the Jupyter notebook, open **Anaconda prompt** from the start function through the **Anaconda package**.



Figure 1.22

After running the **Anaconda prompt**, write your **Jupyter notebook** to be taken to the Jupyter program.

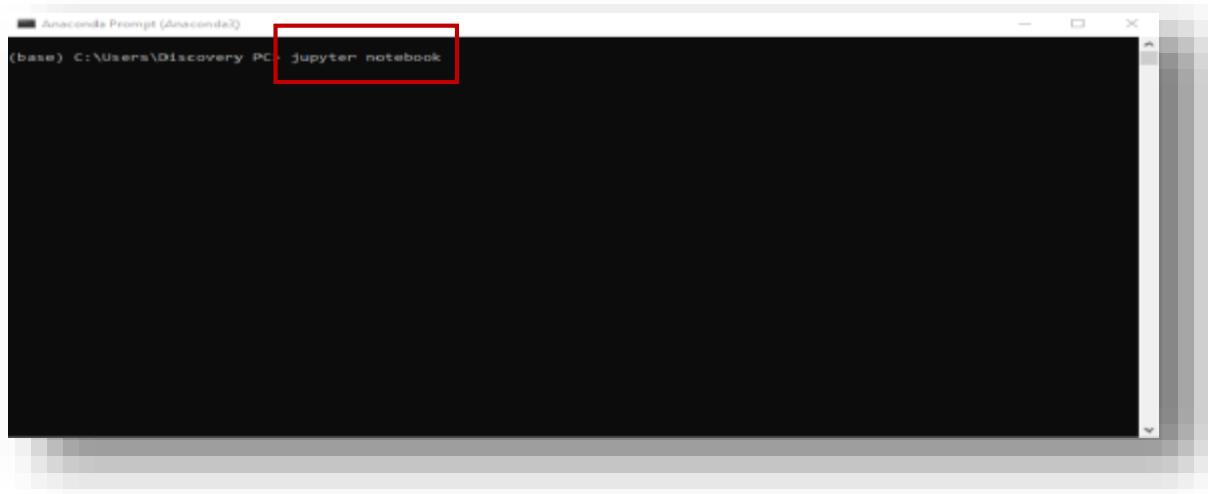
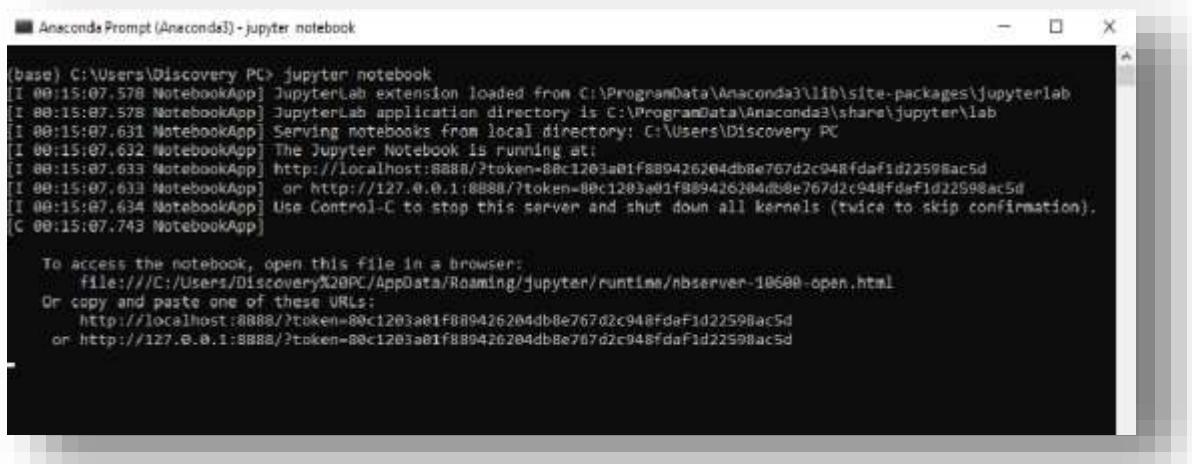


Figure 1.23



```
[base] C:\Users\Discovery PC> jupyter notebook
[1 00:15:07.578 NotebookApp] JupyterLab extension loaded from C:\ProgramData\Anaconda3\lib\site-packages\jupyterlab
[1 00:15:07.578 NotebookApp] JupyterLab application directory is C:\ProgramData\Anaconda3\share\jupyter\lab
[1 00:15:07.631 NotebookApp] Serving notebooks from local directory: C:\Users\Discovery PC
[1 00:15:07.632 NotebookApp] The Jupyter Notebook is running at:
[1 00:15:07.633 NotebookApp] http://localhost:8888/?token=80c1203a01f889426204db8e767d2c948fdaF1d22598ac5d
[1 00:15:07.633 NotebookApp] or http://127.0.0.1:8888/?token=80c1203a01f889426204db8e767d2c948fdaF1d22598ac5d
[1 00:15:07.634 NotebookApp] Use Control-C to stop this server and shut down all kernels (twice to skip confirmation).
[C 00:15:07.743 NotebookApp]

To access the notebook, open this file in a browser:
  file:///C:/Users/Discovery%20PC/AppData/Roaming/jupyter/runtime/nbserver-10500-open.html
Or copy and paste one of these URLs:
  http://localhost:8888/?token=80c1203a01f889426204db8e767d2c948fdaF1d22598ac5d
  or http://127.0.0.1:8888/?token=80c1203a01f889426204db8e767d2c948fdaF1d22598ac5d
```

Figure 1.24

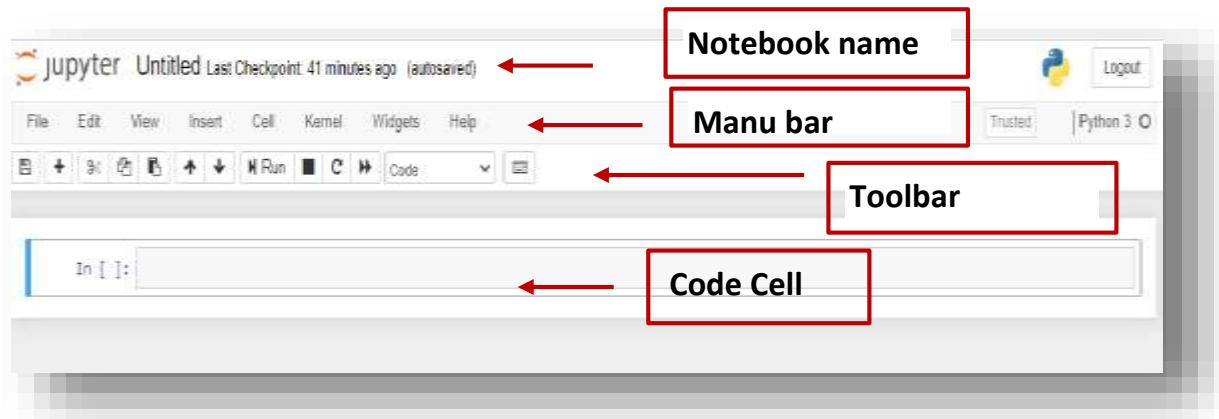
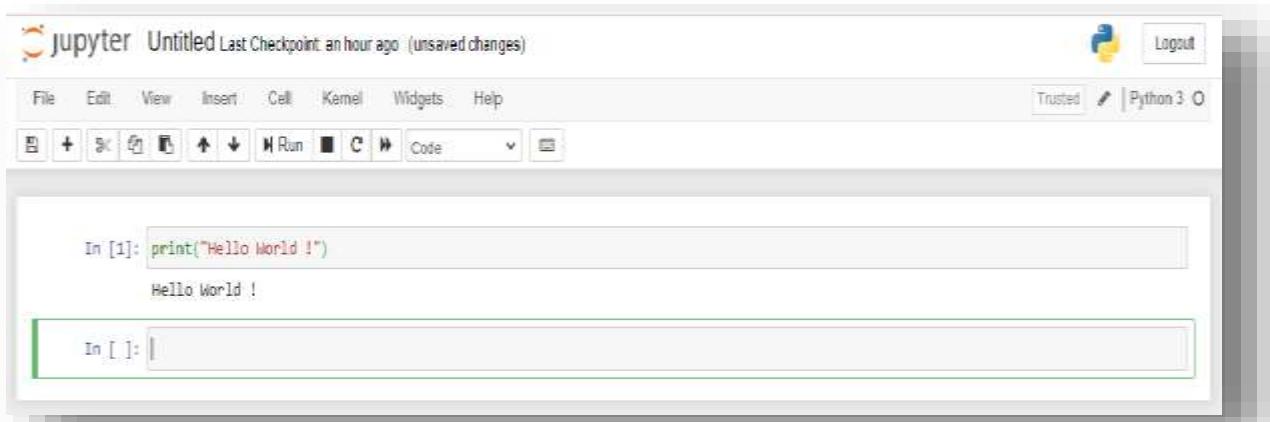


Figure 1.25

**Notebook name:** The name displayed at the top of the page, next to the Jupyter logo, reflects the name of the `.ipynb` file. Clicking on the notebook name brings up a dialog which allows you to



rename it. Thus, renaming a notebook from “Untitled0” to “My first notebook” in the browser, renames the `Untitled0.ipynb` file to `My first notebook.ipynb`.

**Menu bar:** The menu bar presents different options that may be used to manipulate the way the notebook functions.

**Toolbar:** The tool bar gives a quick way of performing the most-used operations within the notebook, by clicking on an icon.

**Code cell:** the default type of cell; read on for an explanation of cells.

Figure 1.26

---

# CHAPTER 2

---

## Python Packages (Numpy and Pandas)

### 2.1 Numpy

#### 2.1.1 Overview

Numpy is the core library for scientific computing in Python. It provides a high-performance multidimensional array object and tools for working with these arrays. Numpy is a powerful N-dimensional *array* object which is Linear algebra for Python. Numpy arrays essentially come in two flavors: *Vectors* and *Matrices*. *Vectors* are strictly 1-d array whereas *Matrices* are 2-d but *matrices* can have only one row/column.

Numpy provides an N-dimensional array type, the `ndarray`, which describes a collection of “items” of the same type. The items can be indexed using for example N integers.

## 2.1.2 Numpy in python

To install `numpy` library in your system and to know further about python basics you can follow the below link:

```
1. import numpy as np
```

Figure 2.1

Now to use `numpy` in the program we need to import the module. Generally, `numpy` package is defined as `np` of abbreviation for convenience. But you can import it using anything you want.

```
1. import numpy as np
2. np.array([1, 2, 3])          # Create a rank 1 array
3. np.arange(15)               # generate an 1-d array from 0 to 14
4. np.arange(15).reshape(3, 5) # generate array and change dimensions
```

Figure 2.2



```
Out[1]:
array([[ 0,  1,  2,  3,  4],
       [ 5,  6,  7,  8,  9],
       [10, 11, 12, 13, 14]])
```

In [2]:

Figure 2.3

Example array 1 dimension:

```
1. a = np.array([0,1,2,3]) # Create a rank 1 array
2.
3. print(a)                #print array a
4. print(type(a))          #type of array a
5. print(a.ndim)            #dimension of array a
6. print(a.shape )          #shape (row ,column) of array a
7. print(len(a))            #length of array a
```

Figure 2.4



```
Console 10/A
[0 1 2 3]
<class 'numpy.ndarray'>
1
(4,)
4
```

Figure 2.5

In the example figure above we can observe `numpy` is imported first and then a 1-d `numpy` array is defined. Then we can examine the type, dimension, shape, and length of the array using mentioned commands. Here are some important commands for creating arrays:

```

1. print(np.random.randint(1,100,6),'\n')           #create array of 6 random values in
0-100 range.
2. print(np.random.randint(1,100,6).reshape(3,2),'\n') #reshape the array according to row
and column vectors.
3. print(np.random.rand(4),'\n')                   #create an array of uniform distrib
ution (0,1)
4. print(np.eye(3),'\n')                          #create a 3*3 identity matrix
5. print(np.zeros(3),'\n')                        #create array([0,0,0])
6. print(np.zeros((5,5)),'\n')                   #create a 5*5 2-d array of zeros
7. print(np.random.randn(2,2),'\n')                #return standard normal distribution v
center around zero.
8. print(np.empty((2,3)),'\n')                   # uninitialized
9. print(np.arange(0, 2, 0.3),'\n') # from 0 to a number less than 2 with 0.3 intervals
10. print(np.ones((2,3,4), dtype=np.int16),'\n')  # all element are 1
11. print(np.array([[1,2],[3,4]]), dtype=complex),'\n') # complex array
12. print(np.array([(1.5,2,3),(4,5,6)]),'\n')    # two-dimensional array
13. print(np.array([2,3,4]),'\n')

```

Figure 2.6

```

In [6]: 26 7 65 41 29 7
[[82 43]
 [81 91]
 [44 37]]
[0.05721004 0.06689817 0.04226207 0.34687584]
[[1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]]
[0. 0. 0.]
[[0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0.]]
[[0.53866425 1.81479059]
 [-1.16329411 0.26191503]]
[[0. 0. 0.]
 [0. 0. 0.]]
[0. 0.3 0.6 0.9 1.2 1.5 1.8]
[[1 1 1]
 [1 1 1]
 [1 1 1 1]]
[[1 1 1]
 [1 1 1, 1]
 [1 1 1 1, 1]]
[[1 1, 1 1]
 [1 1, 1 1, 1]
 [1 1, 1 1, 1, 1]]
[[1, 1, 1, 1]
 [1, 1, 1, 1, 1]
 [1, 1, 1, 1, 1, 1]]
[[1, 2, 3]
 [4, 5, 6]
 [2, 3, 4]]

```

Figure 2.7

### 2.1.3 Important attributes of the `ndarray` object

`ndarray.shape`: The dimensions of the array. This is a tuple of integers indicating the size of the array in each dimension. For a matrix with  $n$  rows and  $m$  columns, shape will be  $(n, m)$ .

`ndarray.ndim`: The number of axes (dimensions) of the array.

`ndarray.dtype`: If you want to know the data type of an array, you can query the attributes of *dtype*. An object describing the type of

The following attributes contain information about the memory layout of the array:

`ndarray.flags`: Information about the memory layout of the array.

`ndarray.shape`: Tuple of array dimensions.

`ndarray.strides`: Tuple of bytes to step in each dimension when traversing an array.

`ndarray.ndim`: Number of array dimensions.

`ndarray.data`: Python buffer object pointing to the start of the array's data.

**ndarray.size:** Number of elements in the array.

**ndarray.itemsize:** Length of one array element in bytes.

**ndarray nbytes:** Total bytes consumed by the elements of the array.

**ndarray.base:** Base object if memory is from some other object.

The elements in the array. One can create or specify dtype using standard Python types. Additionally, numpy provides types of its own. numpy.int32, numpy.int16, and numpy.float64 are some examples.

```
1. a = np.arange(15).reshape(3,5)
2. print(a.shape)
3. print(a.ndim)
4. print(a.dtype)
5. print(a.itemsize)
6. print(a.size)
```

Figure 2.8

```
Console 6/A
(3, 5)
2
int32
4
15
```

Figure 2.9

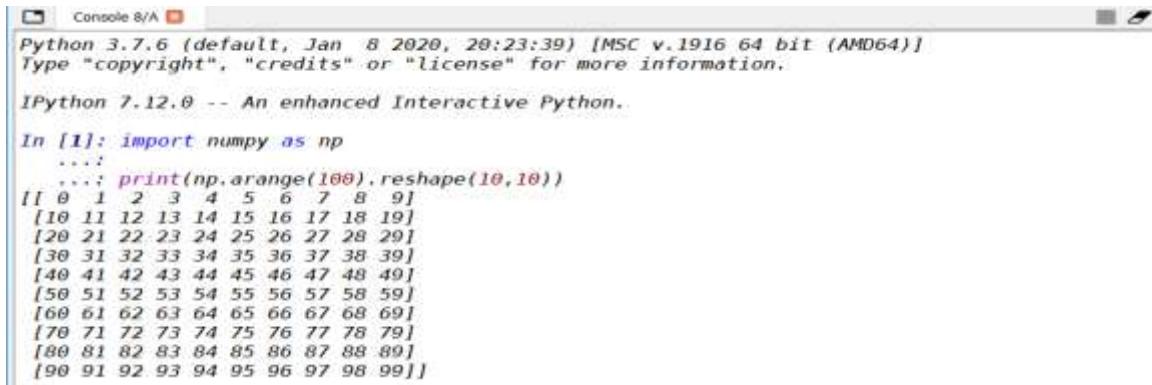
`ndarray.itemsize` : The size in bytes of each element of the array. For example, an array of elements of type `float64` has `itemsize` 8 ( $=64/8$ ), while one of type `complex32` has itemsize 4 ( $=32/8$ ). It is equivalent to `ndarray.dtype.itemsize`.

`ndarray.size`: The total number of elements of the array. This is equal to the product of the elements of shape.

**Printing an array:** When you print an array, numpy displays it in a similar way to nested lists, but with the following layout: the last axis is printed from left to right, the second-to-last is printed from top to bottom, the rest are also printed from top to bottom, with each slice separated from the next by an empty line.

```
1. print(np.arange(100).reshape(10,10))
```

Figure 2.10



```

Python 3.7.6 (default, Jan  8 2020, 20:23:39) [MSC v.1916 64 bit (AMD64)]
Type "copyright", "credits" or "license" for more information.

IPython 7.12.0 -- An enhanced Interactive Python.

In [1]: import numpy as np
.....
....; print(np.arange(100).reshape(10,10))
[[ 0  1  2  3  4  5  6  7  8  9]
 [10 11 12 13 14 15 16 17 18 19]
 [20 21 22 23 24 25 26 27 28 29]
 [30 31 32 33 34 35 36 37 38 39]
 [40 41 42 43 44 45 46 47 48 49]
 [50 51 52 53 54 55 56 57 58 59]
 [60 61 62 63 64 65 66 67 68 69]
 [70 71 72 73 74 75 76 77 78 79]
 [80 81 82 83 84 85 86 87 88 89]
 [90 91 92 93 94 95 96 97 98 99]]

```

Figure 2.11

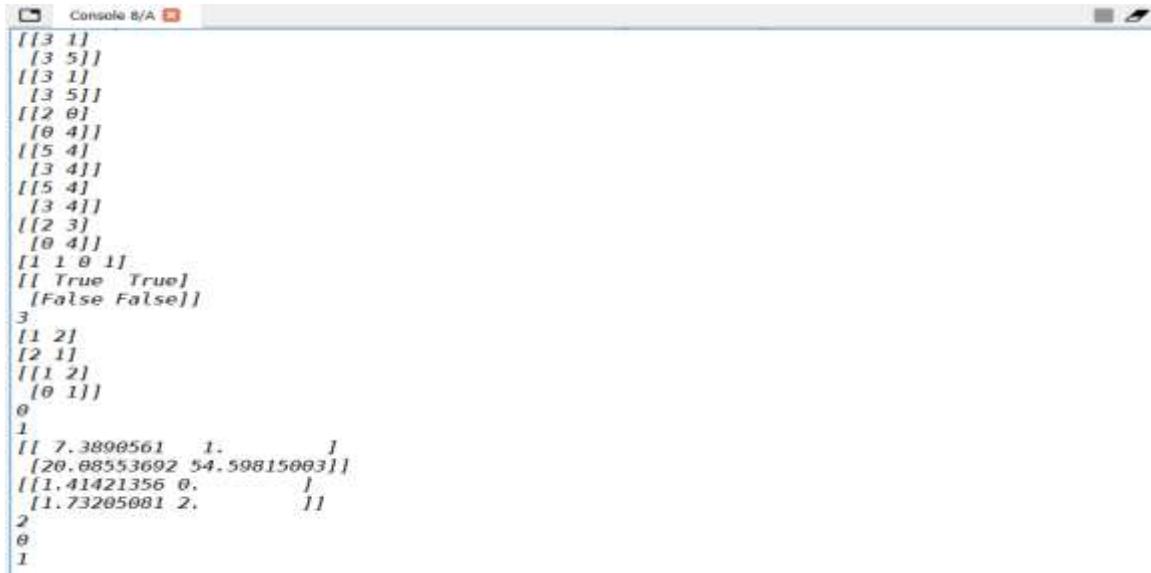
## 2.1.4 Basic Operations of ndarray:

```

1. A = np.array([[1,1],[0,1]])
2. B = np.array([[2,0],[3,4]])
3. print(A+B)           #addition of two array
4. print(np.add(A,B))   #addition of two array
5. print(A * B)         # elementwise product
6. print(A @ B)         # matrix product
7. print(A.dot(B))      # another matrix product
8. print(B.T) #Transpose of B array
9. print(A.flatten())   #form 1-d array
10. print(B < 3)        #Boolean of Matrix B. True for elements less than 3
11. print(A.sum())       # sum of all elements of A
12. print(A.sum(axis=0)) # sum of each column
13. print(A.sum(axis=1)) # sum of each row
14. print(A.cumsum(axis=1)) # cumulative sum along each row
15. print(A.min())       # min value of all elements
16. print(A.max())       # max value of all elements
17. print(np.exp(B))     # exponential
18. print(np.sqrt(B))    # square root
19. print(A.argmin())    #position of min value of elements
20. print(A.argmax())    #position of max value of elements
21. print(A[1,1])        #member of a array in (1,1) position

```

Figure 2.12



```

[[3 1]
 [3 5]
 [[3 1]
 [3 5]]
 [[2 0]
 [0 4]]
 [[5 4]
 [3 4]]
 [[5 4]
 [3 4]]
 [[2 3]
 [0 4]]
 [[1 0 1]
 [True True]
 [False False]]
 3
 [1 2]
 [2 1]
 [[1 2]
 [0 1]]
 0
 1
 [[ 7.3890561  1.
 [20.08553692 54.59815003]]
 [[1.41421356 0.
 [1.73205081 2.
 2
 0
 1

```

Figure 2.13

## Indexing, Slicing and Iterating in numpy:

```

1. a = np.arange(4)**3          # create array a
2. print(a[2])                # member of a array in 2nd position
3. print(a[::-1])              # reversed a
4. print(a[1,...])             # same as a[1,:,:] or a[1]
5. print(a[a>5])              # a with values greater than 5

```

Figure 2.14



```

8
[27  8  1  0]
1
[ 8 27]

```

Figure 2.15

If any position of an array is allocated to another array and its being broadcasted to a new value, then the original array also

changed. This is because *numpy* doesn't want to use more memory for the same array. As shown in the following figure, the value of *a* is changed with changing the values of *x* which is a part of the array *a*.

```
1. a=np.arange( 4)**3
2. print(a)
3. x = a[0:4] #assign x with 4 values of a
4. x[:] = 99 #change the values of x to 99 which will change the values of a also.
5. print(a)
```

Figure 2.16

```
Console 8/A
[0 1 8 27]
[99 99 99 99]
```

Figure 2.17

When we use a comparison operator in an array it returns a Boolean array. Then using the Boolean array we could select elements conditionally from the original array.

- **Operations on a 1d Array**

```
1. array1 = np.array([10,20,30,40,50])
2. array2 = np.arange(5)
3. print(array2)
```

Figure 2.18



Figure 2.19

You can perform arithmetic operations on these arrays. For example, if you add the arrays, the arithmetic operator will work element-wise. The output will be an array of the same dimension.

```
1. array3 = array1 + array2
2. print(array3)
```

Figure 2.20



Figure 2.21

Here is a pictorial representation of the same:

10	20	30	40	50
<i>array1</i>				
0	1	2	3	4
<i>array2</i>				
10	21	32	43	54
<i>array3</i>				

```
1. array1=array1 *2
2. print(array1)
```

Figure 2.22

```
Console 8/A
[ 20 40 60 80 100]
```

Figure 2.23

To find the square of the numbers, use :

```
1. array1=array1 **2
2. print(array1)
```

Figure 2.24

```
Console 8/A
[ 400 1600 3600 6400 10000]
```

Figure 2.25

If you would like to cube the individual elements, or even higher up, use the power function. Here, each element of the array is raised to the power 3.

```
1. print( np.power(array1,3))
```

Figure 2.26



```
64000000.0 -198967296.0 -588640256.0 158994944.0 -727379968.0
```

Figure 2.27

## Using Numpy with Conditional Expressions

You can use conditionals to find the values that match your criteria. Since array1 is an array, the result of a conditional operation is also an array. When we perform a conditional check, the output is an array of booleans.

```
1. array5 = array1 >= 30
2. print(array5)
```

Figure 2.28



```
True True True True True
```

Figure 2.29

## • Operations on a 2D Array

### Arithmetic Operators with Numpy 2D Arrays

Let's create 2 two-dimensional arrays, A and B.

```
1. A = np.array([[3,2],[0,1]])
2. B = np.array([[3,1],[2,1]])
3.
4.
5. print(A, '\n')
6. print(B, '\n')
7.
```

- Now, let's try adding the arrays. Unsurprisingly, the elements at the respective positions in arrays are added together.

```
8. print (A+B, '\n')
9.
```

- All the arithmetic operations work in a similar way. You can also multiply or divide the arrays. The operations are performed element-wise.

```
10. print (A*B, '\n')
```

- Similar to programming languages like C# and Java, you can also use operators like `+=`, `*=` on your Numpy arrays. For example, we have the array:

Doing `+=` operation on the array 'A' is equivalent to adding each element of the array with a specified value. So,

```
11. A +=2
12. print(A, '\n')
```

- The same output can also be achieved by the function `dot`. For example:

```
13. A.dot(B)
14.
15. print(A, '\n')
```

Figure 2.30



```
Console 10/A
```

```
[[3 2]
 [0 1]]
[[3 1]
 [2 1]]
[[6 3]
 [2 2]]
[[9 2]
 [0 1]]
[[5 4]
 [2 3]]
[[5 4]
 [2 3]]
```

Figure 2.31

- Here is the pictorial representation.

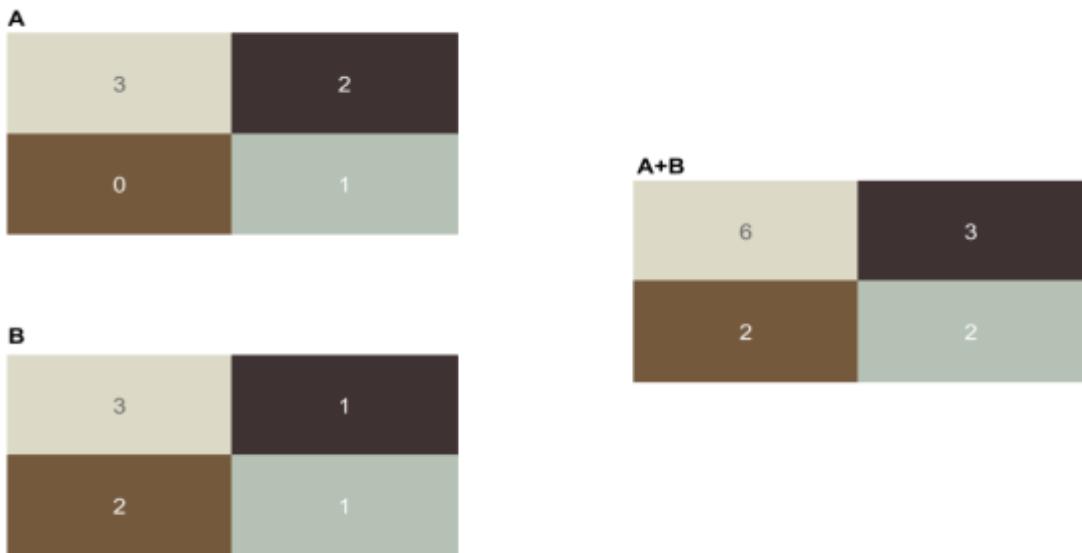


Figure 2.31

- Arithmetic Functions in Numpy

All the arithmetic operations work in a similar way. You can also multiply or divide the arrays. The operations are performed element-wise.

There are several functions that you can use to perform arithmetic operations on this array. For example, the `sum` function adds all the values in the array and gives a scalar output.

```
1. print(array.sum(), '\n')
```

- The `min` function finds the lowest value in the array.

```
2. print(array.min(), '\n')
```

- Using Axis Parameter

If you have more than one dimension in your array, you can define the axis; along which, the arithmetic operations should take place.

For example, for a two-dimensional array, you have two axes. Axis 0 is running vertically downwards across the rows, while Axis 1 is running horizontally from left to right across the columns.

If you want the sum of all the values in a single column, use the Axis parameter with value '0.' The first value in the resulting array represents the sum of all values in the first column and the second value represents the sum of all values in the second column.

```
3. print(array.sum(axis=0), '\n')
```

\* Similarly, to find the lowest value across a particular row, use the Axis parameter with a Value '1.' Each of the values in the resulting array represents the lowest value for that particular row.

```
4. print(array.min(axis=1), '\n')
```

Figure 2.32



```
3.1405596216774785
0.5995741990603158
[1.83631954 1.30424008]
[0.70466589 0.5995742 ]
```

Figure 2.33

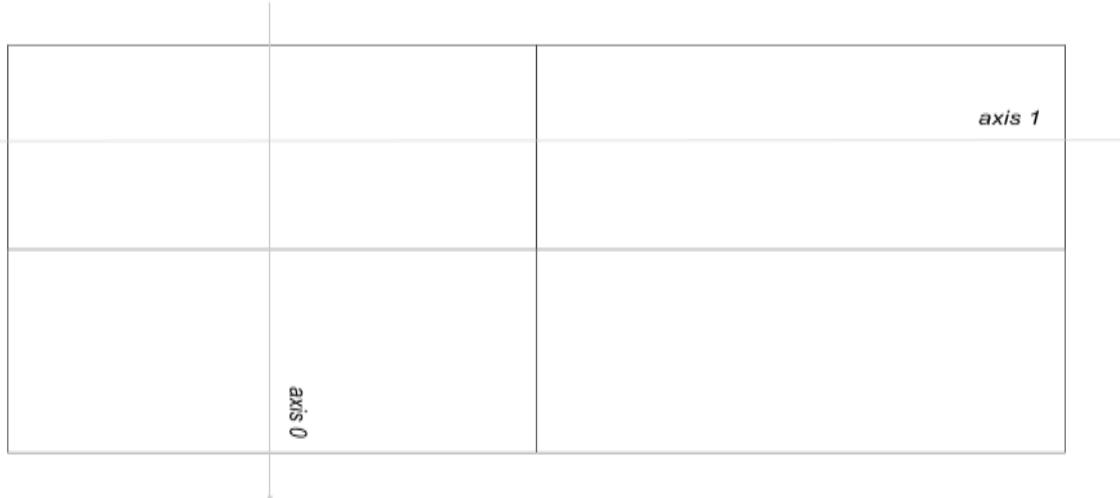


Figure 2.34

- **Logical Operators in Numpy**

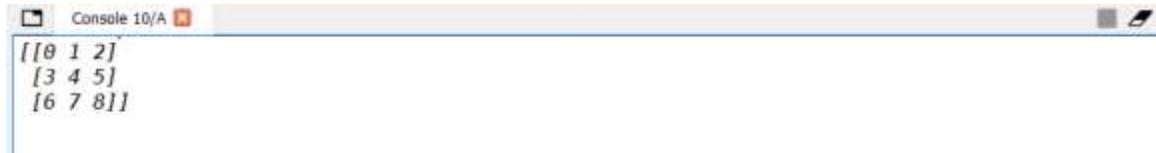
Numpy provides logic functions like `logical_and`, `logical_or` etc., in a similar pattern to perform logical operations. For example:

#### Other Functions in Numpy

In addition to arithmetic operators, Numpy also provides functions to perform arithmetic operations. You can use functions like `add`, `subtract`, `multiply`, `divide` to perform array operations. For example:

```
1. a = np.arange(9).reshape(3,3)
2. print(a, '\n')
```

Figure 2.35

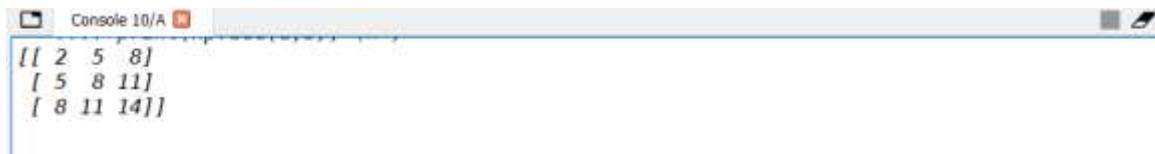


```
[[0 1 2]
 [3 4 5]
 [6 7 8]]
```

Figure 2.36

```
1. b = np.array([2,4,6])
2. print(np.add(a,b), '\n')
```

Figure 2.37



```
[[ 2  5  8]
 [ 5  8 11]
 [ 8 11 14]]
```

Figure 2.38

Note that, array 'b' is added to each row in the array 'a.' So the array dimensions should match.

## 2.2 Pandas

### 2.2.1 Overview

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.

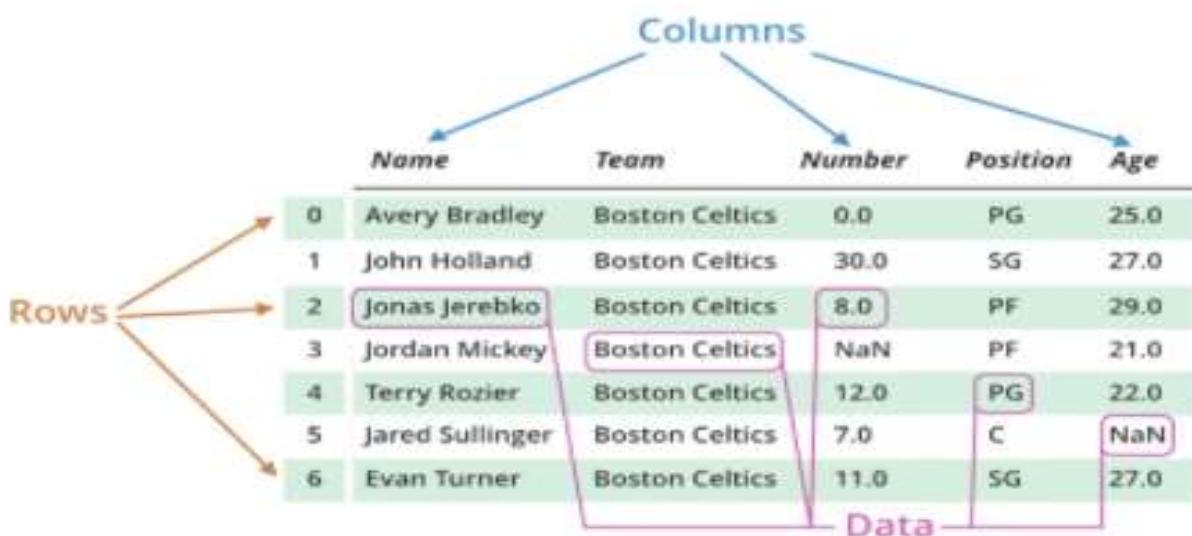


Figure 2.39

## 2.2.2 Pandas in python

- 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.

	Name	Team	Number
0	Avery Bradley	Boston Celtics	0.0
1	John Holland	Boston Celtics	30.0
2	Jonas Jerebko	Boston Celtics	8.0
3	Jordan Mickey	Boston Celtics	NaN
4	Terry Rozier	Boston Celtics	12.0
5	Jared Sullinger	Boston Celtics	7.0

Figure 2.40

**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.

```

In [1]: import pandas as pd
d = {"a":10, "b":20, "c":30}
pd.Series(d)

Out[1]: a    10
         b    20
         c    30
        dtype: int64

In [2]: pd.Series(list(d.values()), list(d.keys()))

Out[2]: a    10
         c    30
         b    20
        dtype: int64

```

Figure 2.41

construction 4 types of objects list , array, dictionary and method of calling from the library

```

In [1]: import pandas as pd
import numpy as np

In [2]: labels = ['a', 'b', 'c']
my_list = [10, 20, 30]
arr = np.array(my_list)
dic = {"a":10, "b":20, "c":30}

In [4]: labels
Out[4]: ['a', 'b', 'c']

In [5]: my_list
Out[5]: [10, 20, 30]

In [6]: arr
Out[6]: array([10, 20, 30])

In [7]: dic
Out[7]: {'a': 10, 'b': 20, 'c': 30}

```

Figure 2.42

Deal with the series , To call the Series, you must stick to the parameter

```
In [8]: pd.Series()  
Out[8]:  
Init signature: pd.Series(self, data=None, index=None, dtype=None, name=None,  
copy=False, fastpath=False)  
Docstring:  
One-dimensional ndarray with axis labels (including time series).
```

Figure 2.43

- Panda differs from numpy by the presence of index
- add element of series using index

```
In [8]: pd.Series(data = my_list)  
Out[8]: 0    10  
1    20  
2    30  
dtype: int64
```

Figure 2.44

- add element of series using data and index

```
In [9]: pd.Series(data = my_list , index= labels)  
Out[9]: a    10  
b    20  
c    30  
dtype: int64
```

Figure 2.45

Notice in the example above that the elements are printed as a dictionary but in fact it is a Series

- Add element of series without using data and index as variables

```
In [10]: pd.Series( my_list , labels)
Out[10]: a    10
          b    20
          c    30
         dtype: int64
```

Figure 2.46

- Add element of series using array

```
In [11]: pd.Series(arr , labels)
Out[11]: a    10
          b    20
          c    30
         dtype: int32
```

Figure 2.47

- Add element of series using dictionary

```
In [10]: pd.Series(dic)
Out[10]: a    10
          b    20
          c    30
         dtype: int64
```

Figure 2.48

- Add two elements of series using list

```
In [11]: ser1=pd.Series([1,2,3,4], index=['USA','Germany','USSR','Japan'])
```

```
In [12]: ser1
```

```
Out[12]: USA      1  
Germany    2  
USSR      3  
Japan      4  
dtype: int64
```

```
In [13]: ser2=pd.Series([1,2,5,4], index=['USA','Germany','Italy','Japan'])
```

```
In [14]: ser2
```

```
Out[14]: USA      1  
Germany    2  
Italy      5  
Japan      4  
dtype: int64
```

Figure 2.49

- When two values differ, the NaN is marked as the value is converted from int to float

```
In [15]: ser1+ser2
```

```
Out[15]: Germany    4.0  
Italy        NaN  
Japan       8.0  
USA         2.0  
USSR        NaN  
dtype: float64
```

Figure 2.50

- **DataFrames**

using random to Return a sample (or samples) from the “standard normal” distribution.

Seed the generator, This method is called when Random State is initialized. It can be called again to re-seed the generator

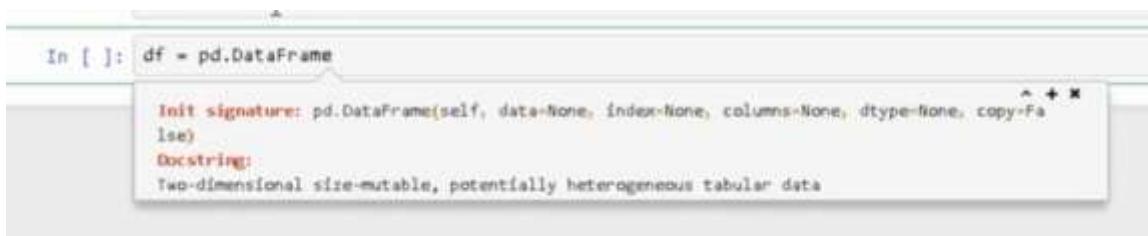
```
In [1]: import pandas as pd
import numpy as np

In [2]: from numpy.random import randn

In [3]: np.random.seed(101)
```

Figure 2.51

Deal with the dataframe , To call the dataframe, you must stick to the parameter



In [ ]: df = pd.DataFrame

Init signature: pd.DataFrame(self, data=None, index=None, columns=None, dtype=None, copy=False)  
Docstring:  
Two-dimensional size-mutable, potentially heterogeneous tabular data

Figure 2.52

- Add element of dataframe using randn

```
In [4]: df=pd.DataFrame(randn(5,4), index= 'A B C D E'.split() , columns='W X Y Z'.split())
In [5]: df
Out[5]:
      W      X      Y      Z
A  2.706850  0.628133  0.907969  0.503826
B  0.651118 -0.319318 -0.848077  0.605965
C -2.018168  0.740122  0.528813 -0.589001
D  0.188695 -0.758872 -0.933237  0.955057
E  0.190794  1.978757  2.605967  0.683509
```

Figure 2.53

- Print element of the dataframe

```
In [6]: df['W']
Out[6]: A    2.706850
         B    0.651118
         C   -2.018168
         D    0.188695
         E    0.190794
Name: W, dtype: float64
```

Figure 2.54

- Type of element in each index of element

```
In [7]: type(df['W'])
Out[7]: pandas.core.series.Series
```

Figure 2.55

- Print type of element in each index of element

```
In [8]: type(df)
Out[8]: pandas.core.frame.DataFrame
```

Figure 2.56

- Note that the data frame is a group of series
- Add new element of dataframe and Combine two series together

```
In [9]: df['New']= df['W'] +df['Y']
In [10]: df['New']
Out[10]: A    3.614819
          B   -0.196959
          C   -1.489355
          D   -0.744542
          E    2.796762
          Name: New, dtype: float64

In [11]: df
Out[11]:
```

	W	X	Y	Z	New
A	2.706850	0.628133	0.907969	0.503826	3.614819
B	0.651118	-0.319318	-0.848077	0.605965	-0.196959
C	-2.018168	0.740122	0.528813	-0.589001	-1.489355
D	0.188695	-0.758872	-0.933237	0.955057	-0.744542
E	0.190794	1.978757	2.605967	0.683509	2.796762

Figure 2.57

- Drop row of dataframe

```
In [12]: df.drop('New',axis=1)
```

Out[12]:

	W	X	Y	Z
A	2.706850	0.628133	0.907969	0.503826
B	0.651118	-0.319318	-0.848077	0.605965
C	-2.018168	0.740122	0.528813	-0.589001
D	0.188695	-0.758872	-0.933237	0.955057
E	0.190794	1.978757	2.605967	0.683509

Figure 2.58

But this method does not delete the element in place

```
In [12]: df.drop('New',axis=1)
```

Out[12]:

	W	X	Y	Z
A	2.706850	0.628133	0.907969	0.503826
B	0.651118	-0.319318	-0.848077	0.605965
C	-2.018168	0.740122	0.528813	-0.589001
D	0.188695	-0.758872	-0.933237	0.955057
E	0.190794	1.978757	2.605967	0.683509

```
In [13]: df
```

Out[13]:

	W	X	Y	Z	New
A	2.706850	0.628133	0.907969	0.503826	3.614819
B	0.651118	-0.319318	-0.848077	0.605965	-0.196959
C	-2.018168	0.740122	0.528813	-0.589001	-1.489355
D	0.188695	-0.758872	-0.933237	0.955057	-0.744542
E	0.190794	1.978757	2.605967	0.683509	2.796762

Figure 2.59

Confirm the deletion process must be activated inplace= True

```
In [14]: df.drop('New',axis=1 , inplace=True)
```

```
In [15]: df
```

Out[15]:

	W	X	Y	Z
A	2.706850	0.628133	0.907969	0.503826
B	0.651118	-0.319318	-0.848077	0.605965
C	-2.018168	0.740122	0.528813	-0.589001
D	0.188695	-0.758872	-0.933237	0.955057
E	0.190794	1.978757	2.605967	0.683509

Figure 2.60

- Drop Column of dataframe

```
In [16]: df.drop('E', axis=0 , inplace=True )
```

```
In [17]: df
```

Out[17]:

	W	X	Y	Z
A	2.706850	0.628133	0.907969	0.503826
B	0.651118	-0.319318	-0.848077	0.605965
C	-2.018168	0.740122	0.528813	-0.589001
D	0.188695	-0.758872	-0.933237	0.955057

Figure 2.61

- Print shape of elements

```
In [18]: df.shape
```

Out[18]: (4, 4)

Figure 2.62

- Print location of element

```
In [19]: df.loc['C']  
Out[19]: W    -2.018168  
          X     0.740122  
          Y     0.528813  
          Z    -0.589001  
         Name: C, dtype: float64
```

Figure 2.63

- Print index location of element

```
In [20]: df.iloc[2]  
Out[20]: W    -2.018168  
          X     0.740122  
          Y     0.528813  
          Z    -0.589001  
         Name: C, dtype: float64
```

Figure 2.64

- Print location of element in each index of element

```
In [21]: df.loc[['A','C'] , ['W','Y']]  
Out[21]:  
          W      Y  
A  2.706850  0.907969  
C -2.018168  0.528813
```

Figure 2.65

- Comparison Operators

```
In [24]: df[df>0]
```

```
Out[24]:
```

	W	X	Y	Z
A	2.706850	0.628133	0.907969	0.503826
B	0.651118	NaN	NaN	0.605965
C	NaN	0.740122	0.528813	NaN
D	0.188695	NaN	NaN	0.955057
E	0.190794	1.978757	2.605967	0.683509

```
In [25]: df['W']>0
```

```
Out[25]: A    True
          B    True
          C    False
          D    True
          E    True
Name: W, dtype: bool
```

```
In [26]: df[df['W']>0]
```

```
Out[26]:
```

	W	X	Y	Z
A	2.706850	0.628133	0.907969	0.503826
B	0.651118	-0.319318	-0.848077	0.605965
D	0.188695	-0.758872	-0.933237	0.955057
E	0.190794	1.978757	2.605967	0.683509

```
In [27]: df
```

```
Out[27]:
```

	W	X	Y	Z
A	2.706850	0.628133	0.907969	0.503826
B	0.651118	-0.319318	-0.848077	0.605965
C	-2.018168	0.740122	0.528813	-0.589001
D	0.188695	-0.758872	-0.933237	0.955057
E	0.190794	1.978757	2.605967	0.683509

```
In [28]: df[df['Z']<0]
```

```
Out[28]:
```

	W	X	Y	Z
C	-2.018168	0.740122	0.528813	-0.589001

```
In [29]: resultdf=df[df['Z']<0]
```

```
In [30]: resultdf['X']
```

```
Out[30]: C    0.740122
Name: X, dtype: float64
```

```
In [31]: df[df['Z']>0]['X']
```

```
Out[31]: A    0.628133
B   -0.319318
D   -0.758872
E    1.978757
Name: X, dtype: float64
```

Figure 2.66

- You can use the two methods, but the second method is faster and does not require more space

```
In [34]: boolser=df['W']>0
result=df[boolser]
mycol=['X','Y']
result[mycol]
```

Out[34]:

	X	Y
A	0.628133	0.907969
B	-0.319318	-0.848077
D	-0.758872	-0.933237
E	1.978757	2.605967

```
In [35]: df[df['W']>0][['X','Y']]
```

Out[35]:

	X	Y
A	0.628133	0.907969
B	-0.319318	-0.848077
D	-0.758872	-0.933237
E	1.978757	2.605967

Figure 2.67

- The interpreter cannot compare logical operators more than two series at the same time so we refer to Bitwise Operators

- Logical operators – AND

```
In [36]: df[(df['W']>0) and (df['Y']>0) ]
```

```
-----
ValueError                                Traceback (most recent call last)
<ipython-input-36-0526fa1b9c77> in <module>
----> 1 df[(df['W']>0) and (df['Y']>0) ]
```

```
C:\ProgramData\Anaconda3\lib\site-packages\pandas\core\generic.py in __nonzero__(self)
    1477     def __nonzero__(self):
    1478         raise ValueError(
-> 1479             "The truth value of a {type(self).__name__} is ambiguous. "
    1480             "Use a.empty, a.bool(), a.item(), a.any() or a.all()."
    1481         )
```

```
ValueError: The truth value of a Series is ambiguous. Use a.empty, a.bool(), a.item(), a.any() or a.all().
```

Figure 2.68

- Bitwise Operators – Ampersand

```
In [37]: df[(df['W']>0) & (df['Y']>0) ]
```

```
Out[37]:
```

	W	X	Y	Z
A	2.706850	0.628133	0.907969	0.503826
E	0.190794	1.978757	2.605967	0.683509

Figure 2.69

- Bitwise Operators – Pipe

```
In [38]: df[(df['W']>0) | (df['Y']>0) ]
```

```
Out[38]:
```

	W	X	Y	Z
A	2.706850	0.628133	0.907969	0.503826
B	0.651118	-0.319318	-0.848077	0.605965
C	-2.018168	0.740122	0.528813	-0.589001
D	0.188695	-0.758872	-0.933237	0.955057
E	0.190794	1.978757	2.605967	0.683509

Figure 2.70

- Logical operators – OR

```
In [39]: df[(df['W']>0) or (df['Y']>0)]
```

```
-----
ValueError                                Traceback (most recent call last)
<ipython-input-39-9f60e210bf65> in <module>
----> 1 df[(df['W']>0) or (df['Y']>0)]
```

```
C:\ProgramData\Anaconda3\lib\site-packages\pandas\core\generic.py in __nonzero__(self)
    1477     def __nonzero__(self):
    1478         raise ValueError(
-> 1479             f"The truth value of a {type(self).__name__} is ambiguous. "
    1480             "Use a.empty, a.bool(), a.item(), a.any() or a.all()."
    1481     )
```

```
ValueError: The truth value of a Series is ambiguous. Use a.empty, a.bool(), a.item(), a.any() or a.all().
```

Figure 2.71

- Reset index as column

```
In [41]: df.reset_index()
```

```
Out[41]:
```

	index	W	X	Y	Z
0	A	2.706850	0.628133	0.907969	0.503826
1	B	0.651118	-0.319318	-0.848077	0.605965
2	C	-2.018168	0.740122	0.528813	-0.589001
3	D	0.188695	-0.758872	-0.933237	0.955057
4	E	0.190794	1.978757	2.605967	0.683509

Figure 2.72

- Set index as column of dataframe

```
In [36]: newind='CA NY OR CO WR'.split()
```

```
In [37]: df['States']=newind
```

```
In [39]: df.set_index('States')
```

```
Out[39]:
```

	W	X	Y	Z
States				
CA	2.706850	0.628133	0.907969	0.503826
NY	0.651118	-0.319318	-0.848077	0.605965
OR	-2.018168	0.740122	0.528813	-0.589001
CO	0.188695	-0.758872	-0.933237	0.955057
WR	0.190794	1.978757	2.605967	0.683509

Figure 2.73

- Multi-index and index hierarchy

- How to work with the index

Let us go over how to work with Multi-Index, first we'll create a quick example of what a Multi-Indexed DataFrame would look like:

```
In [2]: import pandas as pd
import numpy as np

# Index Levels
outside = ['G1','G1','G1','G2','G2','G2']
inside = [1,2,3,1,2,3]
hier_index = list(zip(outside,inside))
hier_index = pd.MultiIndex.from_tuples(hier_index)
```

```
In [ ]: | I
```

Figure 2.74

- How to work an index hierarchy

```

hier_index = pd.MultiIndex.from_tuples(hier_index)

In [3]: outside
Out[3]: ['G1', 'G1', 'G1', 'G2', 'G2', 'G2']

In [4]: inside
Out[4]: [1, 2, 3, 1, 2, 3]

In [ ]:

```

Figure 2.75

- Illustration the List(outside,inside)

```

In [5]: hier_index
Out[5]: MultiIndex(levels=[[ 'G1', 'G2'], [1, 2, 3]],
                   labels=[[0, 0, 0, 1, 1, 1], [0, 1, 2, 0, 1, 2]])

In [6]: list(zip(outside,inside))
Out[6]: [('G1', 1), ('G1', 2), ('G1', 3), ('G2', 1), ('G2', 2), ('G2', 3)]

In [ ]:

```

Figure 2.76

- Hierarchical multiIndex



```

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3
In [6]: list(zip(outside,inside))
Out[6]: [('G1', 1), ('G1', 2), ('G1', 3), ('G2', 1), ('G2', 2), ('G2', 3)]

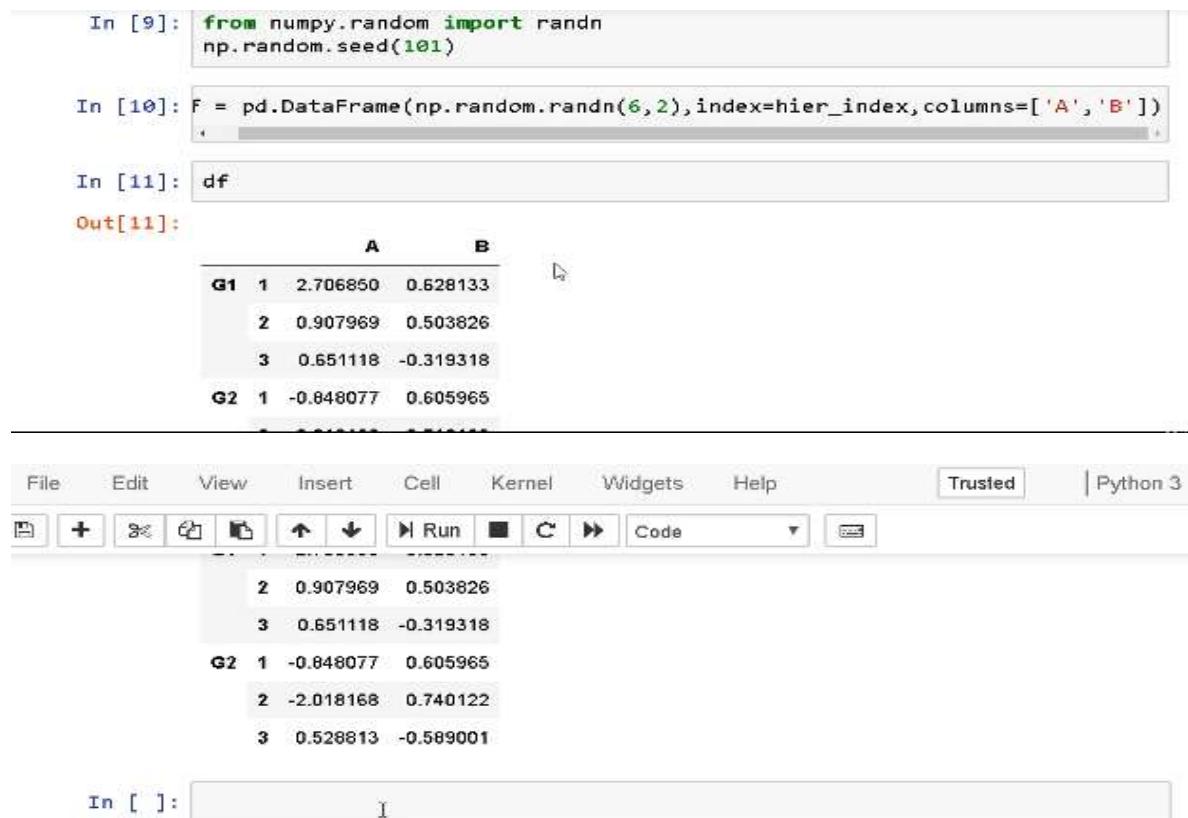
In [7]: hier_index = pd.MultiIndex.from_tuples(hier_index)

In [8]: hier_index
Out[8]: MultiIndex(levels=[[ 'G1', 'G2'], [1, 2, 3]],
                   labels=[[0, 0, 0, 1, 1, 1], [0, 1, 2, 0, 1, 2]])

```

Figure 2.77

- The following example shows how to print random numbers inside index:



In [9]: `from numpy.random import randn  
np.random.seed(101)`

In [10]: `F = pd.DataFrame(np.random.randn(6,2),index=hier_index,columns=['A','B'])`

In [11]: `df`

Out[11]:

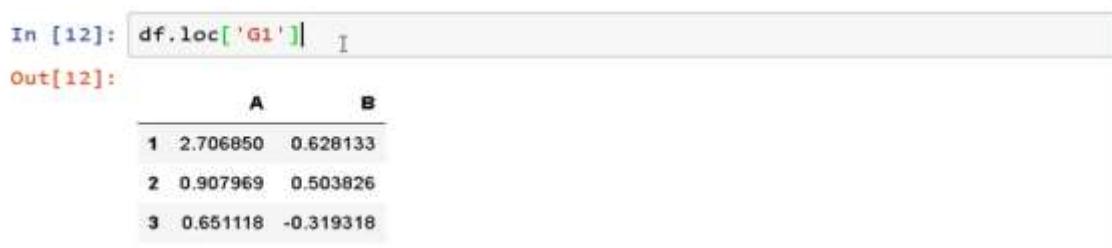
	A	B
<b>G1</b>		
1	2.706850	0.628133
2	0.907969	0.503826
3	0.651118	-0.319318
<b>G2</b>		
1	-0.848077	0.605965
2	-2.018168	0.740122
3	0.528813	-0.589001

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

In [ ]:

Figure 2.78

- Calling data, here you answer all the following data on the G1



In [12]: `df.loc['G1']`

Out[12]:

	A	B
1	2.706850	0.628133
2	0.907969	0.503826
3	0.651118	-0.319318

Figure 2.79

- Only want to answer the data in number 1. Use the loc

```
In [13]: df.loc['G1'].lpc[1]
Out[13]: A      2.706850
          B      0.628133
          Name: 1, dtype: float64
```

Figure 2.80

- Call A&B series with data and rname index

```
In [15]: df.index.names = ['Group', 'Num']
In [16]: df
Out[16]:
```

Group	Num	A	B
G1	1	2.706850	0.628133
	2	0.907969	0.503826
	3	0.651118	-0.319318

Figure 2.81

- I want to summon the number shaded in blue

Group	Num	A	B
G1	1	2.706850	0.628133
	2	0.907969	0.503826
	3	0.651118	-0.319318
G2	1	-0.848077	0.605965
	2	-2.018168	0.740122
	3	0.528813	-0.589001

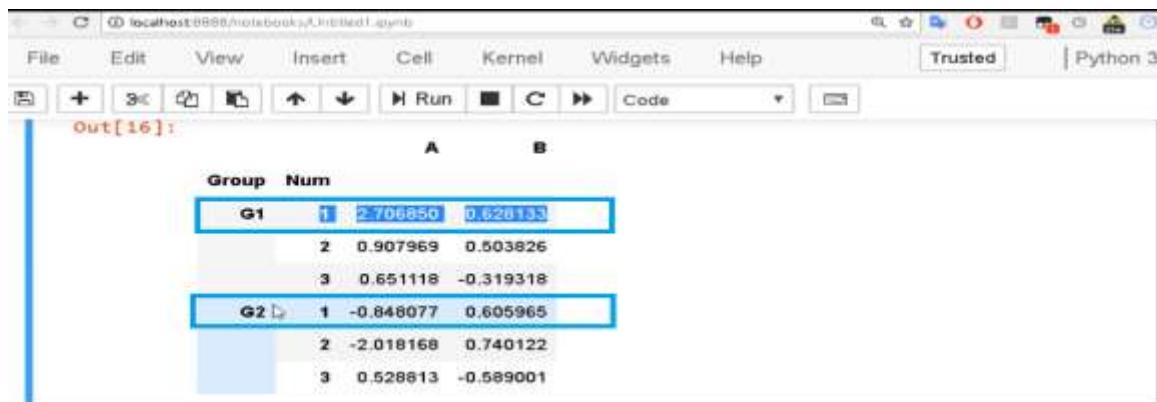
Output:

```
In [19]: df.loc['G2'].loc[2].loc['B']
Out[19]: 0.7401220570561068
```

```
In [ ]:
```

Figure 2.82

- I want to answer all the data following their number 1 in each G2 & G1 group



Group	Num	A	B
G1	1	2.706850	0.628133
	2	0.907969	0.503826
	3	0.651118	-0.319318
G2	1	-0.848077	0.605965
	2	-2.018168	0.740122
	3	0.528613	-0.589001

We well use function XS

```
In [22]: df.xs(1,level = 'Num')
Out[22]:
```

Group	A	B
G1	2.706850	0.628133
G2	-0.848077	0.605965

Figure 2.83

- Pandas missing data

- There are spaces in the table with null or Null value. I want to replace Null

Example :

**Panada - Missing Data**

```
In [2]: import pandas as pd
import numpy as np

In [4]: df = pd.DataFrame({'A':[1,2,np.nan],
                           'B':[5,np.nan,np.nan],
                           'C':[1,2,3]})

In [5]: df
Out[5]:
       A    B    C
0  1.0  5.0  1
1  2.0  NaN  2
2  NaN  NaN  3
```

Figure 2.84

Nan==null value

To remove nan we use methode dropna

Exis=0 > work on rows

```
In [6]: df.dropna()
Out[6]:
Signature: df.dropna(axis=0, how='any', thresh=None, subset=None, inplace=False)
Docstring:
Return object with labels on given axis omitted where alternately any
or all of the data are missing
```

Figure 2.85

```
In [5]: df
Out[5]:
      A      B      C
0  1.0  5.0  1
1  2.0  NaN  2
2  NaN  NaN  3

In [6]: df.dropna()
Out[6]:
      A      B      C
0  1.0  5.0  1

In [ ]:
```

Figure 2.86

Nan : In the event that the columns appear, specify the axes

Exis=1 : Inside the column is its data

```
In [5]: df
Out[5]:
      A      B      C
0  1.0  5.0  1
1  2.0  NaN  2
2  NaN  NaN  3

In [7]: df.dropna(axis=1)
Out[7]:
      C
0  1
1  2
2  3
```

Figure 2.87

Inplace=true

- You must set its value to true for this to run
- Parameter thresh:2

```
In [9]: df.dropna(thresh=2)
Out[9]:
      A      B      C
0  1.0  5.0  1
1  2.0  NaN  2
```

Figure 2.88

- Fillna method: the job is replacing

```
In [10]: df
Out[10]:
      A      B      C
0  1.0  5.0  1
1  2.0  NaN  2
2  NaN  NaN  3

In [11]: df.fillna(value = 'VALUE')
Out[11]:
      A      B      C
0    1    5    1
1    2  VALUE    2
2  VALUE  VALUE    3
```

Figure 2.89

- Calculate an average for specific values

```
In [13]: df['A'].fillna(value=df['A'].mean())
Out[13]: 0    1.0
1    2.0
2    1.5
Name: A, dtype: float64
In [ ]:
```

Figure 2.90

- Pandas-Groupby

Pandas dataframe. groupby() function is used to split the data into groups based on some criteria. pandas objects can be split on any of their axes. The abstract definition of grouping is to provide a mapping of labels to group names

- Data for a specific company and we will work on it

```
In [2]: data = {'Company': ['GOOG', 'GOOG', 'MSFT', 'MSFT', 'FB', 'FB'],
              'Person': ['Sam', 'Charlie', 'Amy', 'Vanessa', 'Carl', 'Sarah'],
              'Sales': [200, 120, 340, 124, 243, 350]}
In [3]: df = pd.DataFrame(data)
In [4]: df
Out[4]:
   Company Person  Sales
0    GOOG    Sam    200
1    GOOG  Charlie    120
2    MSFT    Amy    340
3    MSFT  Vanessa    124
4      FB    Carl    243
5      FB    Sarah    350
In [5]: df.groupby('Company')
Out[5]: <pandas.core.groupby.DataFrameGroupBy object at 0x00000000088DB908>
In [ ]:
```

Figure 2.91

- Define an object

```

jupyter Untitled Last Checkpoint: 4 hours ago (autosaved)
File Edit View Insert Cell Kernel Widgets Help
In [3]: df = pd.DataFrame(data)
In [4]: df
Out[4]:
   Company Person  Sales
0    GOOG    Sam    200
1    GOOG  Charlie   120
2    MSFT    Amy    340
3    MSFT  Vanessa    24
4      FB    Carl    243
5      FB   Sarah    350

In [6]: byComp = df.groupby("Company")
In [7]: byComp.mean()
Out[7]:
   Sales
   Company
   FB    296.5
   GOOG  160.0
   MSFT  232.0

```

Figure 2.92

- Give me the average for values except for Person, which is a string

Sum function: it collects the goods in the company

```

In [8]: byComp.sum()
Out[8]:
   Sales
   Company
   FB    593
   GOOG  320
   MSFT  464

```

Figure 2.93

- When you want to collect a certain thing that you use Loc

```
In [10]: byComp.sum().loc['FB']
Out[10]: Sales      593
          Name: FB, dtype: int64
```

Figure 2.94

- We need sum all data of company

```
In [11]: df.groupby('Company').sum().loc['FB']
Out[11]: Sales      593
          Name: FB, dtype: int64
```

Figure 2.95

- We need to find a minimum of each group

```
In [15]: df.groupby('Company').min()
Out[15]:
          Person  Sales
          Company
          FB      Carl    243
          GOOG    Charlie   120
          MSFT    Amy     124
```

Figure 2.96

- We need to find a maximum of each group

```
In [16]: df.groupby("Company").max()
```

```
Out[16]:
```

Company	Person	Sales
FB	Sarah	350
GOOG	Sam	200
MSFT	Vanessa	340

```
In [ ]:
```

Figure 2.97

- We need to describe of company

```
In [17]: df.groupby("Company").describe()
```

```
Out[17]:
```

Sales									
	count	mean	std	min	25%	50%	75%	max	
Company									
FB	2.0	296.5	75.660426	243.0	269.75	296.5	323.25	350.0	
GOOG	2.0	160.0	56.568542	120.0	140.00	160.0	180.00	200.0	
MSFT	2.0	232.0	152.735065	124.0	178.00	232.0	286.00	340.0	

```
In [ ]:
```

Figure 2.98

Transpose: inside it has a set of function

```
In [20]: df.groupby('Company').describe().transpose()
```

```
Out[20]:
```

Company	FB	GOOG	MSFT	
Sales	count	2.000000	2.000000	2.000000
	mean	296.500000	160.000000	232.000000
	std	75.660426	56.568542	152.735065
	min	243.000000	120.000000	124.000000
	25%	269.750000	140.000000	178.000000
	50%	296.500000	160.000000	232.000000
	75%	323.250000	180.000000	286.000000
	max	350.000000	200.000000	340.000000

```
In [ ]:
```

Figure 2.99

- Pandas- Merging ,Joining and concatenating

## Pandas - Merging, Joining, and Concatenating

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

In [2]: df1 = pd.DataFrame({'A': ['A0', 'A1', 'A2', 'A3'],
                           'B': ['B0', 'B1', 'B2', 'B3'],
                           'C': ['C0', 'C1', 'C2', 'C3'],
                           'D': ['D0', 'D1', 'D2', 'D3']},
                           index=[0, 1, 2, 3])

In [3]: df2 = pd.DataFrame({'A': ['A4', 'A5', 'A6', 'A7'],
                           'B': ['B4', 'B5', 'B6', 'B7'],
                           'C': ['C4', 'C5', 'C6', 'C7'],
                           'D': ['D4', 'D5', 'D6', 'D7']},
                           index=[4, 5, 6, 7])

In [4]: df3 = pd.DataFrame({'A': ['A8', 'A9', 'A10', 'A11'],
                           'B': ['B8', 'B9', 'B10', 'B11'],
                           'C': ['C8', 'C9', 'C10', 'C11'],
                           'D': ['D8', 'D9', 'D10', 'D11']},
                           index=[8, 9, 10, 11])

In [ ]:
```

Figure 2.100

Have 3 data frame:

```
Out[9]:
```

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

```
In [10]: df2
```

```
Out[10]:
```

	A	B	C	D
4	A4	B4	C4	D4
5	A5	B5	C5	D5
6	A6	B6	C6	D6
7	A7	B7	C7	D7

```
In [11]: df3
```

```
Out[11]:
```

	A	B	C	D
8	A8	B8	C8	D8
9	A9	B9	C9	D9
10	A10	B10	C10	D10
11	A11	B11	C11	D11

Figure 2.101

- We want to concat these 3 of the data frame in one data frame by rows:

```
In [12]: pd.concat([df1,df2,df3])
Out[12]:
   A   B   C   D
0  A0  B0  C0  D0
1  A1  B1  C1  D1
2  A2  B2  C2  D2
3  A3  B3  C3  D3
4  A4  B4  C4  D4
5  A5  B5  C5  D5
6  A6  B6  C6  D6
7  A7  B7  C7  D7
8  A8  B8  C8  D8
9  A9  B9  C9  D9
10 A10 B10 C10 D10
11 A11 B11 C11 D11
```

Figure 2.102

- We want to concat these 3 of the data frame in one data frame by columns:

```
In [13]: pd.concat([df1,df2,df3], axis=1)
Out[13]:
   A   B   C   D   A   B   C   D   A   B   C   D
0  A0  B0  C0  D0  NaN  NaN  NaN  NaN  NaN  NaN  NaN  NaN
1  A1  B1  C1  D1  NaN  NaN  NaN  NaN  NaN  NaN  NaN  NaN
2  A2  B2  C2  D2  NaN  NaN  NaN  NaN  NaN  NaN  NaN  NaN
3  A3  B3  C3  D3  NaN  NaN  NaN  NaN  NaN  NaN  NaN  NaN
4  NaN  NaN  NaN  NaN  A4  B4  C4  D4  NaN  NaN  NaN  NaN
5  NaN  NaN  NaN  NaN  A5  B5  C5  D5  NaN  NaN  NaN  NaN
6  NaN  NaN  NaN  NaN  A6  B6  C6  D6  NaN  NaN  NaN  NaN
7  NaN  NaN  NaN  NaN  A7  B7  C7  D7  NaN  NaN  NaN  NaN
8  NaN  NaN  NaN  NaN  NaN  NaN  NaN  NaN  A8  B8  C8  D8
9  NaN  NaN  NaN  NaN  NaN  NaN  NaN  NaN  A9  B9  C9  D9
10 NaN  NaN  NaN  NaN  NaN  NaN  NaN  NaN  A10 B10 C10 D10
11 NaN  NaN  NaN  NaN  NaN  NaN  NaN  NaN  A11 B11 C11 D11
```

Figure 2.103

- Here 2 data frame left& right:

```
In [15]: left = pd.DataFrame({'key': ['K0', 'K1', 'K2', 'K3'],
                             'A': ['A0', 'A1', 'A2', 'A3'],
                             'B': ['B0', 'B1', 'B2', 'B3']})

right = pd.DataFrame({'key': ['K0', 'K1', 'K2', 'K3'],
                      'C': ['C0', 'C1', 'C2', 'C3'],
                      'D': ['D0', 'D1', 'D2', 'D3']})
```

In [ ]: lef

Figure 2.104

Left:

```
In [16]: left
```

Out[16]:

	A	B	key
0	A0	B0	K0
1	A1	B1	K1
2	A2	B2	K2
3	A3	B3	K3

Figure 2.105

Right:

```
In [17]: right
```

Out[17]:

	C	D	key
0	C0	D0	K0
1	C1	D1	K1
2	C2	D2	K2
3	C3	D3	K3

Figure 2.106

## Join Left and Right:

	A	B	key	C	D
0	A0	B0	K0	C0	D0
1	A1	B1	K1	C1	D1
2	A2	B2	K2	C2	D2
3	A3	B3	K3	C3	D3

```
In [ ]: left = pd.DataFrame({'key1': ['K0', 'K0', 'K1', 'K2'],
   'key2': ['K0', 'K1', 'K0', 'K1'],
   'A': ['A0', 'A1', 'A2', 'A3'],
   'B': ['B0', 'B1', 'B2', 'B3']})

right = pd.DataFrame({'key1': ['K0', 'K1', 'K1', 'K2'],
   'key2': ['K0', 'K0', 'K0', 'K0'],
   'C': ['C0', 'C1', 'C2', 'C3'],
   'D': ['D0', 'D1', 'D2', 'D3']})
```

Figure 2.107

## SQL Joins

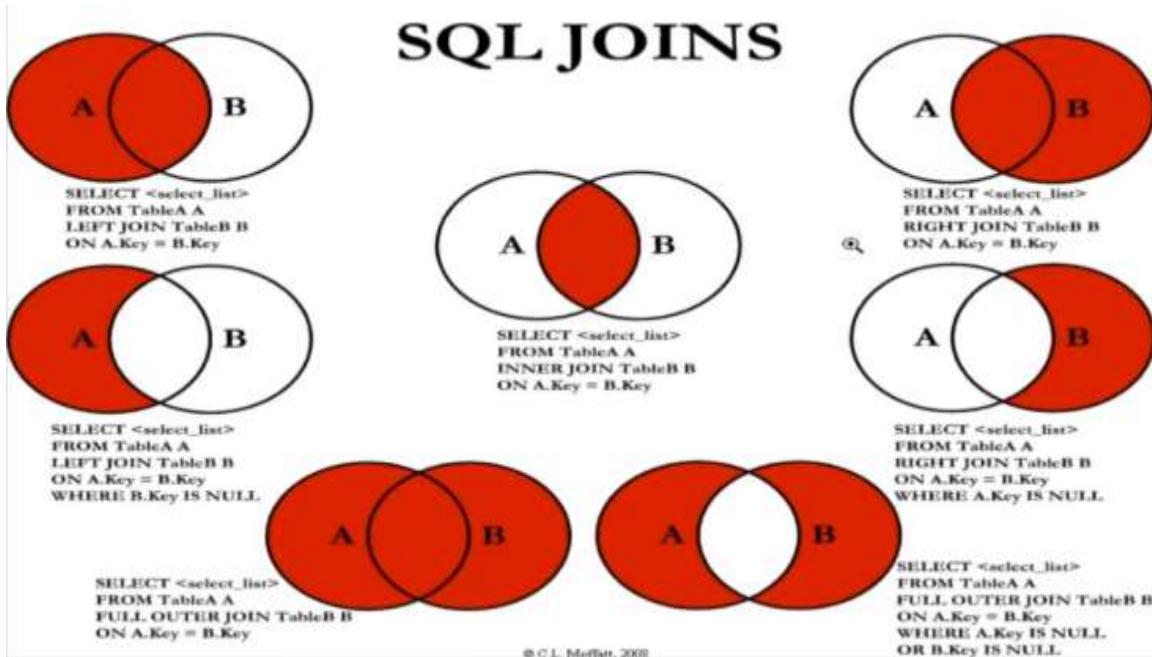


Figure 2.108

- Left and Right

```
In [20]: left
Out[20]:
      A   B  key1  key2
0  A0  B0    K0    K0
1  A1  B1    K0    K1
2  A2  B2    K1    K0
3  A3  B3    K2    K1

In [21]: right
Out[21]:
      C   D  key1  key2
0  C0  D0    K0    K0
1  C1  D1    K1    K0
2  C2  D2    K1    K0
3  C3  D3    K2    K0
```

Figure 2.109

- First type of join

```
In [22]: pd.merge(left, right, on=['key1', 'key2'])
Out[22]:
   A   B  key1  key2   C   D
0  A0  B0    K0    K0  C0  D0
1  A2  B2    K1    K0  C1  D1
2  A2  B2    K1    K0  C2  D2

In [23]: pd.merge(left, right, how='outer', on=['key1', 'key2'])
Out[23]:
   A   B  key1  key2      C      D
0  A0  B0    K0    K0  C0  D0
1  A1  B1    K0    K1  NaN  NaN
2  A2  B2    K1    K0  C1  D1
3  A2  B2    K1    K0  C2  D2
4  A3  B3    K2    K1  NaN  NaN
5  NaN  NaN    K2    K0  C3  D3
```

Figure 2.110

- Second type of join

```
In [24]: pd.merge(left, right, how='right', on=['key1', 'key2'])
Out[24]:
   A   B  key1  key2   C   D
0  A0  B0    K0    K0  C0  D0
1  A2  B2    K1    K0  C1  D1
2  A2  B2    K1    K0  C2  D2
3  NaN  NaN    K2    K0  C3  D3

In [25]: left = pd.DataFrame({'A': ['A0', 'A1', 'A2'],
                           'B': ['B0', 'B1', 'B2'],
                           index=['K0', 'K1', 'K2'])

right = pd.DataFrame({'C': ['C0', 'C2', 'C3'],
                      'D': ['D0', 'D2', 'D3']},
                      index=['K0', 'K2', 'K3'])

In [26]: left
Out[26]:
   A   B
K0  A0  B0
K1  A1  B1
K2  A2  B2
```

Figure 2.111

- Third type of join

```
In [27]: right
Out[27]:
      C   D
K0  C0  D0
K2  C2  D2
K3  C3  D3

In [28]: left.join(right)
Out[28]:
      A   B   C   D
K0  A0  B0  C0  D0
K1  A1  B1  NaN  NaN
K2  A2  B2  C2  D2
```

---

```
In [ ]:
```

Figure 2.112

- Concatenation of data frame

```
In [34]: pd.concat([df1,df2,df3], axis=1)
Out[34]:
      A   B   C   D   A   B   C   D   A   B   C   D
0   A0  B0  C0  D0  NaN  NaN  NaN  NaN  NaN  NaN  NaN  NaN
1   A1  B1  C1  D1  NaN  NaN  NaN  NaN  NaN  NaN  NaN  NaN
2   A2  B2  C2  D2  NaN  NaN  NaN  NaN  NaN  NaN  NaN  NaN
3   A3  B3  C3  D3  NaN  NaN  NaN  NaN  NaN  NaN  NaN  NaN
4   NaN  NaN  NaN  NaN  A4  B4  C4  D4  NaN  NaN  NaN  NaN
5   NaN  NaN  NaN  NaN  A5  B5  C5  D5  NaN  NaN  NaN  NaN
6   NaN  NaN  NaN  NaN  A6  B6  C6  D6  NaN  NaN  NaN  NaN
7   NaN  NaN  NaN  NaN  A7  B7  C7  D7  NaN  NaN  NaN  NaN
8   NaN  NaN  NaN  NaN  NaN  NaN  NaN  NaN  A8  B8  C8  D8
9   NaN  NaN  NaN  NaN  NaN  NaN  NaN  NaN  A9  B9  C9  D9
10  NaN  NaN  NaN  NaN  NaN  NaN  NaN  NaN  A10  B10  C10  D10
11  NaN  NaN  NaN  NaN  NaN  NaN  NaN  NaN  A11  B11  C11  D11
```

---

Figure 2.113

- Pandas-Operation
- How to retrieve unique value from the data frame

**Panada - Operations**

```
In [2]: import pandas as pd
import numpy as np

In [3]: df = pd.DataFrame({'col1':[1,2,3,4], 'col2':[444,555,666,444], 'col3':['abc','def','ghi','xyz']})

In [4]: df

Out[4]:
   col1  col2  col3
0     1    444    abc
1     2    555    def
2     3    666    ghi
3     4    444    xyz
```

Figure 2.114

- Unique method: Responds to non-repeated values

```
In [4]: df

Out[4]:
   col1  col2  col3
0     1    444    abc
1     2    555    def
2     3    666    ghi
3     4    444    xyz

In [6]: len(df['col2'].unique())
Out[6]: array([444, 555, 666], dtype=int64)

In [ ]:
```

Figure 2.115

- `nunique` method: Gives the number of non-repeated values

```
Out[4]:
```

	col1	col2	col3
0	1	444	abc
1	2	555	def
2	3	666	ghi
3	4	444	xyz

```
In [10]: df['col2'].nunique()
```

```
Out[10]: 3
```

Figure 2.116

- `Value counts` : It counts iterating values

```
In [13]: df['col2'].value_counts()
```

```
Out[13]: 444    2
555    1
666    1
Name: col2, dtype: int64
```

```
In [ ]:
```

Figure 2.117

- There are ways to determine the data
- 3 forms for determining data
- First form

```
In [20]: df[df['col1'] > 2]
```

```
Out[20]:
```

	col1	col2	col3
2	3	666	ghi
3	4	444	xyz

```
In [ ]:
```

Figure 2.118

- Second form

```
In [23]: df[(df['col1']>2) & (df['col2']==444)]
```

```
Out[23]:
```

	col1	col2	col3
3	4	444	xyz

```
In [ ]:
```

Figure 2.119

- Third form

```
In [24]: df[(df['col1']>2)& (df['col2']==444)]
```

```
Out[24]:
```

	col1	col2	col3
2	3	666	ghi
3	4	444	xyz

Figure 2.120

- Apply method: Square items, length

```
In [26]: def times2(x):  
    return x**2
```

```
In [29]: df['col1'].apply(times2)
```

```
Out[29]: 0    2  
1    4  
2    6  
3    8  
Name: col1, dtype: int64
```

```
In [ ]:
```

Figure 2.121

```
In [30]: df['col3'].apply(len)  
Out[30]: 0    3  
1    3  
2    3  
3    3  
Name: col3, dtype: int64
```

```
In [ ]:
```

Figure 2.122

```
In [31]: df['col1'].apply(lambda x: x**2)  
Out[31]: 0    2  
1    4  
2    6  
3    8  
Name: col1, dtype: int64
```

```
In [ ]:
```

Figure 2.123

- Inplace : cancel the column

```
In [36]: df.drop('col1', axis = 1, inplace= True)  
Out[36]:  
   col1  col2  col3  
0    1    444  abc  
1    2    555  def  
2    3    666  ghi  
3    4    444  xyz
```

Figure 2.124

- Some of class attribute

```

In [37]: df.columns
Out[37]: Index([u'col1', u'col2', u'col3'], dtype='object')

In [38]: df
Out[38]:
   col1  col2  col3
0     1    444    abc
1     2    555    def
2     3    666    ghi
3     4    444    xyz

In [39]: df.index
Out[39]: RangeIndex(start=0, stop=4, step=1)

In [40]: df.sort_values('col2')
Out[40]:
   col1  col2  col3
3     4    444    xyz
1     2    555    def
2     3    666    ghi
0     1    444    abc

```

Figure 2.125

- Pivot table: Group

```

In [43]: data = {'A': ['foo', 'foo', 'foo', 'bar', 'bar', 'bar'],
              'B': ['one', 'one', 'two', 'two', 'one', 'one'],
              'C': ['x', 'y', 'x', 'y', 'x', 'y'],
              'D': [1, 3, 2, 5, 4, 1]}
df = pd.DataFrame(data)

In [44]: df
Out[44]:
   A    B    C    D
0  foo  one  x  1
1  foo  one  y  3
2  foo  two  x  2
3  bar  two  y  5
4  bar  one  x  4
5  bar  one  y  1

```

Figure 2.126

```
In [46]: df.pivot_table(values='D', index=['A', 'B'], columns=['C'])  
Out[46]:  
      C   x   y  
A B  
bar one  4.0  1.0  
     two  NaN  5.0  
foo one  1.0  3.0  
     two  2.0  NaN
```

Figure 2.127

- Pandas-data input&output



- CSV
- Excel
- HTML
- SQL

Figure 2.128

- Pwd : By giving you the following file that you are working on

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

In [2]: pd
Out[2]: u'C:\\\\Users\\\\tamer\\\\Desktop\\\\Python-Data-Science-and-Machine-Learning-Bootcamp\\\\Python-Data-Science-and-Machine-Learning-Bootcamp\\\\Python-for-Data-Analysis\\\\Pandas'

In [ ]:
```

Figure 2.129

- Read file CSV

```
In [5]: pd.read_csv('example')

Out[5]:
   a   b   c   d
0  0   1   2   3
1  4   5   6   7
2  8   9  10  11
3 12  13  14  15
```

Figure 2.130

---

# CHAPTER 3

---

## Data Reprocessing

### 3.1 Overview

Data preprocessing in Machine Learning is a crucial step that helps enhance the quality of data to promote the extraction of meaningful insights from the data. Data preprocessing in Machine Learning refers to the technique of preparing (cleaning and organizing) the raw data to make it suitable for a building and training Machine Learning models. In simple words, data preprocessing in Machine Learning is a data mining technique that transforms raw data into an understandable and readable format.

### Why Data Preprocessing in Machine Learning?

When it comes to creating a Machine Learning model, data preprocessing is the first step marking the initiation of the process. Typically, real-world data is incomplete, inconsistent, inaccurate (contains errors or outliers), and often lacks specific attribute values/trends. This is where data preprocessing enters the scenario – it helps to clean, format, and organize the raw data, thereby making it ready-to-go for Machine Learning models.

## 3.2 Steps in data preprocessing in machine learning

It involves below steps:

1. Getting the dataset
2. Importing libraries
3. Importing datasets
4. Finding Missing Data
5. Encoding Categorical Data
6. Splitting dataset into training and test set
7. Feature scaling

### 1. Get the Dataset

To create a machine learning model, the first thing we required is a dataset as a machine learning model completely works on data. The collected data for a particular problem in a proper format is known as the dataset.

Dataset may be of different formats for different purposes, such as, if we want to create a machine learning model for business purpose, then dataset will be different with the dataset required for a liver patient. So each dataset is different from another dataset. To use the dataset in our code, we usually put it into a CSV file. However, sometimes, we may also need to use an HTML or xlsx file.

## What is a CSV File?

CSV stands for "**Comma-Separated Values**" files; it is a file format which allows us to save the tabular data, such as spreadsheets. It is useful for huge datasets and can use these datasets in programs.

## 2. Importing Libraries

In order to perform data preprocessing using Python, we need to import some predefined Python libraries. These libraries are used to perform some specific jobs. There are three specific libraries that we will use for data preprocessing, which are:

**Numpy:** Numpy Python library is used for including any type of mathematical operation in the code. It is the fundamental package for scientific calculation in Python. It also supports to add large, multidimensional arrays and matrices. So, in Python, we can import it as:

```
1. import numpy as nm
```

Figure 3.1

Here we have used **nm**, which is a short name for Numpy, and it will be used in the whole program.

**Matplotlib:** The second library is **matplotlib**, which is a Python 2D plotting library, and with this library, we need to import a sub-library **pyplot**. This library is used to plot any type of charts in Python for the code. It will be imported as below:

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

Figure 3.2

Here we have used plt as a short name for this library.

**Pandas:** The last library is the Pandas library, which is one of the most famous Python libraries and used for importing and managing the datasets. It is an open-source data manipulation and analysis library. It will be imported as below:

Here, we have used pd as a short name for this library. Consider the below image:

```
1. import numpy as np
2. import matplotlib.pyplot as plt
3. import pandas as pd
```

Figure 3.3

### 3. Importing the Datasets

Now we need to import the datasets which we have collected for our machine learning project. But before importing a dataset, we need to set the current directory as a working directory. To set a working directory in Spyder IDE, we need to follow the below steps:

1. Save your Python file in the directory which contains dataset.
2. Go to File explorer option in Spyder IDE, and select the required directory.
3. Click on F5 button or run option to execute the file.

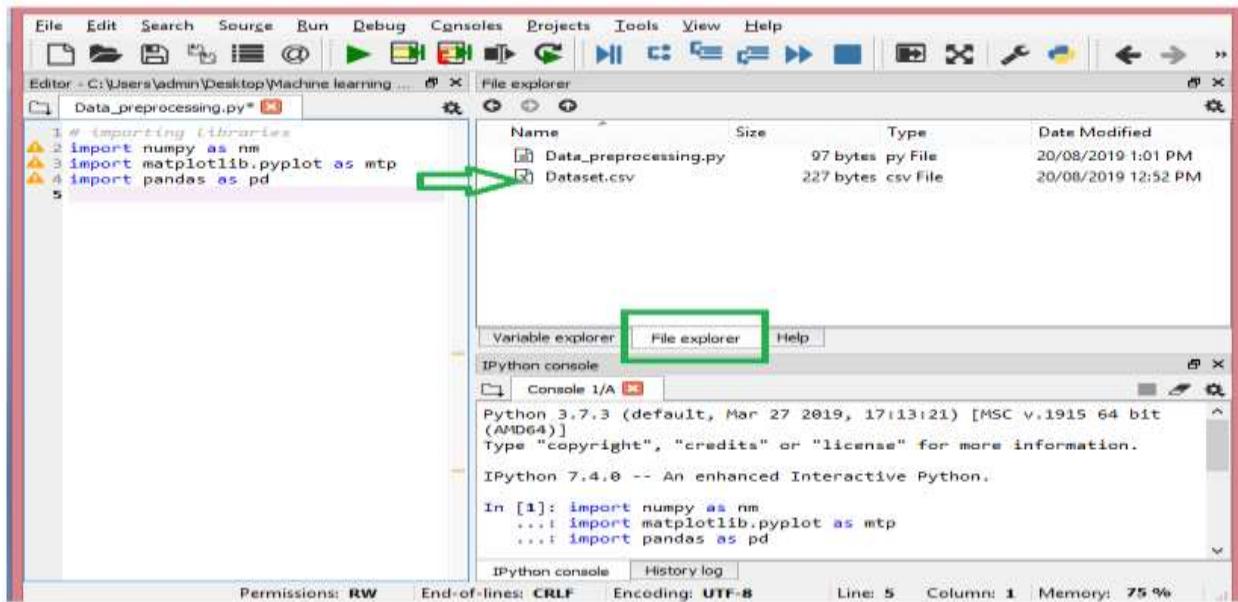


Figure 3.4

### `read_csv()` function:

Now to import the dataset, we will use `read_csv()` function of pandas library, which is used to read a csv file and performs various operations on it. Using this function, we can read a csv file locally as well as through an URL.

We can use `read_csv` function as below:

```
1. dataset = pd.read_csv('Data.csv')
```

Figure 3.5

Here, `data_set` is a name of the variable to store our dataset, and inside the function, we have passed the name of our dataset. Once we execute the above line of code, it will successfully import the dataset in our code. We can also check the imported dataset by clicking on the section **variable explorer**, and then double click on `data_set`. Consider the below image:

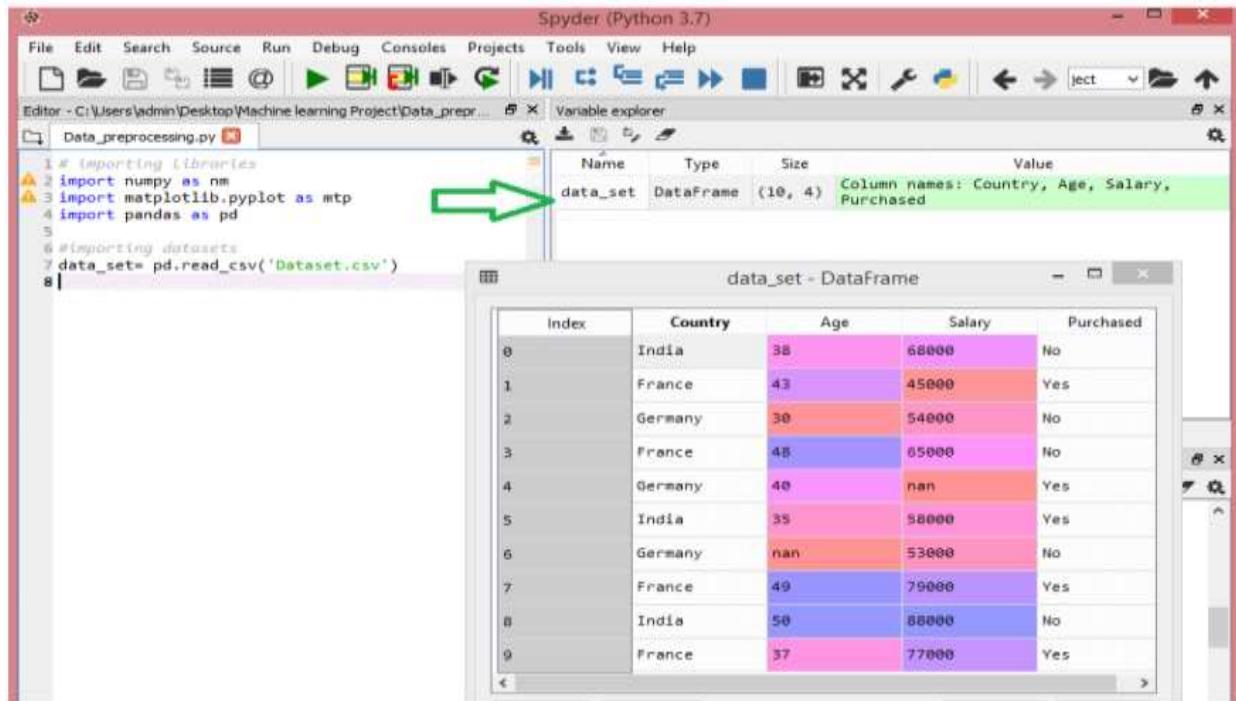


Figure 3.6

As in the above image, indexing is started from 0, which is the default indexing in Python. We can also change the format of our dataset by clicking on the format option.

### Extracting dependent and independent variables:

In machine learning, it is important to distinguish the matrix of features (independent variables) and dependent variables from dataset. In our dataset, there are three independent variables that are **Country**, **Age**, and **Salary**, and one is a dependent variable which is **Purchased**.

### Extracting independent variable:

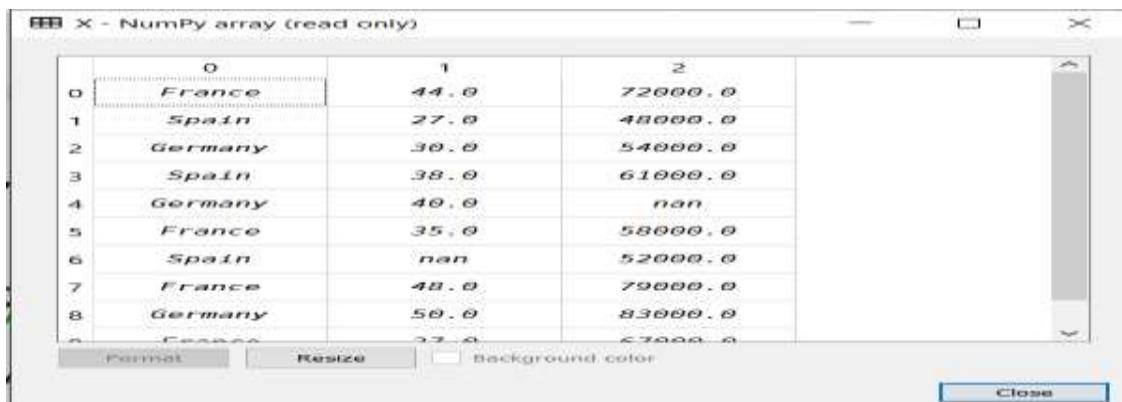
To extract an independent variable, we will use `iloc[ ]` method of Pandas library. It is used to extract the required rows and columns from the dataset.

```
1. X = dataset.iloc[:, :-1].values
```

Figure 3.7

In the above code, the first colon(:) is used to take all the rows, and the second colon(:) is for all the columns. Here we have used `:-1`, because we don't want to take the last column as it contains the dependent variable. So by doing this, we will get the matrix of features.

By executing the above code, we will get output as:



X - NumPy array (read only)			
	0	1	2
0	France	44.0	72000.0
1	Spain	27.0	48000.0
2	Germany	30.0	54000.0
3	Spain	38.0	61000.0
4	Germany	40.0	nan
5	France	35.0	58000.0
6	Spain	nan	52000.0
7	France	48.0	79000.0
8	Germany	50.0	83000.0
9	France	32.0	67000.0

Figure 3.8

As we can see in the above output, there are only three variables.

### Extracting dependent variable:

To extract dependent variables, again, we will use Pandas `.iloc[ ]` method.

```
1. y = dataset.iloc[:, -1].values
```

Figure 3.9

Here we have taken all the rows with the last column only. It will give the array of dependent variables.

By executing the above code, we will get output as:

### Output:



	0
0	0
1	Yes
2	No
3	No
4	Yes
5	Yes
6	No
7	Yes
8	No
9	Yes

Figure 3.10

## 4. Handling Missing data:

The next step of data preprocessing is to handle missing data in the datasets. If our dataset contains some missing data, then it may create a huge problem for our machine learning model. Hence it is necessary to handle missing values present in the dataset.

### Ways to handle missing data:

There are mainly two ways to handle missing data, which are:

**By deleting the particular row:** The first way is used to commonly deal with null values. In this way, we just delete the specific row or column which consists of null values. But this way is not so efficient and removing data may lead to loss of information which will not give the accurate output.

**By calculating the mean:** In this way, we will calculate the mean of that column or row which contains any missing value and will put it on the place of missing value. This strategy is useful for the features which have numeric data such as age, salary, year, etc. Here, we will use this approach.

To handle missing values, we will use **Scikit-learn** library in our code, which contains various libraries for building machine learning models. Here we will use **Imputer** class of **sklearn.preprocessing** library. Below is the code for it:

```
1. #handling missing data (Replacing missing data with the mean value)
2.
3. from sklearn.impute import SimpleImputer
4.
5. imputer = SimpleImputer(missing_values=np.nan, strategy='mean')
6.
7. #Fitting imputer object to the independent variables x
8.
9. imputer.fit(X[:, 1:3])
10.
11. #Replacing missing data with the calculated mean value
12.
13. X[:, 1:3] = imputer.transform(X[:, 1:3])
```

Figure 3.11

## 5. Encoding Categorical data:

Categorical data is data which has some categories such as, in our dataset; there are two categorical variable, **Country**, and **Purchased**.

Since machine learning model completely works on mathematics and numbers, but if our dataset would have a categorical variable, then it may create trouble while building the model. So it is necessary to encode these categorical variables into numbers.

**For Country variable:**

Firstly, we will convert the country variables into categorical data. So to do this, we will use **LabelEncoder()** class from **preprocessing** library.

```
1. # Encoding categorical data
2.
3. # Encoding the Independent Variable
4.
5. from sklearn.compose import ColumnTransformer
6. from sklearn.preprocessing import OneHotEncoder
7. ct = ColumnTransformer(transformers=[('encoder', OneHotEncoder(), [0])], remainder='passthrough')
8. X = np.array(ct.fit_transform(X))
9.
10. # Encoding the Dependent Variable
11.
12. from sklearn.preprocessing import LabelEncoder
13. le = LabelEncoder()
14. y = le.fit_transform(y)
```

Figure 3.12

## 6. Splitting the Dataset into the Training set and Test set

In machine learning data preprocessing, we divide our dataset into a training set and test set. This is one of the crucial steps of data preprocessing as by doing this, we can enhance the performance of our machine learning model.

Suppose, if we have given training to our machine learning model by a dataset and we test it by a completely different dataset. Then, it will create difficulties for our model to understand the correlations between the models.

If we train our model very well and its training accuracy is also very high, but we provide a new dataset to it, then it will decrease the performance. So we always try to make a machine learning model which performs well with the training set and also with the test dataset. Here, we can define these datasets as:

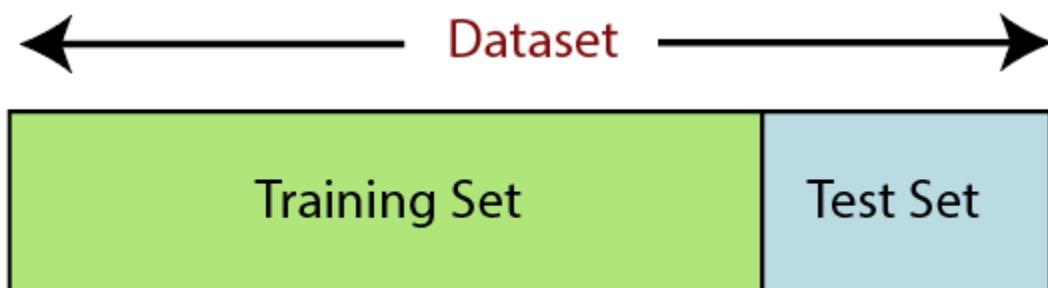


Figure 3.13

**Training Set:** A subset of dataset to train the machine learning model, and we already know the output.

**Test set:** A subset of dataset to test the machine learning model, and by using the test set, model predicts the output.

For splitting the dataset, we will use the below lines of code:

```
1. # Splitting the dataset into the Training set and Test set
2.
3. from sklearn.model_selection import train_test_split
4. X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2,
5. random_state = 1)
```

Figure 3.14

## Explanation:

In the above code, the first line is used for splitting arrays of the dataset into random train and test subsets.

In the second line, we have used four variables for our output that are

x\_train: features for the training data

x\_test: features for testing data

y\_train: Dependent variables for training data

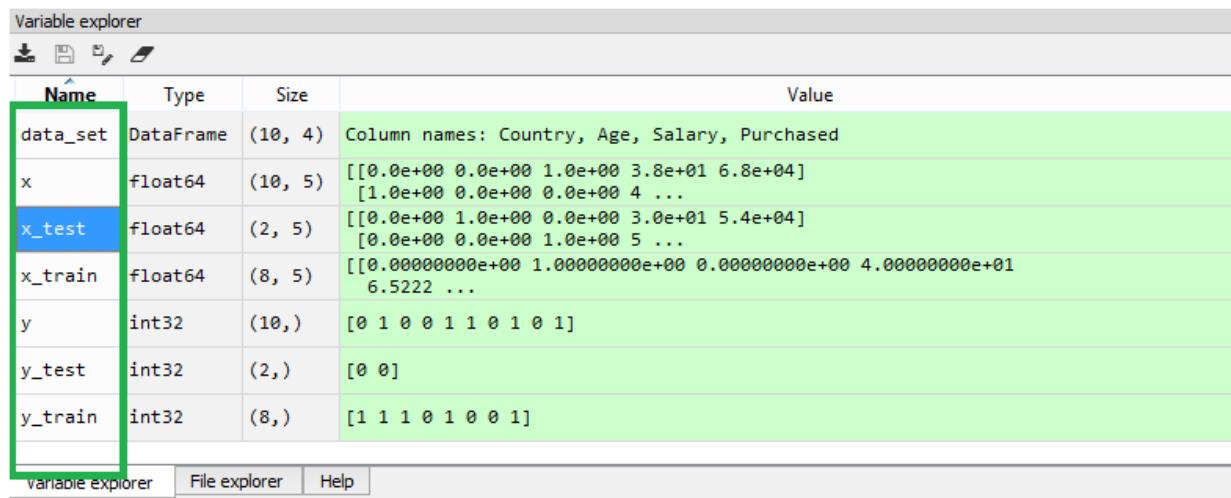
y\_test: Independent variable for testing data

In `train_test_split()` function, we have passed four parameters in which first two are for arrays of data, and `test_size` is for specifying the size of the test set. The `test_size` maybe `.5`, `.3`, or `.2`, which tells the dividing ratio of training and testing sets.

The last parameter `random_state` is used to set a seed for a random generator so that you always get the same result, and the most used value for this is `42`.

## Output:

By executing the above code, we will get 4 different variables, which can be seen under the variable explorer section.



Name	Type	Size	Value
data_set	DataFrame	(10, 4)	Column names: Country, Age, Salary, Purchased
x	float64	(10, 5)	[[0.0e+00 0.0e+00 1.0e+00 3.8e+01 6.8e+04] [1.0e+00 0.0e+00 0.0e+00 4 ...]
x_test	float64	(2, 5)	[[0.0e+00 1.0e+00 0.0e+00 3.0e+01 5.4e+04] [0.0e+00 0.0e+00 1.0e+00 5 ...]
x_train	float64	(8, 5)	[[0.0000000e+00 1.0000000e+00 0.0000000e+00 4.0000000e+01 6.5222 ...]
y	int32	(10,)	[0 1 0 0 1 1 0 1 0 1]
y_test	int32	(2,)	[0 0]
y_train	int32	(8,)	[1 1 1 0 1 0 0 1]

Figure 3.15

As we can see in the above image, the x and y variables are divided into 4 different variables with corresponding values.

## 7. Feature Scaling

Feature scaling is the final step of data preprocessing in machine learning. It is a technique to standardize the independent variables of the dataset in a specific range. In feature scaling, we put our variables in the same range and in the same scale so that no any variable dominate the other variable.

Consider the below dataset:

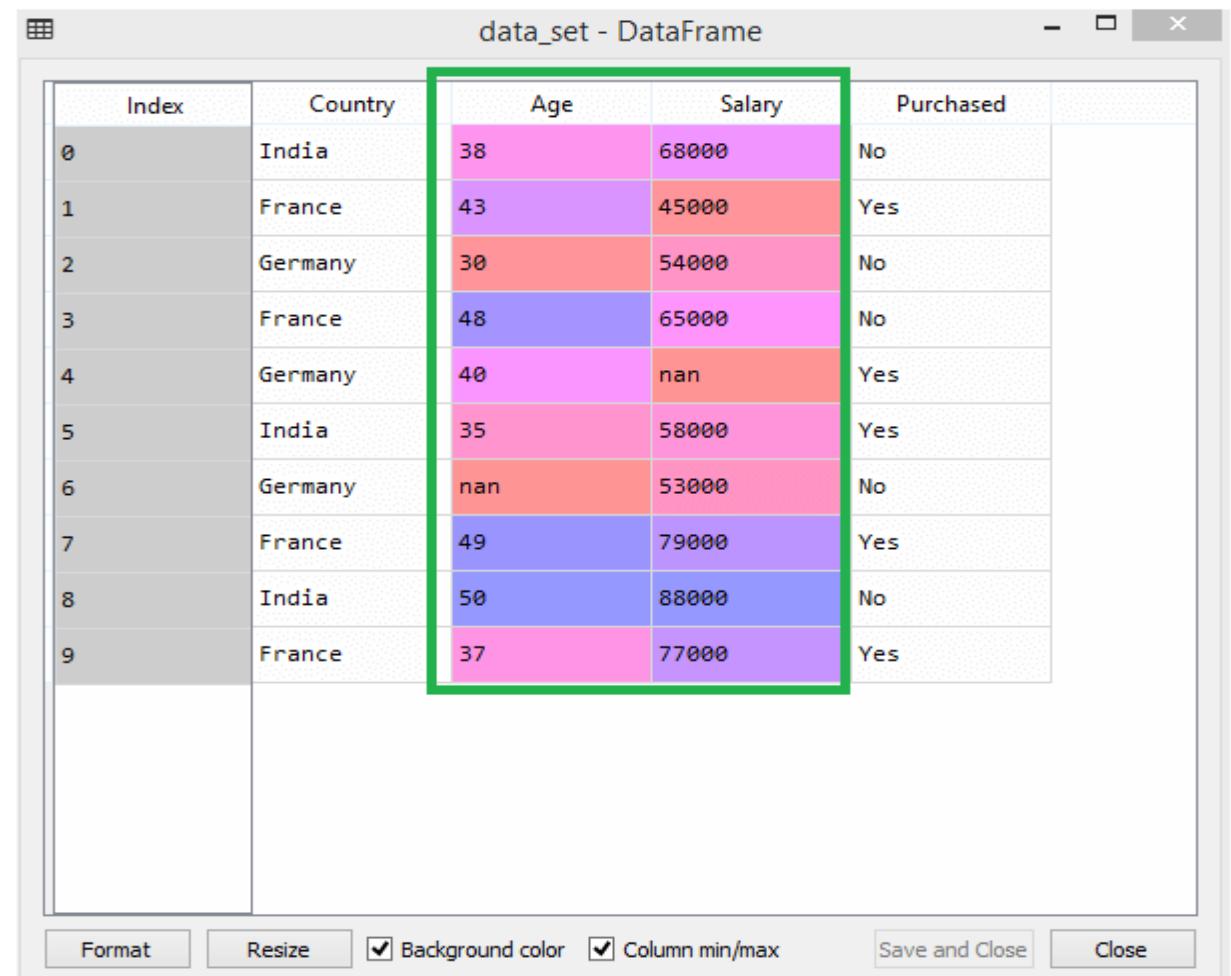
```
1. # Feature Scaling
2.
3. from sklearn.preprocessing import StandardScaler
4. sc = StandardScaler()
5. X_train[:, 3:] = sc.fit_transform(X_train[:, 3:])
```

```
6. X_test[:, 3:] = sc.transform(X_test[:, 3:])
```

Figure 3.16

## Output:

data\_set - DataFrame



Index	Country	Age	Salary	Purchased
0	India	38	68000	No
1	France	43	45000	Yes
2	Germany	30	54000	No
3	France	48	65000	No
4	Germany	40	nan	Yes
5	India	35	58000	Yes
6	Germany	nan	53000	No
7	France	49	79000	Yes
8	India	50	88000	No
9	France	37	77000	Yes

Format    Resize     Background color     Column min/max    Save and Close    Close

Figure 3.17

---

# CHAPTER 4

---

## Regression

### 4.1 Overview

Regression models (both linear and non-linear) are used for predicting a real value, like salary for example. If your independent variable is time, then you are forecasting future values, otherwise your model is predicting present but unknown values. Regression technique vary from Linear Regression to SVR and Random Forests Regression.

The primary focus of machine learning is building algorithms that can receive input data, and use statistical analysis; To predict outputs within an acceptable range.

In this part, you will understand and learn how to implement the following Machine Learning Regression models:

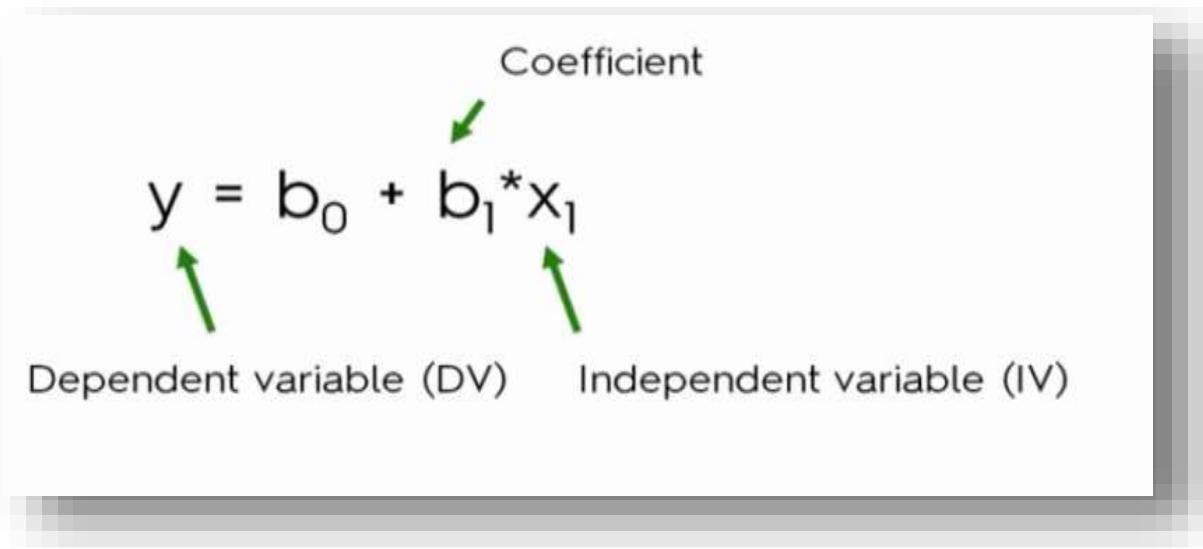
1. Simple Linear Regression
  2. Multiple Linear Regression
  3. Polynomial Regression
  4. Support Vector for Regression (SVR)
  5. Decision Tree Classification
  6. Random Forest Classification
- 
-

# CHAPTER 5

## Simple Linear Regression

### 5.1 Overview

Assume the equation given below is used to fit a straight line to our graph, people who studied statistics will be familiar with it, as it is a basic straight-line equation. So here as we can see, **b0** is constant, which is fixed, and **b1** is the score to plot the graph or its merely a coefficient.



The diagram shows the simple linear regression equation  $y = b_0 + b_1 * x_1$ . A green arrow points to the term  $b_1$  with the label "Coefficient". Another green arrow points to the term  $x_1$  with the label "Independent variable (IV)". A third green arrow points to the term  $y$  with the label "Dependent variable (DV)".

Figure 5.1

So let's see what base **b0** means in our example. Assuming every fresher in the company gets **30K** as starting salary we can set it as the base price.

### Simple Linear Regression:

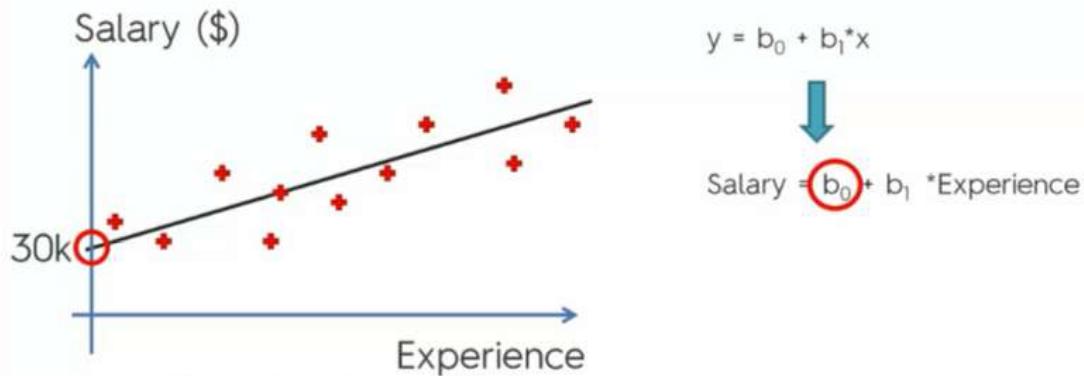


Figure 5.2

So now let's take the salary of an employee with 1-year experience so that we can see changes in the plot. We can observe an increase in pay by **10K** in the green line, so if we have an experience value, for example, take nine years for which there is no salary specified, we can easily predict.

### Simple Linear Regression:

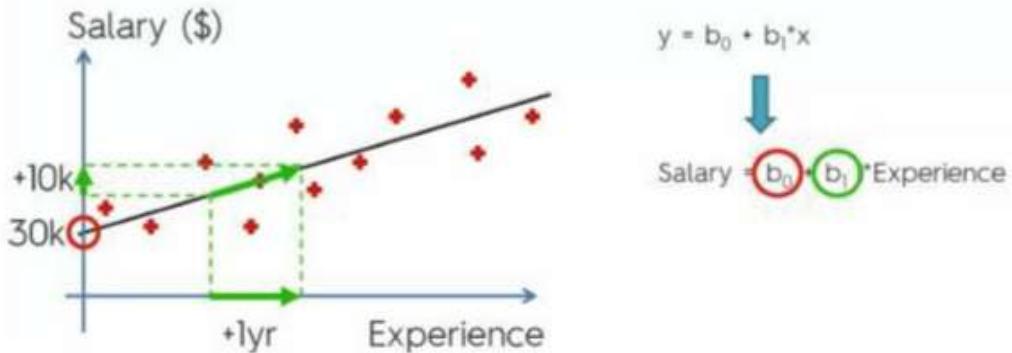


Figure 5.3

Finally, our objective is to minimize the distance between actual and observed values in the graph as we can see in the graph below if we reduce the cost more the chances of our better prediction.

### Simple Linear Regression:

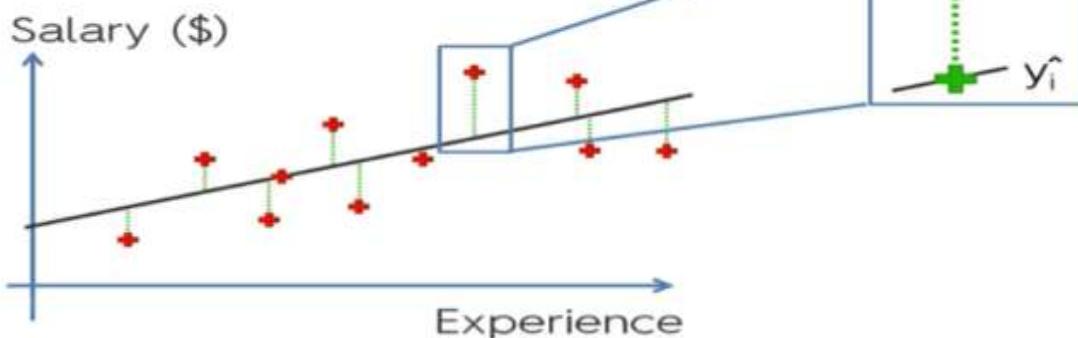


Figure 5.4

Here  $Y_i$  is the original value, while  $\hat{y}$  is the observed value. So here is the equation used to minimize the distance between Actual and Observed values. In below formula  $h_0(x_i)$  is nothing but our original cost, we take summation of squared differences and divide it by  $2m$  where  $m$  is the total number of features or rows in our data set, to get significant value.

Out of various Cost function values, the **minimum** is chosen, and its line of the graph is selected as the **Best fit Line**.

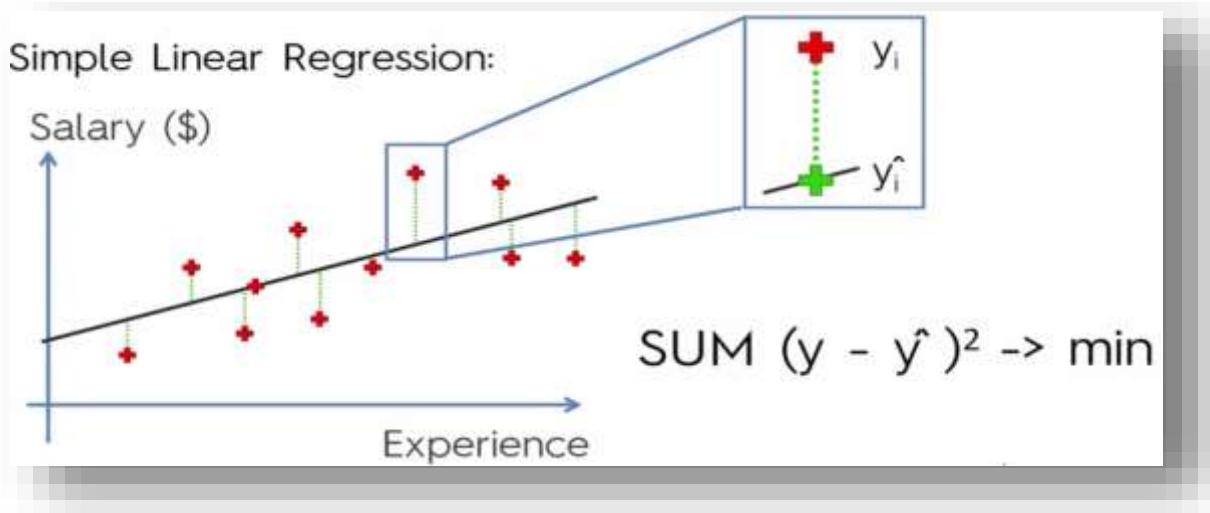


Figure 5.5

## 5.2 Simple linear regression in python

Given below is the python implementation of above technique on our small dataset:

Initially the dataset is filled out in by Excel , so suppose we fill out a dataset to calculate the employees 'salaries and years experiences

	YearsExperience	Salary
1	1.1	33343
2	1.2	43207
3	1.3	37731
4	1.4	43525
5	2.2	39591
6	2.9	56642
7	3.1	60150
8	3.2	54445
9	3.2	64445
10	3.2	57219
11	3.7	62111
12	3.9	58784
13	4	56657
14	4.1	57091
15	4.5	61111
16	4.8	67938
17	5.1	66029
18	5.3	63089
19	5.4	61111
20	5.8	63840
21	6.6	91736
22	7.1	96273
23	7.9	101302

Figure 5.6

## 1. Data Pre-processing

The first step for creating the Simple Linear Regression model is data pre-processing. We have already done it earlier in this tutorial. But there will be some changes, which are given in the below steps:

- First, we will import the three important libraries, which will help us for loading the dataset, plotting the graphs, and creating the Simple Linear Regression model.

The main libraries are called to handle the model , the following command imports the CSV dataset using pandas:

```
1. import numpy as np
2. import matplotlib.pyplot as plt
3. import pandas as pd
4.
5. # Importing the dataset
6. dataset = pd.read_csv('Salary_Data.csv')
```

Figure 5.7

To ensure that the dataset has been called, highlight the item, and then press F9

By executing the above line of code (ctrl+ENTER), we can read the dataset on our Spyder IDE screen by clicking on the variable explorer option.

data\_set - DataFrame

Index	YearsExperience	Salary
0	1	32383
1	1.1	45287
2	1.3	39751
3	2	43525
4	2.2	39891
5	2.7	56642
6	3	60150
7	3.2	54445
8	3.2	64445
9	3.7	57169
10	3.9	63218
11	4	55794
12	4	56957
13	4.1	57881

Format    Resize     Background color     Column min/max    Save and Close    Close

Figure 5.8

The above output shows the dataset, which has two variables: Salary and Experience.

Referred to Years of experience are (independent), while salaries are (dependent)

As such , the more features in years of experience, the better the prediction of models.

To find a correlation relationship between the independent and dependent , We call up the index location to convert from vector to matrix

```
7. X = dataset.iloc[:, :-1].values
8. y = dataset.iloc[:, 1].values
```

Figure 5.9

In the above lines of code, for x variable, we have taken -1 value since we want to remove the last column from the dataset. For y variable, we have taken 1 value as a parameter, since we want to extract the second column and indexing starts from the zero.

By executing the above line of code, we will get the output for X and Y variable as:

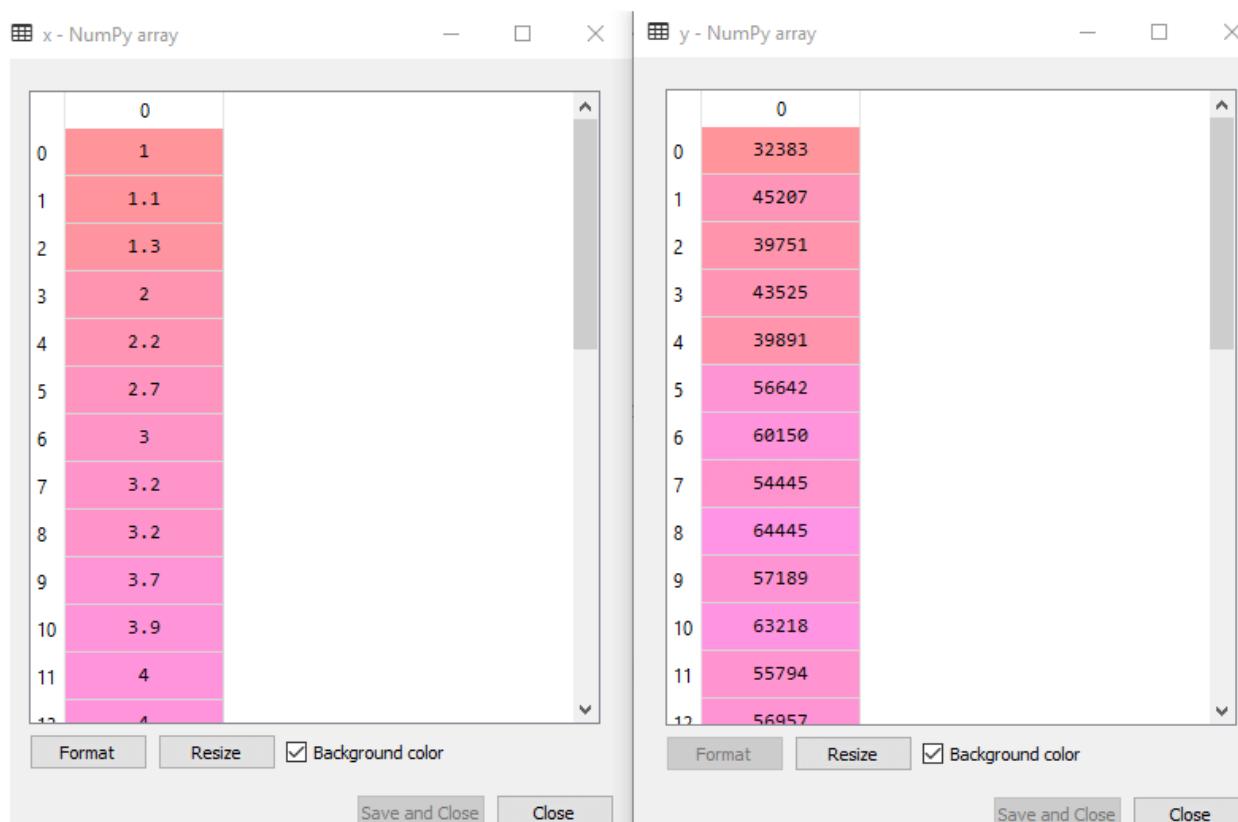


Figure 5.10

In the above output image, we can see the X (independent) variable and Y (dependent) variable has been extracted from the given dataset.

- Splitting the dataset into the Training set and Test set :

Next, we split 80% of the data to the training set while 20% of the data to test set using below code.

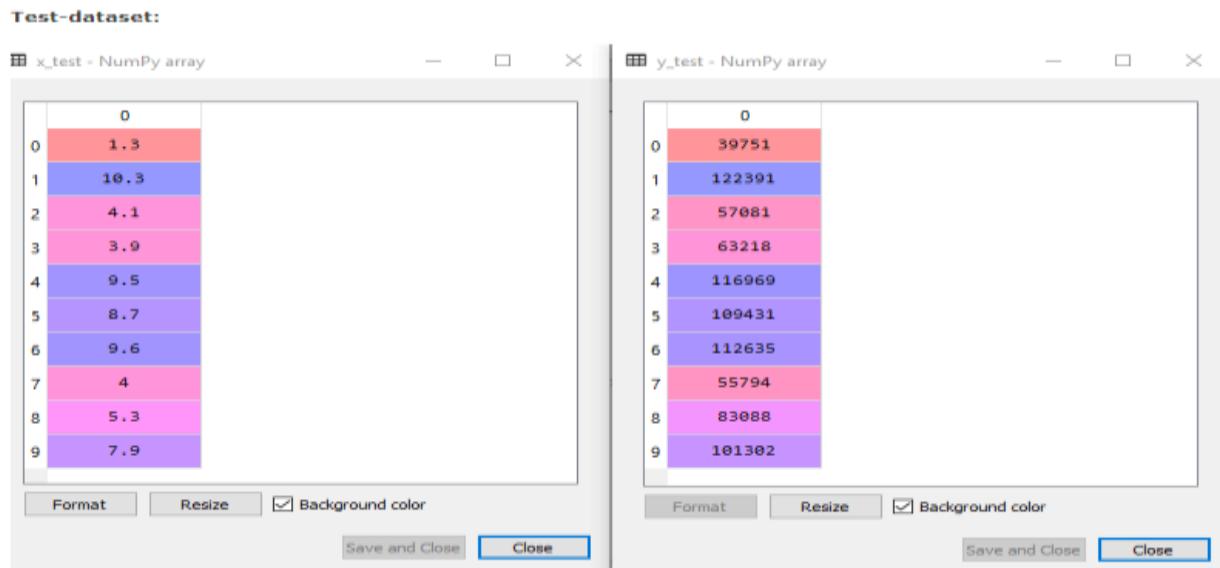
The test\_size variable is where we actually specify the proportion of the test set.

1.

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.5, random_state = 0)
```

Figure 5.11

Output:



Training Dataset:

x\_train - NumPy array

	0
0	2.7
1	5.1
2	3.2
3	4.5
4	8.2
5	6.8
6	1.1
7	10.5
8	3
9	2.2
10	5.6
11	6
12	5.7

y\_train - NumPy array

	0
0	56642
1	66029
2	64445
3	61111
4	113812
5	91738
6	45207
7	121872
8	60150
9	39891
10	81363
11	93940
12	57189

Figure 5.12

- Fitting Simple Linear Regression to the Training set:

Now the second step is to fit our model to the training dataset. To do so, we will import the **LinearRegression** class of the **linear\_model** library from the **scikit learn**. After importing the class, we are going to create an object of the class named as a **regressor**. The code for this is given below:

```
1. from sklearn.linear_model import LinearRegression
2. regressor = LinearRegression()
3. regressor.fit(X_train,y_train)
```

Figure 5.13

In the above code, we have used a **fit()** method to fit our Simple Linear Regression object to the training set. In the **fit()** function, we have passed the **x\_train** and **y\_train**, which is our training

dataset for the dependent and an independent variable. We have fitted our regressor object to the training set so that the model can easily learn the correlations between the predictor and target variables. After executing the above lines of code, we will get the below output.

- Prediction of test set result:

dependent (salary) and an independent variable (Experience). So, now, our model is ready to predict the output for the new observations. In this step, we will provide the test dataset (new observations) to the model to check whether it can predict the correct output or not.

We will create a prediction vector **y\_pred**, and **x\_pred**, which will contain predictions of test dataset, and prediction of training set respectively.

```
1. y_pred = regressor.predict(X_test)
2. y_pred_train = regressor.predict(X_train)
```

Figure 5.14

On executing the above lines of code, two variables named **y\_pred** and **x\_pred** will generate in the variable explorer options that contain salary predictions for the training set and test set.

**Output:**

You can check the variable by clicking on the variable explorer option in the IDE, and also compare the result by comparing values from `y_pred` and `y_test`. By comparing these values, we can check how good our model is performing.

- **visualizing the Training set results:**

Now in this step, we will visualize the training set result. To do so, we will use the `scatter()` function of the `pyplot` library, which we have already imported in the pre-processing step. The **scatter () function** will create a scatter plot of observations.

In the x-axis, we will plot the Years of Experience of employees and on the y-axis, salary of employees. In the function, we will pass the real values of training set, which means a year of experience `x_train`, training set of Salaries `y_train`, and color of the observations. Here we are taking a green color for the observation, but it can be any color as per the choice.

Now, we need to plot the regression line, so for this, we will use the **plot()** function of the `pyplot` library. In this function, we will pass the years of experience for training set, predicted salary for training set `x_pred`, and color of the line.

Next, we will give the title for the plot. So here, we will use the **title()** function of the `pyplot` library and pass the name ("Salary vs Experience (Training Dataset)").

After that, we will assign labels for x-axis and y-axis using **`xlabel()` and `ylabel()` function**.

Finally, we will represent all above things in a graph using `show()`. The code is given below:

```
1. plt.scatter(X_train ,y_train ,color = 'red')
2. plt.plot(X_train,y_pred_train ,color = 'blue')
3. plt.plot(X_train,regressor.predict(X_train), color = 'blue')
4. plt.title('Salary VS Experience (Training set) ')
5. plt.xlabel('Years of Experience')
6. plt.ylabel('Salary')
7. plt.show()
```

Figure 5.15

## Output:

By executing the above lines of code, we will get the below graph plot as an output.

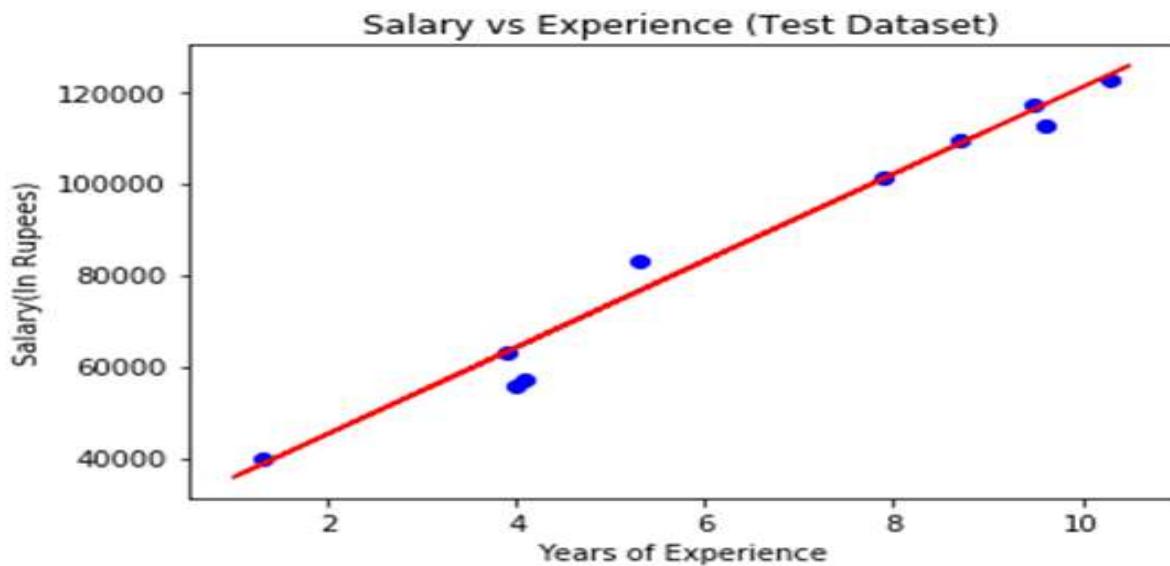


Figure 5.16

In the above plot, there are observations given by the blue color, and prediction is given by the red regression line. As we can see,

most of the observations are close to the regression line, hence we can say our Simple Linear Regression is a good model and able to make good predictions.

---

# CHAPTER 6

---

## Multiple Linear Regression

### 6.1 Overview

In the previous topic, we checked about Simple Linear Regression. You must be clear that Regression Models are Supervised Learning Models which can predict continuous variable. In the last topic, we saw how one variable may effect the result. Similarly, Multiple Linear Regression is the extension of Linear Regression in which we have number of dependent variables.

To be more clear, if you want to predict the marks of student, discipline only may not be sufficient. There may be many factors affecting the marks of any student. It may be Internal Marks, Assignments Completed, Attendance and more. Multiple Linear Regression is **used when we have multiple values affecting the result**. The effect may be positive or negative.

Simple  
Linear  
Regression

$$y = b_0 + b_1 * x_1$$

Multiple  
Linear  
Regression

$$y = b_0 + b_1 * x_1 + b_2 * x_2 + \dots + b_n * x_n$$

Dependent variable (DV)      Independent variables (IVs)

Constant      Coefficients

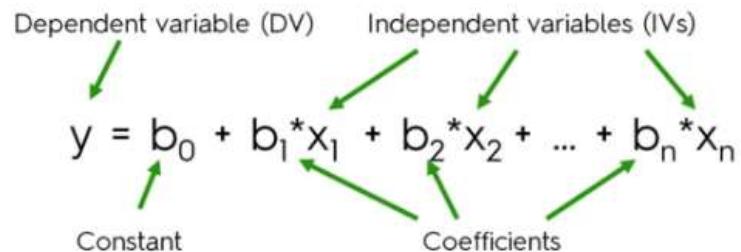


Figure 6.1

Hope you are now clear about the Multiple Linear Regression Problem.

Before start coding our model. We must be clear that Multiple Linear Regression have some assumptions. The data-set must be Linear, lack of multi-collinearity, independence of errors, and so on.

The more fun part is we will today pre process our data. It is very important part of building any machine learning model. And today we will handle the categorical data.

## 6.2 How to Construct a Multivariate Linear Regression Model

we will talk about how to build a multiple linear regression model step by step. In practical application, we often encounter many independent variables  $x_1, x_2$  and so on for a dependent variable  $y$ , but not all of these independent variables are very helpful for the prediction of  $Y$ . We need to remove the useless elements from them to get the most suitable model. Then there is a problem. How to choose these independent variables? Here are five ways to build models:

- 1 All-in
- 2 Backward Elimination
- 3 Forward Selection
- 4 Bidirectional Elimination Two-way Elimination
- 5 Score Comparison Information Quantity Comparison

Among them, 2, 3, 4 are also the most commonly used methods. They are called Step Regression, which is called stepwise regression. Their algorithms are similar, but the application order is somewhat different. The next one introduces these methods.

## 1- All-in :

The so-called all-in is to throw all the independent variables we think into it. Generally, it is used in several cases:

- 1 Prior Knowledge We already know all the information in advance, knowing that all these independent variables will have an impact on the results of the model.
- 2 You have to use these variables
- 3 Preparing for Backward Elimination for reverse phase-out

All-in method is very simple, but it is usually used in some special cases or when some external forces interfere. This method is not recommended.

## 2- Backward Elimination:

The essence of the reverse elimination algorithm is that for each independent variable of the model, it actually has an influence on the prediction results of our model, which is described by a statistical concept called P-value. Here, we first define a threshold of whether this influence is significant or not, that is, a significance level (SL), which is defined as 0.05. Then the second step uses all the independent variables to fit a model. The third step is to calculate the P-value of each independent variable in the model to show how much influence it has on our model. Then we take the highest P-value and assume the  $P > SL$ , we go on to the fourth step, otherwise we will end the method. The fourth step, then, is to remove the independent variable corresponding to the highest P value from our model. Fifth step, after removing an independent variable, the model is fitted

again with the remaining independent variables, so here is a cycle from the third step to the fifth step, until all the remaining P values are smaller than SL, which shows that the model has been fitted. Details of the steps are as follows:

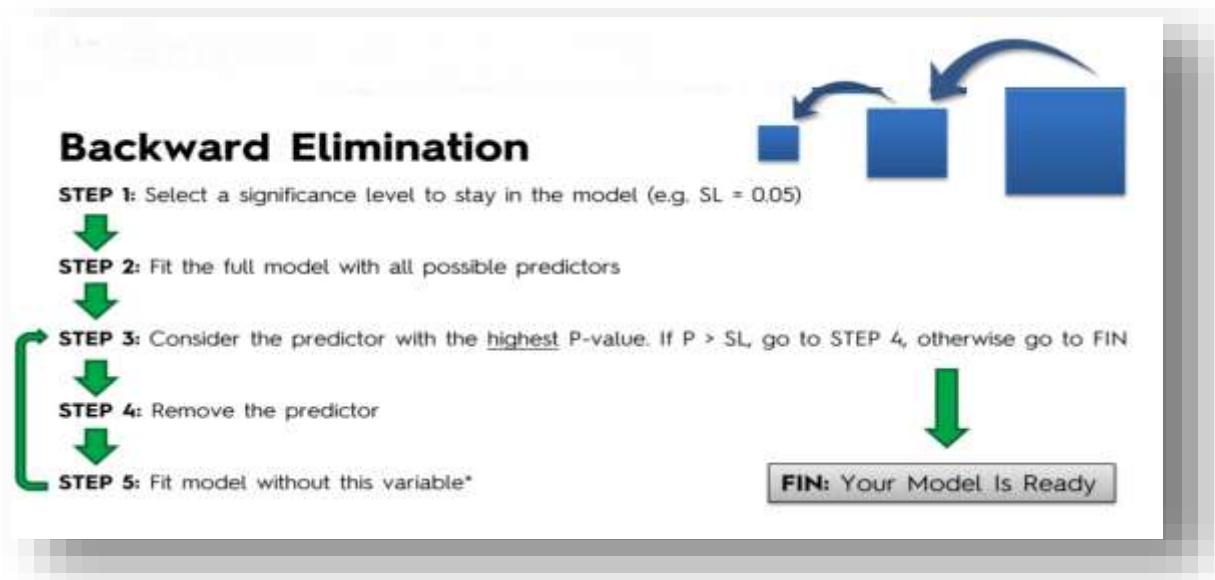


Figure 6.2

### 3- Forward Selection:

The algorithm idea of forward selection and reverse elimination is very close, but the order is reversed. The first step is to set a significant threshold  $SL = 0.05$ . The second step is to make simple linear regression fitting for each independent variable,  $\$x_n$ , and get their P values respectively, and then get the lowest among them. In the third step, for the lowest P value, our conclusion is that this independent variable has the greatest impact on the model we are going to fit, so we will retain this independent variable. In the fourth step, we will see which of

the remaining independent variables will bring us the smallest P value.

If the new P value is smaller than the SL we defined earlier, we will go back to the third step, that is, adding a new variable in the third

step, then finding the largest P value in the remaining variables, and then adding it to the model, and so on. Until the remaining variables that have not been added to the model, their P values are all greater than SL, that is to say, the remaining variables may not have a significant impact on the model, then these will not be adopted, then the model will be fitted. Details of the steps are as follows:

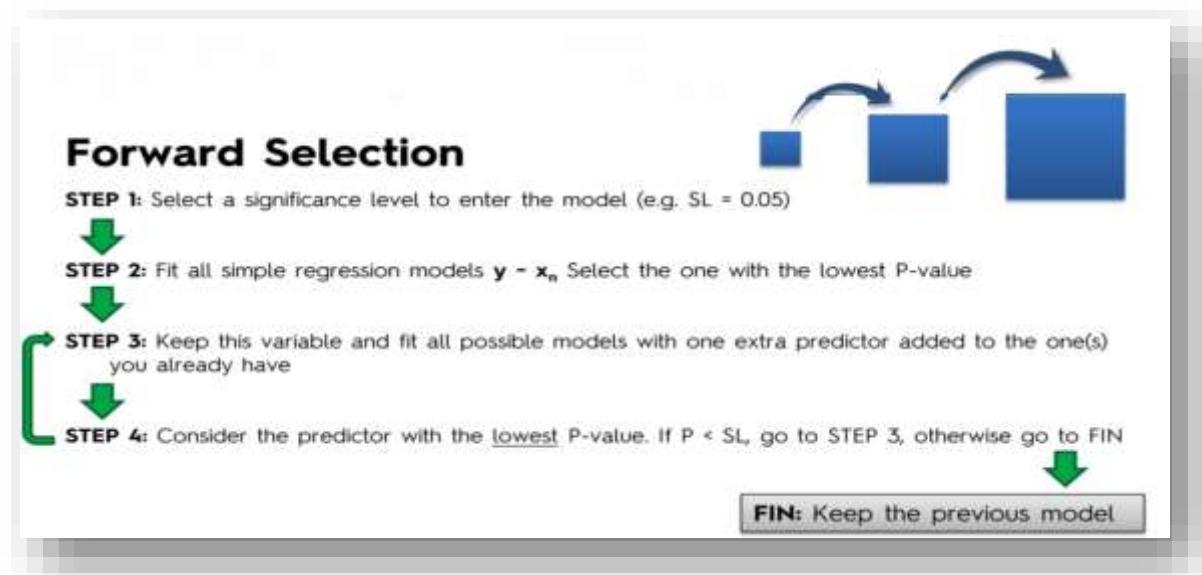


Figure 6.3

#### 4- Bidirectional Elimination:

The so-called two-way elimination, in fact, is the combination of the two previous algorithms. In the first step, we need to choose two saliency thresholds: whether an old variable should be excluded and whether a new variable that has not yet been adopted should enter our model. In the second step, we make a forward choice to decide whether to adopt a new independent variable. The third step is reverse elimination, that is, we may have to eliminate the old variables, and then cycle between the second and third steps. Because two thresholds have been defined, but new ones can not go out, when the old ones can not come in, the model has been fitted. Details of the steps are as follows:

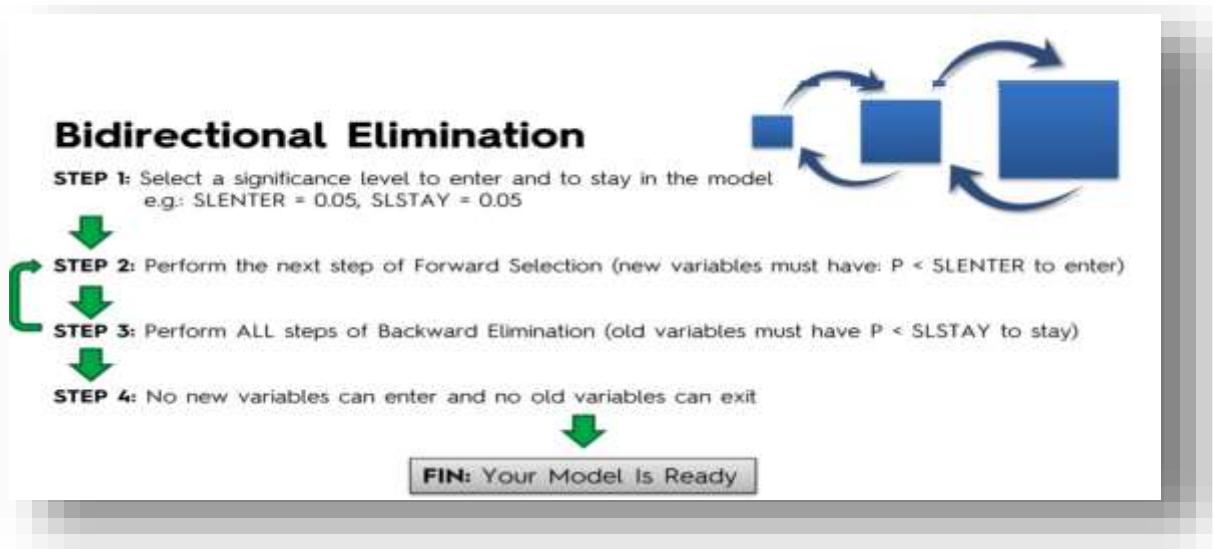


Figure 6.4

## 5- Score Comparison:

Finally, the amount of information is compared. The so-called information quantity is a way of evaluating a multivariate linear regression model. Give it a score, for example. So there are many ways of scoring. For example, the most common one is called Akaike criterion. For all possible models, we scored them one by one. For multiple linear regression, if there were  $N$  independent variables, there would be  $2^N - 1$  different models. After scoring these models, we chose the model with the highest score. Then there will be a problem, if  $N$  is large, the number of models will be very large. So although this method is intuitively well understood, it is not suitable to use this method when the number of independent variables is large.

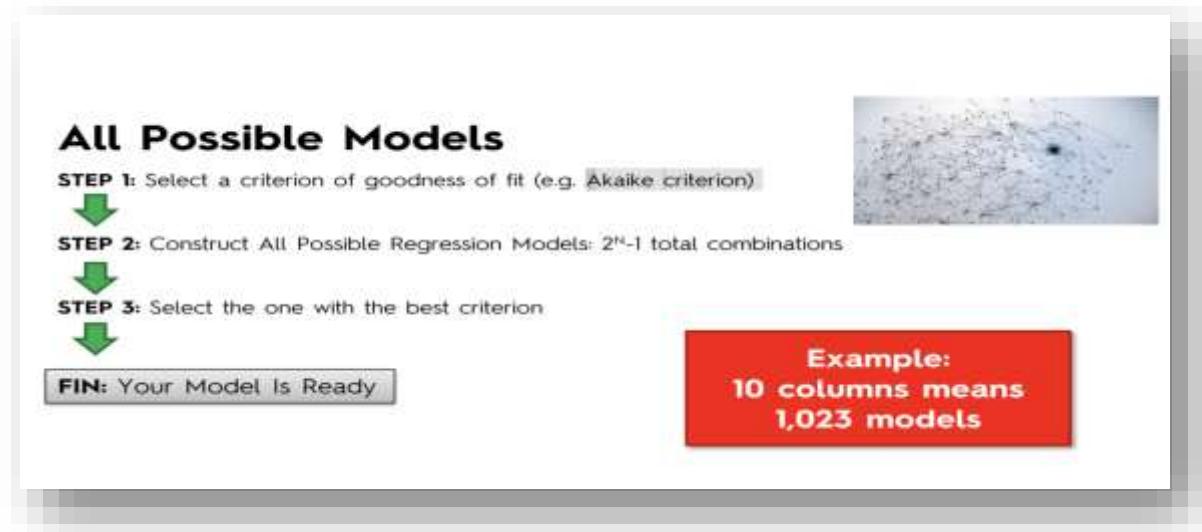


Figure 6.5

## 6.3 Multiple Linear Regression in python

## Problem Description:

We have a dataset of 50 start-up companies. This dataset contains five main information: R&D Spend, Administration Spend, Marketing Spend, State, and Profit for a financial year. Our goal is to create a model that can easily determine which company has a maximum profit, and which is the most affecting factor for the profit of a company.

Since we need to find the Profit, so it is the dependent variable, and the other four variables are independent variables. Below are the main steps of deploying the MLR model:

1. Data Pre-processing Steps
2. Fitting the MLR model to the training set
3. Predicting the result of the test set

- **Data Pre-processing Step:**

The very first step is data pre-processing, which we have already discussed in this tutorial. This process contains the below steps:

- Importing libraries: Firstly we will import the library which will help in building the model. Below is the code for it:

```
1. # Importing the libraries
2.
```

```
3. import numpy as np
4. import matplotlib.pyplot as plt
5. import pandas as pd
```

Figure 6.6

**Importing dataset:** Now we will import the dataset(50\_CompList), which contains all the variables. Below is the code for it:

```
1. # Importing the dataset
2. dataset = pd.read_csv('50_Startups.csv')
```

Figure 6.7

**Output:** We will get the dataset as:

Index	R&D Spend	Administration	Marketing Spend	State	Profit
0	165349	136898	471784	New York	192262
1	162598	151378	443899	California	191792
2	153442	101146	487935	Florida	191050
3	144372	118672	383208	New York	182902
4	142107	91391.8	366168	Florida	166188
5	131877	99814.7	362861	New York	156991
6	134615	147199	127717	California	156123
7	130298	145530	323877	Florida	155753
8	128543	148719	311613	New York	152212
9	123335	108679	304982	California	149760
10	101913	110594	229161	Florida	146122
11	100672	91790.6	249745	California	144259
12	93863.8	127320	249839	Florida	141586
13	91992.4	135495	252665	California	134307

Figure 6.8

In above output, we can clearly see that there are five variables, in which four variables are continuous and one is categorical variable.

- **Extracting dependent and independent Variables:**

```
1. #Extracting Independent and dependent Variable
2.
3. x= data_set.iloc[:, :-1].values
4. y= data_set.iloc[:, 4].values
```

Figure 6.9

- **Encoding Dummy Variables:**

As we have one categorical variable (State), which cannot be directly applied to the model, so we will encode it. To encode the categorical variable into numbers, we will use the **LabelEncoder** class. But it is not sufficient because it still has some relational order, which may create a wrong model. So in order to remove this problem, we will use **OneHotEncoder**, which will create the dummy variables. Below is code for it:

```
1. #Categorical data
2.
3. from sklearn.preprocessing import LabelEncoder, OneHotEncoder
4.
5. labelencoder_x= LabelEncoder()
6. x[:, 3]= labelencoder_x.fit_transform(x[:,3])
7. onehotencoder= OneHotEncoder(categorical_features= [3])
8. x= onehotencoder.fit_transform(x).toarray()
```

Figure 6.10

Here we are only encoding one independent variable, which is state as other variables are continuous.

## Output:

x - NumPy array

	0	1	2	3	4	5
0	0	0	1	165349	136898	471784
1	1	0	0	162598	151378	443899
2	0	1	0	153442	101146	407935
3	0	0	1	144372	118672	383200
4	0	1	0	142107	91391.8	366168
5	0	0	1	131877	99814.7	362861
6	1	0	0	134615	147199	127717
7	0	1	0	130298	145530	323877
8	0	0	1	120543	148719	311613
9	1	0	0	123335	108679	304982
10	0	1	0	101913	110594	229161
11	1	0	0	100672	91790.6	249745
12	0	1	0	93863.8	127320	249839
13	1	0	0	91992.4	135495	252665

Format    Resize     Background color

Figure 6.11

As we can see in the above output, the state column has been converted into dummy variables (0 and 1). **Here each dummy variable column is corresponding to the one State.** We can check by comparing it with the original dataset. The first column corresponds to the California State, the second column corresponds to the Florida State, and the third column corresponds to the New York State.

- Now, we are writing a single line of code just to avoid the dummy variable trap:

- #avoiding the dummy variable trap:
- 
- `x = x[:, 1:]`

Figure 6.12

If we do not remove the first dummy variable, then it may introduce multicollinearity in the model.



	0	1	2	3	4
0	0	1	165349	136898	471784
1	0	0	162598	151378	443899
2	1	0	153442	101146	407935
3	0	1	144372	118672	383200
4	1	0	142107	91391.8	366168
5	0	1	131877	99814.7	362861
6	0	0	134615	147199	127717
7	1	0	130298	145530	323877
8	0	1	120543	148719	311613
9	0	0	123335	108679	304982
10	1	0	101913	110594	229161
11	0	0	100672	91790.6	249745
12	1	0	93863.8	127320	249839

Figure 6.13

As we can see in the above output image, the first column has been removed.

- Now we will split the dataset into training and test set. The code for this is given below:

```

1. # Splitting the dataset into training and test set.
2.
3. from sklearn.model_selection import train_test_split
4. x_train, x_test, y_train, y_test= train_test_split(x, y, test_size= 0.2, random_state=0
)

```

Figure 6.14

The above code will split our dataset into a training set and test set.

**Output:** The above code will split the dataset into training set and test set. You can check the output by clicking on the variable explorer option given in Spyder IDE. The test set and training set will look like the below image:

Test set:

y_test - NumPy array		x_test - NumPy array				
	0	0	1	2	3	4
0	103282	1	0	66051.5	182646	118148
1	144259	0	0	100672	91790.6	249745
2	146122	1	0	101913	110594	229161
3	77798.8	1	0	27892.9	84710.8	164471
4	191050	1	0	153442	101146	407935
5	105008	0	1	72107.6	127865	353184
6	81229.1	0	1	20229.6	65947.9	185265
7	97483.6	0	1	61136.4	152702	88218.2
8	110352	1	0	73994.6	122783	303319
9	166188	1	0	142107	91391.8	366168

Format    Resize     Background color    Save and Close    Close

Training set:

x\_train - NumPy array

y\_train - NumPy array

Format    Resize     Background color    Save and Close    Close

Format    Resize     Background color    Save and Close    Close

Figure 6.15

- Fitting our MLR model to the Training set:

Now, we have well prepared our dataset in order to provide training, which means we will fit our regression model to the training set. It will be similar to as we did in Simple Linear Regression model. The code for this will be:

```

1. #Fitting the MLR model to the training set:
2.
3. from sklearn.linear_model import LinearRegression
4. regressor= LinearRegression()
5. regressor.fit(x_train, y_train)

```

Figure 6.16

## Output:

```
Out[9]: LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)
```

Figure 6.17

Now, we have successfully trained our model using the training dataset. In the next step, we will test the performance of the model using the test dataset.

- **Prediction of Test set results:**

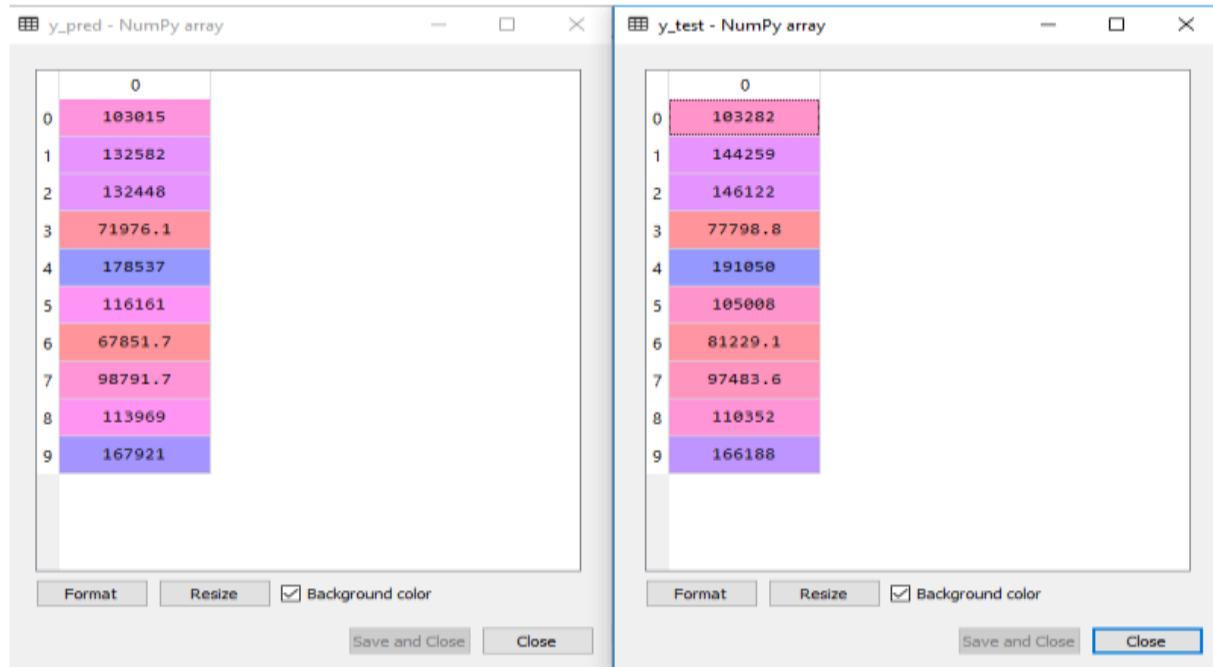
The last step for our model is checking the performance of the model. We will do it by predicting the test set result. For prediction, we will create a **y\_pred** vector. Below is the code for it:

```
1. #Predicting the Test set result
2.
3. y_pred= regressor.predict(x_test)
```

Figure 6.18

By executing the above lines of code, a new vector will be generated under the variable explorer option. We can test our model by comparing the predicted values and test set values.

## Output:



The image shows two separate windows, each titled "y\_pred - NumPy array" and "y\_test - NumPy array". Both windows display a 10x2 grid of data. The left window (y\_pred) contains integer values: 103015, 132582, 132448, 71976.1, 178537, 116161, 67851.7, 98791.7, 113969, and 167921. The right window (y\_test) contains floating-point values: 103282, 144259, 146122, 77798.8, 191050, 105008, 81229.1, 97483.6, 110352, and 166188. Each window has a "Format" button, a "Resize" button, a checked "Background color" checkbox, and a "Save and Close" button. The right window also has a "Close" button.

	0
0	103015
1	132582
2	132448
3	71976.1
4	178537
5	116161
6	67851.7
7	98791.7
8	113969
9	167921

	0
0	103282
1	144259
2	146122
3	77798.8
4	191050
5	105008
6	81229.1
7	97483.6
8	110352
9	166188

Figure 6.19

In the above output, we have predicted result set and test set. We can check model performance by comparing these two value index by index. For example, the first index has a predicted value of **103015\$** profit and test/real value of **103282\$** profit. The difference is only of **267\$**, which is a good prediction, so, finally, our model is completed here.

- We can also check the score for training dataset and test dataset. Below is the code for it:

```
1. print('Train Score: ', regressor.score(x_train, y_train))
2.
3. print('Test Score: ', regressor.score(x_test, y_test))
```

Figure 6.20

## Applications of Multiple Linear Regression:

There are mainly two applications of Multiple Linear Regression:

- Effectiveness of Independent variable on prediction:
- Predicting the impact of changes:

# CHAPTER 7

## Polynomial Regression

### 7.1 Overview

Polynomial Regression is a regression algorithm that models the relationship between a dependent(y) and independent variable(x) as nth degree polynomial. The Polynomial Regression equation is given below:

$$y = b_0 + b_1 x_1 + b_2 x_1^2 + b_3 x_1^3 + \dots + b_n x_1^n$$

Figure 7.1

- It is also called the special case of Multiple Linear Regression in ML. Because we add some polynomial terms to the Multiple Linear regression equation to convert it into Polynomial Regression.
- It is a linear model with some modification in order to increase the accuracy.
- The dataset used in Polynomial regression for training is of non-linear nature.
- It makes use of a linear regression model to fit the complicated and non-linear functions and datasets.

- Hence, "In Polynomial regression, the original features are converted into Polynomial features of required degree (2,3,..,n) and then modeled using a linear model."

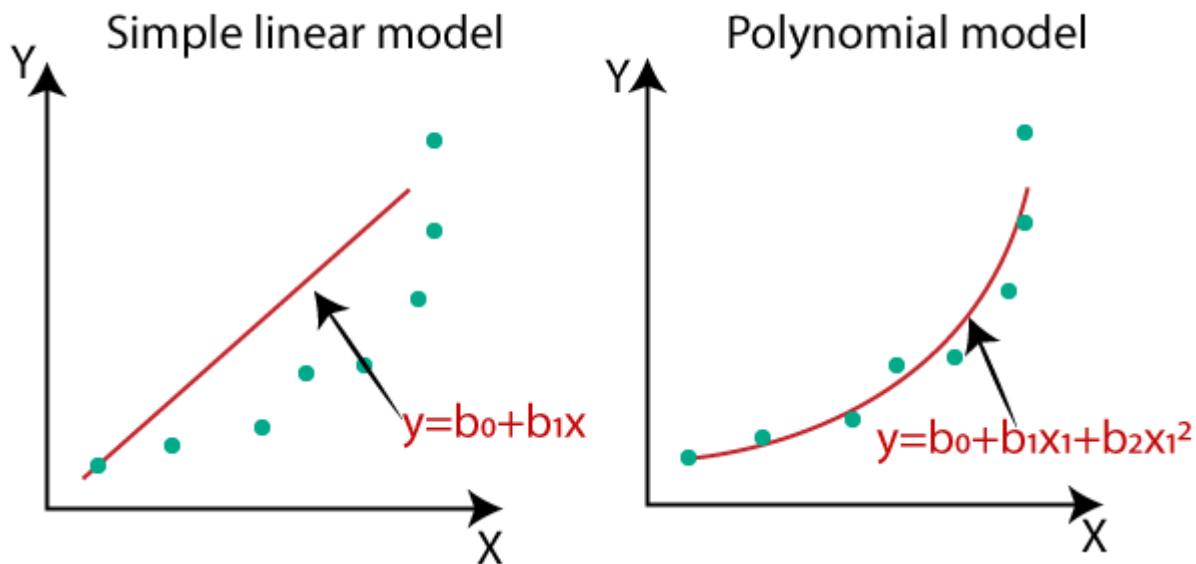


Figure 7.2

- In the above image, we have taken a dataset which is arranged non-linearly. So if we try to cover it with a linear model, then we can clearly see that it hardly covers any data point. On the other hand, a curve is suitable to cover most of the data points, which is of the Polynomial model.
- Hence, if the datasets are arranged in a non-linear fashion, then we should use the Polynomial Regression model instead of Simple Linear Regression.

- **Equation of the Polynomial Regression Model:**

1- Simple Linear Regression equation:

(a)  $y = b_0 + b_1 x$

2- Multiple Linear Regression equation:

(b)  $y = b_0 + b_1 x_1 + b_2 x_2 + b_3 x_3 + \dots + b_n x_n$

3- Polynomial Regression equation:

(c)  $y = b_0 + b_1 x + b_2 x^2 + b_3 x^3 + \dots + b_n x^n$

When we compare the above three equations, we can clearly see that all three equations are Polynomial equations but differ by the degree of variables. The Simple and Multiple Linear equations are also Polynomial equations with a single degree, and the Polynomial regression equation is Linear equation with the nth degree. So if we add a degree to our linear equations, then it will be converted into Polynomial Linear equations.

## 7.2 Simple linear regression in python

Here we will implement the Polynomial Regression using Python. We will understand it by comparing Polynomial Regression model with the Simple Linear Regression model. So first, let's understand the problem for which we are going to build the model.

**Problem Description:** There is a Human Resource company, which is going to hire a new candidate. The candidate has told his previous salary 160K per annum, and the HR have to check whether he is telling the truth or bluff. So to identify this, they only have a dataset of his previous company in which the salaries of the top 10 positions are mentioned with their levels. By checking the dataset available, we have found that there is a **non-linear relationship between the Position levels and the salaries**. Our goal is to build a **Bluffing detector regression** model, so HR can hire an honest candidate. Below are the steps to build such a model.

Position	Level(X-variable)	Salary(Y-Variable)
Business Analyst	1	45000
Junior Consultant	2	50000
Senior Consultant	3	60000
Manager	4	80000
Country Manager	5	110000
Region Manager	6	150000
Partner	7	200000
Senior Partner	8	300000
C-level	9	500000
CEO	10	1000000

Figure 7.3

- **Steps for Polynomial Regression:**

The main steps involved in Polynomial Regression are given below:

- Data Pre-processing
- Build a Linear Regression model and fit it to the dataset
- Build a Polynomial Regression model and fit it to the dataset
- Visualize the result for Linear Regression and Polynomial Regression model.
- Predicting the output.

- **Data Pre-processing Step:**

The data pre-processing step will remain the same as in previous regression models, except for some changes. In the Polynomial Regression model, we will not use feature scaling, and also we will not split our dataset into training and test set. It has two reasons:

- The dataset contains very less information which is not suitable to divide it into a test and training set, else our model will not be able to find the correlations between the salaries and levels.
- In this model, we want very accurate predictions for salary, so the model should have enough information.

The code for pre-processing step is given below:

```

1. # Polynomial Regression
2.
3. # Importing the libraries
4.
5. import numpy as np
6. import matplotlib.pyplot as plt
7. import pandas as pd
8.
9. # Importing the dataset
10.
11. dataset = pd.read_csv('Position_Salaries.csv')
12. X = dataset.iloc[:, 1:-1].values
13. y = dataset.iloc[:, -1].values

```

Figure 7.4

■■■ X - NumPy array

	0
0	1
1	2
2	3
3	4
4	5
5	6
6	7
7	8
8	9
9	10

Format    Resize     Background color

Save and Close    Close

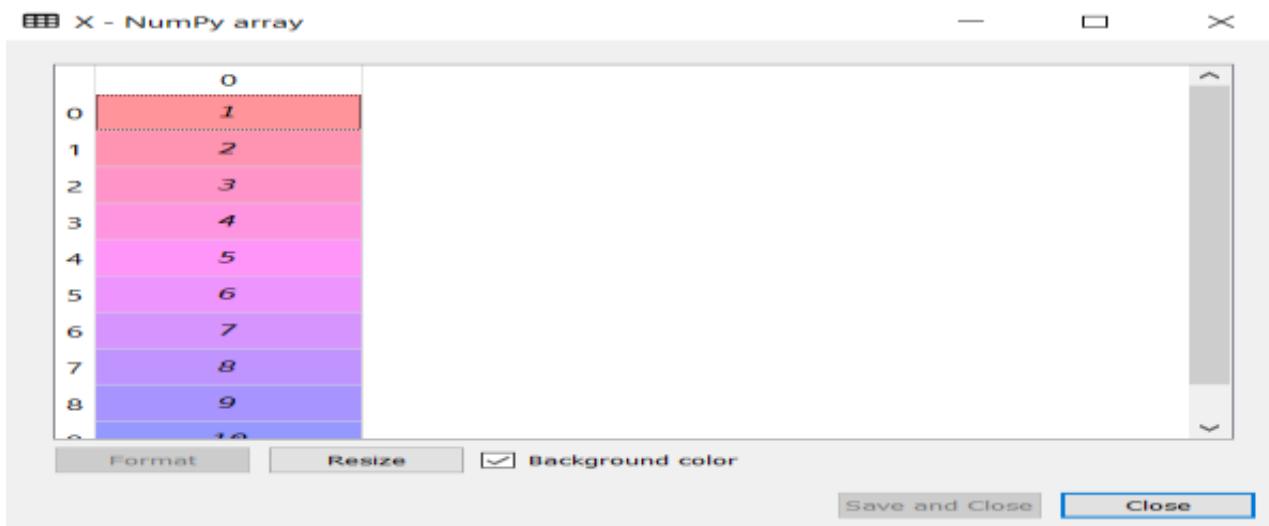


Figure 7.5

■■■ y - NumPy array

	0
0	450000
1	500000
2	600000
3	800000
4	1100000
5	1500000
6	2000000
7	3000000
8	5000000
9	10000000

Format    Resize     Background color

Save and Close    Close

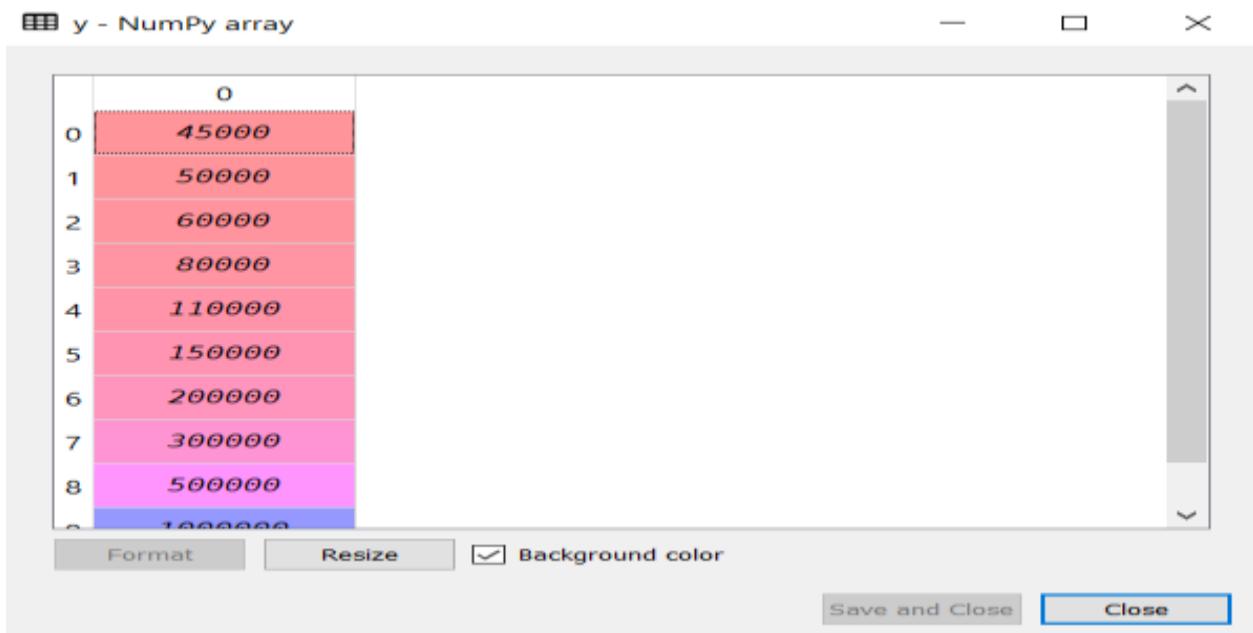


Figure 7.6

## Explanation:

- In the above lines of code, we have imported the important Python libraries to import dataset and operate on it.
- Next, we have imported the dataset '**Position\_Salaries.csv**', which contains three columns (Position, Levels, and Salary), but we will consider only two columns (Salary and Levels).
- After that, we have extracted the dependent(Y) and independent variable(X) from the dataset. For x-variable, we have taken parameters as `[:,1:2]`, because we want 1 index(levels), and included `:2` to make it as a matrix.

- **Building the Linear regression model:**

Now, we will build and fit the Linear regression model to the dataset. In building polynomial regression, we will take the Linear regression model as reference and compare both the results. The code is given below:

```
1. # Training the Linear Regression model on the whole dataset
2.
3. from sklearn.linear_model import LinearRegression
4. lin_reg = LinearRegression()
5. lin_reg.fit(X, y)
```

Figure 7.7

In the above code, we have created the Simple Linear model using `lin_regs` object of `LinearRegression` class and fitted it to the dataset variables (x and y).

```

1. # Training the Polynomial Regression model on the whole dataset
2.
3. from sklearn.preprocessing import PolynomialFeatures
4. poly_reg = PolynomialFeatures(degree = 4)
5. X_poly = poly_reg.fit_transform(X)
6. lin_reg_2 = LinearRegression()
7. lin_reg_2.fit(X_poly, y)

```

Figure 7.8

In the above lines of code, we have used `poly_regs.fit_transform(x)`, because first we are converting our feature matrix into polynomial feature matrix, and then fitting it to the Polynomial regression model. The parameter value(`degree= 2`) depends on our choice. We can choose it according to our Polynomial features.

After executing the code, we will get another matrix `x_poly`, which can be seen under the variable explorer option:

**X\_poly - NumPy array**

	0	1	2	3	
0	1	1	1	1	
1	1	2	4	8	
2	1	3	9	27	
3	1	4	16	64	
4	1	5	25	125	
5	1	6	36	216	
6	1	7	49	343	
7	1	8	64	512	
8	1	9	81	729	
<					

Format    Resize     Background color    Save and Close    Close

Figure 7.9

Next, we have used another `LinearRegression` object, namely `lin_reg_2`, to fit our `x_poly` vector to the linear model.

- **Visualizing the result for Linear regression:**

Now we will visualize the result for Linear regression model as we did in Simple Linear Regression. Below is the code for it:

```
1. # Visualising the Linear Regression results
2.
3. plt.scatter(X, y, color = 'red')
4. plt.plot(X, lin_reg.predict(X), color = 'blue')
5. plt.title('Truth or Bluff (Linear Regression)')
6. plt.xlabel('Position Level')
7. plt.ylabel('Salary')
8. plt.show()
```

Figure 7.10

Output:

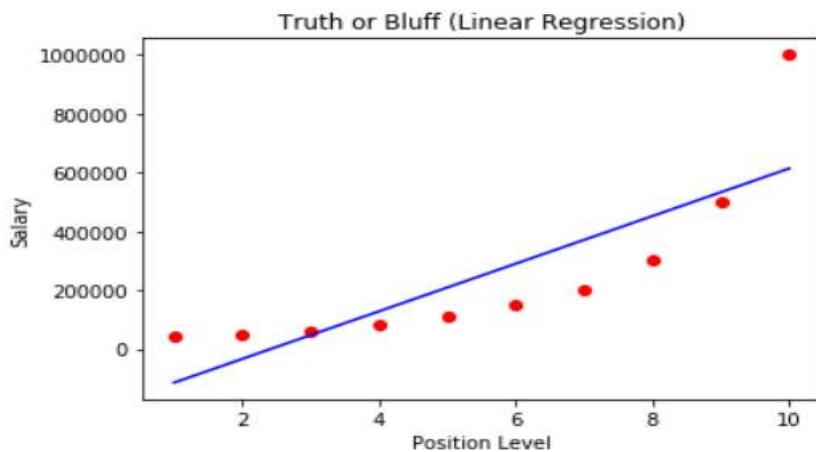


Figure 7.11

In the above output image, we can clearly see that the regression line is so far from the datasets. Predictions are in a red straight line, and blue points are actual values. If we consider this output to predict the value of CEO, it will give a salary of approx. 600000\$, which is far away from the real value.

So we need a curved model to fit the dataset other than a straight line.

- **Visualizing the result for Polynomial Regression**

Here we will visualize the result of Polynomial regression model, code for which is little different from the above model.

Code for this is given below:

```
1. # Visualising the Polynomial Regression results
2.
3. plt.scatter(X, y, color = 'red')
4. plt.plot(X, lin_reg_2.predict(poly_reg.fit_transform(X)), color = 'blue')
5. plt.title('Truth or Bluff (Polynomial Regression)')
6. plt.xlabel('Position level')
7. plt.ylabel('Salary')
8. plt.show()
```

Figure 7.12

In the above code, we have taken `lin_reg_2.predict(poly_regs.fit_transform(x))`, instead of `x_poly`, because we want a Linear regressor object to predict the polynomial features matrix.

**Output:**

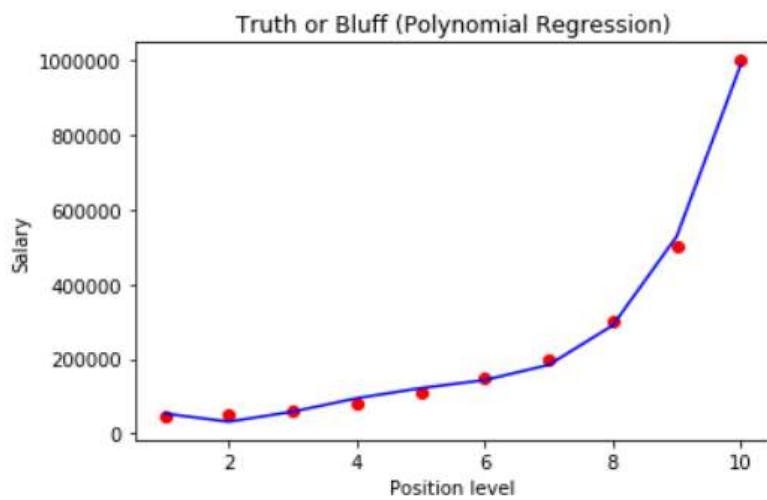


Figure 7.13

As we can see in the above output image, the predictions are close to the real values. The above plot will vary as we will change the degree.

**For degree = 3:**

If we change the degree=3, then we will give a more accurate plot, as shown in the below image.

```
1. # Visualising the Polynomial Regression results (for higher resolution and smoother curve)
2. X_grid = np.arange(min(X), max(X), 0.1)
3. X_grid = X_grid.reshape((len(X_grid), 1))
4. plt.scatter(X, y, color = 'red')
5. plt.plot(X_grid, lin_reg_2.predict(poly_reg.fit_transform(X_grid)), color = 'blue')
6. plt.title('Truth or Bluff (Polynomial Regression)')
7. plt.xlabel('Position level')
8. plt.ylabel('Salary')
9. plt.show()
```

Figure 7.14

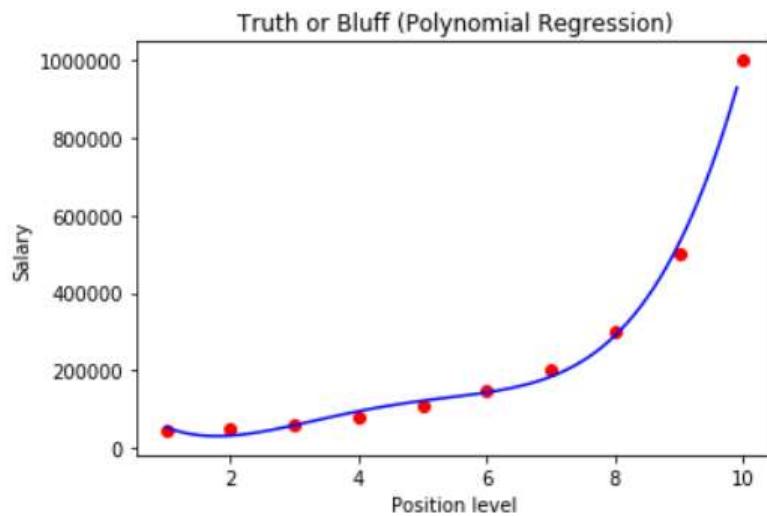


Figure 7.15

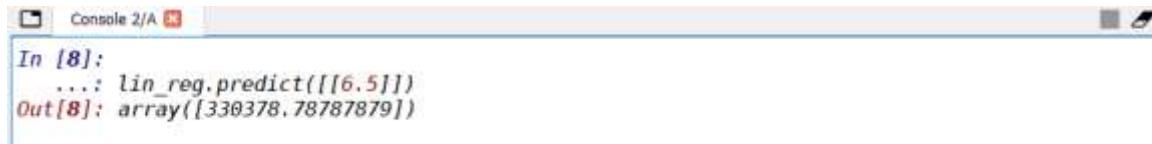
- **Predicting the final result with the Linear Regression model:**

Now, we will predict the final output using the Linear regression model to see whether an employee is saying truth or bluff. So, for this, we will use the predict() method and will pass the value 6.5. Below is the code for it:

```
1. # Predicting a new result with Linear Regression
2.
3. lin_reg.predict([[6.5]])
```

Figure 7.16

Output:



The screenshot shows a Jupyter Notebook cell titled 'Console 2/A'. The cell contains the following code and output:

```
In [8]: lin_reg.predict([[6.5]])
Out[8]: array([330378, 78787879])
```

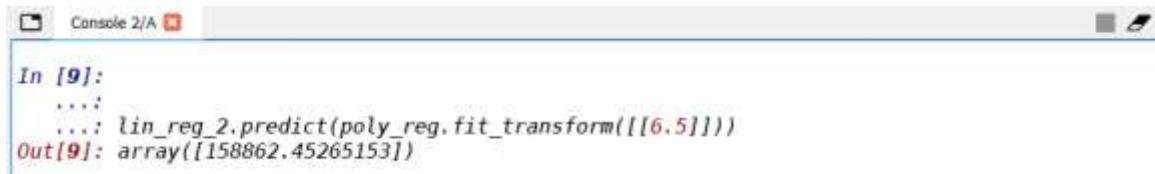
Figure 7.17

- **Predicting the final result with the Polynomial Regression model:**

Now, we will predict the final output using the Polynomial Regression model to compare with Linear model. Below is the code for it:

```
1. # Predicting a new result with Polynomial Regression
2.
3. lin_reg_2.predict(poly_reg.fit_transform([[6.5]]))
```

Figure 7.18



```
In [9]:  
...:  
...: lin_reg_2.predict(poly_reg.fit_transform([[6.5]]))  
Out[9]: array([158862.45265153])
```

Figure 7.19

As we can see, the predicted output for the Polynomial Regression is [158862.45265153], which is much closer to real value hence, we can say that future employee is saying true.

---

# CHAPTER 8

---

## Support Vector Regression

### 8.1 Overview

Support Vector Machine or SVM is one of the most popular Supervised Learning algorithms, which is used for Classification as well as Regression problems. However, primarily, it is used for Classification problems in Machine Learning.

The goal of the SVM algorithm is to create the best line or decision boundary that can segregate n-dimensional space into classes so that we can easily put the new data point in the correct category in the future. This best decision boundary is called a hyperplane.

SVM chooses the extreme points/vectors that help in creating the hyperplane. These extreme cases are called as support vectors, and hence algorithm is termed as Support Vector Machine. Consider the below diagram in which there are two different categories that are classified using a decision boundary or hyperplane:

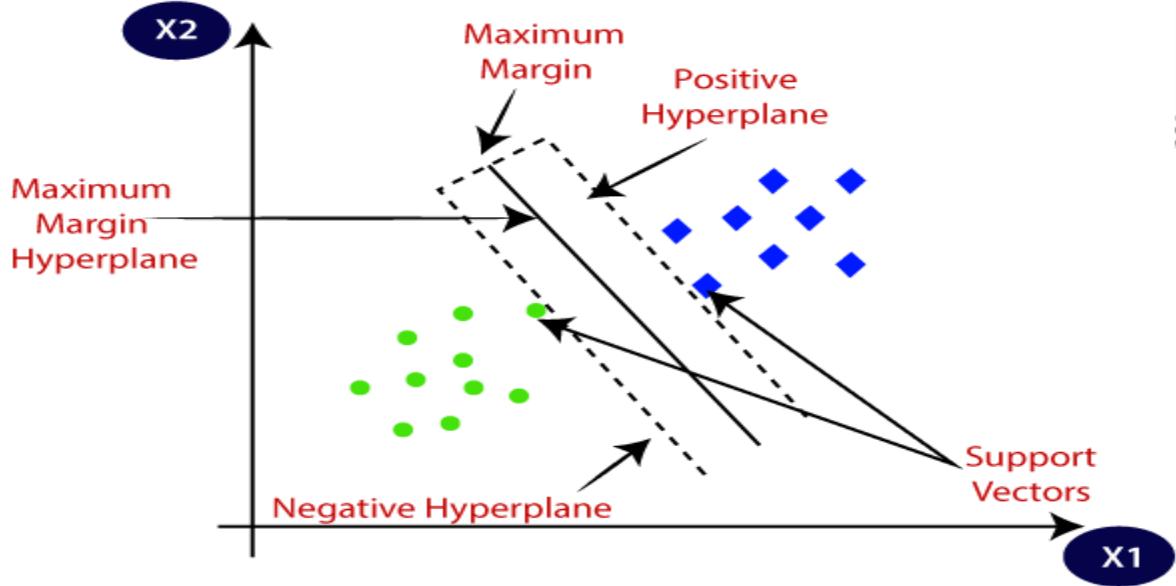


Figure 8.1

- **Types of SVM:**

- **Linear SVM:** Linear SVM is used for linearly separable data, which means if a dataset can be classified into two classes by using a single straight line, then such data is termed as linearly separable data, and classifier is used called as Linear SVM classifier.
- **Non-linear SVM:** Non-Linear SVM is used for non-linearly separated data, which means if a dataset cannot be classified by using a straight line, then such data is termed as non-linear data and classifier used is called as Non-linear SVM classifier.

- How does SVM works?

### Linear SVM:

The working of the SVM algorithm can be understood by using an example. Suppose we have a dataset that has two tags (green and blue), and the dataset has two features  $x_1$  and  $x_2$ . We want a classifier that can classify the pair  $(x_1, x_2)$  of coordinates in either green or blue. Consider the below image:

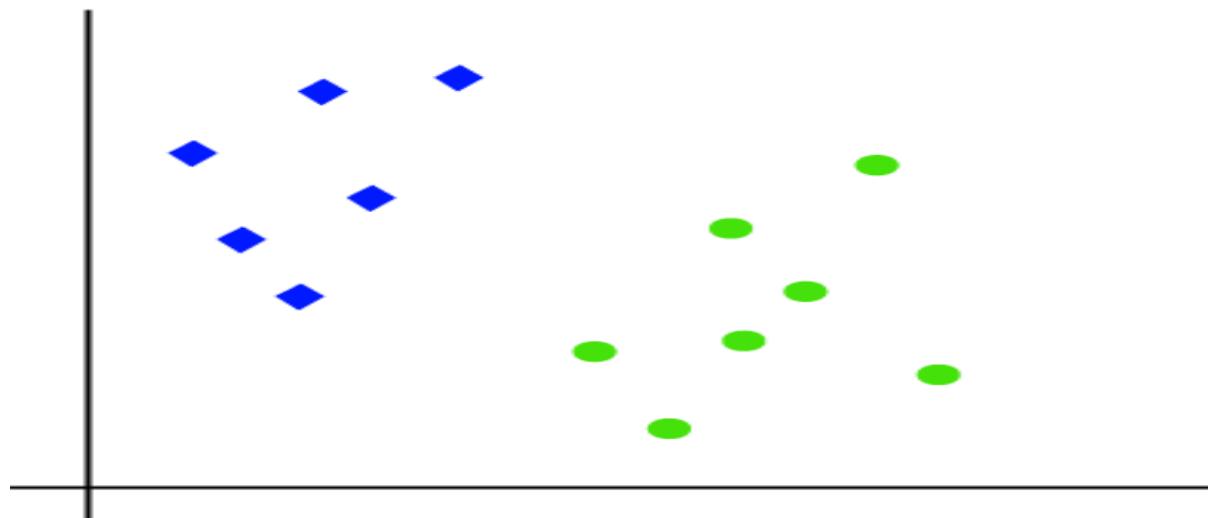


Figure 8.2

Hence, the SVM algorithm helps to find the best line or decision boundary; this best boundary or region is called as a **hyperplane**. SVM algorithm finds the closest point of the lines from both the classes. These points are called support vectors. The distance between the vectors and the hyperplane is called as **margin**. And the goal of SVM is to maximize this margin. The **hyperplane** with maximum margin is called the **optimal hyperplane**.

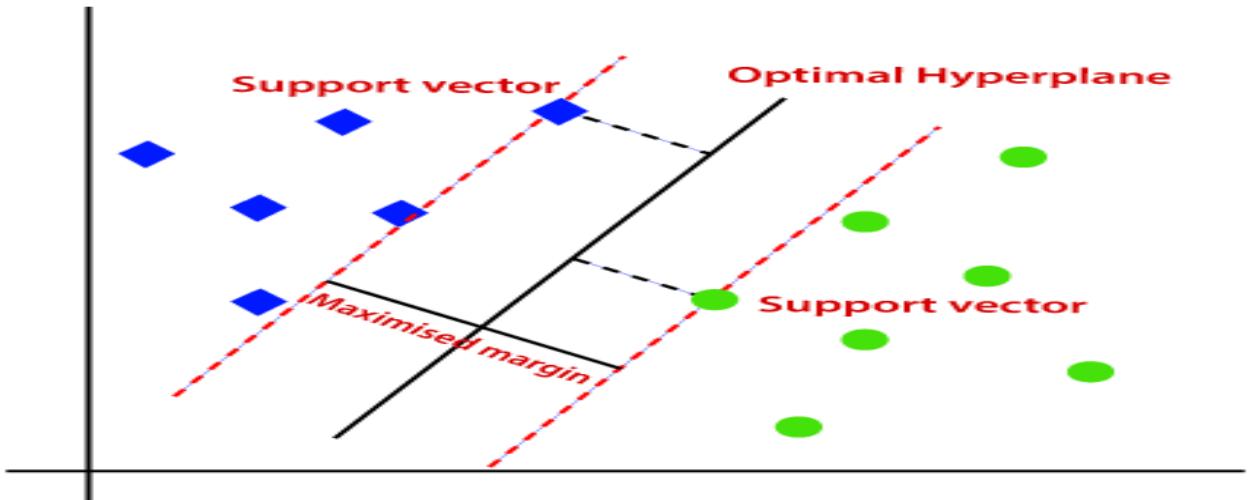


Figure 8.3

- **Non-Linear SVM:**

If data is linearly arranged, then we can separate it by using a straight line, but for non-linear data, we cannot draw a single straight line. Consider the below image:

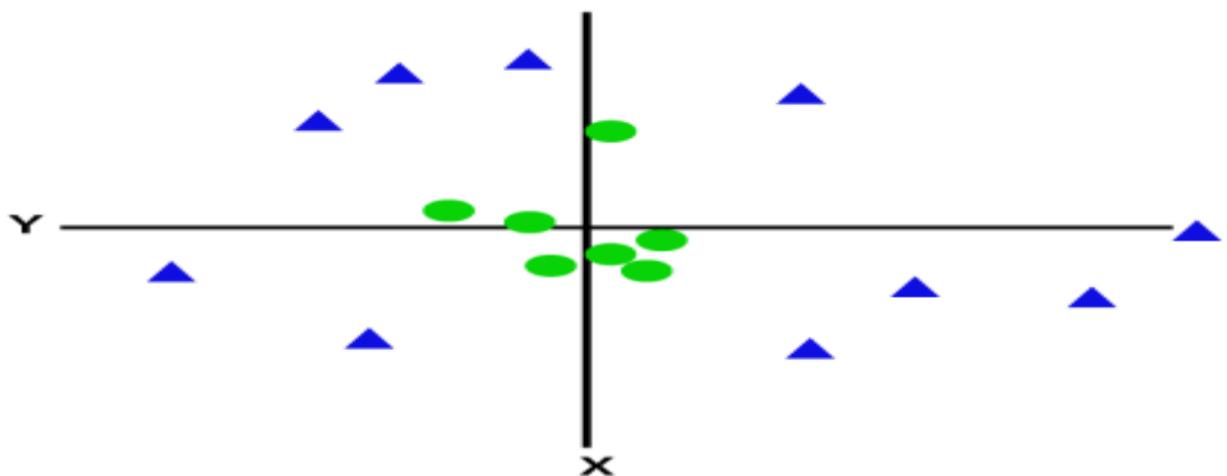


Figure 8.4

## 8.2 Support vector regression in python

Now we will implement the SVM algorithm using Python. Here we will use the same dataset `user_data`, which we have used in Logistic regression and KNN classification.

- **Data preprocessing step**

Till the Data pre-processing step, the code will remain the same. Below is the code:

Step 1: Importing the libraries:

```
1. # importing libraries
2.
3. import numpy as np
4. import matplotlib.pyplot as plt
5. import pandas as pd
6.
7. dataset = pd.read_csv('Position_Salaries.csv')
```

Figure 8.5

Output:



Index	Position	Level	Salary
0	Business Analyst	1	45000
1	Junior Consultant	2	50000
2	Senior Consultant	3	60000
3	Manager	4	80000
4	Country Manager	5	110000
5	Region Manager	6	150000
6	Partner	7	200000
7	Senior Partner	8	300000
8	C-level	9	500000
9	CEO	10	1000000

Figure 8.6

## Step 2: Reading the dataset

```
1. X = dataset.iloc[:, 1:-1].values  
2. y = dataset.iloc[:, -1].values  
3. y = y.reshape(len(y),1)
```

Figure 8.7

Output:

X - NumPy array	
0	0
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9
10	10

Format    Resize     Background color    Save and Close    Close

Figure 8.8

y - NumPy array	
0	450000
1	500000
2	600000
3	800000
4	1100000
5	1500000
6	2000000
7	3000000
8	5000000
9	10000000

Format    Resize     Background color    Save and Close    Close

Figure 8.9

## Step 3: Feature the data

Feature Scaling basically helps to normalize the data within a particular range. Normally several common class types contain the feature scaling function so that they make feature scaling automatically. However, the SVR class is not a commonly used class type so we should perform feature scaling using Python.

```
• # Feature Scaling
•
• from sklearn.preprocessing import StandardScaler
• sc_X = StandardScaler()
• sc_y = StandardScaler()
• X = sc_X.fit_transform(X)
• y = sc_y.fit_transform(y)
```

Figure 8.10

## Step 4: Fitting and Predicting the dataset

### • Fitting and Predicting the dataset

```
1. # Training the SVR model on the whole dataset
2. from sklearn.svm import SVR
3. regressor = SVR(kernel = 'rbf')
4. regressor.fit(X, y)
```

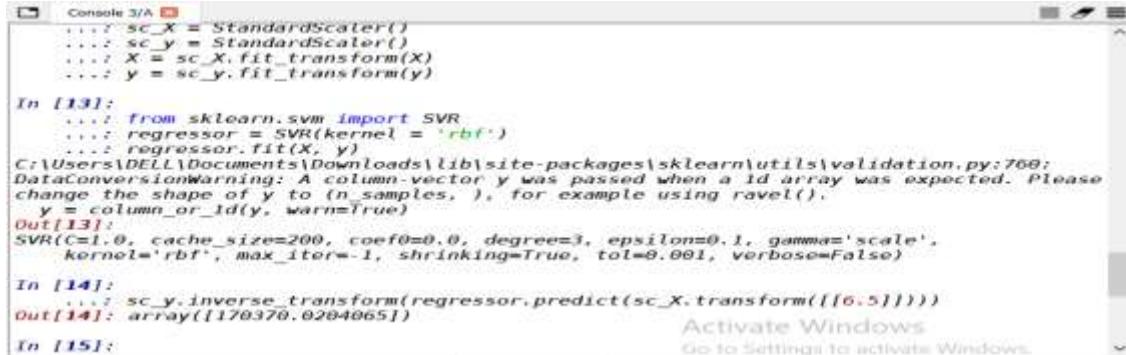
Figure 8.11

- Predicting a new result

```
1. # Predicting a new result
2. sc_y.inverse_transform(regressor.predict(sc_X.transform([[6.5]])))
```

Figure 8.12

## Output:



```
1. sc_X = StandardScaler()
2. sc_y = StandardScaler()
3. X = sc_X.fit_transform(X)
4. y = sc_y.fit_transform(y)

In [13]:
1. from sklearn.svm import SVR
2. regressor = SVR(kernel = 'rbf')
3. regressor.fit(X, y)
C:\Users\DELL\Documents\Downloads\lib\site-packages\sklearn\utils\validation.py:760:
DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please
change the shape of y to (n_samples, ), for example using ravel().
y = column_or_1d(y, warn=True)

Out[13]:
SVR(C=1.0, cache_size=200, coef0=0.0, degree=3, epsilon=0.1, gamma='scale',
kernel='rbf', max_iter=-1, shrinking=True, tol=0.001, verbose=False)

In [14]:
1. sc_y.inverse_transform(regressor.predict(sc_X.transform([[6.5]])))
Out[14]: array([170370.0204065])
```

Figure 8.13

## Step 5: Visualising the SVR results

```
1. # Visualising the SVR results
2.
3. plt.scatter(sc_X.inverse_transform(X), sc_y.inverse_transform(y), color = 'red')
4. plt.plot(sc_X.inverse_transform(X), sc_y.inverse_transform(regressor.predict(X)), color
   = 'blue')
5. plt.title('Truth or Bluff (SVR)')
6. plt.xlabel('Position level')
7. plt.ylabel('Salary')
8. plt.show()
```

Figure 8.14

## Output:

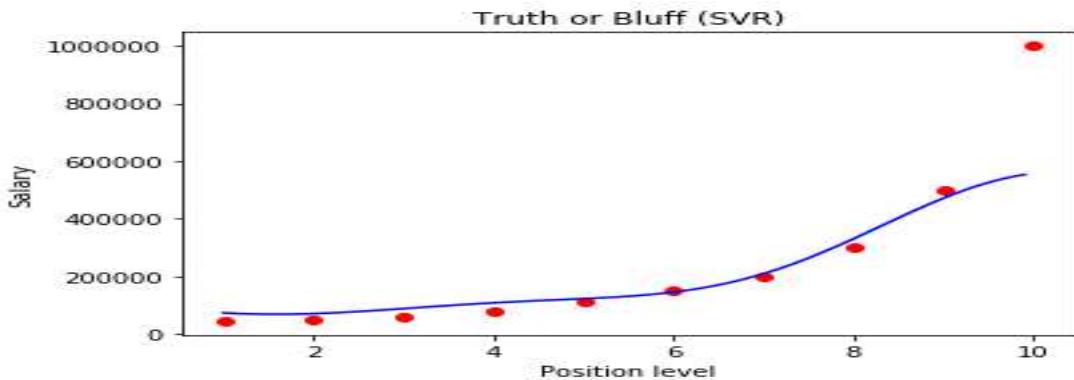


Figure 8.15

- Visualizing the SVR results (for higher resolution and smoother curve)

```
1. # Visualising the SVR results (for higher resolution and smoother curve)
2.
3. X_grid = np.arange(min(sc_X.inverse_transform(X)), max(sc_X.inverse_transform(X)), 0.1)

4. X_grid = X_grid.reshape((len(X_grid), 1))
5. plt.scatter(sc_X.inverse_transform(X), sc_y.inverse_transform(y), color = 'red')
6. plt.plot(X_grid, sc_y.inverse_transform(regressor.predict(sc_X.transform(X_grid))), color = 'blue')
7. plt.title('Truth or Bluff (SVR)')
8. plt.xlabel('Position level')
9. plt.ylabel('Salary')
10. plt.show()
```

Figure 8.16

Output:

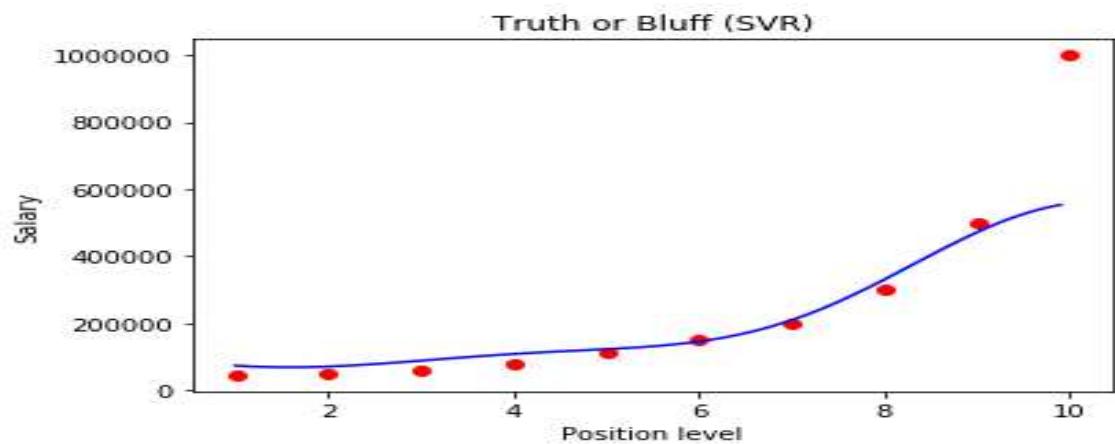


Figure 8.17

# CHAPTER 9

## Decision Tree Regression

### 9.1 Overview

Decision Tree is one of the most commonly used, practical approaches for supervised learning. It can be used to solve both Regression and Classification tasks with the latter being put more into practical application.

### 9.2 Decision Tree Regression in python

Step 1: Importing the libraries and dataset

The first step will always consist of importing the libraries that are needed to develop the ML model. The NumPy, matplotlib and the Pandas libraries are imported.

```
1. # Decision Tree Regression
2.
3. # Importing the libraries
4. import numpy as np
5. import matplotlib.pyplot as plt
6. import pandas as pd
7.
8. # Importing the dataset
9. dataset = pd.read_csv('Position_Salaries.csv')
```

Figure 9.1

## Output:

Index	Position	Level	Salary
0	Business Analyst	1	45000
1	Junior Consultant	2	50000
2	Senior Consultant	3	60000
3	Manager	4	80000
4	Country Manager	5	110000
5	Region Manager	6	150000
6	Partner	7	200000
7	Senior Partner	8	300000
8	C-level	9	500000
9	CEO	10	1000000

Figure 9.2

## Step 2 :Importing the X and Y

In this step, we shall use pandas to store the data obtained from my github repository and store it as a Pandas DataFrame using the function ‘Position\_salaries.csv’. In this, we assign the independent variable (X) to the ‘Level’ column and the dependent variable (y) to the ‘Salary’.

```
1. X = dataset.iloc[:, 1:-1].values
2. y = dataset.iloc[:, -1].values
```

Figure 9.3

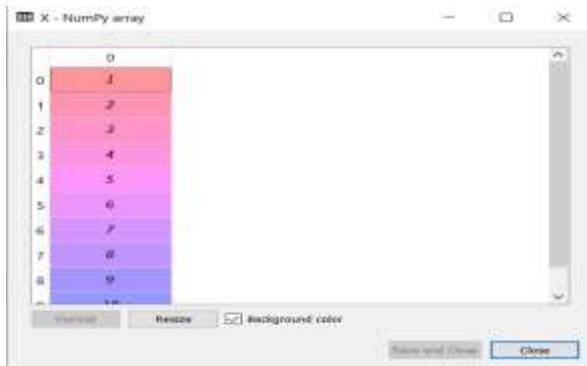


Figure 9.4



Figure 9.5

Step 3: Training the decision tree regression model on the training set:

We import the DecisionTreeRegressor class from `sklearn.tree` and assign it to the variable 'regressor'.

```
1. # Training the Decision Tree Regression model on the whole dataset
2. from sklearn.tree import DecisionTreeRegressor
3. regressor = DecisionTreeRegressor(random_state = 0)
4. regressor.fit(X, y)
```

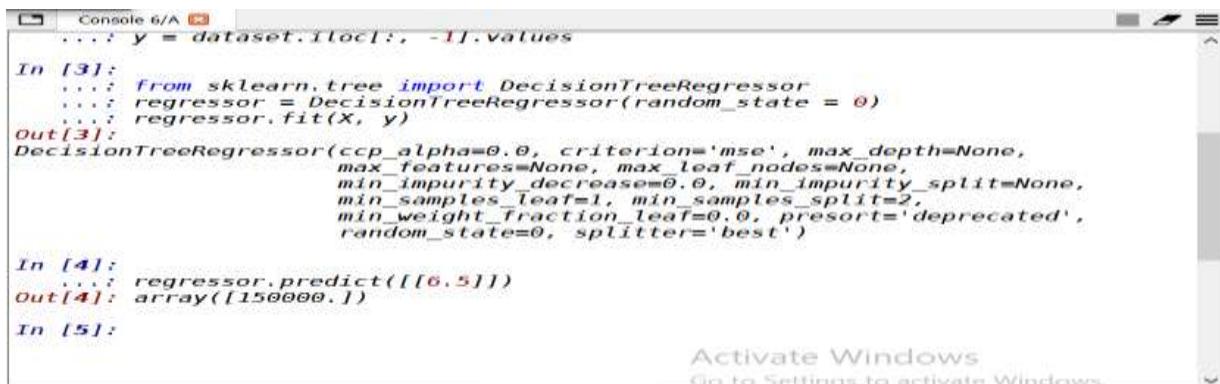
Figure 9.6

## Step 4: Predicting the results

In this step, we predict the results of the test set with the model trained on the training set values using the `regressor.predict` function and assign it to '`regressor`'.

```
1. # Predicting a new result
2. regressor.predict([[6.5]])
```

Figure 9.7



```
...: y = dataset.iloc[:, -1].values
...:
In [3]:
...: from sklearn.tree import DecisionTreeRegressor
...: regressor = DecisionTreeRegressor(random_state = 0)
...: regressor.fit(X, y)
Out[3]:
DecisionTreeRegressor(ccp_alpha=0.0, criterion='mse', max_depth=None,
                      max_features=None, max_leaf_nodes=None,
                      min_impurity_decrease=0.0, min_impurity_split=None,
                      min_samples_leaf=1, min_samples_split=2,
                      min_weight_fraction_leaf=0.0, presort='deprecated',
                      random_state=0, splitter='best')
In [4]:
...: regressor.predict([[6.5]])
Out[4]:
array([150000.1])
In [5]:
```

Figure 9.8

## Step 5: Visualising the Decision Tree Regression Results

In this graph, the Real values are plotted with “Red” color and the Predicted values are plotted with “Green” color. The plot of the Decision Tree Regression model is also drawn in “Black” color.

```
1. # Visualising the Decision Tree Regression results (higher resolution)
```

```

2.
3. X_grid = np.arange(min(X), max(X), 0.01)
4. X_grid = X_grid.reshape((len(X_grid), 1))
5. plt.scatter(X, y, color = 'red')
6. plt.plot(X_grid, regressor.predict(X_grid), color = 'blue')
7. plt.title('Truth or Bluff (Decision Tree Regression)')
8. plt.xlabel('Position level')
9. plt.ylabel('Salary')
10. plt.show()

```

Figure 9.9

Output:



Figure 9.10

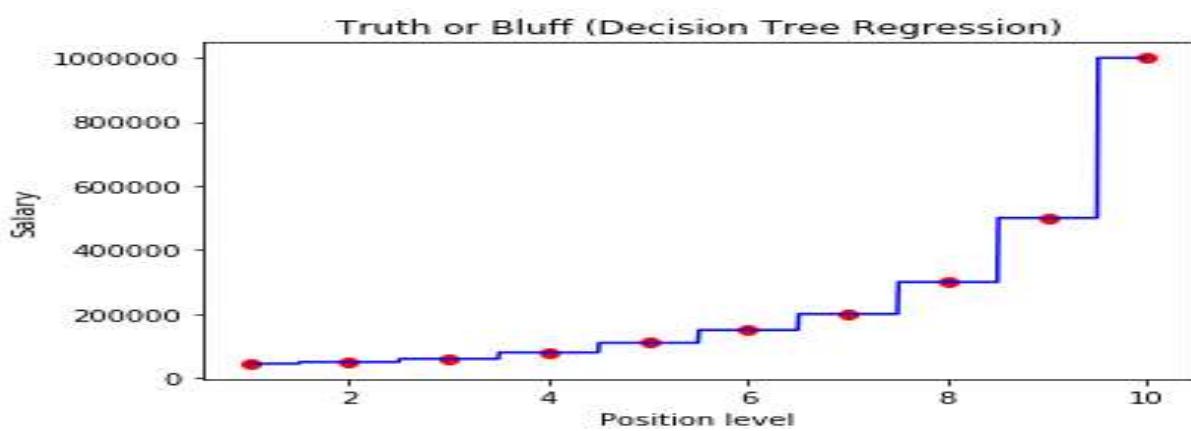


Figure 9.11

# CHAPTER 10

---

## Random Forest Regression

### 10.1 Overview

A Random Forest is an ensemble technique capable of performing both regression and classification tasks with the use of multiple decision trees and a technique called Bootstrap and Aggregation, commonly known as bagging. The basic idea behind this is to combine multiple decision trees in determining the final output rather than relying on individual decision trees.

Random Forest has multiple decision trees as base learning models. We randomly perform row sampling and feature sampling from the dataset forming sample datasets for every model. This part is called Bootstrap.

We need to approach the Random Forest regression technique like any other machine learning technique

Design a specific question or data and get the source to determine the required data.

Make sure the data is in an accessible format else convert it to the required format.

- Specify all noticeable anomalies and missing data points that may be required to achieve the required data.

- Create a machine learning model
- Set the baseline model that you want to achieve
- Train the data machine learning model.
- Provide an insight into the model with test data
- Now compare the performance metrics of both the test data and the predicted data from the model.
- If it doesn't satisfy your expectations, you can try improving your model accordingly or dating your data or use another data modeling technique.

At this stage you interpret the data you have gained and report accordingly.

## 10.2 Decision Tree Regression in python

## Step 1: How to Import The Libraries and Data Set

The main libraries are called to handle the model , the following command imports the CSV dataset using pandas:

```
1. # Decision Tree Regression
2.
3. # Importing the libraries
4. import numpy as np
5. import matplotlib.pyplot as plt
6. import pandas as pd
7.
8. # Importing the dataset
9. dataset = pd.read_csv('Position_Salaries.csv')
```

Figure 10.1

To ensure that the dataset has been called, highlight the item, and then press F9



Index	Position	Level	Salary
0	<i>Business Analyst</i>	1	45000
1	<i>Junior Consultant</i>	2	50000
2	<i>Senior Consultant</i>	3	60000
3	<i>Manager</i>	4	80000
4	<i>Country Manager</i>	5	110000
5	<i>Region Manager</i>	6	150000
6	<i>Partner</i>	7	200000
7	<i>Senior Partner</i>	8	300000
8	<i>C-level</i>	9	500000
9	<i>CEO</i>	10	1000000

Figure 10.2

## Step 2: Importing the X and Y:

Dividing the dataset into 2 components

```
1. X = dataset.iloc[:, 1:-1].values
2. y = dataset.iloc[:, -1].values
```

Figure 10.3



Figure 10.4



Figure 10.5

### Step 3: Training the random forest regression model on the whole dataset:

```
1. # Training the Decision Tree Regression model on the whole dataset
2.
3. from sklearn.tree import DecisionTreeRegressor
4. regressor = DecisionTreeRegressor(random_state = 0)
5. regressor.fit(X, y)
```

Figure 10.6

### Step 4: predicting a new result:

```
1. # Predicting a new result
2.
3. regressor.predict([[6.5]])
```

Figure 10.7



```
Out[4]: RandomForestRegressor(bootstrap=True, ccp_alpha=0.0, criterion='mse',
max_depth=None, max_features='auto',
max_leaf_nodes=None,
max_samples=None, min_impurity_decrease=0.0,
min_impurity_split=None, min_samples_leaf=1,
min_samples_split=2, min_weight_fraction_leaf=0.0,
n_estimators=10, n_jobs=None, oob_score=False,
random_state=0, verbose=0, warm_start=False)

In [5]: regressor.predict([[6.5]])
Out[5]: array([167000.])

In [6]:
```

Activate Windows  
Go to Settings to activate Windows.

Figure 10.8

## Step 5: Visualising the random forest regression results:

```
1. # Visualising the Decision Tree Regression results (higher resolution)
2.
3. X_grid = np.arange(min(X), max(X), 0.01)
4. X_grid = X_grid.reshape((len(X_grid), 1))
5. plt.scatter(X, y, color = 'red')
6. plt.plot(X_grid, regressor.predict(X_grid), color = 'blue')
7. plt.title('Truth or Bluff (Decision Tree Regression)')
8. plt.xlabel('Position level')
9. plt.ylabel('Salary')
10. plt.show()
```

Figure 10.9

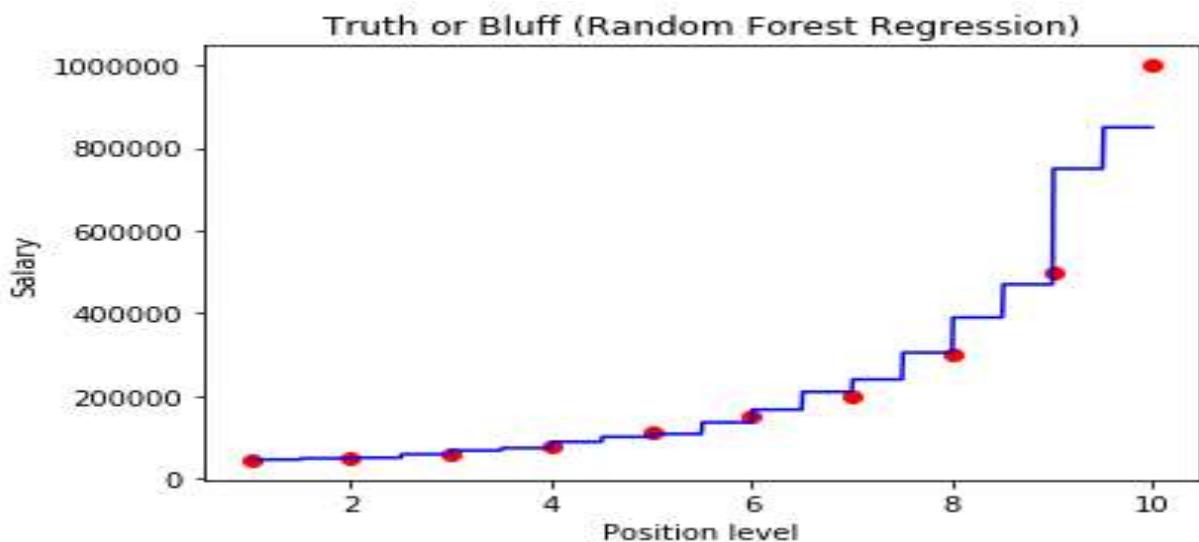


Figure 10.10

---

# CHAPTER 11

---

## Evaluating Regression Models Performance

### 11.1 Overview

Model evaluation is very important in data science. It helps you to understand the performance of your model and makes it easy to present your model to other people. There are many different evaluation metrics out there but only some of them are suitable to be used for regression. This article will cover the different metrics for regression model and difference between them. Hopefully after you read this post, you are clear on which metrics to apply to your future regression model.

Every time when I tell my friends: “Hey, I have built a machine learning model to predict XXX.” Their first reaction would be: “Cool, so what is the accuracy of your model prediction?” Well, unlike classification, accuracy in regression model is slightly harder to illustrate. It is impossible for you to predict the exact value but rather **how close your prediction is against the real value.**

There are 3 main metrics for model evaluation in regression:

1. R Square/Adjusted R Square
2. Mean Square Error(MSE)/Root Mean Square Error(RMSE)
3. Mean Absolute Error(MAE)

**R-squared** is a statistical measure that represents the goodness of fit of a regression model. The ideal value for r-square is 1. The closer the value of r-square to 1, the better is the model fitted.

R-square is a comparison of residual sum of squares ( $SS_{res}$ ) with total sum of squares( $SS_{tot}$ ). Total sum of squares is calculated by summation of squares of perpendicular distance between data points and the average line.

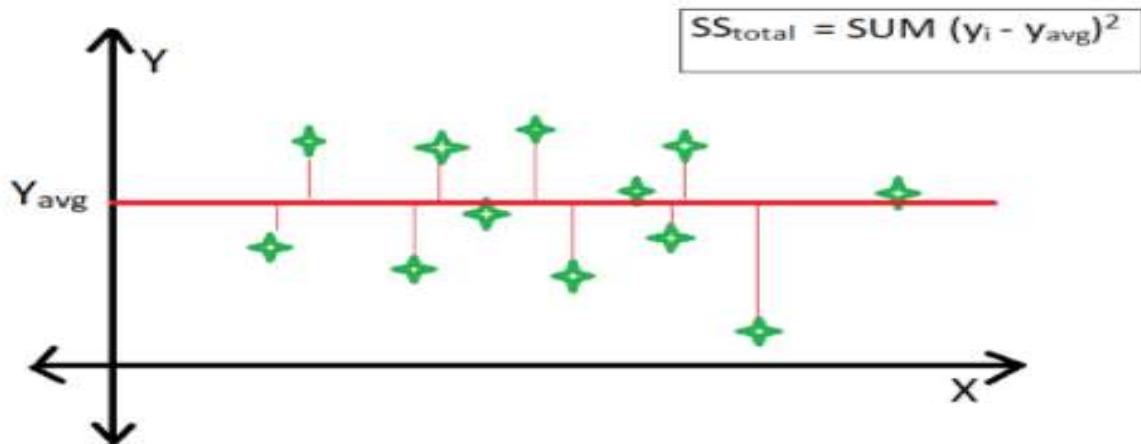


Figure 11.1

Residual sum of squares is calculated by the summation of squares of perpendicular distance between data points and the best fitted line.

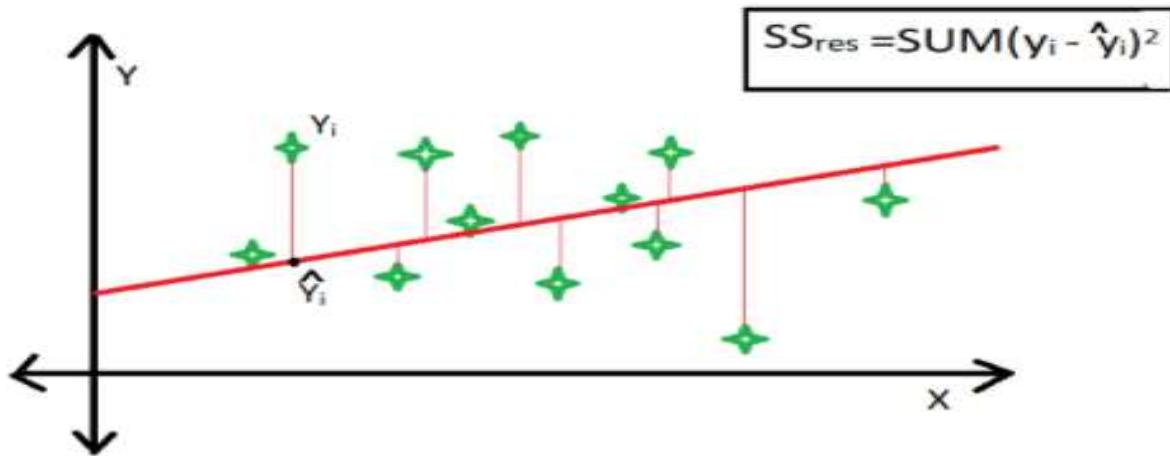


Figure 11.2

R square is calculated by using the following formula :

$$R^2 = 1 - \frac{SS_{res}}{SS_{tot}}$$

Figure 11.3

Where  $SS_{res}$  is the residual sum of squares and  $SS_{tot}$  is the total sum of squares.

The goodness of fit of regression models can be analyzed on the basis of R-square method. The more the value of r-square near to 1, the better is the model.

**Note :** The value of R-square can also be negative when the models fitted is worse than the average fitted model.

### Limitation of using R-square method :

- The value of r-square always increases or remains same as new variables are added to the model, without detecting the significance of this newly added variable (i.e value of r-square never decreases on addition of new attributes to the model). As a result, non-significant attributes can also be added to the model with an increase in r-square value.
- This is because  $SS_{tot}$  is always constant and regression model tries to decrease the value of  $SS_{res}$  by finding some correlation with this new attribute and hence the overall value of r-square increases, which can lead to a poor regression model.

- **Adjusted R-squared**

There is a drawback of  $R^2$  that it improves every time when we add new variables in the model.

Think about it, whenever you add a new variable there can be two circumstances, either the new variable improves your model or not. When the new variable improves your model then it is ok. But what if it does not improve your model? Then the problem occurs. The value of  $R^2$  keeps on increasing with the addition of more independent variables even though they may not have a significant impact on the prediction.

To solve this pitfall, an Adjusted  $R^2$  value is used instead of  $R^2$  value. The mathematical representation for Adjusted  $R^2$  is-

$$\text{Adj } R^2 = 1 - (1 - R^2) \frac{n - 1}{n - p - 1}$$

p - number of regressors

n - sample size

Figure 11.4

Adjusted  $R^2$  will penalize the model whenever you add a new variable to it. From the equation, you can understand that clearly. Whenever you add a new variable, the value of  $R^2$  increases and it also increases the denominator( $n - p - 1$ ) on the left of the equation. As the denominator increases, it increases the value of  $1 - R^2$  on the left of the equation. Which in turn decreases the value of Adjusted  $R^2$ . Hence, using an Adjusted  $R^2$  value you can better understand the effect of the additional variables to your model.

**Note:** Adjusted  $R^2$  is greatly helpful when your dataset contains more features and you need to choose the most effective features to train your model.

### **Which Regression Model is the Best?**

Well, this is subjective to your dataset and the model you choose. Each machine learning model solves a problem with a different objective using a different dataset. Hence, you must understand the context of using that model before choosing a metric.

---

# CHAPTER 12

---

## Classification

### 12.1 Overview

Unlike regression where you predict a continuous number, you use classification to predict a category. There is a wide variety of classification applications from medicine to marketing. Classification models include linear models like Logistic Regression, SVM, and nonlinear ones like K-NN, Kernel SVM and Random Forests.

In this part, you will understand and learn how to implement the following Machine Learning Classification models:

1. Logistic Regression
2. K-Nearest Neighbors (K-NN)
3. Support Vector Machine (SVM)
4. Kernel SVM
5. Naive Bayes
6. Decision Tree Classification
7. Random Forest Classification

---

# CHAPTER 13

---

## Logistic Regression

### 13.1 Overview

Logistic regression is a Machine Learning classification algorithm that is used to predict the probability of a categorical dependent variable. In logistic regression, the dependent variable is a binary variable that contains data coded as 1 (yes, success, etc.) or 0 (no, failure, etc.). In other words, the logistic regression model predicts  $P(Y=1)$  as a function of  $X$ .

### 13.2 Logistic Regression Intuition

Classification is a very important area of supervised machine learning. A large number of important machine learning problems fall within this area. There are many classification methods, and logistic regression is one of them.

#### What Is Classification?

Supervised machine learning algorithms define models that capture relationships among data. Classification is an area of supervised machine learning that tries to predict which class or category some entity belongs to, based on its features.

For example, you might analyze the employees of some company and try to establish a dependence on the features or variables, such as the level of education, number of years in a current position, age, salary, odds for being promoted, and so on. The set of data related to a single employee is one observation. The features or variables can take one of two forms:

Independent variables, also called inputs or predictors, don't depend on other features of interest (or at least you assume so for the purpose of the analysis).

Dependent variables, also called outputs or responses, depend on the independent variables.

## Math Prerequisites

You'll need an understanding of the sigmoid function and the natural logarithm function to understand what logistic regression is and how it works.

### 13.3 Import libraries and dataset

In this step, we will pre-process/prepare the data so that we can use it in our code efficiently. It will be the same as we have done in Data pre-processing topic. The code for this is given below:

```
1. # Logistic Regression
2.
3. # Importing the libraries
4. import numpy as np
5. import matplotlib.pyplot as plt
6. import pandas as pd
7.
8. # Importing the dataset
9. dataset = pd.read_csv('Social_Network_Ads.csv')
```

Figure 13.1

Output: By executing the above lines of code, we will get the dataset as the output. Consider the given image:

Index	User ID	Gender	Age	matedSal	Purchased
0	15624510	Male	19	190000	0
1	15810944	Male	35	200000	0
2	15668575	Female	26	430000	0
3	15603246	Female	27	570000	0
4	15804002	Male	19	760000	0
5	15728773	Male	27	580000	0
6	15598044	Female	27	840000	0
7	15694829	Female	32	1500000	1
8	15600575	Male	25	330000	0
9	15727311	Female	35	650000	0
10	15570769	Female	26	800000	0

Figure 13.2

## 13.4 Dividing the dataset into 2 components

Now, we will extract the dependent and independent variables from the given dataset. Below is the code for it:

```
1. X = dataset.iloc[:, [2, 3]].values
2. y = dataset.iloc[:, -1].values
```

Figure 13.3

## Output:

X - NumPy array		
	0	1
0	19	190000
1	35	200000
2	26	430000
3	27	570000
4	19	760000
5	27	580000
6	27	840000
7	32	1500000
8	25	330000
9	35	650000

Figure 13.4

y - NumPy array		
	0	
0	0	
1	0	
2	0	
3	0	
4	0	
5	0	
6	0	
7	1	
8	0	
9	0	

Figure 13.5

## 13.5 Splitting the dataset into the Training set and Test set

Now we will split the dataset into a training set and test set. Below is the code for it:

```
1. # Splitting the dataset into the Training set and Test set
2.
3. from sklearn.model_selection import train_test_split
4. X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_state = 0)
```

Figure 13.6

**Output:**

X\_test - NumPy array

	0	1
0	39	870000
1	38	500000
2	35	750000
3	38	790000
4	35	500000
5	27	200000
6	31	150000
7	36	1440000
8	18	680000
9	42	430000

Format Resize  Background color Save and Close Close

Figure 13.7

X\_train - NumPy array

	0	1
0	44	390000
1	32	1200000
2	38	500000
3	32	1350000
4	52	210000
5	53	1040000
6	39	420000
7	38	610000
8	36	500000
9	36	530000

Format Resize  Background color Save and Close Close

Figure 13.8

y_test - NumPy array	
0	0
1	0
2	0
3	0
4	0
5	0
6	0
7	1
8	0
9	0

Figure 13.9

y_train - NumPy array	
0	0
1	1
2	0
3	1
4	1
5	1
6	0
7	0
8	0
9	0

Figure 13.10

## 13.6 Feature Scaling

In logistic regression, we will do feature scaling because we want accurate result of predictions. Here we will only scale the independent variable because dependent variable have only 0 and 1 values. Below is the code for it:

```
1. # Feature Scaling
2.
3. from sklearn.preprocessing import StandardScaler
4. sc = StandardScaler()
5. X_train = sc.fit_transform(X_train)
6. X_test = sc.transform(X_test)
```

Figure 13.11

### 13.7 Training the Logistic Regression model on the Training set

We have well prepared our dataset, and now we will train the dataset using the training set. For providing training or fitting the model to the training set, we will import the **LogisticRegression** class of the **sklearn** library.

After importing the class, we will create a classifier object and use it to fit the model to the logistic regression. Below is the code for it:

```
1. # Training the Logistic Regression model on the Training set
2.
3. from sklearn.linear_model import LogisticRegression
4. classifier = LogisticRegression(random_state = 0)
5. classifier.fit(X_train, y_train)
```

Figure 13.12

### 13.8 Predicting the Test set results

Our model is well trained on the training set, so we will now predict the result by using test set data. Below is the code for it:

```
1. # Predicting the Test set results
```

```
2.  
3. y_pred = classifier.predict(X_test)
```

Figure 13.13



Figure 13.14

### 13.9 Making the Confusion Matrix

Now we will create the confusion matrix here to check the accuracy of the classification. To create it, we need to import the `confusion_matrix` function of the `sklearn` library. After importing the function, we will call it using a new variable `cm`. The function takes two parameters, mainly `y_true` (the actual values) and `y_pred` (the targeted value return by the classifier). Below is the code for it:

```
1. # Making the Confusion Matrix  
2.  
3. from sklearn.metrics import confusion_matrix  
4. cm = confusion_matrix(y_test, y_pred)  
5. print(cm)
```

Figure 13.15

**Output:**



Figure 13.16

### 13.10 Visualising the Training set results

Finally, we will visualize the training set result. To visualize the result, we will use **ListedColormap** class of matplotlib library. Below is the code for it:

```

1. # Visualising the Training set results
2.
3. from matplotlib.colors import ListedColormap
4. X_set, y_set = X_train, y_train
5. X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 1, stop = X_set[:, 0].max() + 1, step = 0.01),
6.                      np.arange(start = X_set[:, 1].min() - 1, stop = X_set[:, 1].max() + 1, step = 0.01))
7. plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape),
8.               alpha = 0.75, cmap = ListedColormap(('red', 'green')))
9. plt.xlim(X1.min(), X1.max())
10. plt.ylim(X2.min(), X2.max())
11. for i, j in enumerate(np.unique(y_set)):
12.     plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
13.                 c = ListedColormap(('red', 'green'))(i), label = j)
14. plt.title('Logistic Regression (Training set)')
15. plt.xlabel('Age')
16. plt.ylabel('Estimated Salary')
17. plt.legend()
18. plt.show()

```

Figure 13.17

**Output:**

X1 - NumPy array

	0	1	2	3	^
0	-2.99319	-2.98319	-2.97319	-2.96319	<
1	-2.99319	-2.98319	-2.97319	-2.96319	>
2	-2.99319	-2.98319	-2.97319	-2.96319	
3	-2.99319	-2.98319	-2.97319	-2.96319	
4	-2.99319	-2.98319	-2.97319	-2.96319	
5	-2.99319	-2.98319	-2.97319	-2.96319	
6	-2.99319	-2.98319	-2.97319	-2.96319	
7	-2.99319	-2.98319	-2.97319	-2.96319	
8	-2.99319	-2.98319	-2.97319	-2.96319	

Format    Resize     Background color    Save and Close    Close

Figure 13.18

X2 - NumPy array

	0	1	2	3	^
0	-2.58254	-2.58254	-2.58254	-2.58254	<
1	-2.57254	-2.57254	-2.57254	-2.57254	>
2	-2.56254	-2.56254	-2.56254	-2.56254	
3	-2.55254	-2.55254	-2.55254	-2.55254	
4	-2.54254	-2.54254	-2.54254	-2.54254	
5	-2.53254	-2.53254	-2.53254	-2.53254	
6	-2.52254	-2.52254	-2.52254	-2.52254	
7	-2.51254	-2.51254	-2.51254	-2.51254	
8	-2.50254	-2.50254	-2.50254	-2.50254	

Format    Resize     Background color    Save and Close    Close

Figure 13.19

X\_set - NumPy array

	0	1
0	0.581649	-0.886707
1	-0.606738	1.46174
2	-0.0125441	-0.567782
3	-0.606738	1.89663
4	1.37391	-1.40858
5	1.47294	0.997847
6	0.0864882	-0.799728
7	-0.0125441	-0.248858
8	-0.210609	-0.567782
9	-0.310609	-0.100072

Format Resize  Background color

Save and Close Close

Figure 13.20

y\_set - NumPy array

	0
0	0
1	1
2	0
3	1
4	1
5	1
6	0
7	0
8	0
9	0

Format Resize  Background color

Save and Close Close

Figure 13.21

X1 - NumPy array

	0	1	2	3
0	-2.99319	-2.98319	-2.97319	-2.96319
1	-2.99319	-2.98319	-2.97319	-2.96319
2	-2.99319	-2.98319	-2.97319	-2.96319
3	-2.99319	-2.98319	-2.97319	-2.96319
4	-2.99319	-2.98319	-2.97319	-2.96319
5	-2.99319	-2.98319	-2.97319	-2.96319
6	-2.99319	-2.98319	-2.97319	-2.96319
7	-2.99319	-2.98319	-2.97319	-2.96319
8	-2.99319	-2.98319	-2.97319	-2.96319

Format    Resize     Background color    Save and Close    Close

Figure 13.22



Figure 13.23

### 13.11 Visualising the test set results

Our model is well trained using the training dataset. Now, we will visualize the result for new observations (Test set). The code for the test set will remain same as above except that here we will use `x_test` and `y_test` instead of `x_train` and `y_train`. Below is the code for it:

```

1. # Visualising the Test set results
2.
3. from matplotlib.colors import ListedColormap
4. X_set, y_set = X_test, y_test
5. X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 1, stop = X_set[:, 0].max() + 1, step = 0.01),
6.                      np.arange(start = X_set[:, 1].min() - 1, stop = X_set[:, 1].max() + 1, step = 0.01))
7. plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape),
8.                 alpha = 0.75, cmap = ListedColormap(('red', 'green')))
9. plt.xlim(X1.min(), X1.max())
10. plt.ylim(X2.min(), X2.max())
11. for i, j in enumerate(np.unique(y_set)):
12.     plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
13.                 c = ListedColormap(('red', 'green'))(i), label = j)
14. plt.title('Logistic Regression (Test set)')
15. plt.xlabel('Age')
16. plt.ylabel('Estimated Salary')
17. plt.legend()
18. plt.show()

```

Figure 13.24

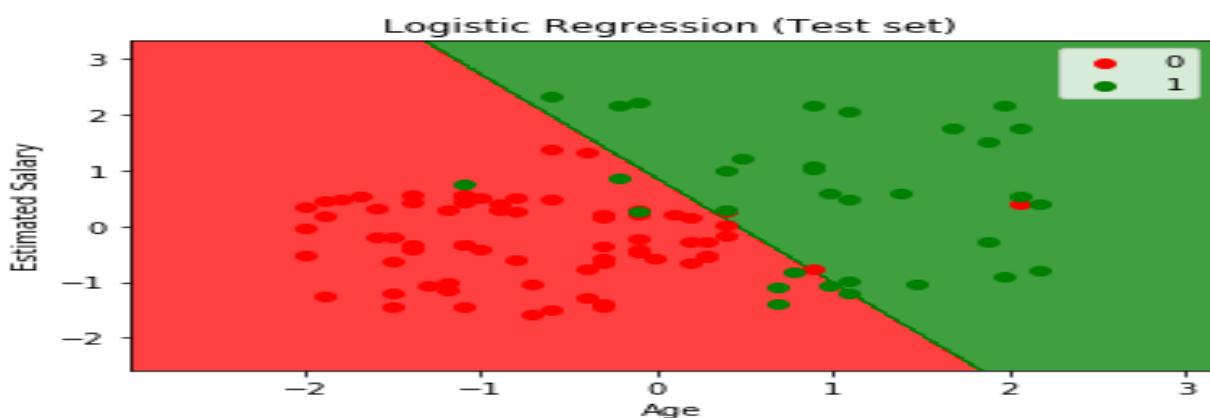


Figure 13.25

---

---

# CHAPTER 14

---

## K-Nearest Neighbors(K-NN)

### 14.1 Overview

- K-Nearest Neighbour is one of the simplest Machine Learning algorithms based on Supervised Learning technique.
- K-NN algorithm assumes the similarity between the new case/data and available cases and put the new case into the category that is most similar to the available categories.
- K-NN algorithm stores all the available data and classifies a new data point based on the similarity. This means when new data appears then it can be easily classified into a well suited category by using K- NN algorithm.
- K-NN algorithm can be used for Regression as well as for Classification but mostly it is used for the Classification problems.

**Why do we need a K-NN Algorithm?**

Suppose there are two categories, i.e., Category A and Category B, and we have a new data point  $x_1$ , so this data point will lie in which of these categories. To solve this type of problem, we need a K-NN algorithm. With the help of K-NN, we can easily identify the category or class of a particular dataset. Consider the below diagram:

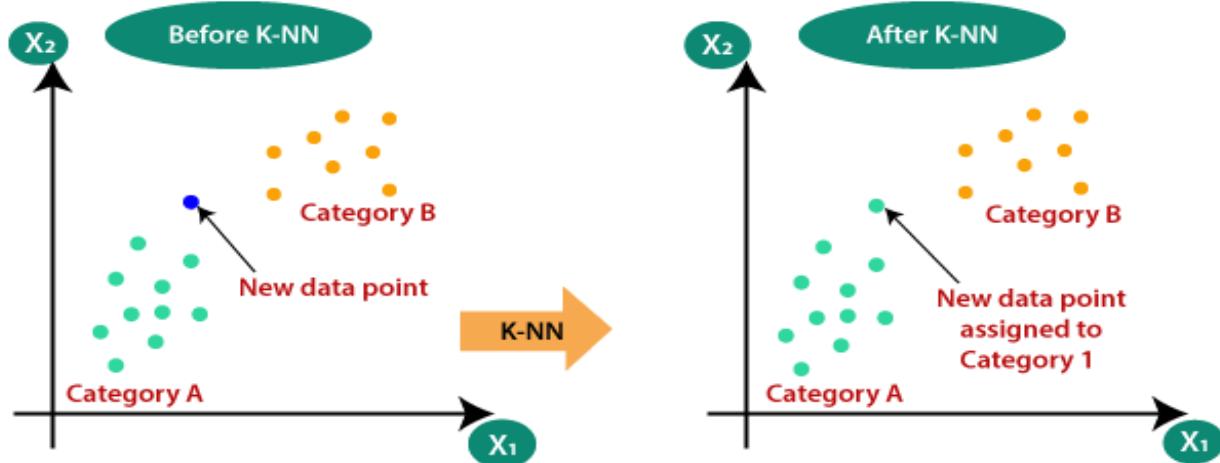


Figure 14.1

- K-Nearest Neighbor(KNN) Algorithm for Machine Learning
  - K-Nearest Neighbour is one of the simplest Machine Learning algorithms based on Supervised Learning technique.
  - K-NN algorithm assumes the similarity between the new case/data and available cases and put the new case into

the category that is most similar to the available categories.

- K-NN algorithm stores all the available data and classifies a new data point based on the similarity. This means when new data appears then it can be easily classified into a well suited category by using K-NN algorithm.
- K-NN algorithm can be used for Regression as well as for Classification but mostly it is used for the Classification problems.
- K-NN is a **non-parametric algorithm**, which means it does not make any assumption on underlying data.
- It is also called a **lazy learner algorithm** because it does not learn from the training set immediately instead it stores the dataset and at the time of classification, it performs an action on the dataset.
- KNN algorithm at the training phase just stores the dataset and when it gets new data, then it classifies that data into a category that is much similar to the new data.

## Why do we need a K-NN Algorithm?

Suppose there are two categories, i.e., Category A and Category B, and we have a new data point  $x_1$ , so this data point will lie in which of these categories. To solve this type of problem, we need a K-NN algorithm. With the help of K-NN, we can easily identify the category or class of a particular dataset. Consider the below diagram:

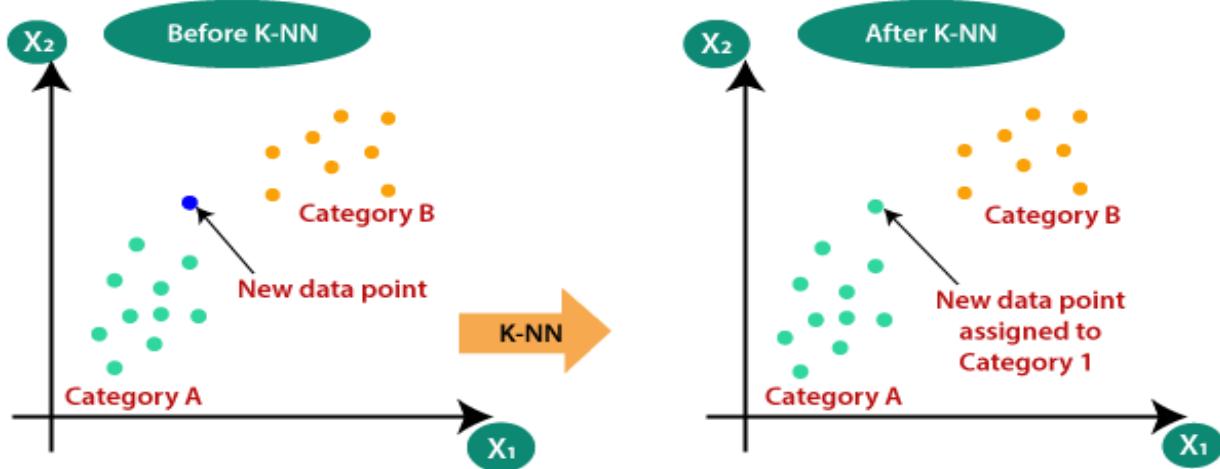


Figure 14.2

## How does K-NN work?

The K-NN working can be explained on the basis of the below algorithm:

**Step-1:** Select the number K of the neighbors

**Step-2:** Calculate the Euclidean distance of K number of neighbors

**Step-3:** Take the K nearest neighbors as per the calculated Euclidean distance.

**Step-4:** Among these  $k$  neighbors, count the number of the data points in each category.

**Step-5:** Assign the new data points to that category for which the number of the neighbor is maximum.

**Step-6:** Our model is ready.

Suppose we have a new data point and we need to put it in the required category. Consider the below image:

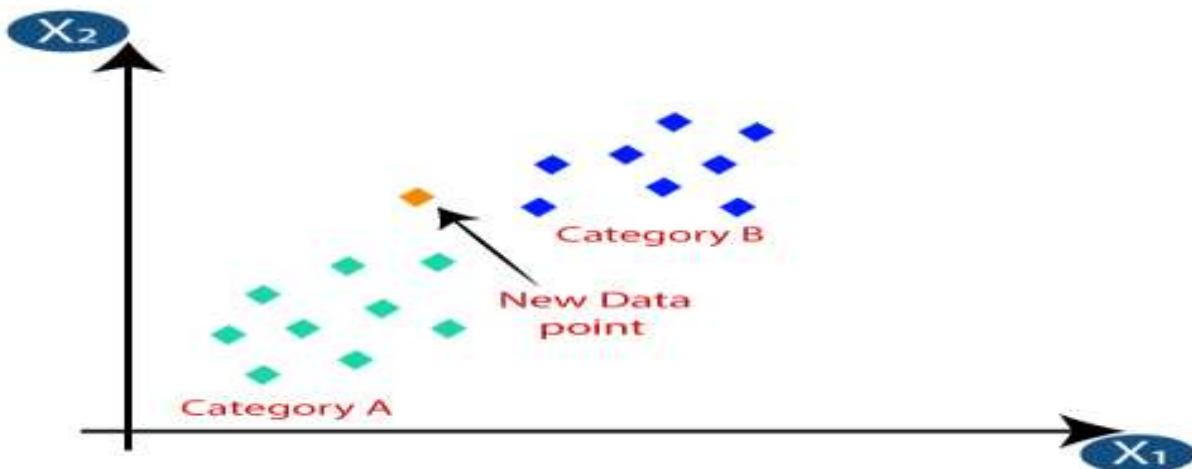


Figure 14.3

- Firstly, we will choose the number of neighbors, so we will choose the  $k=5$ .
- Next, we will calculate the **Euclidean distance** between the data points. The Euclidean distance is the distance between two points, which we have already studied in geometry. It can be calculated as:

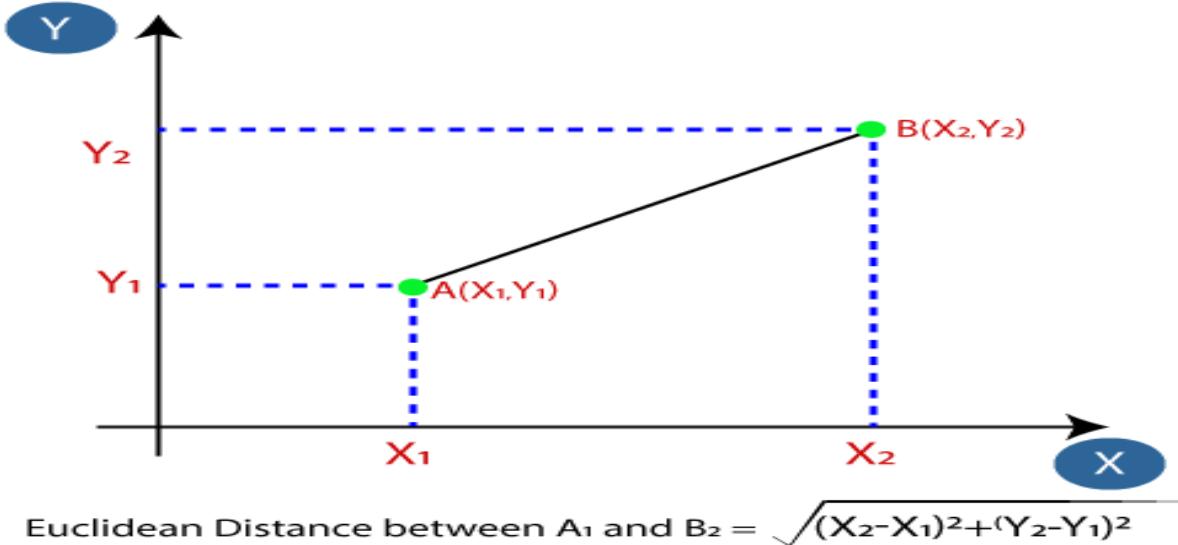


Figure 14.4

By calculating the Euclidean distance we got the nearest neighbors, as three nearest neighbors in category A and two nearest neighbors in category B. Consider the below image:

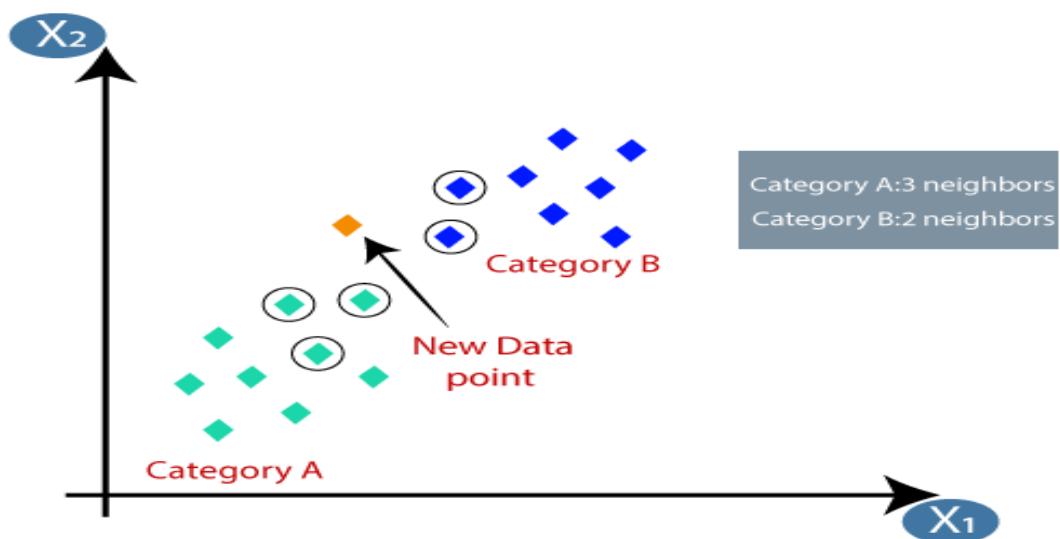


Figure 14.5

As we can see the 3 nearest neighbors are from category A, hence this new data point must belong to category A.

## **How to select the value of K in the K-NN Algorithm?**

Below are some points to remember while selecting the value of K in the K-NN algorithm:

- There is no particular way to determine the best value for "K", so we need to try some values to find the best out of them. The most preferred value for K is 5.
- A very low value for K such as K=1 or K=2, can be noisy and lead to the effects of outliers in the model.
- Large values for K are good, but it may find some difficulties.

## **Advantages of KNN Algorithm:**

- It is simple to implement.
- It is robust to the noisy training data
- It can be more effective if the training data is large.

## **Disadvantages of KNN Algorithm:**

- Always needs to determine the value of K which may be complex some time.
- The computation cost is high because of calculating the distance between the data points for all the training samples.

- **Steps in K-Nearest Neighbour in Machine Learning**

To do the Python implementation of the K-NN algorithm, we will use the same problem and dataset which we have used in Logistic Regression. But here we will improve the performance of the model. Below is the problem description:

**Problem for K-NN Algorithm:** There is a Car manufacturer company that has manufactured a new SUV car. The company wants to give the ads to the users who are interested in buying that SUV. So for this problem, we have a dataset that contains multiple user's information through the social network. The dataset contains lots of information but the Estimated Salary and Age we will consider for the independent variable and the Purchased variable is for the dependent variable. Below is the dataset:

## **14.2 Steps to implement the K-NN algorithm:**

- Data Pre-processing step
- Fitting the K-NN algorithm to the Training set
- Predicting the test result
- Test accuracy of the result(Creation of Confusion matrix)
- Visualizing the test set result.

## Data Pre-Processing Step:

The Data Pre-processing step will remain exactly the same as Logistic Regression. Below is the code for it:

- Importing the libraries

```

14     # K-Nearest Neighbors (K-NN)
15
16     # Importing the libraries
17     import numpy as np
18     import matplotlib.pyplot as plt
19     import pandas as pd
20
21
22
23     # Importing the dataset
24     dataset = pd.read_csv('Social_Network_Ads.csv')

```

Figure 14.6

dataset - DataFrame

Index	User ID	Gender	Age	EstimatedSalary	Purchased
0	15624510	Male	19	19000	0
1	15810944	Male	35	20000	0
2	15668575	Female	26	43000	0
3	15603246	Female	27	57000	0
4	15804002	Male	19	76000	0
5	15728773	Male	27	58000	0
6	15598044	Female	27	84000	0
7	15694829	Female	32	150000	1
8	15600575	Male	25	33000	0
9	15727311	Female	35	65000	0
10	15570769	Female	26	80000	0

Format    Resize     Background  Column min/  Row max/  Save and Close  Close

Figure 14.7

- Splitting the dataset into the Training set and Test set and Feature Scaling

By executing the above code, our dataset is imported to our program and well pre-processed. After feature scaling our test dataset will look like:

```

1. X = dataset.iloc[:, [2, 3]].values
2. y = dataset.iloc[:, -1].values
3.
4. # Splitting the dataset into the Training set and Test set
5. from sklearn.model_selection import train_test_split
6. X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_state = 0)
7.

```

```
8. # Feature Scaling
9. from sklearn.preprocessing import StandardScaler
10. sc = StandardScaler()
11. X_train = sc.fit_transform(X_train)
12. X_test = sc.transform(X_test)
```

Figure 14.8

X - NumPy array		
	0	1
0	19	190000
1	35	200000
2	26	430000
3	27	570000
4	19	760000
5	27	580000
6	27	840000
7	32	1500000
8	25	330000
9	38	420000

Figure 14.9

y - NumPy array	
	0
0	0
1	0
2	0
3	0
4	0
5	0
6	0
7	1
8	0
9	0

Figure 14.10

X\_test - NumPy array

	0	1
0	30	87000
1	38	50000
2	35	75000
3	30	79000
4	35	50000
5	27	20000
6	31	15000
7	36	144000
8	18	68000
9	42	43000

Format Resize  Background color

Save and Close Close

Figure 14.11

X\_train - NumPy array

	0	1
0	44	39000
1	32	120000
2	38	50000
3	32	135000
4	52	21000
5	53	104000
6	39	42000
7	38	61000
8	36	50000
9	32	53000

Format Resize  Background color

Save and Close Close

Figure 14.12

Figure 14.13 shows a table titled "y\_test - NumPy array". The table has a single column with 10 rows. The first 8 rows are colored red and contain the value "0" in black text. The 9th row is colored blue and contains the value "1" in black text. The 10th row is colored red and contains the value "0" in black text. The table includes standard Excel-style buttons for "Format", "Resize", and "Background color". At the bottom, there are "Save and Close" and "Close" buttons.

y_test - NumPy array	
0	0
1	0
2	0
3	0
4	0
5	0
6	0
7	1
8	0
9	0

Format      Resize       Background color

Save and Close      Close

Figure 14.13

Figure 14.14 shows a table titled "y\_train - NumPy array". The table has a single column with 10 rows. The first 2 rows are colored red and contain the value "0" in black text. The 3rd row is colored blue and contains the value "1" in black text. The 4th row is colored red and contains the value "1" in black text. The 5th row is colored blue and contains the value "1" in black text. The 6th row is colored blue and contains the value "1" in black text. The 7th row is colored red and contains the value "0" in black text. The 8th row is colored red and contains the value "0" in black text. The 9th row is colored red and contains the value "0" in black text. The 10th row is colored red and contains the value "0" in black text. The table includes standard Excel-style buttons for "Format", "Resize", and "Background color". At the bottom, there are "Save and Close" and "Close" buttons.

y_train - NumPy array	
0	0
1	0
2	1
3	0
4	1
5	1
6	1
7	0
8	0
9	0

Format      Resize       Background color

Save and Close      Close

Figure 14.14

- Training the K-NN model on the Training set

From the above output image, we can see that our data is successfully scaled.

- **Fitting K-NN classifier to the Training data:**

Now we will fit the K-NN classifier to the training data. To do this we will import the **KNeighborsClassifier** class of **Sklearn Neighbors** library. After importing the class, we will create the **Classifier** object of the class. The Parameter of this class will be

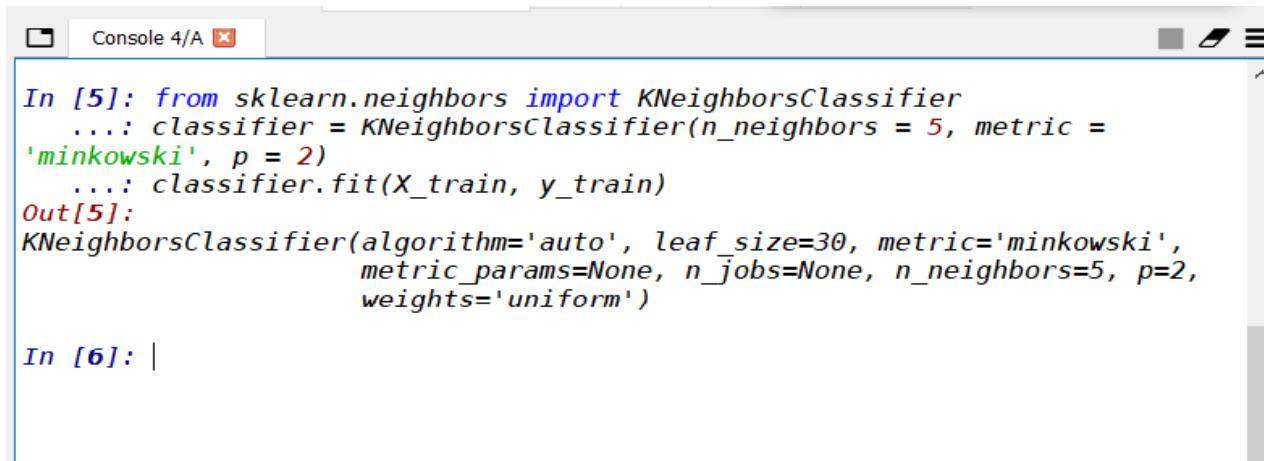
- **n\_neighbors:** To define the required neighbors of the algorithm. Usually, it takes 5.
- **metric='minkowski':** This is the default parameter and it decides the distance between the points.
- **p=2:** It is equivalent to the standard Euclidean metric.

And then we will fit the classifier to the training data. Below is the code for it:

```
1. # Training the K-NN model on the Training set
2.
3. from sklearn.neighbors import KNeighborsClassifier
4. classifier = KNeighborsClassifier(n_neighbors = 5, metric = 'minkowski', p = 2)
5. classifier.fit(X_train, y_train)
```

Figure 14.15

Output: By executing the above code, we will get the output as:



```
In [5]: from sklearn.neighbors import KNeighborsClassifier
.... classifier = KNeighborsClassifier(n_neighbors = 5, metric =
'minkowski', p = 2)
.... classifier.fit(X_train, y_train)
Out[5]:
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
metric_params=None, n_jobs=None, n_neighbors=5, p=2,
weights='uniform')

In [6]:
```

Figure 14.16

**Predicting the Test Result:** To predict the test set result, we will create a **y\_pred** vector as we did in Logistic Regression. Below is the code for it:

- **Predicting the Test set results**

```
01. # Predicting the Test set results
02. y_pred = classifier.predict(X_test)
```

Figure 14.17

## Output:

The output for the above code will be:

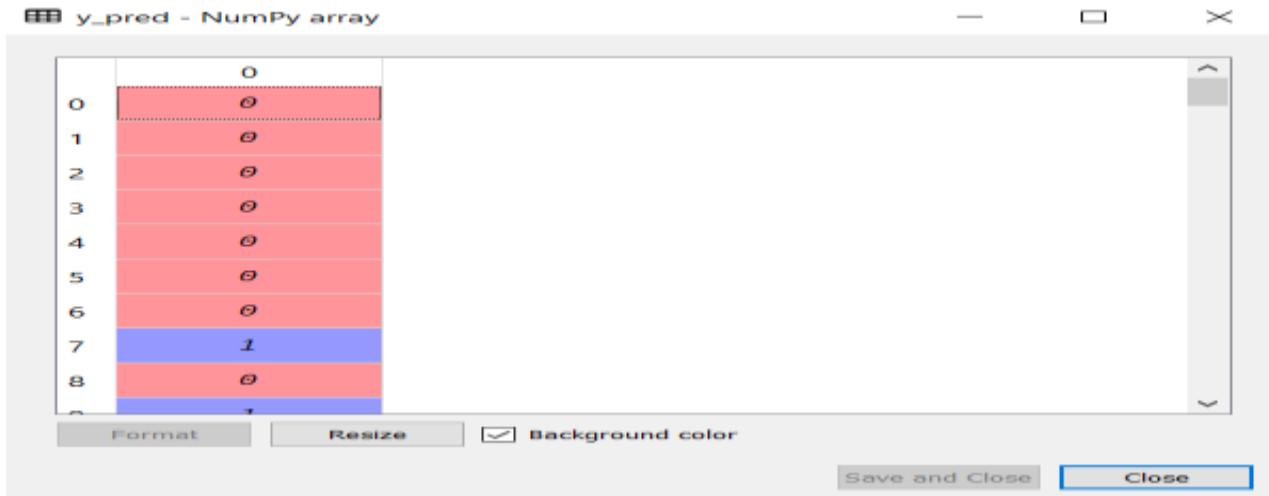


Figure 14.18

## Creating the Confusion Matrix:

Now we will create the Confusion Matrix for our K-NN model to see the accuracy of the classifier. Below is the code for it:

- **Making the Confusion Matrix**

```

1. # Predicting the Test set results
2. y_pred = classifier.predict(X_test)
3.
4. # Making the Confusion Matrix
5. from sklearn.metrics import confusion_matrix
6. cm = confusion_matrix(y_test, y_pred)
7. print(cm)

```

Figure 14.19

In above code, we have imported the `confusion_matrix` function and called it using the variable `cm`.

**Output:** By executing the above code, we will get the matrix as below:

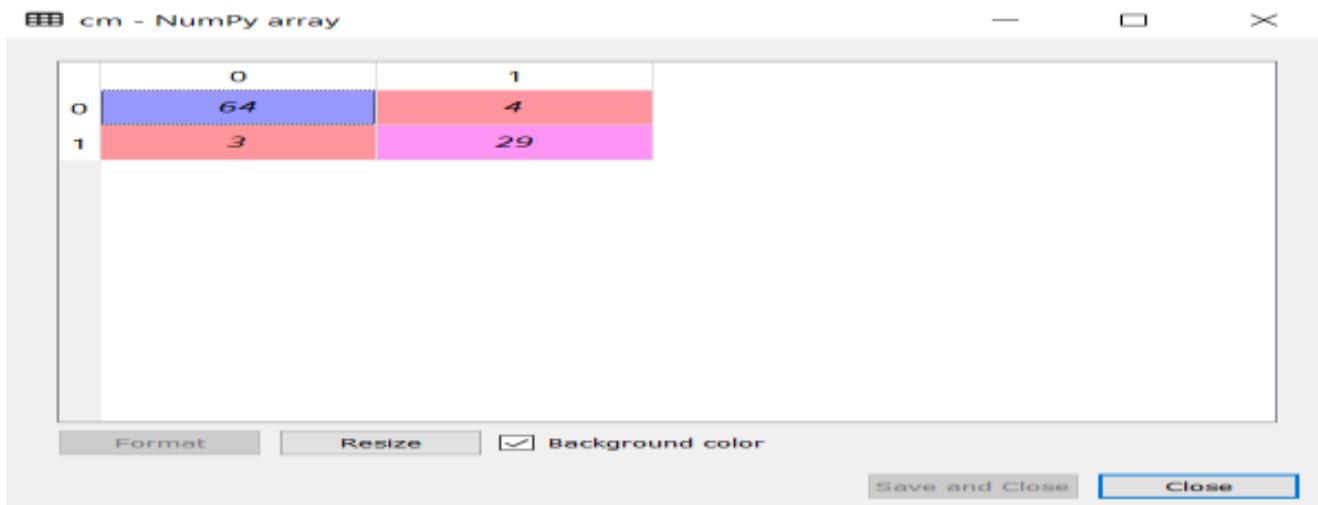


Figure 14.20

In the above image, we can see there are  $64+29= 93$  correct predictions and  $3+4= 7$  incorrect predictions, whereas, in Logistic Regression, there were 11 incorrect predictions. So we can say that the performance of the model is improved by using the K-NN algorithm.

- **Visualising the Training set results**

Now, we will visualize the training set result for K-NN model. The code will remain same as we did in Logistic

Regression, except the name of the graph. Below is the code for it:

```
1. # Visualising the Training set results
2.
3. from matplotlib.colors import ListedColormap
4. X_set, y_set = X_train, y_train
5. X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 1, stop = X_set[:, 0].max() + 1, step = 0.01),
6.                      np.arange(start = X_set[:, 1].min() - 1, stop = X_set[:, 1].max() + 1, step = 0.01))
7. plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape),
8.                 alpha = 0.75, cmap = ListedColormap(('red', 'green')))
9. plt.xlim(X1.min(), X1.max())
10. plt.ylim(X2.min(), X2.max())
11. for i, j in enumerate(np.unique(y_set)):
12.     plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
13.                 c = ListedColormap(('red', 'green'))(i), label = j)
14. plt.title('K-NN (Training set)')
15. plt.xlabel('Age')
16. plt.ylabel('Estimated Salary')
17. plt.legend()
18. plt.show()
```

Figure 14.21

## Output:

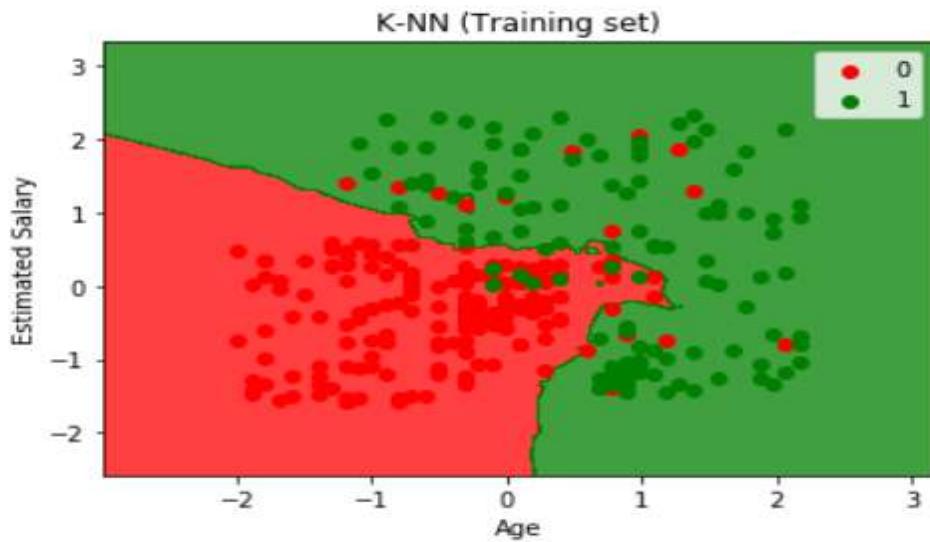


Figure 14.22

By executing the above code, we will get the below graph:

- Visualising the Test set results

The output graph is different from the graph which we have occurred in Logistic Regression. It can be understood in the below points:

- As we can see the graph is showing the red point and green points. The green points are for Purchased(1) and Red Points for not Purchased(0) variable.
- The graph is showing an irregular boundary instead of showing any straight line or any curve because it is a K-NN algorithm, i.e., finding the nearest neighbor.

- The graph has classified users in the correct categories as most of the users who didn't buy the SUV are in the red region and users who bought the SUV are in the green region.
  - The graph is showing good result but still, there are some green points in the red region and red points in the green region. But this is no big issue as by doing this model is prevented from overfitting issues.
  - Hence our model is well trained.
- Visualizing the Test set result:

After the training of the model, we will now test the result by putting a new dataset, i.e., Test dataset. Code remains the same except some minor changes: such as **x\_train** and **y\_train** will be replaced by **x\_test** and **y\_test**.

Below is the code for it:

```
1. # Visualising the Test set results
2.
3. from matplotlib.colors import ListedColormap
4. X_set, y_set = X_test, y_test
5. X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 1, stop = X_set[:, 0].max()
+ 1, step = 0.01),
6.                      np.arange(start = X_set[:, 1].min() - 1, stop = X_set[:, 1].max()
+ 1, step = 0.01))
```

```

7. plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X
1.shape),
8.                 alpha = 0.75, cmap = ListedColormap(('red', 'green')))
9. plt.xlim(X1.min(), X1.max())
10. plt.ylim(X2.min(), X2.max())
11. for i, j in enumerate(np.unique(y_set)):
12.     plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
13.                 c = ListedColormap(('red', 'green'))(i), label = j)
14. plt.title('K-NN (Test set)')
15. plt.xlabel('Age')
16. plt.ylabel('Estimated Salary')
17. plt.legend()
18. plt.show()

```

Figure 14.23

## Output:

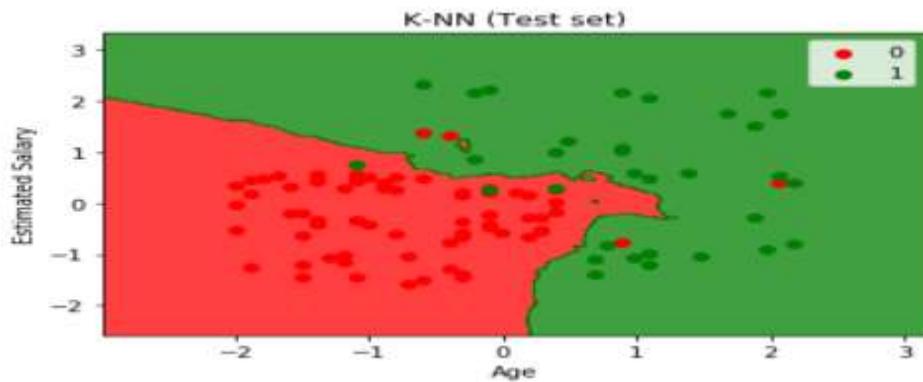


Figure 14.24

The above graph is showing the output for the test data set. As we can see in the graph, the predicted output is well good as most of the red points are in the red region and most of the green points are in the green region.

---

However, there are few green points in the red region and a few red points in the green region. So these are the incorrect observations that we have observed in the confusion matrix(7 Incorrect output).

---

## **CHAPTER 15**

---

### **Support Vector Machine (SVM)**

#### **15.1 Overview**

Support Vector Machine or SVM is one of the most popular Supervised Learning algorithms, which is used for Classification as well as Regression problems. However, primarily, it is used for Classification problems in Machine Learning.

The goal of the SVM algorithm is to create the best line or decision boundary that can segregate n-dimensional space into classes so that we can easily put the new data point in the correct category in the future. This best decision boundary is called a hyperplane.

SVM chooses the extreme points/vectors that help in creating the hyperplane. These extreme cases are called as support vectors, and hence algorithm is termed as Support Vector Machine. Consider the below diagram in which there are two different categories that are classified using a decision boundary or hyperplane:

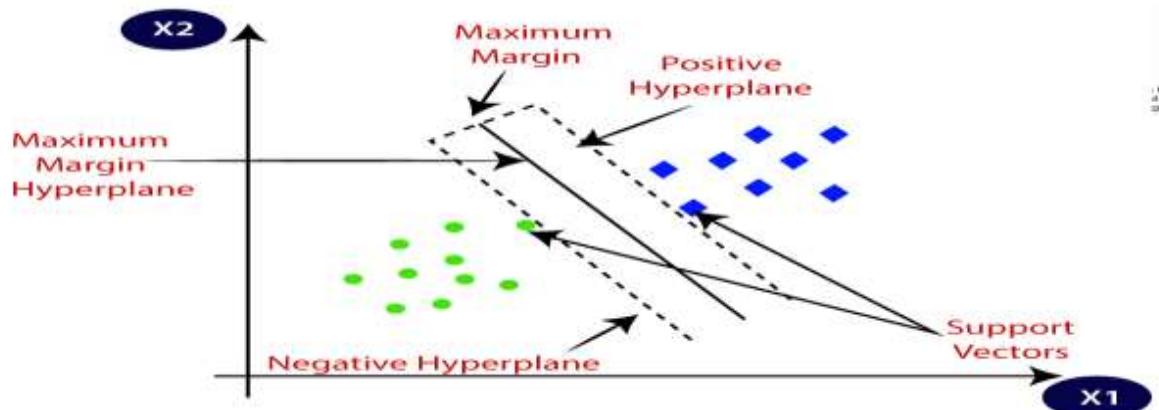


Figure 15.1

- **Types of SVM:**

1. SVM can be of two types:

**Linear SVM:** Linear SVM is used for linearly separable data, which means if a dataset can be classified into two classes by using a single straight line, then such data is termed as linearly separable data, and classifier is used called as Linear SVM classifier.

**Non-linear SVM:** Non-Linear SVM is used for non-linearly separated data, which means if a dataset cannot be classified by using a straight line, then such data is termed as non-linear data and classifier used is called as Non-linear SVM classifier.

## 2. Linear SVM:

The working of the SVM algorithm can be understood by using an example. Suppose we have a dataset that has two tags (green and blue), and the dataset has two features  $x_1$  and  $x_2$ . We want a classifier that can classify the pair  $(x_1, x_2)$  of coordinates in either green or blue. Consider the below image:

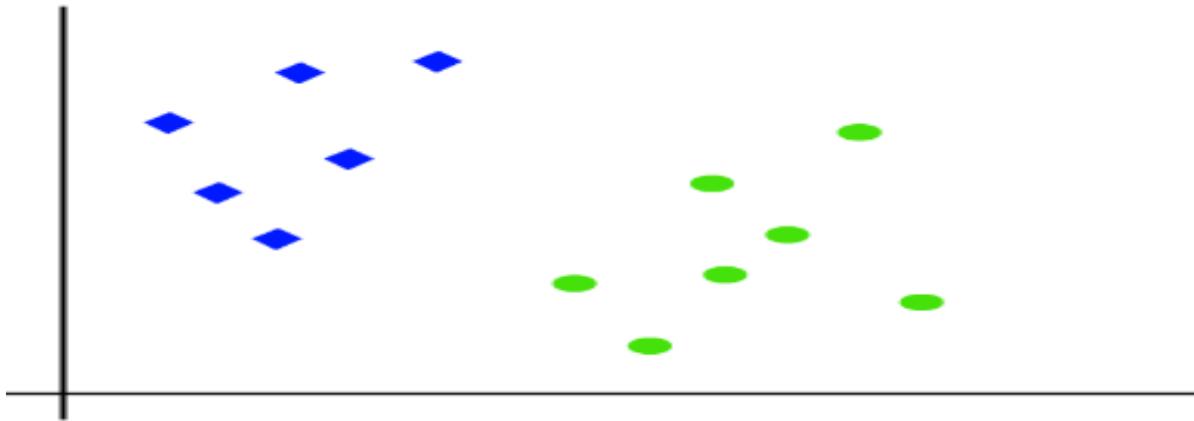


Figure 15.2

So as it is 2-d space so by just using a straight line, we can easily separate these two classes. But there can be multiple lines that can separate these classes. Consider the below image:

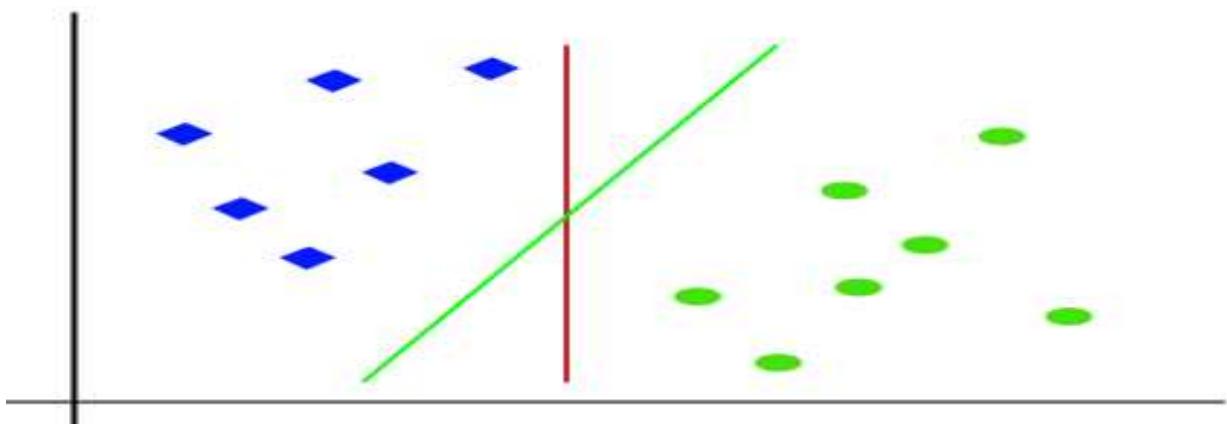


Figure 15.3

Hence, the SVM algorithm helps to find the best line or decision boundary; this best boundary or region is called as a hyperplane.

SVM algorithm finds the closest point of the lines from both the classes. These points are called support vectors. The distance between the vectors and the hyperplane is called as margin. And the goal of SVM is to maximize this margin. The hyperplane with maximum margin is called the optimal hyperplane.

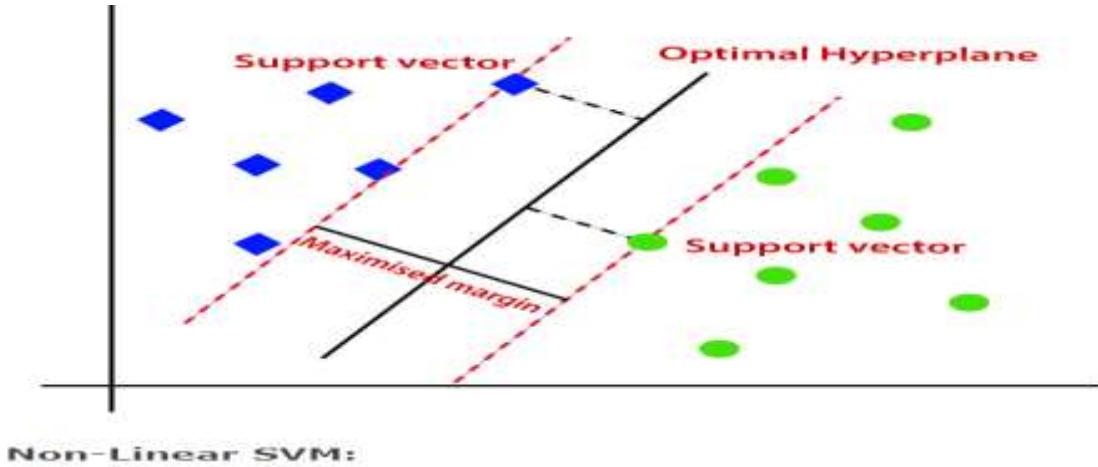


Figure 15.4

## 15.2 Python Implementation of Support Vector Machine

- **Data Pre-processing step**

Till the Data pre-processing step, the code will remain the same. Below is the code:

```

1. # Support Vector Machine (SVM)
2.
3. # Importing the libraries
4. import numpy as np
5. import matplotlib.pyplot as plt
6. import pandas as pd
7.
8. # Importing the dataset
9. dataset = pd.read_csv('Social_Network_Ads.csv')

```

Figure 15.5

After executing the above code, we will pre-process the data. The code will give the dataset as:

Index	User ID	Gender	Age	EstimatedSalary	Purchased
0	15624510	Male	19	190000	0
1	15810944	Male	35	200000	0
2	15668575	Female	26	430000	0
3	15603246	Female	27	570000	0
4	15804002	Male	19	760000	0
5	15728773	Male	27	580000	0
6	15598044	Female	27	840000	0
7	15694829	Female	32	1500000	1
8	15600575	Male	25	330000	0
9	15727311	Female	35	650000	0
10	15570769	Female	26	800000	0

Figure 15.6

```

1. X = dataset.iloc[:, [2, 3]].values
2. y = dataset.iloc[:, -1].values

```

Figure 15.7

The scaled output for the test set will be:

X - NumPy array		
	0	1
0	19	19000
1	35	20000
2	26	43000
3	27	57000
4	19	76000
5	27	58000
6	27	84000
7	32	150000
8	25	33000
9	35	65000

Format    Resize     Background color    Save and Close    Close

Figure 15.8



Figure 15.9

- **Fitting the SVM classifier to the training set:**

Now the training set will be fitted to the SVM classifier. To create the SVM classifier, we will import **SVC** class from **Sklearn.svm** library. Below is the code for it:

```

1. # Splitting the dataset into the Training set and Test set
2. from sklearn.model_selection import train_test_split
3. X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25,
4. random_state = 0)
5.
6. # Feature Scaling
7. from sklearn.preprocessing import StandardScaler
8. sc = StandardScaler()
9. X_train = sc.fit_transform(X_train)
10. X_test = sc.transform(X_test)

```

Figure 15.10

## Output:

Figure 15.11 shows a NumPy array named X\_test. The array has two columns: 0 and 1. The data is as follows:

	0	1
0	30	870000
1	38	500000
2	35	750000
3	30	790000
4	35	500000
5	27	200000
6	31	150000
7	36	1440000
8	18	680000
9	47	430000

Buttons at the bottom: Format, Resize,  Background color, Save and Close, Close.

Figure 15.11

Figure 15.12 shows a NumPy array named X\_train. The array has two columns: 0 and 1. The data is as follows:

	0	1
0	44	390000
1	32	1200000
2	38	500000
3	32	1350000
4	52	210000
5	53	1040000
6	39	420000
7	38	610000
8	36	500000
9	36	630000

Buttons at the bottom: Format, Resize,  Background color, Save and Close, Close.

Figure 15.12

Figure 15.13 shows a NumPy array named `y_test` with 10 rows and 1 column. The values are: 0, 0, 0, 0, 0, 0, 1, 0, 0, 0. The cell for value 1 is highlighted with a blue background.

y_test - NumPy array	
0	0
1	0
2	0
3	0
4	0
5	0
6	0
7	1
8	0
9	0

Format    Resize     Background color    Save and Close    Close

Figure 15.13

Figure 15.14 shows a NumPy array named `y_train` with 10 rows and 1 column. The values are: 0, 1, 0, 1, 1, 1, 0, 0, 0, 0. The cells for values 1 and 0 at indices 1 and 6 are highlighted with blue and red backgrounds respectively.

y_train - NumPy array	
0	0
1	1
2	0
3	1
4	1
5	1
6	0
7	0
8	0
9	0

Format    Resize     Background color    Save and Close    Close

Figure 15.14

```

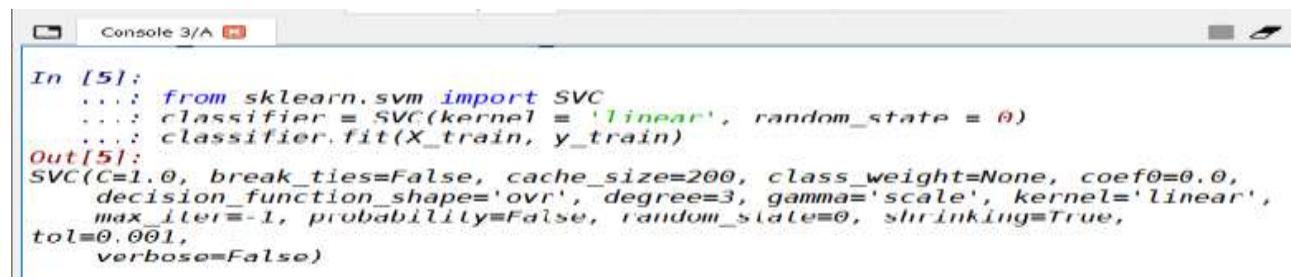
1. # Training the SVM model on the Training set
2. from sklearn.svm import SVC
3. classifier = SVC(kernel = 'linear', random_state = 0)
4. classifier.fit(X_train, y_train)

```

Figure 15.15

In the above code, we have used `kernel='linear'`, as here we are creating SVM for linearly separable data. However, we can change it for non-linear data. And then we fitted the classifier to the training dataset(`x_train`, `y_train`)

## Output:



```
In [5]:  
...: from sklearn.svm import SVC  
...: classifier = SVC(kernel = 'linear', random_state = 0)  
...: classifier.fit(X_train, y_train)  
Out[5]:  
SVC(C=1.0, break_ties=False, cache_size=200, class_weight=None, coef0=0.0,  
decision_function_shape='ovr', degree=3, gamma='scale', kernel='linear',  
max_iter=-1, probability=False, random_state=0, shrinking=True,  
tol=0.001,  
verbose=False)
```

Figure 15.16

- **Predicting the test set result:**

Now, we will predict the output for test set. For this, we will create a new vector `y_pred`. Below is the code for it:

```
1. # Predicting the Test set results  
2. y_pred = classifier.predict(X_test)
```

Figure 15.17

After getting the `y_pred` vector, we can compare the result of `y_pred` and `y_test` to check the difference between the actual value and predicted value.

## Output:

Below is the output for the prediction of the test set:



Figure 15.18

- **Creating the confusion matrix:**

Now we will see the performance of the SVM classifier that how many incorrect predictions are there as compared to the Logistic regression classifier. To create the confusion matrix, we need to import the `confusion_matrix` function of the `sklearn` library. After importing the function, we will call it using a new variable `cm`. The function takes two parameters, mainly `y_true`( the actual values) and `y_pred` (the targeted value return by the classifier). Below is the code for it:

```
1. # Making the Confusion Matrix
2. from sklearn.metrics import confusion_matrix
3. cm = confusion_matrix(y_test, y_pred)
4. print(cm)
```

Figure 15.19

## Output:

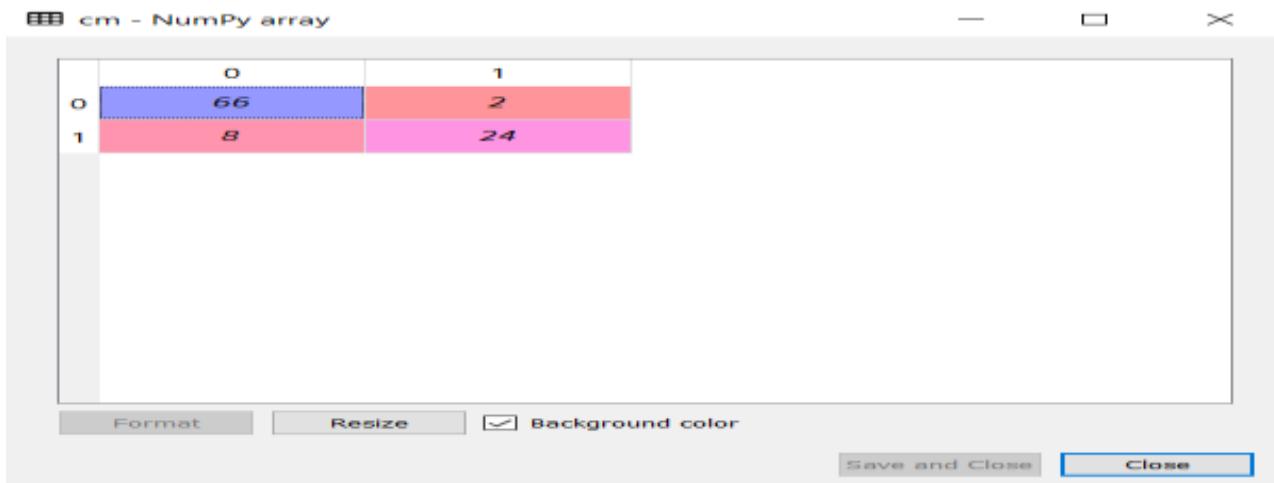


Figure 15.20

As we can see in the above output image, there are  $66+24= 90$  correct predictions and  $8+2= 10$  correct predictions. Therefore we can say that our SVM model improved as compared to the Logistic regression model.

- **Visualizing the training set result:**

Now we will visualize the training set result, below is the code for it:

```
1. # Visualising the Training set results
2. from matplotlib.colors import ListedColormap
3. X_set, y_set = X_train, y_train
4. X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 1, stop = X_set[:, 0].max() + 1, step = 0.01),
```

```

5.         np.arange(start = X_set[:, 1].min() - 1, stop = X_set[:, 1].max()
6.                     + 1, step = 0.01))
7.         plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X
8.                     .shape),
9.                     alpha = 0.75, cmap = ListedColormap(('red', 'green')))
10.        plt.xlim(X1.min(), X1.max())
11.        plt.ylim(X2.min(), X2.max())
12.        for i, j in enumerate(np.unique(y_set)):
13.            plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
14.                        c = ListedColormap(('red', 'green'))(i), label = j)
15.        plt.title('SVM (Training set)')
16.        plt.xlabel('Age')
17.        plt.ylabel('Estimated Salary')
18.        plt.legend()
19.        plt.show()

```

Figure 15.21

## Output:

By executing the above code, we will get the output as:

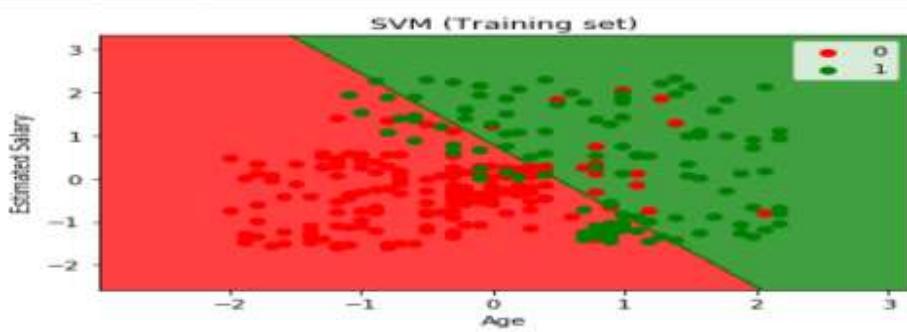


Figure 15.22

As we can see, the above output is appearing similar to the Logistic regression output. In the output, we got the straight line as hyperplane because we have **used a linear kernel in the classifier**. And we have also discussed above that for the 2d space, the hyperplane in SVM is a straight line.

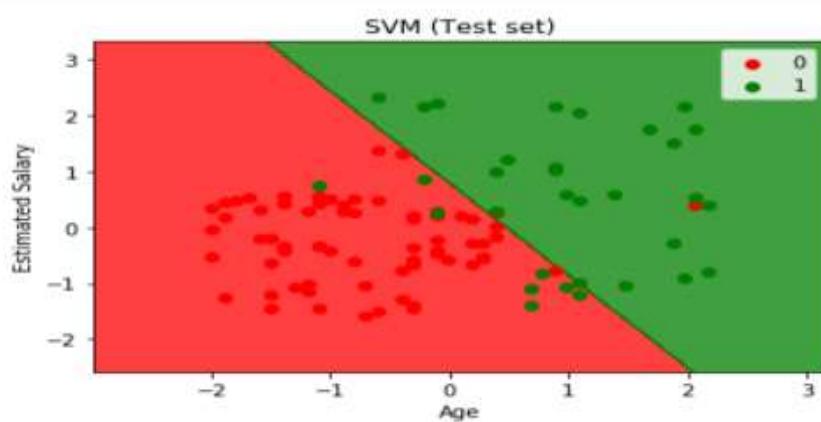
- **Visualizing the test set result:**

```
• # Visualising the Test set results
•
• from matplotlib.colors import ListedColormap
• X_set, y_set = X_test, y_test
• X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 1, stop = X_set[:, 0].max() + 1, step = 0.01),
•                      np.arange(start = X_set[:, 1].min() - 1, stop = X_set[:, 1].max() + 1, step = 0.01))
• plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape),
•              alpha = 0.75, cmap = ListedColormap(('red', 'green')))
• plt.xlim(X1.min(), X1.max())
• plt.ylim(X2.min(), X2.max())
• for i, j in enumerate(np.unique(y_set)):
•     plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
•                 c = ListedColormap(('red', 'green'))(i), label = j)
• plt.title('SVM (Test set)')
• plt.xlabel('Age')
• plt.ylabel('Estimated Salary')
• plt.legend()
• plt.show()
```

Figure 15.23

## Output:

By executing the above code, we will get the output as:



## CHAPTER 16

---

### Kernel SVM

#### 16.1 Overview

In our previous Machine Learning blog we have discussed about SVM (Support Vector Machine) in Machine Learning. Now we are going to provide you a detailed description of SVM Kernel and Different Kernel Functions and its examples such as linear, nonlinear, polynomial, Gaussian kernel, Radial basis function (RBF), sigmoid etc.

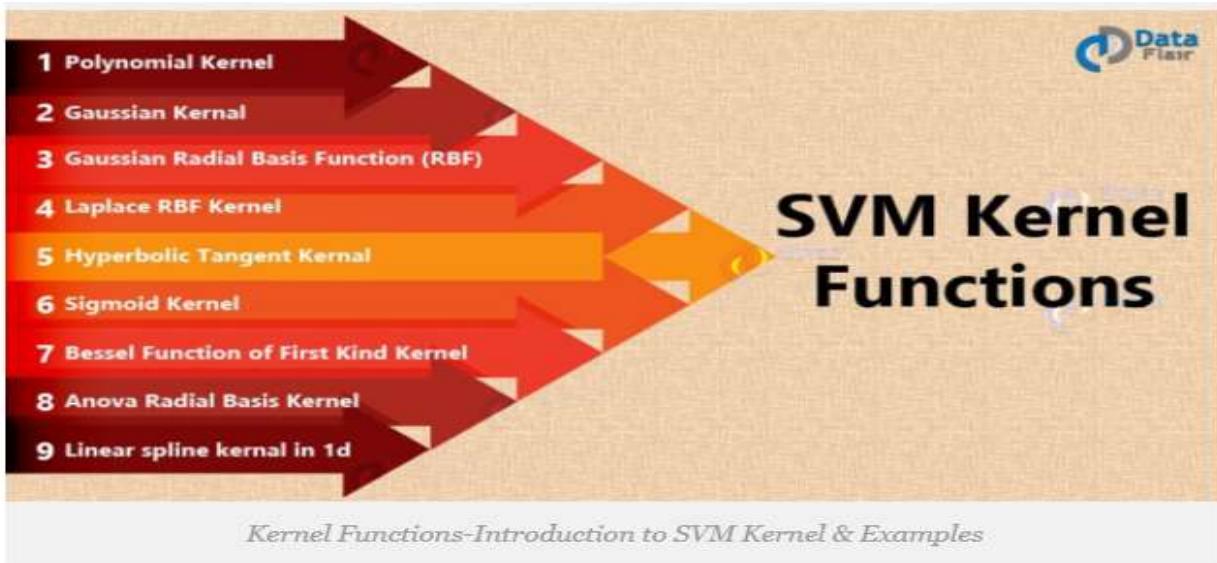


Figure 16.1

## SVM Kernel Functions

SVM algorithms use a set of mathematical functions that are defined as the kernel. The function of kernel is to take data as input and transform it into the required form. Different SVM algorithms use different types of kernel functions. These functions can be different types. For example linear, nonlinear, polynomial, radial basis function (RBF), and sigmoid.

### 15.1 Implementing Kernel SVM with Python

## Parameters

**C** : float, optional (default=1.0)

Penalty parameter C of the error term.

**kernel** : string, optional (default='rbf')

Specifies the kernel type to be used in the algorithm. It must be one of 'linear', 'poly', 'rbf', 'sigmoid', 'precomputed' or a callable. If none is given, 'rbf' will be used. If a callable is given it is used to pre-compute the kernel matrix from data matrices; that matrix should be an array of shape (n\_samples, n\_samples).

**degree** : int, optional (default=3)

Degree of the polynomial kernel function ('poly'). Ignored by all other kernels.

Figure 16.2

```
1. # Kernel SVM
2.
3. # Importing the libraries
4. import numpy as np
5. import matplotlib.pyplot as plt
6. import pandas as pd
7.
8. # Importing the dataset
9. dataset = pd.read_csv('Social_Network_Ads.csv')
10. X = dataset.iloc[:, [2, 3]].values
11. y = dataset.iloc[:, -1].values
12.
13. # Splitting the dataset into the Training set and Test set
14. from sklearn.model_selection import train_test_split
15. X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_state = 0)
16.
17. # Feature Scaling
18. from sklearn.preprocessing import StandardScaler
19. sc = StandardScaler()
20. X_train = sc.fit_transform(X_train)
21. X_test = sc.transform(X_test)
22.
23. # Training the Kernel SVM model on the Training set
24. from sklearn.svm import SVC
25. classifier = SVC(kernel = 'linear', random_state = 0)
```

```

26. classifier.fit(X_train, y_train)
27.
28. # Predicting the Test set results
29. y_pred = classifier.predict(X_test)
30.
31. # Making the Confusion Matrix
32. from sklearn.metrics import confusion_matrix
33. cm = confusion_matrix(y_test, y_pred)
34. print(cm)
35.
36. # Visualising the Training set results
37. from matplotlib.colors import ListedColormap
38. X_set, y_set = X_train, y_train
39. X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 1, stop = X_set[:, 0].max()
+ 1, step = 0.01),
40.                      np.arange(start = X_set[:, 1].min() - 1, stop = X_set[:, 1].max()
+ 1, step = 0.01))
41. plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X
1.shape),
42.                 alpha = 0.75, cmap = ListedColormap(('red', 'green')))
43. plt.xlim(X1.min(), X1.max())
44. plt.ylim(X2.min(), X2.max())
45. for i, j in enumerate(np.unique(y_set)):
46.     plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
47.                 c = ListedColormap(('red', 'green'))(i), label = j)
48. plt.title('Kernel SVM (Training set)')
49. plt.xlabel('Age')
50. plt.ylabel('Estimated Salary')
51. plt.legend()
52. plt.show()
53.
54. # Visualising the Test set results
55. from matplotlib.colors import ListedColormap
56. X_set, y_set = X_test, y_test
57. X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 1, stop = X_set[:, 0].max()
+ 1, step = 0.01),
58.                      np.arange(start = X_set[:, 1].min() - 1, stop = X_set[:, 1].max()
+ 1, step = 0.01))
59. plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X
1.shape),
60.                 alpha = 0.75, cmap = ListedColormap(('red', 'green')))
61. plt.xlim(X1.min(), X1.max())
62. plt.ylim(X2.min(), X2.max())
63. for i, j in enumerate(np.unique(y_set)):
64.     plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
65.                 c = ListedColormap(('red', 'green'))(i), label = j)
66. plt.title('Kernel SVM (Test set)')
67. plt.xlabel('Age')
68. plt.ylabel('Estimated Salary')
69. plt.legend()
70. plt.show()

```

Figure 16.3

## Types Kernel :

### 1. Polynomial Kernel

It is popular in image processing. Equation is:

$$k(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i \cdot \mathbf{x}_j + 1)^d$$

*Polynomial kernel equation*

Figure 16.4

where  $d$  is the degree of the polynomial.

```
1. # Training the Kernel SVM model on the Training set
2. from sklearn.svm import SVC
3. classifier = SVC(kernel = 'poly', random_state = 0)
4. classifier.fit(X_train, y_train)
```

Figure 16.5

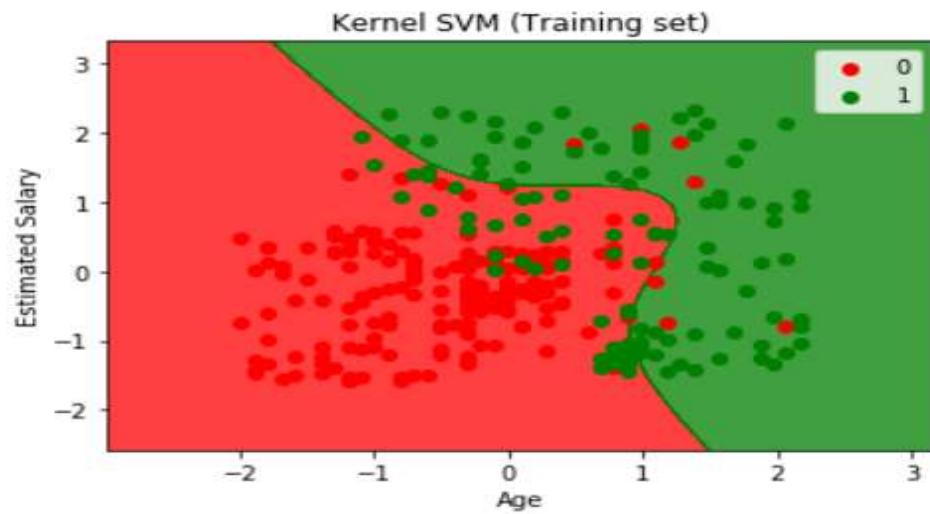


Figure 16.6

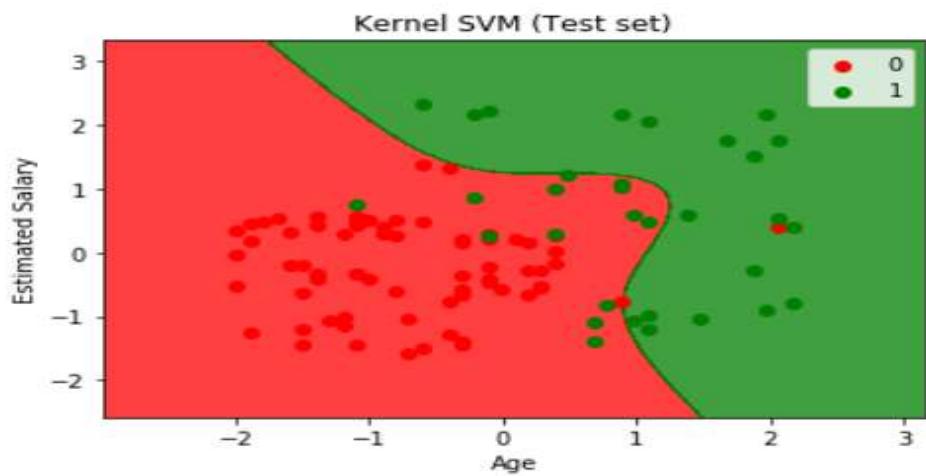


Figure 16.7

## 2. Gaussian kernel

It is a general-purpose kernel; used when there is no prior knowledge about the data. Equation is:

$$k(x, y) = \exp\left(-\frac{\|x - y\|^2}{2\sigma^2}\right)$$

*Gaussian kernel equation*

Figure 16.8

## Radial basis function (RBF)

```

1. # Training the Kernel SVM model on the Training set
2. from sklearn.svm import SVC
3. classifier = SVC(kernel = 'rbf', random_state = 0)
4. classifier.fit(X_train, y_train)

```

Figure 16.9

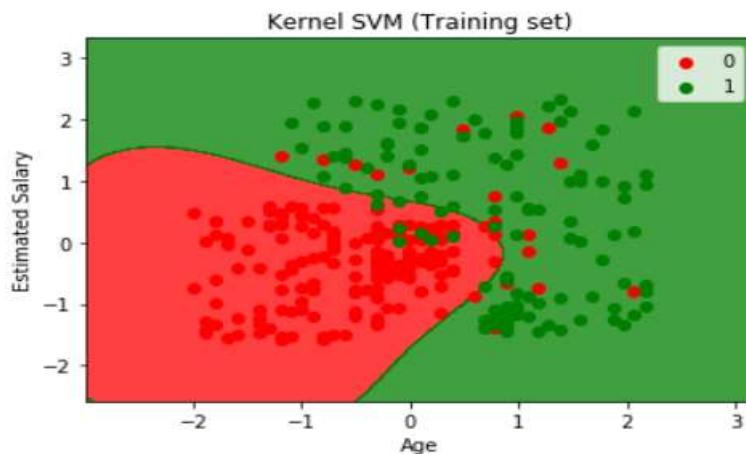


Figure 16.10

## 3.Sigmoid Kernel

We can use it as the proxy for neural networks. Equation is

$$k(x, y) = \tanh(\alpha x^T y + c)$$

### Sigmoid kernel equation

Figure 16.11

```
1. # Training the Kernel SVM model on the Training set
2. from sklearn.svm import SVC
3. classifier = SVC(kernel = 'Sigmoid', random_state = 0)
4. classifier.fit(X_train, y_train)
```

Figure 16.12

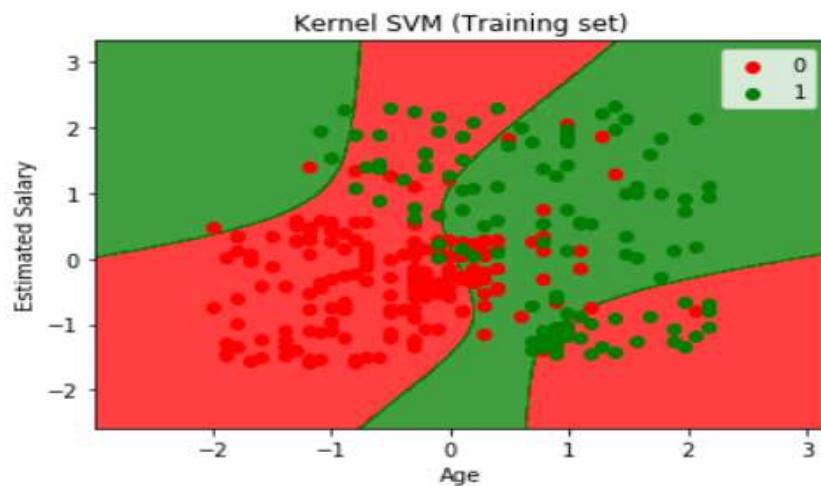


Figure 16.13

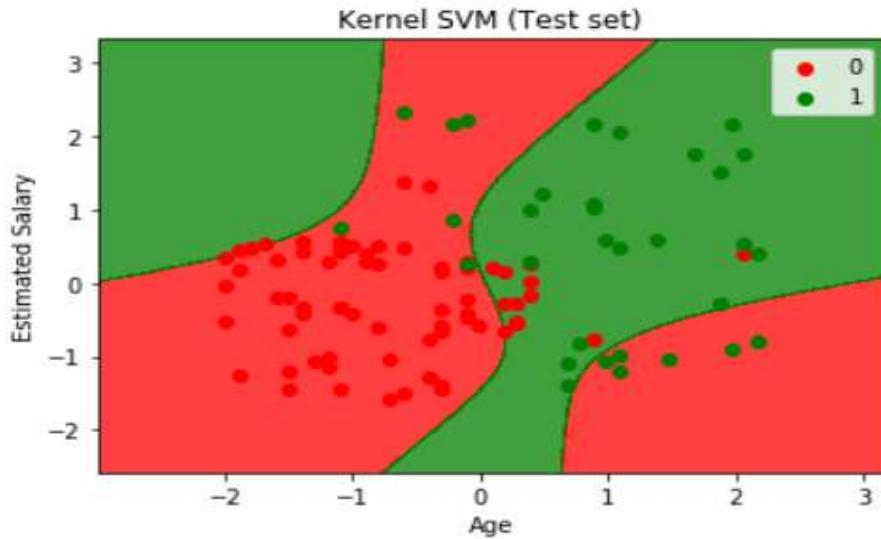


Figure 16.14

## 4. Linear Kernel

It is useful when dealing with large sparse data vectors. It is often used in text categorization. The splines kernel also performs well in regression problems. Equation is:

$$k(x, y) = 1 + xy + xy \min(x, y) - \frac{x + y}{2} \min(x, y)^2 + \frac{1}{3} \min(x, y)^3$$

*Linear splines kernel equation in one-dimension*

Figure 16.15

```
1. # Training the Kernel SVM model on the Training set
2. from sklearn.svm import SVC
3. classifier = SVC(kernel = 'Linear', random_state = 0)
4. classifier.fit(X_train, y_train)
```

Figure 16.16

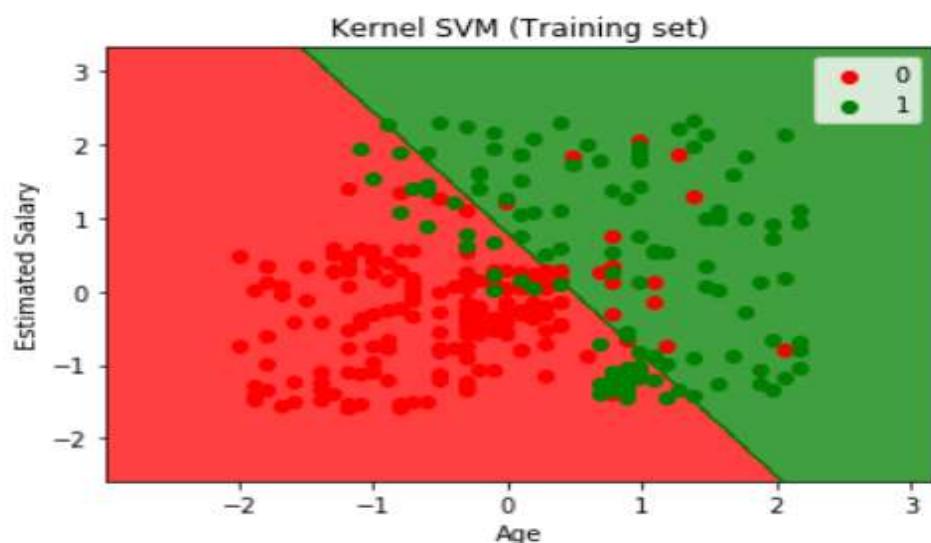


Figure 16.17

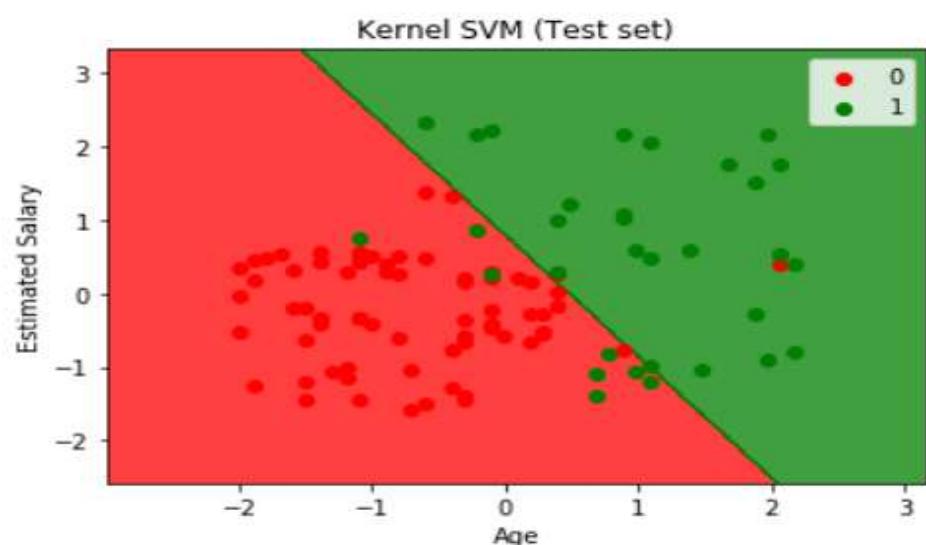


Figure 16.18

# CHAPTER 17

---

## Naive Bayes

### 17.1 Overview

Naive Bayes is a statistical classification technique based on Bayes Theorem. It is one of the simplest supervised learning algorithms. Naive Bayes classifier is the fast, accurate and reliable algorithm. Naive Bayes classifiers have high accuracy and speed on large datasets.

- Naïve Bayes algorithm is a supervised learning algorithm, which is based on **Bayes theorem** and used for solving classification problems.
- It is mainly used in *text classification* that includes a high-dimensional training dataset.
- Naïve Bayes Classifier is one of the simple and most effective Classification algorithms which helps in building the fast machine learning models that can make quick predictions.

- It is a probabilistic classifier, which means it predicts on the basis of the probability of an object.
- Some popular examples of Naïve Bayes Algorithm are **spam filtration**, **Sentimental analysis**, and **classifying articles**.

## 17.2 Bayes' Theorem:

- Bayes' theorem is also known as **Bayes' Rule** or **Bayes' law**, which is used to determine the probability of a hypothesis with prior knowledge. It depends on the conditional probability.
- The formula for Bayes' theorem is given as:

$$P(A|B) = \frac{P(B|A) P(A)}{P(B)}$$

Figure 17.1

**P(A | B) is Posterior probability:** Probability of hypothesis A on the observed event B.

**P(B | A) is Likelihood probability:** Probability of the evidence given that the probability of a hypothesis is true.

**P(A) is Prior Probability:** Probability of hypothesis before observing the evidence.

**P(B) is Marginal Probability:** Probability of Evidence.

**Advantages of Naïve Bayes Classifier:**

- Naïve Bayes is one of the fast and easy ML algorithms to predict a class of datasets.
- It can be used for Binary as well as Multi-class Classifications.
- It performs well in Multi-class predictions as compared to the other Algorithms.
- It is the most popular choice for **text classification problems**.

**Disadvantages of Naïve Bayes Classifier:**

- Naive Bayes assumes that all features are independent or unrelated, so it cannot learn the relationship between features.

## 17.3 Python Implementation of the Naïve Bayes algorithm:

Now we will implement a Naive Bayes Algorithm using Python. So for this, we will use the "user\_data" dataset, which we have used in our other classification model. Therefore we can easily compare the Naive Bayes model with the other models.

### Steps to implement:

- Data Pre-processing step
- Fitting Naive Bayes to the Training set
- Predicting the test result
- Test accuracy of the result(Creation of Confusion matrix)
- Visualizing the test set result.

#### 1. Data Pre-processing step:

In this step, we will pre-process/prepare the data so that we can use it efficiently in our code. It is similar as we did in **data-pre-processing**. The code for this is given below:

```
1. # Naive Bayes
2.
3. # Importing the libraries
4. import numpy as np
5. import matplotlib.pyplot as plt
6. import pandas as pd
7.
8. # Importing the dataset
9. dataset = pd.read_csv('Social_Network_Ads.csv')
```

```
10. X = dataset.iloc[:, [2, 3]].values  
11. y = dataset.iloc[:, -1].values
```

Figure 17.2

dataset - DataFrame

Index	User ID	Gender	Age	matedSal	Purchased
0	15624510	Male	19	19000	0
1	15810944	Male	35	20000	0
2	15668575	Female	26	43000	0
3	15603246	Female	27	57000	0
4	15804082	Male	19	76000	0
5	15728773	Male	27	58000	0
6	15598044	Female	27	84000	0
7	15694829	Female	32	150000	1
8	15600575	Male	25	33000	0
9	15727311	Female	35	65000	0
10	15570769	Female	26	80000	0

Format Resize  Background color  Column min/ Save and Close Close

Figure 17.3

X - NumPy array

	0	1
0	19	19000
1	35	20000
2	26	43000
3	27	57000
4	19	76000
5	27	58000
6	27	84000
7	32	150000
8	25	33000
9	35	65000

Format Resize  Background color Save and Close Close

Figure 17.4

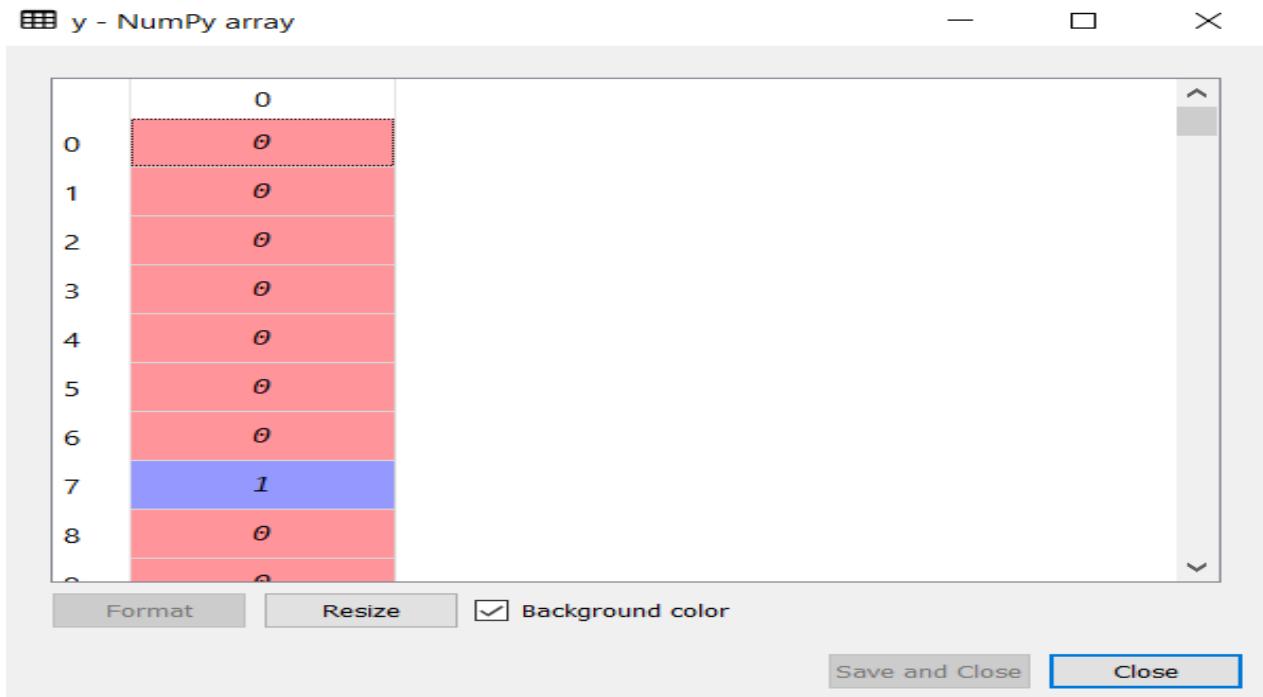


Figure 17.5

```
1. # Splitting the dataset into the Training set and Test set
2. from sklearn.model_selection import train_test_split
3. X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25,
4.
5. random_state = 0)
6.
7. # Feature Scaling
8. from sklearn.preprocessing import StandardScaler
9. sc = StandardScaler()
10. X_train = sc.fit_transform(X_train)
11. X_test = sc.transform(X_test)
```

Figure 17.6

X\_test - NumPy array

	0	1
0	30	87000
1	38	50000
2	35	75000
3	30	79000
4	35	50000
5	27	20000
6	31	15000
7	36	144000
8	18	68000
9	47	42000

Format Resize  Background color

Save and Close Close

Figure 17.7

X\_train - NumPy array

	0	1
0	44	39000
1	32	120000
2	38	50000
3	32	135000
4	52	21000
5	53	104000
6	39	42000
7	38	61000
8	36	50000
9	26	62000

Format Resize  Background color

Save and Close Close

Figure 17.8



Figure 17.9

## 2. Fitting Naive Bayes to the Training Set:

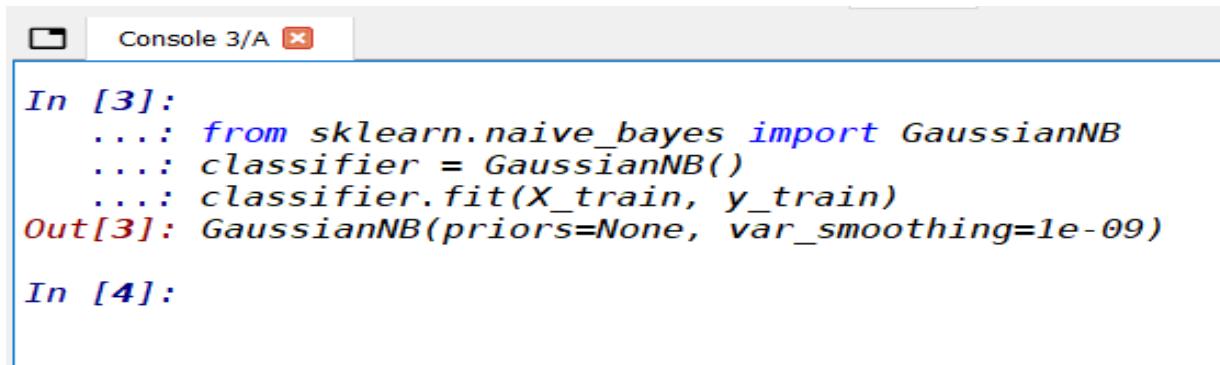
After the pre-processing step, now we will fit the Naive Bayes model to the Training set. Below is the code for it:

```
1. # Training the Naive Bayes model on the Training set
2. from sklearn.naive_bayes import GaussianNB
3. classifier = GaussianNB()
4. classifier.fit(X_train, y_train)
```

Figure 17.10

In the above code, we have used the **GaussianNB classifier** to fit it to the training dataset. We can also use other classifiers as per our requirement.

## Output:



```
Console 3/A
```

```
In [3]:  
....: from sklearn.naive_bayes import GaussianNB  
....: classifier = GaussianNB()  
....: classifier.fit(X_train, y_train)  
Out[3]: GaussianNB(priors=None, var_smoothing=1e-09)  
  
In [4]:
```

Figure 17.11

### 3. Prediction of the test set result:

Now we will predict the test set result. For this, we will create a new predictor variable `y_pred`, and will use the `predict` function to make the predictions.

```
1. # Predicting the Test set results  
2. y_pred = classifier.predict(X_test)
```

Figure 17.12

## Output:



Figure 17.13



Figure 17.14

The above output shows the result for prediction vector `y_pred` and real vector `y_test`. We can see that some predictions are different from the real values, which are the incorrect predictions.

#### 4. Creating Confusion Matrix:

Now we will check the accuracy of the Naive Bayes classifier using the Confusion matrix. Below is the code for it:

```
1. # Making the Confusion Matrix
2. from sklearn.metrics import confusion_matrix
3. cm = confusion_matrix(y_test, y_pred)
```

Figure 17.15

#### Output:

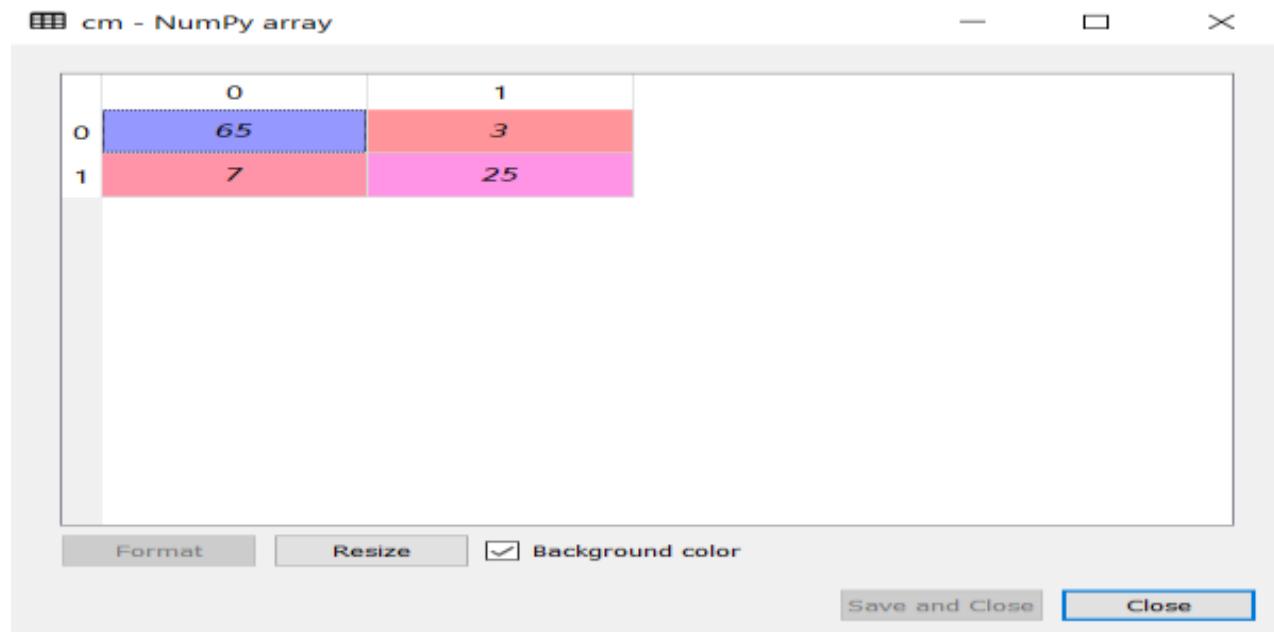


Figure 17.16

As we can see in the above confusion matrix output, there are  $7+3=10$  incorrect predictions, and  $65+25=90$  correct predictions.

## 5) Visualizing the training set result:

Next we will visualize the training set result using Naïve Bayes Classifier. Below is the code for it:

```
1. # Visualising the Training set results
2. # Visualising the Training set results
3. from matplotlib.colors import ListedColormap
4. X_set, y_set = X_train, y_train
5. X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 1, stop = X_set[:, 0].max() + 1, step = 0.01),
6.                      np.arange(start = X_set[:, 1].min() - 1, stop = X_set[:, 1].max() + 1, step = 0.01))
7. plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape),
8.               alpha = 0.75, cmap = ListedColormap(('red', 'green')))
9. plt.xlim(X1.min(), X1.max())
10. plt.ylim(X2.min(), X2.max())
11. for i, j in enumerate(np.unique(y_set)):
12.     plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
13.                 c = ListedColormap(('red', 'green'))(i), label = j)
14. plt.title('Naive Bayes (Training set)')
15. plt.xlabel('Age')
16. plt.ylabel('Estimated Salary')
17. plt.legend()
18. plt.show()
```

Figure 17.17

## Output:

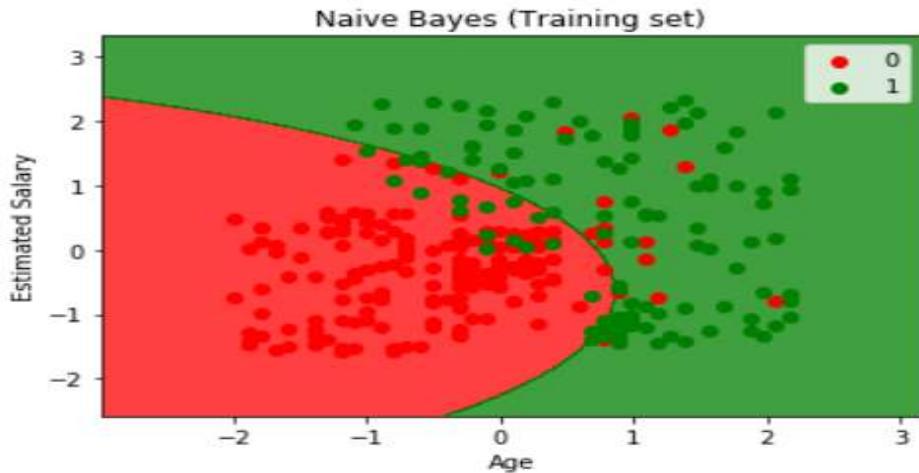


Figure 17.18

In the above output we can see that the Naïve Bayes classifier has segregated the data points with the fine boundary. It is Gaussian curve as we have used **GaussianNB** classifier in our code.

## 6. Visualizing the Test set result:

```

1. # Visualising the Test set results
2. from matplotlib.colors import ListedColormap
3. X_set, y_set = X_test, y_test
4. X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 1, stop = X_set[:, 0].max() + 1, step = 0.01),
5.                      np.arange(start = X_set[:, 1].min() - 1, stop = X_set[:, 1].max() + 1, step = 0.01))
6. plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape),
7.                 alpha = 0.75, cmap = ListedColormap(('red', 'green')))
8. plt.xlim(X1.min(), X1.max())
9. plt.ylim(X2.min(), X2.max())
10. for i, j in enumerate(np.unique(y_set)):
11.     plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
12.                 c = ListedColormap(('red', 'green'))(i), label = j)
13. plt.title('Naive Bayes (Test set)')
14. plt.xlabel('Age')
15. plt.ylabel('Estimated Salary')
16. plt.legend()
17. plt.show()

```

Figure 17.19

**Output:**

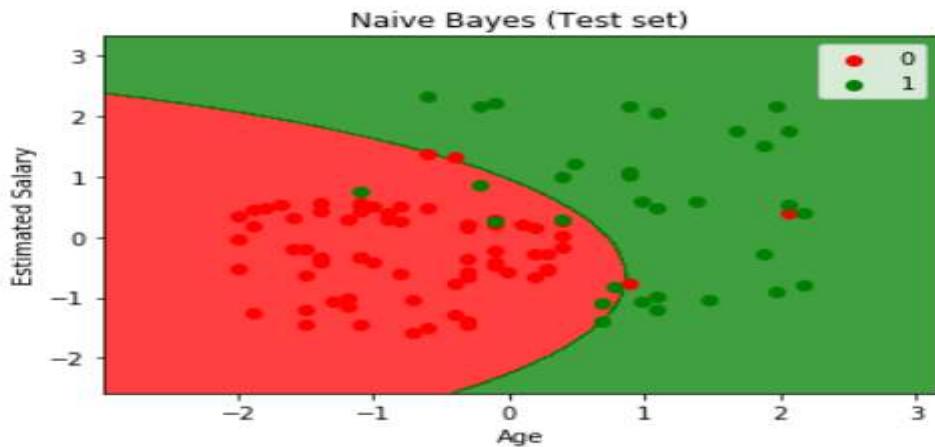


Figure 17.20

The above output is final output for test set data. As we can see the classifier has created a Gaussian curve to divide the "purchased" and "not purchased" variables. There are some wrong predictions which we have calculated in Confusion matrix. But still it is pretty good classifier

---

# CHAPTER 18

---

## Decision Tree Classification

### 18.1 Overview

1. Decision Tree is a **Supervised learning technique** that can be used for both classification and Regression problems, but mostly it is preferred for solving Classification problems. It is a tree-structured classifier, where **internal nodes represent the features of a dataset, branches represent the decision rules and each leaf node represents the outcome**.
2. In a Decision tree, there are two nodes, which are the **Decision Node** and **Leaf Node**. Decision nodes are used to make any decision and have multiple branches, whereas Leaf nodes are the output of those decisions and do not contain any further branches.
3. The decisions or the test are performed on the basis of features of the given dataset.
4. It is a graphical representation for getting all the possible solutions to a problem/decision based on given conditions.

5. It is called a decision tree because, similar to a tree, it starts with the root node, which expands on further branches and constructs a tree-like structure.
6. In order to build a tree, we use the **CART algorithm**, which stands for **Classification and Regression Tree algorithm**.
7. A decision tree simply asks a question, and based on the answer (Yes/No), it further split the tree into subtrees.

Below diagram explains the general structure of a decision tree:

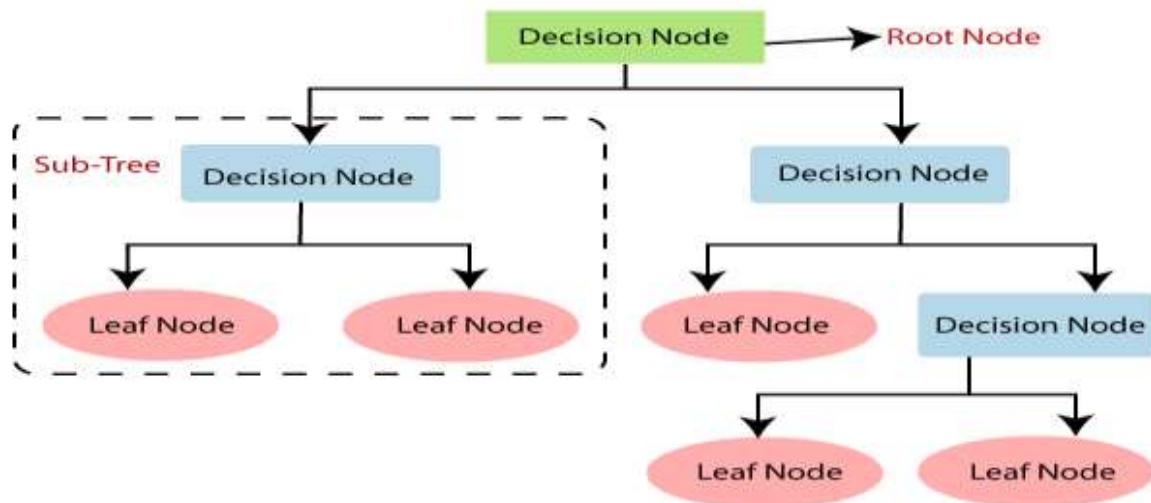


Figure 18.1

## Why use Decision Trees?

There are various algorithms in Machine learning, so choosing the best algorithm for the given dataset and problem is the main point to remember while creating a machine learning model. Below are the two reasons for using the Decision tree:

- Decision Trees usually mimic human thinking ability while making a decision, so it is easy to understand.
- The logic behind the decision tree can be easily understood because it shows a tree-like structure.

## Decision Tree Terminologies

- **Root Node:** Root node is from where the decision tree starts. It represents the entire dataset, which further gets divided into two or more homogeneous sets.
- **Leaf Node:** Leaf nodes are the final output node, and the tree cannot be segregated further after getting a leaf node.
- **Splitting:** Splitting is the process of dividing the decision node/root node into sub-nodes according to the given conditions.
- **Branch/Sub Tree:** A tree formed by splitting the tree.
- **Pruning:** Pruning is the process of removing the unwanted branches from the tree.

- **Parent/Child node:** The root node of the tree is called the parent node, and other nodes are called the child nodes.

## How does the Decision Tree algorithm Work?

In a decision tree, for predicting the class of the given dataset, the algorithm starts from the root node of the tree. This algorithm compares the values of root attribute with the record (real dataset) attribute and, based on the comparison, follows the branch and jumps to the next node.

For the next node, the algorithm again compares the attribute value with the other sub-nodes and move further. It continues the process until it reaches the leaf node of the tree. The complete process can be better understood using the below algorithm:

**Step-1:** Begin the tree with the root node, says S, which contains the complete dataset.

**Step-2:** Find the best attribute in the dataset using **Attribute Selection Measure (ASM)**.

**Step-3:** Divide the S into subsets that contains possible values for the best attributes.

**Step-4:** Generate the decision tree node, which contains the best attribute.

**Step-5:** Recursively make new decision trees using the subsets of the dataset created in step -3. Continue this process until a stage is reached where you cannot further classify the nodes and called the final node as a leaf node.

## Advantages of the Decision Tree

- It is simple to understand as it follows the same process which a human follow while making any decision in real-life.
- It can be very useful for solving decision-related problems.
- It helps to think about all the possible outcomes for a problem.
- There is less requirement of data cleaning compared to other algorithms.

## Disadvantages of the Decision Tree

- The decision tree contains lots of layers, which makes it complex.
- It may have an overfitting issue, which can be resolved using the **Random Forest algorithm**.
- For more class labels, the computational complexity of the decision tree may increase.

Now we will implement the Decision tree using Python. For this, we will use the dataset "user\_data.csv," which we have used in previous classification models. By using the same dataset, we can compare the Decision tree classifier with other classification models such as KNN SVM, LogisticRegression, etc.

Steps will also remain the same, which are given below:

1. **Data Pre-processing step**
2. **Fitting a Decision-Tree algorithm to the Training set**
3. **Predicting the test result**
4. **Test accuracy of the result(Creation of Confusion matrix)**
5. **Visualizing the test set result.**

## 1. Data Pre-Processing Step:

Below is the code for the pre-processing step:

```
1. # Decision Tree Classification
2.
3. # Importing the libraries
4. import numpy as np
5. import matplotlib.pyplot as plt
6. import pandas as pd
7.
8. # Importing the dataset
9. dataset = pd.read_csv('Social_Network_Ads.csv')
10. X = dataset.iloc[:, [2, 3]].values
11. y = dataset.iloc[:, -1].values
```

Figure 18.2

In the above code, we have pre-processed the data. Where we have loaded the dataset, which is given as:

dataset - DataFrame

Index	User ID	Gender	Age	matedSal	purchased
0	15624510	Male	19	19000	0
1	15816944	Male	35	20000	0
2	15668575	Female	26	43000	0
3	15603246	Female	27	57000	0
4	15894002	Male	19	76000	0
5	15728773	Male	27	58000	0
6	15598044	Female	27	84000	0
7	15694829	Female	32	150000	1
8	15600575	Male	25	33000	0
9	15727311	Female	35	65000	0
10	15576769	Female	26	80000	0

Figure 18.3

X - NumPy array

	0	1
0	19	19000
1	35	20000
2	26	43000
3	27	57000
4	19	76000
5	27	58000
6	27	84000
7	32	150000
8	25	33000
9	26	65000

Figure 18.4

y - NumPy array

	0
0	0
1	0
2	0
3	0
4	0
5	0
6	0
7	1
8	0
9	0

Figure 18.5

```
1. # Splitting the dataset into the Training set and Test set
2.
3. from sklearn.model_selection import train_test_split
4. X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_state = 0)
5.
6. # Feature Scaling
7.
8. from sklearn.preprocessing import StandardScaler
9. sc = StandardScaler()
10. X_train = sc.fit_transform(X_train)
11. X_test = sc.transform(X_test)
```

Figure 18.6

X_test - NumPy array	
0	1
0	38
1	50000
2	75000
3	79000
4	50000
5	20000
6	15000
7	144000
8	68000
9	12000

Figure 18.7

■■■ X\_train - NumPy array

	0	1
0	44	39000
1	32	120000
2	38	50000
3	32	135000
4	52	21000
5	53	104000
6	39	42000
7	38	61000
8	36	50000
9	35	63000

Format    Resize     Background color

Save and Close    Close

Figure 18.8

■■■ y\_train - NumPy array

	0
0	0
1	1
2	0
3	1
4	1
5	1
6	0
7	0
8	0
9	0

Format    Resize     Background color

Save and Close    Close

Figure 18.9

## 2. Fitting a Decision-Tree algorithm to the Training set

Now we will fit the model to the training set. For this, we will import the **DecisionTreeClassifier** class from **sklearn.tree** library.

Below is the code for it:

```
1. # Training the Decision Tree Classification model on the Training set
2.
3. from sklearn.tree import DecisionTreeClassifier
4. classifier = DecisionTreeClassifier(criterion = 'entropy', random_state = 0)
5. classifier.fit(X_train, y_train)
```

Figure 18.10

In the above code, we have created a classifier object, in which we have passed two main parameters;

- **"criterion='entropy'"**: Criterion is used to measure the quality of split, which is calculated by information gain given by entropy.
- **random\_state=0"**: For generating the random states.

Below is the output for this:

```
Console 4/A
...: X_train = sc.fit_transform(X_train)
...: X_test = sc.transform(X_test)

In [5]:
...: from sklearn.tree import DecisionTreeClassifier
...: classifier = DecisionTreeClassifier(criterion = 'entropy',
...: random_state = 0)
...: classifier.fit(X_train, y_train)
Out[5]:
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None,
criterion='entropy',
max_depth=None, max_features=None,
max_leaf_nodes=None,
min_impurity_decrease=0.0, min_impurity_split=None,
min_samples_leaf=1, min_samples_split=2,
min_weight_fraction_leaf=0.0, presort='deprecated',
random_state=0, splitter='best')

In [6]:
```

Figure 18.11

### 3. Predicting the test result

Now we will predict the test set result. We will create a new prediction vector **y\_pred**. Below is the code for it:

```
1. # Predicting the Test set results
2.
3. y_pred = classifier.predict(X_test)
```

Figure 18.12

### Output:

In the below output image, the predicted output and real test output are given. We can clearly see that there are some values in the prediction vector, which are different from the real vector values. These are prediction errors.

Figure 18.13 shows a table titled "y\_pred - NumPy array". The table has 9 rows and 2 columns. The first 8 rows have a red background color, and the 9th row has a blue background color. The data is as follows:

	0
0	0
1	0
2	0
3	0
4	0
5	0
6	0
7	1
8	0
9	0

Buttons at the bottom: Format, Resize,  Background color, Save and Close, Close.

Figure 18.13

Figure 18.14 shows a table titled "y\_test - NumPy array". The table has 9 rows and 2 columns. The first 8 rows have a red background color, and the 9th row has a blue background color. The data is as follows:

	0
0	0
1	0
2	0
3	0
4	0
5	0
6	0
7	1
8	0
9	0

Buttons at the bottom: Format, Resize,  Background color, Save and Close, Close.

Figure 18.14

#### 4. Test accuracy of the result (Creation of Confusion matrix)

In the above output, we have seen that there were some incorrect predictions, so if we want to know the number of correct and incorrect predictions, we need to use the confusion matrix. Below is the code for it:

```
1. # Making the Confusion Matrix
2.
3. from sklearn.metrics import confusion_matrix
4. cm = confusion_matrix(y_test, y_pred)
```

Figure 18.15

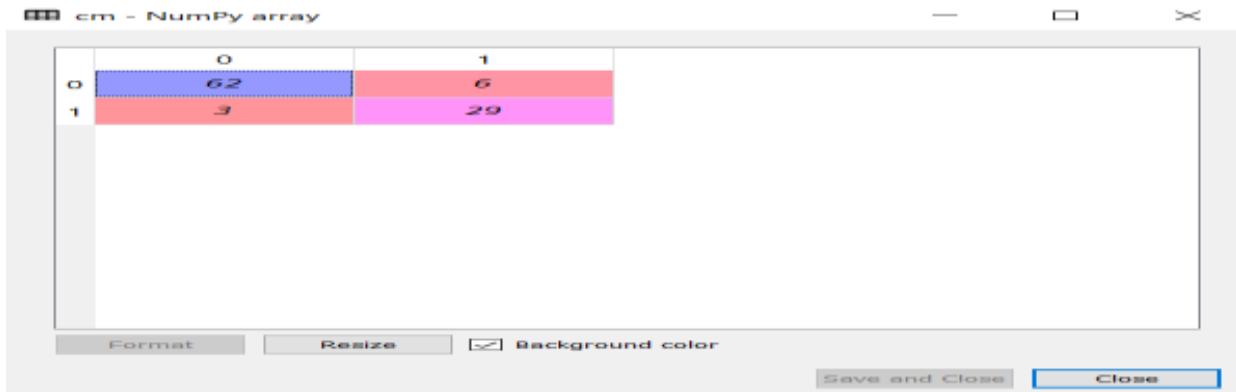


Figure 18.16

In the above output image, we can see the confusion matrix, which has  $6+3=9$  incorrect predictions and  $62+29=91$  correct predictions. Therefore, we can say that compared to other classification models, the Decision Tree classifier made a good prediction.

## 5. Visualizing the training set result:

Here we will visualize the training set result. To visualize the training set result we will plot a graph for the decision tree classifier. The classifier will predict yes or No for the users who have either Purchased or Not purchased the SUV car as we did in Logistic Regression. Below is the code for it:

```
1. # Visualising the Training set results
2.
3. from matplotlib.colors import ListedColormap
4. X_set, y_set = X_train, y_train
5. X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 1, stop = X_set[:, 0].max() + 1, step = 0.01),
6.                      np.arange(start = X_set[:, 1].min() - 1, stop = X_set[:, 1].max() + 1, step = 0.01))
7. plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape),
8.               alpha = 0.75, cmap = ListedColormap(('red', 'green')))
9. plt.xlim(X1.min(), X1.max())
10. plt.ylim(X2.min(), X2.max())
11. for i, j in enumerate(np.unique(y_set)):
12.     plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
13.                 c = ListedColormap(('red', 'green'))(i), label = j)
14. plt.title('Decision Tree Classification (Training set)')
15. plt.xlabel('Age')
16. plt.ylabel('Estimated Salary')
17. plt.legend()
18. plt.show()
```

Figure 18.17

**Output:**

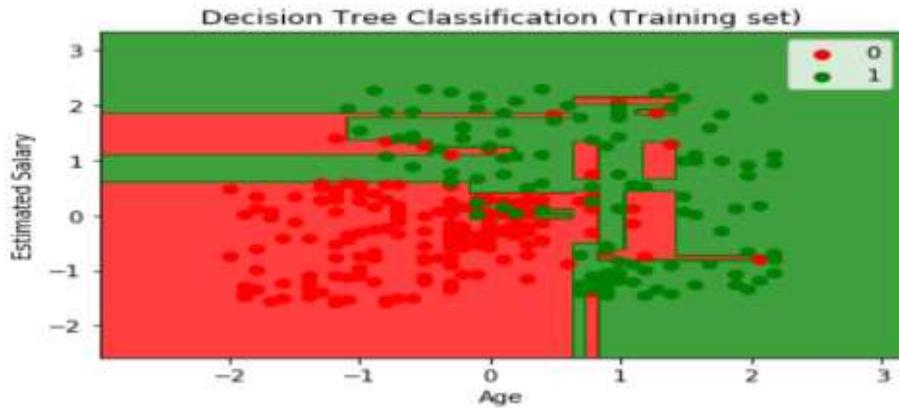


Figure 18.18

The above output is completely different from the rest classification models. It has both vertical and horizontal lines that are splitting the dataset according to the age and estimated salary variable.

As we can see, the tree is trying to capture each dataset, which is the case of overfitting.

## 6. Visualizing the test set result:

Visualization of test set result will be similar to the visualization of the training set except that the training set will be replaced with the test set.

```

1. # Visualising the Test set results
2.
3. from matplotlib.colors import ListedColormap
4. X_set, y_set = X_test, y_test
5. X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 1, stop = X_set[:, 0].max() + 1, step = 0.01),
6.                      np.arange(start = X_set[:, 1].min() - 1, stop = X_set[:, 1].max() + 1, step = 0.01))

```

```

7. plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X
1.shape),
8.                 alpha = 0.75, cmap = ListedColormap(('red', 'green')))
9. plt.xlim(X1.min(), X1.max())
10. plt.ylim(X2.min(), X2.max())
11. for i, j in enumerate(np.unique(y_set)):
12.     plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
13.                 c = ListedColormap(('red', 'green'))(i), label = j)
14. plt.title('Decision Tree Classification (Test set)')
15. plt.xlabel('Age')
16. plt.ylabel('Estimated Salary')
17. plt.legend()
18. plt.show()

```

Figure 18.19

## Output:



Figure 18.20

As we can see in the above image that there are some green data points within the purple region and vice versa. So, these are the incorrect predictions which we have discussed in the confusion matrix.

---

# CHAPTER 19

---

## Random Forest Classification

### 19.1 Overview

Random Forest is a popular machine learning algorithm that belongs to the supervised learning technique. It can be used for both Classification and Regression problems in ML. It is based on the concept of ensemble learning, which is a process of combining multiple classifiers to solve a complex problem and to improve the performance of the model.

As the name suggests, "Random Forest is a classifier that contains a number of decision trees on various subsets of the given dataset and takes the average to improve the predictive accuracy of that dataset." Instead of relying on one decision tree, the random forest takes the prediction from each tree and based on the majority votes of predictions, and it predicts the final output.

The greater number of trees in the forest leads to higher accuracy and prevents the problem of overfitting.

The below diagram explains the working of the Random Forest algorithm:

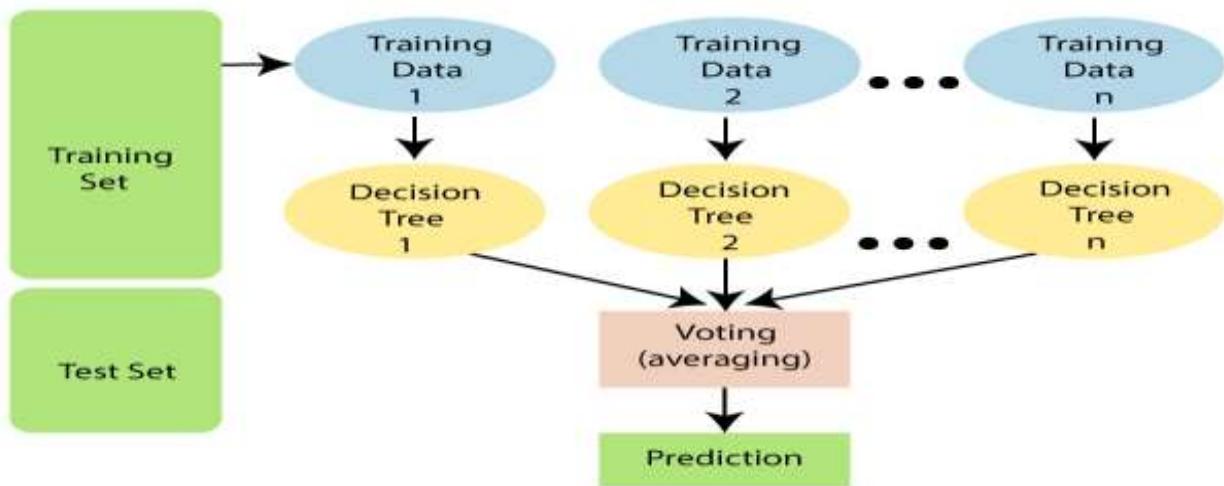


Figure 19.1

- **Assumptions for Random Forest**

Since the random forest combines multiple trees to predict the class of the dataset, it is possible that some decision trees may predict the correct output, while others may not. But together, all the trees predict the correct output. Therefore, below are two assumptions for a better Random forest classifier:

- There should be some actual values in the feature variable of the dataset so that the classifier can predict accurate results rather than a guessed result.
- The predictions from each tree must have very low correlations.

## Why use Random Forest?

Below are some points that explain why we should use the Random Forest algorithm:

- It takes less training time as compared to other algorithms.
- It predicts output with high accuracy, even for the large dataset it runs efficiently.
- It can also maintain accuracy when a large proportion of data is missing.

## How does Random Forest algorithm work?

Random Forest works in two-phase first is to create the random forest by combining N decision tree, and second is to make predictions for each tree created in the first phase.

The Working process can be explained in the below steps and diagram:

**Step-1:** Select random K data points from the training set.

**Step-2:** Build the decision trees associated with the selected data points (Subsets).

**Step-3:** Choose the number N for decision trees that you want to build.

**Step-4:** Repeat Step 1 & 2.

**Step-5:** For new data points, find the predictions of each decision tree, and assign the new data points to the category that wins the majority votes.

## **Applications of Random Forest**

There are mainly four sectors where Random forest mostly used:

1. **Banking:** Banking sector mostly uses this algorithm for the identification of loan risk.
2. **Medicine:** With the help of this algorithm, disease trends and risks of the disease can be identified.
3. **Land Use:** We can identify the areas of similar land use by this algorithm.
4. **Marketing:** Marketing trends can be identified using this algorithm.

## **Advantages of Random Forest**

- Random Forest is capable of performing both Classification and Regression tasks.
- It is capable of handling large datasets with high dimensionality.
- It enhances the accuracy of the model and prevents the overfitting issue.

## **Disadvantages of Random Forest**

- Although random forest can be used for both classification and regression tasks, it is not more suitable for Regression tasks.

## **Python Implementation of Random Forest Algorithm**

Now we will implement the Random Forest Algorithm tree using Python. For this, we will use the same dataset "user\_data.csv", which we have used in previous classification models. By using the same dataset, we can compare the Random Forest classifier with other classification models such as Decision tree Classifier, KNN, SVM, Logistic Regression, etc.

- Implementation Steps are given below:
  1. Data Pre-processing step
  2. Fitting the Random forest algorithm to the Training set
  3. Predicting the test result
  4. Test accuracy of the result (Creation of Confusion matrix)
  5. Visualizing the test set result.

## 1. Data Pre-Processing Step:

Below is the code for the pre-processing step:

```
1. # Random Forest Classification
2.
3. # Importing the libraries
4. import numpy as np
5. import matplotlib.pyplot as plt
6. import pandas as pd
7.
8. # Importing the dataset
9. dataset = pd.read_csv('Social_Network_Ads.csv')
10. X = dataset.iloc[:, [2, 3]].values
11. y = dataset.iloc[:, -1].values
```

Figure 19.2

In the above code, we have pre-processed the data. Where we have loaded the dataset, which is given as:

Index	User ID	Gender	Age	matchdSal	Purchased
0	15624510	Male	19	19000	0
1	15818944	Male	35	20000	0
2	15660575	Female	26	43000	0
3	15603246	Female	27	57000	0
4	15804002	Male	19	76000	0
5	15728772	Male	27	58000	0
6	15598044	Female	27	84000	0
7	15694929	Female	22	150000	1
8	156009575	Male	25	33000	0
9	15727311	Female	35	65000	0
10	15570769	Female	26	80000	0

Figure 19.3

	0	1
0	19	19000
1	35	20000
2	26	43000
3	27	57000
4	19	76000
5	27	58000
6	27	84000
7	32	150000
8	25	33000
9	35	65000

Figure 19.4



Figure 19.5

```
1. # Splitting the dataset into the Training set and Test set
2.
3. from sklearn.model_selection import train_test_split
4. X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_state = 0)
5.
6. # Feature Scaling
7. from sklearn.preprocessing import StandardScaler
8. sc = StandardScaler()
9. X_train = sc.fit_transform(X_train)
10. X_test = sc.transform(X_test)
```

Figure 19.6

■■■ X\_test - NumPy array

	0	1
0	38	870000
1	38	500000
2	35	750000
3	30	790000
4	35	500000
5	27	200000
6	31	150000
7	36	1440000
8	18	680000
9	47	420000

Format    Resize     Background color

Save and Close    Close

Figure 19.7

■■■ X\_train - NumPy array

	0	1
0	44	390000
1	32	1200000
2	38	500000
3	32	1350000
4	52	21000
5	53	1040000
6	39	420000
7	38	61000
8	36	500000
9	36	620000

Format    Resize     Background color

Save and Close    Close

Figure 19.8



Figure 19.9

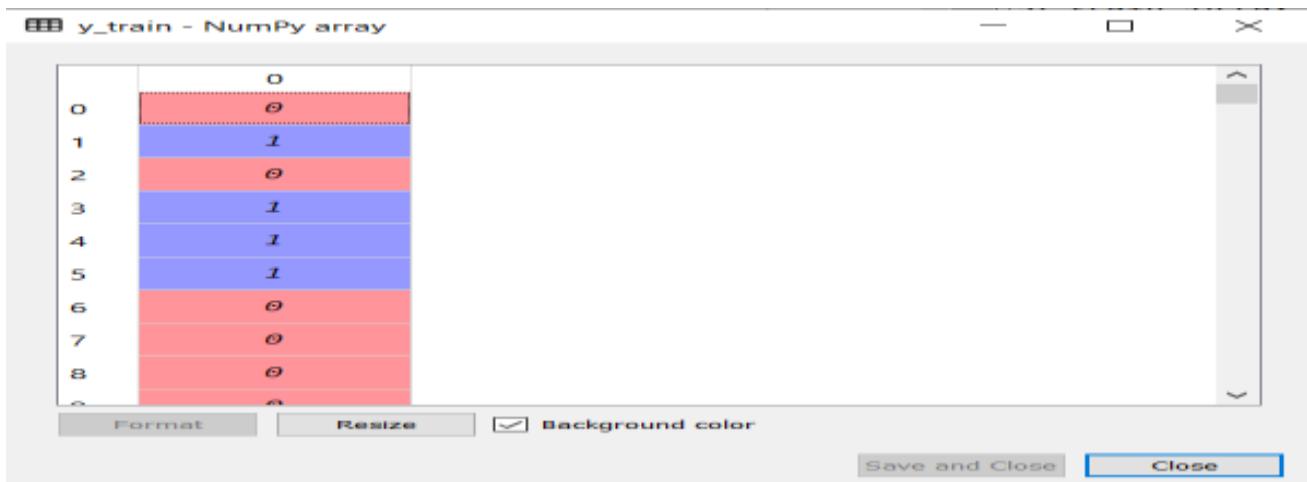


Figure 19.10

## 2. Fitting the Random Forest algorithm to the training set:

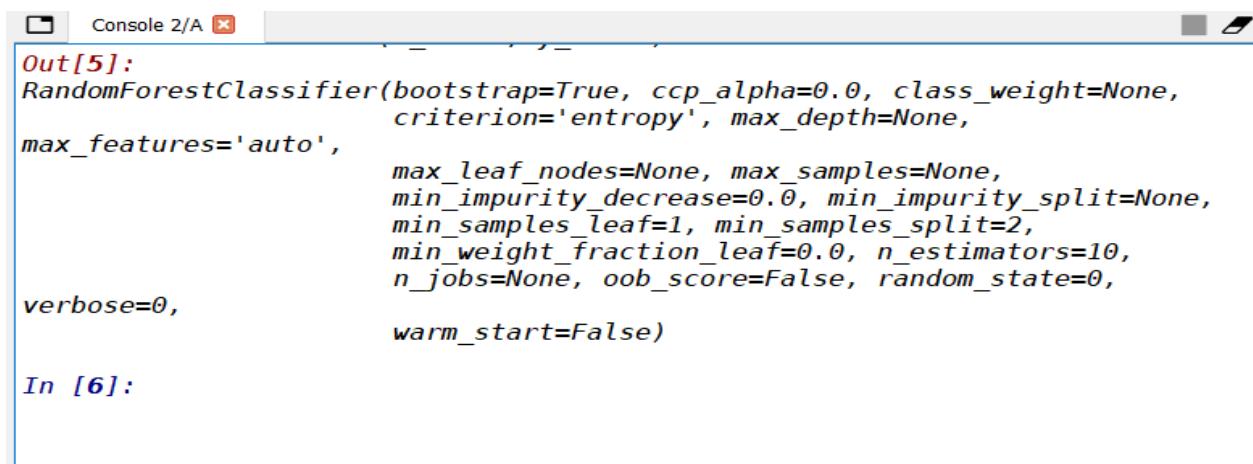
Now we will fit the Random forest algorithm to the training set. To fit it, we will import the **RandomForestClassifier** class from the **sklearn.ensemble** library. The code is given below:

```
1. # Training the Random Forest Classification model on the Training set
2.
3. from sklearn.ensemble import RandomForestClassifier
4. classifier = RandomForestClassifier(n_estimators = 10, criterion = 'entropy', random_state = 0)
5. classifier.fit(X_train, y_train)
```

Figure 19.10

In the above code, the classifier object takes below parameters:

- **n\_estimators=** The required number of trees in the Random Forest. The default value is 10. We can choose any number but need to take care of the overfitting issue.
- **criterion=** It is a function to analyze the accuracy of the split. Here we have taken "entropy" for the information gain.



The image shows a Jupyter Notebook cell titled "Console 2/A". The output (Out[5]) displays the configuration of a `RandomForestClassifier` object. The parameters set are: `bootstrap=True`, `ccp_alpha=0.0`, `class_weight=None`, `criterion='entropy'`, `max_depth=None`, `max_features='auto'`, `max_leaf_nodes=None`, `max_samples=None`, `min_impurity_decrease=0.0`, `min_impurity_split=None`, `min_samples_leaf=1`, `min_samples_split=2`, `min_weight_fraction_leaf=0.0`, `n_estimators=10`, `n_jobs=None`, `oob_score=False`, `random_state=0`, `verbose=0`, and `warm_start=False`. The input (In [6]) is shown as a blank line.

Figure 19.11

### 3. Predicting the Test Set result

Since our model is fitted to the training set, so now we can predict the test result. For prediction, we will create a new prediction vector `y_pred`. Below is the code for it:

```
1. # Predicting the Test set results
2.
3. y_pred = classifier.predict(X_test)
```

Figure 19.12

#### Output:

The prediction vector is given as:

y_pred - NumPy array	
0	0
1	0
2	0
3	0
4	0
5	0
6	0
7	1
8	0
9	1

Figure 19.13

By checking the above prediction vector and test set real vector, we can determine the incorrect predictions done by the classifier.

#### 4. Creating the Confusion Matrix

Now we will create the confusion matrix to determine the correct and incorrect predictions. Below is the code for it:

```
1. # Making the Confusion Matrix
2.
3. from sklearn.metrics import confusion_matrix
4. cm = confusion_matrix(y_test, y_pred)
```

Figure 19.14

#### Output:

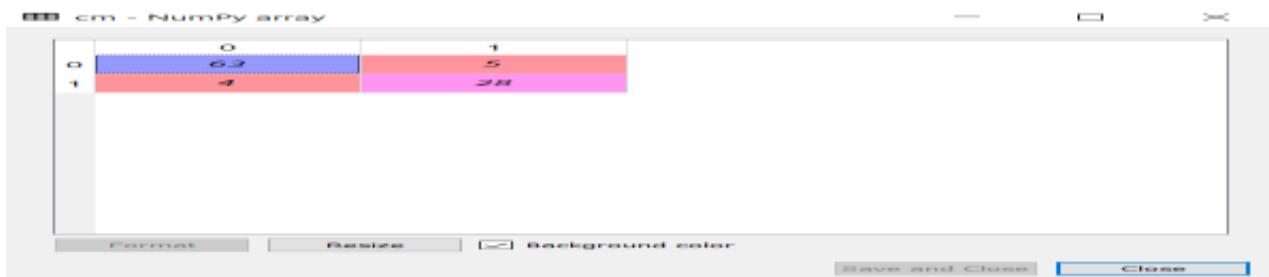


Figure 19.15

As we can see in the above matrix, there are  $4+4=8$  incorrect predictions and  $64+28=92$  correct predictions.

#### 5. Visualizing the training Set result

Here we will visualize the training set result. To visualize the training set result we will plot a graph for the Random forest classifier. The classifier will predict yes or No for the users who

have either Purchased or Not purchased the SUV car as we did in Logistic Regression. Below is the code for it:

```
1. # Visualising the Training set results
2.
3. from matplotlib.colors import ListedColormap
4. X_set, y_set = X_train, y_train
5. X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 1, stop = X_set[:, 0].max() + 1, step = 0.01),
6.                      np.arange(start = X_set[:, 1].min() - 1, stop = X_set[:, 1].max() + 1, step = 0.01))
7. plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape),
8.               alpha = 0.75, cmap = ListedColormap(('red', 'green')))
9. plt.xlim(X1.min(), X1.max())
10. plt.ylim(X2.min(), X2.max())
11. for i, j in enumerate(np.unique(y_set)):
12.     plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
13.                 c = ListedColormap(('red', 'green'))(i), label = j)
14. plt.title('Random Forest Classification (Training set)')
15. plt.xlabel('Age')
16. plt.ylabel('Estimated Salary')
17. plt.legend()
18. plt.show()
```

Figure 19.16

## Output:

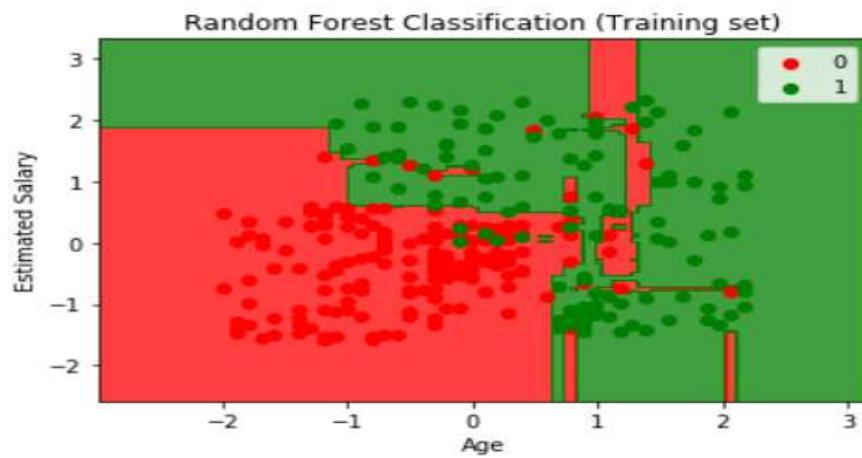


Figure 19.17

The above image is the visualization result for the Random Forest classifier working with the training set result. It is very much similar to the Decision tree classifier. Each data point corresponds to each user of the user\_data, and the purple and green regions are the prediction regions. The purple region is classified for the users who did not purchase the SUV car, and the green region is for the users who purchased the SUV.

So, in the Random Forest classifier, we have taken 10 trees that have predicted Yes or NO for the Purchased variable. The classifier took the majority of the predictions and provided the result.

## 6. Visualizing the test set result

Now we will visualize the test set result. Below is the code for it:

```
1. # Visualising the Test set results
2.
3. from matplotlib.colors import ListedColormap
4. X_set, y_set = X_test, y_test
5. X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 1, stop = X_set[:, 0].max()
+ 1, step = 0.01),
6.                      np.arange(start = X_set[:, 1].min() - 1, stop = X_set[:, 1].max()
+ 1, step = 0.01))
7. plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X
1.shape),
8.                 alpha = 0.75, cmap = ListedColormap(('red', 'green')))
9. plt.xlim(X1.min(), X1.max())
10. plt.ylim(X2.min(), X2.max())
11. for i, j in enumerate(np.unique(y_set)):
12.     plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
13.                 c = ListedColormap(('red', 'green'))(i), label = j)
14. plt.title('Random Forest Classification (Test set)')
15. plt.xlabel('Age')
16. plt.ylabel('Estimated Salary')
17. plt.legend()
18. plt.show()
```

Figure 19.18

## Output:

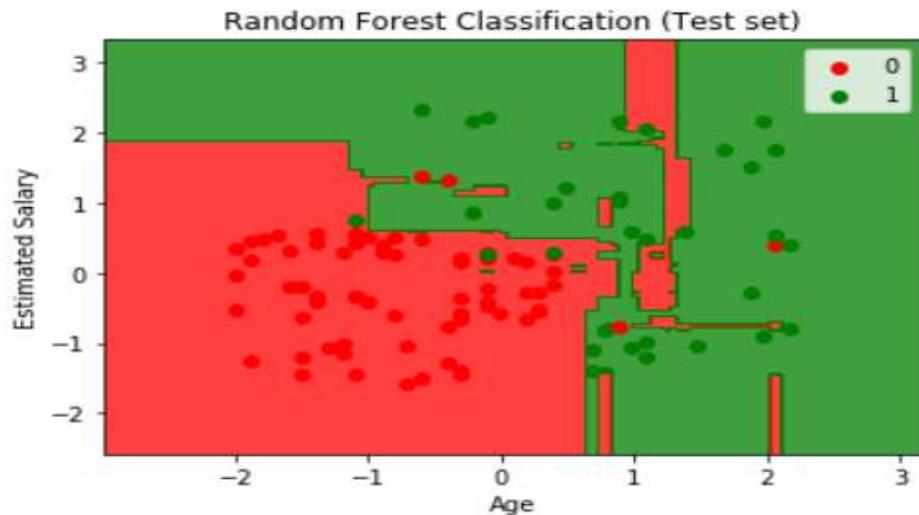


Figure 19.19

The above image is the visualization result for the test set. We can check that there is a minimum number of incorrect predictions (8) without the Overfitting issue. We will get different results by changing the number of trees in the classifier.

---

---

# CHAPTER 20

---

## Evaluating Classification Models Performance

### 20.1 Types of evaluating

1. Review of model evaluation
2. Model evaluation procedures
3. Model evaluation metrics
4. Classification accuracy
5. Confusion matrix
6. Metrics computed from a confusion matrix
7. Adjusting the classification threshold
8. Receiver Operating Characteristic (ROC) Curves
9. Area Under the Curve (AUC)

## 1. Review of model evaluation

- Need a way to choose between models: different model types, tuning parameters, and features
- Use a **model evaluation procedure** to estimate how well a model will generalize to out-of-sample data
- Requires a **model evaluation metric** to quantify the model performance

## 2. Model evaluation procedures

### 1- Training and testing on the same data

- Rewards overly complex models that "overfit" the training data and won't necessarily generalize

### 2- Train/test split

- Split the dataset into two pieces, so that the model can be trained and tested on different data
- Better estimate of out-of-sample performance, but still a "high variance" estimate
- Useful due to its speed, simplicity, and flexibility

### 3- K-fold cross-validation

- Systematically create "K" train/test splits and average the results together
- Even better estimate of out-of-sample performance
- Runs "K" times slower than train/test split

### 3. Model evaluation metrics

- **Regression problems:** Mean Absolute Error, Mean Squared Error, Root Mean Squared Error
- **Classification problems:** Classification accuracy
  - There are many more metrics, and we will discuss them today

### 4. Classification accuracy

[Pima Indian Diabetes Dataset](#) - PIDD from the UCI Machine Learning Repository. The dataset includes 768 instances and 8 attributes, plus the value of one class '0' which is treated as a negative test for diabetes, and the value of another class '1' which is treated as a positive test.

```
1. # read the data into a Pandas DataFrame
2. import pandas as pd
3.
4. url = 'https://archive.ics.uci.edu/ml/machine-learning-databases/pima-indians-
diabetes/pima-indians-diabetes.data'
5.
6. col_names = ['pregnant', 'glucose', 'bp', 'skin', 'insulin', 'bmi', 'pedigree', 'age', ''
label']
7.
8. pima = pd.read_csv(url, header=None, names=col_names)

1. # print the first 5 rows of data from the dataframe
2. pima.head()
```

Figure 20.1

## Output:

	<b>pregnant</b>	<b>glucose</b>	<b>bp</b>	<b>skin</b>	<b>insulin</b>	<b>bmi</b>	<b>pedigree</b>	<b>age</b>	<b>label</b>
<b>0</b>	6	148	72	35	0	33.6	0.627	50	1
<b>1</b>	1	85	66	29	0	26.6	0.351	31	0
<b>2</b>	8	183	64	0	0	23.3	0.672	32	1
<b>3</b>	1	89	66	23	94	28.1	0.167	21	0
<b>4</b>	0	137	40	35	168	43.1	2.288	33	1

Figure 20.2

- label
  - 1: diabetes
  - 0: no diabetes
- pregnant
  - number of times pregnant

## Question:

Can we predict the diabetes status of a patient given their health measurements?

```
1. # define X and y
2. feature_cols = ['pregnant', 'insulin', 'bmi', 'age']
3.
4. # X is a matrix, hence we use [] to access the features we want in feature_cols
5. X = pima[feature_cols]
6.
7. # y is a vector, hence we use dot to access 'label'
8. y = pima.label
```

```
1. # split X and y into training and testing sets
2. from sklearn.cross_validation import train_test_split
3. X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)
4. # train a logistic regression model on the training set
5. from sklearn.linear_model import LogisticRegression
6.
7. # instantiate model
8. logreg = LogisticRegression()
9.
10. # fit model
11. logreg.fit(X_train, y_train)
```

Figure 20.3

## Output:

```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
                   penalty='l2', random_state=None, solver='liblinear', tol=0.0001,
                   verbose=0, warm_start=False)
```

Figure 20.4

```
1. # make class predictions for the testing set
2. y_pred_class = logreg.predict(X_test)
```

Figure 20.5

## Classification accuracy:

percentage of correct predictions

```
1. # calculate accuracy
2. from sklearn import metrics
3. print(metrics.accuracy_score(y_test, y_pred_class))
```

Figure 20.6

**Output :**

0.692708333333

Classification accuracy is 69%

**Null accuracy:** accuracy that could be achieved by always predicting the most frequent class

- We must always compare with this

```
1. # examine the class distribution of the testing set (using a Pandas Series method)
2. y_test.value_counts()
```

Figure 20.7

**Output:**

0 130  
1 62

Name: label, dtype: int64

```
1. # calculate the percentage of ones
2. # because y_test only contains ones and zeros, we can simply calculate the mean = percentage of ones
3. y_test.mean()
```

Figure 20.8

**Output:**

0.3229166666666667

32% of the

```
1. # calculate the percentage of zeros
2. 1 - y_test.mean()
```

Figure 20.9

**Output:**

0.6770833333333333

```
1. # calculate null accuracy in a single line of code
2. # only for binary classification problems coded as 0/1
3. max(y_test.mean(), 1 - y_test.mean())
```

Figure 20.10

**Output:**

0.6770833333333333

This means that a dumb model that always predicts 0 would be right 68% of the time

- This shows how classification accuracy is not that good as it's close to a dumb model

- It's a good way to know the minimum we should achieve with our models

```
1. # calculate null accuracy (for multi-class classification problems)
2. y_test.value_counts().head(1) / len(y_test)
```

Figure 20.11

### Output:

```
0    0.677083
Name: label, dtype: float64
```

Comparing the **true** and **predicted** response values

```
1. # print the first 25 true and predicted responses
2. print('True:', y_test.values[0:25])
3. print('False:', y_pred_class[0:25])
```

Figure 20.12

### Output:

```
True: [1 0 0 1 0 0 1 1 0 0 1 1 0 0 0 0 1 0 0 0 1 1 0 0 0]
False: [0 0 0 0 0 0 0 1 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
```

### Conclusion:

- Classification accuracy is the **easiest classification metric to understand**
- But, it does not tell you the **underlying distribution** of response values

- We examine by calculating the null accuracy
- And, it does not tell you what "types" of errors your classifier is making

## 5. Confusion matrix

Table that describes the performance of a classification model

```

1. # IMPORTANT: first argument is true values, second argument is predicted values
2. # this produces a 2x2 numpy array (matrix)
3. print(metrics.confusion_matrix(y_test, y_pred_class))

```

Figure 20.13

Output:

```
[[118 12]
 [ 47 15]]
```

		<b>Predicted:</b>	
		<b>0</b>	<b>1</b>
<b>Actual:</b>	<b>0</b>	<b>118</b>	<b>12</b>
	<b>1</b>	<b>47</b>	<b>15</b>

Figure 20.14

- Every observation in the testing set is represented in **exactly one box**
- It's a 2x2 matrix because there are **2 response classes**
- The format shown here is **not** universal
  - Take attention to the format when interpreting a confusion matrix

## Basic terminology

- **True Positives (TP):** we *correctly* predicted that they *do* have diabetes
  - 15
- **True Negatives (TN):** we *correctly* predicted that they *don't* have diabetes
  - 118
- **False Positives (FP):** we *incorrectly* predicted that they *do* have diabetes (a "Type I error")
  - 12
  - Falsely predict positive
  - Type I error
- **False Negatives (FN):** we *incorrectly* predicted that they *don't* have diabetes (a "Type II error")
  - 47

- Falsely predict negative
- Type II error
- 0: negative class
- 1: positive class

```

1. # print the first 25 true and predicted responses
2. print('True', y_test.values[0:25])
3. print('Pred', y_pred_class[0:25])

```

Figure 20.15

## Output:

```

True [1 0 0 1 0 0 1 1 0 0 1 1 0 0 0 0 1 0 0 0 1 1 0 0 0]
Pred [0 0 0 0 0 0 0 1 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0]

```

```

1. # save confusion matrix and slice into four pieces
2. confusion = metrics.confusion_matrix(y_test, y_pred_class)
3. print(confusion)
4. #[row, column]
5. TP = confusion[1, 1]
6. TN = confusion[0, 0]
7. FP = confusion[0, 1]
8. FN = confusion[1, 0]

```

Figure 20.16

## Output:

```

[[118 12]
 [ 47 15]]

```

n=192	Predicted:		130
	0	1	
Actual: 0	TN = 118	FP = 12	130
Actual: 1	FN = 47	TP = 15	62
165		27	

Figure 20.17

```

1. # use float to perform true division, not integer division
2. print((TP + TN) / float(TP + TN + FP + FN))
3. print(metrics.accuracy_score(y_test, y_pred_class))

```

Figure 20.18

## Output:

0.692708333333  
0.692708333333

## Classification Error:

Overall, how often is the classifier incorrect?

- Also known as "Misclassification Rate"

```

classification_error = (FP + FN) / float(TP + TN + FP + FN)
print(classification_error)
print(1 - metrics.accuracy_score(y_test, y_pred_class))

```

Figure 20.19

### Output:

```
0.307291666667
0.307291666667
```

### Sensitivity:

When the actual value is positive, how often is the prediction correct?

- Something we want to maximize
- How "sensitive" is the classifier to detecting positive instances?
- Also known as "True Positive Rate" or "Recall"
- $TP / \text{all positive}$ 
  - $\text{all positive} = TP + FN$

```
sensitivity = TP / float(FN + TP)
print(sensitivity)
print(metrics.recall_score(y_test, y_pred_class))
```

Figure 20.20

## Output:

```
0.241935483871
0.241935483871
```

## Specificity:

When the actual value is negative, how often is the prediction correct?

- Something we want to maximize
- How "specific" (or "selective") is the classifier in predicting positive instances?
- $TN / \text{all negative}$ 
  - $\text{all negative} = TN + FP$

```
specificity = TN / (TN + FP)
print(specificity)
```

Figure 20.21

## Output:

```
0.907692307692
```

## Our classifier

- Highly specific
- Not sensitive

## False Positive Rate:

When the actual value is negative, how often is the prediction incorrect?

```
false_positive_rate = FP / float(TN + FP)
print(false_positive_rate)
print(1 - specificity)
```

Figure 20.22

## Output:

```
0.0923076923077
0.0923076923077
```

## Precision:

When a positive value is predicted, how often is the prediction correct?

- How "precise" is the classifier when predicting positive instances?

```
precision = TP / float(TP + FP)
print(precision)
print(metrics.precision_score(y_test, y_pred_class))
```

Figure 20.23

## Output:

0.555555555556  
0.555555555556

Many other metrics can be computed: F1 score, Matthews correlation coefficient, etc.

## Conclusion:

- Confusion matrix gives you a **more complete picture** of how your classifier is performing
- Also allows you to compute various **classification metrics**, and these metrics can guide your model selection

## Which metrics should you focus on?

- Choice of metric depends on your **business objective**
  - Identify if FP or FN is more important to reduce
  - Choose metric with relevant variable (FP or FN in the equation)

- **Spam filter** (positive class is "spam"):
  - Optimize for **precision or specificity**
    - precision
      - false positive as variable
    - specificity
      - false positive as variable
  - Because false negatives (spam goes to the inbox) are more acceptable than false positives (non-spam is caught by the spam filter)
- **Fraudulent transaction detector** (positive class is "fraud"):
  - Optimize for **sensitivity**
    - FN as a variable
  - Because false positives (normal transactions that are flagged as possible fraud) are more acceptable than false negatives (fraudulent transactions that are not detected)

## 7. Adjusting the classification threshold

```
# print the first 10 predicted responses
# 1D array (vector) of binary values (0, 1)
logreg.predict(X_test)[0:10]
```

Figure 20.24

### Output:

```
array([0, 0, 0, 0, 0, 0, 0, 1, 0, 1])
```

```
# print the first 10 predicted probabilities of class membership
logreg.predict_proba(X_test) [0:10]
```

Figure 20.25

## Output:

```
array([[ 0.63247571,  0.36752429],
       [ 0.71643656,  0.28356344],
       [ 0.71104114,  0.28895886],
       [ 0.5858938 ,  0.4141062 ],
       [ 0.84103973,  0.15896027],
       [ 0.82934844,  0.17065156],
       [ 0.50110974,  0.49889026],
       [ 0.48658459,  0.51341541],
       [ 0.72321388,  0.27678612],
       [ 0.32810562,  0.67189438]])
```

- Row: observation
  - Each row, numbers sum to 1
- Column: class
  - 2 response classes there 2 columns
    - column 0: predicted probability that each observation is a member of class 0
    - column 1: predicted probability that each observation is a member of class 1
- Importance of predicted probabilities
  - We can rank observations by probability of diabetes
    - Prioritize contacting those with a higher probability

- predict\_proba process
  1. Predicts the probabilities
  2. Choose the class with the highest probability
- There is a 0.5 classification threshold
  - Class 1 is predicted if probability  $> 0.5$
  - Class 0 is predicted if probability  $< 0.5$

```
# print the first 10 predicted probabilities for class 1
logreg.predict_proba(X_test) [0:10, 1]
```

Figure 20.26

## Output:

```
array([ 0.36752429, 0.28356344, 0.28895886, 0.4141062 , 0.1
5896027, 0.17065156, 0.49889026, 0.51341541, 0.27678612, 0
.67189438])
```

```
# store the predicted probabilities for class 1
y_pred_prob = logreg.predict_proba(X_test) [:, 1]
```

Figure 20.27

```
# allow plots to appear in the notebook
%matplotlib inline
import matplotlib.pyplot as plt

# adjust the font size
plt.rcParams['font.size'] = 12
```

Figure 20.28

```
# histogram of predicted probabilities
# 8 bins
```

```

plt.hist(y_pred_prob, bins=8)

# x-axis limit from 0 to 1
plt.xlim(0,1)
plt.title('Histogram of predicted probabilities')
plt.xlabel('Predicted probability of diabetes')
plt.ylabel('Frequency')

```

Figure 20.29

## Output:

<matplotlib.text.Text at 0x11c2b1128>

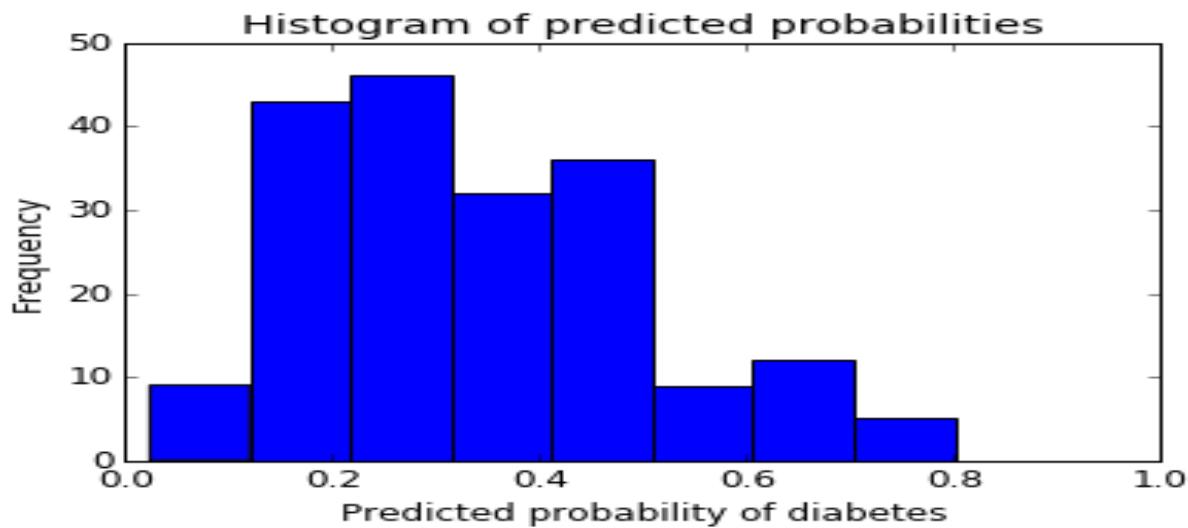


Figure 20.30

- We can see from the third bar

- About 45% of observations have probability from 0.2 to 0.3
- Small number of observations with probability  $> 0.5$
- This is below the threshold of 0.5
- Most would be predicted "no diabetes" in this case
- Solution
  - **Decrease the threshold** for predicting diabetes
    - **Increase the sensitivity** of the classifier
  - This would increase the number of TP
  - More sensitive to positive instances
  - Example of metal detector
  - Threshold set to set off alarm for large object but not tiny objects
  - YES: metal, NO: no metal
  - We lower the threshold amount of metal to set it off
  - It is now more sensitive to metal
  - It will then predict YES more often

```
# predict diabetes if the predicted probability is greater than 0.3
from sklearn.preprocessing import binarize
```

```
# it will return 1 for all values above 0.3 and 0 otherwise
# results are 2D so we slice out the first column
y_pred_class = binarize(y_pred_prob, 0.3)[0]
```

Figure 20.31

## Output:

```
/Users/ritchieng/anaconda3/envs/py3k/lib/python3.5/site-packages/sklearn/u
tills/validation.py:386: DeprecationWarning: Passing 1d arrays as data is d
eprecated in 0.17 and will raise ValueError in 0.19. Reshape your data eith
er using X.reshape(-1, 1) if your data has a single feature or X.reshape(1
, -1) if it contains a single sample.
  DeprecationWarning)
```

```
# print the first 10 predicted probabilities
y_pred_prob[0:10]
```

Figure 20.32

## Output:

```
array([ 0.36752429,  0.28356344,  0.28895886,  0.4141062 ,  0.1
5896027, 0.17065156,  0.49889026,  0.51341541,  0.27678612,  0
.67189438])
```

```
# print the first 10 predicted classes with the lower threshold
y_pred_class[0:10]
```

Figure 20.33

## Output:

```
array([ 1.,  0.,  0.,  1.,  0.,  0.,  1.,  1.,  0.,  1.])
```

```
# previous confusion matrix (default threshold of 0.5)
```

```
print(confusion)
```

Figure 20.34

## Output:

```
[[118 12]
 [ 47 15]]
```

```
# new confusion matrix (threshold of 0.3)
print(metrics.confusion_matrix(y_test, y_pred_class))
```

Figure 20.35

## Output:

```
[[80 50]
 [16 46]]
```

- The row totals are the same
- The rows represent actual response values
  - 130 values top row
  - 62 values bottom row
- Observations from the left column moving to the right column because we will have more TP and FP

```
# sensitivity has increased (used to be 0.24)
print (46 / float(46 + 16))
```

Figure 20.36

## Output:

0.7419354838709677

```
# specificity has decreased (used to be 0.91)
print(80 / float(80 + 50))
```

Figure 20.37

## Output:

0.6153846153846154

## Conclusion:

- **Threshold of 0.5** is used by default (for binary problems) to convert predicted probabilities into class predictions
- Threshold can be **adjusted** to increase sensitivity or specificity
- Sensitivity and specificity have an **inverse relationship**
  - Increasing one would always decrease the other
- Adjusting the threshold should be one of the last step you do in the model-building process
  - The most important steps are
    - Building the models
    - Selecting the best model

## 8. Receiver Operating Characteristic (ROC) Curves

**Question:** Wouldn't it be nice if we could see how sensitivity and specificity are affected by various thresholds, without actually changing the threshold?

**Answer:** Plot the ROC curve.

- Receiver Operating Characteristic (ROC)

```
# IMPORTANT: first argument is true values, second argument is predicted probabilities

# we pass y_test and y_pred_prob
# we do not use y_pred_class, because it will give incorrect results without generating an error
# roc_curve returns 3 objects fpr, tpr, thresholds
# fpr: false positive rate
# tpr: true positive rate
fpr, tpr, thresholds = metrics.roc_curve(y_test, y_pred_prob)

plt.plot(fpr, tpr)
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.0])
plt.rcParams['font.size'] = 12
plt.title('ROC curve for diabetes classifier')
plt.xlabel('False Positive Rate (1 - Specificity)')
plt.ylabel('True Positive Rate (Sensitivity)')
plt.grid(True)
```

Figure 20.38

**Output:**

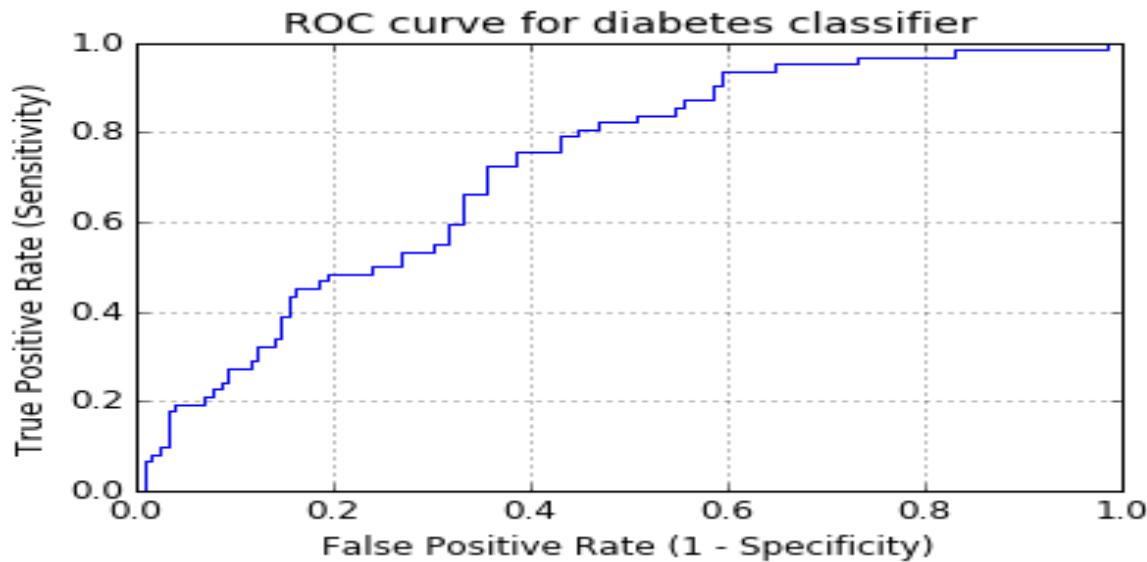


Figure 20.39

- ROC curve can help you to **choose a threshold** that balances sensitivity and specificity in a way that makes sense for your particular context
- You can't actually **see the thresholds** used to generate the curve on the ROC curve itself

```
# define a function that accepts a threshold and prints sensitivity and specificity
def evaluate_threshold(threshold):
    print('Sensitivity:', tpr[thresholds > threshold][-1])
    print('Specificity:', 1 - fpr[thresholds > threshold][-1])
```

Figure 20.41

```
evaluate_threshold(0.5)
```

Figure 20.42

**Output:**

```
Sensitivity: 0.241935483871
Specificity: 0.907692307692
```

```
evaluate_threshold(0.3)
```

Figure 20.43

## Output:

```
Sensitivity: 0.725806451613
Specificity: 0.615384615385
```

## 9. AUC

AUC is the **percentage** of the ROC plot that is **underneath the curve**:

```
# IMPORTANT: first argument is true values, second argument is predicted probabilities
print(metrics.roc_auc_score(y_test, y_pred_prob))
```

Figure 20.44

## Output:

```
0.724565756824
```

- AUC is useful as a **single number summary** of classifier performance
- Higher value = better classifier
- If you randomly chose one positive and one negative observation, AUC represents the likelihood that your classifier will assign a **higher predicted probability** to the positive observation
- AUC is useful even when there is **high class imbalance** (unlike classification accuracy)
  - Fraud case
    - Null accuracy almost 99%
    - AUC is useful here

```
# calculate cross-validated AUC
from sklearn.cross_validation import cross_val_score
cross_val_score(logreg, X, y, cv=10, scoring='roc_auc').mean()
```

Figure 20.45

## Output:

0.73782336182336183

Use both of these whenever possible

### 1. Confusion matrix advantages:

- Allows you to calculate a **variety of metrics**
- Useful for **multi-class problems** (more than two response classes)

---

## 2. ROC/AUC advantages:

- Does not require you to set a classification threshold
  - Still useful when there is high class imbalance
- 

# CHAPTER 21

---

## Clustering

### 21.1 Overview

It is basically a type of unsupervised learning method . An unsupervised learning method is a method in which we draw references from datasets consisting of input data without labelled responses. Generally, it is used as a process to find meaningful structure, explanatory underlying processes,

generative features, and groupings inherent in a set of examples.

Clustering is similar to classification, but the basis is different. In Clustering you don't know what you are looking for, and you are trying to identify some segments or clusters in your data. When you use clustering algorithms on your dataset, unexpected things can suddenly pop up like structures, clusters and groupings you would have never thought of otherwise.

In this part, you will understand and learn how to implement the following Machine Learning Clustering models:

- K-Means Clustering
- Hierarchical Clustering

## 21.2 Why Clustering

Clustering is very much important as it determines the intrinsic grouping among the unlabeled data present. There are no criteria for a good clustering. It depends on the user, what is the criteria they may use which satisfy their need. For instance, we could be interested in finding representatives for homogeneous groups (data reduction), in finding “natural clusters” and describe their unknown properties (“natural” data types), in

finding useful and suitable groupings (“useful” data classes) or in finding unusual data objects (outlier detection). This algorithm must make some assumptions which constitute the similarity of points and each assumption make different and equally valid clusters.

Clustering is very much important as it determines the intrinsic grouping among the unlabeled data present. There are no criteria for a good clustering. It depends on the user, what is the criteria they may use which satisfy their need. For instance, we could be interested in finding representatives for homogeneous groups (data reduction), in finding “natural clusters” and describe their unknown properties (“natural” data types), in finding useful and suitable groupings (“useful” data classes) or in finding unusual data objects (outlier detection). This algorithm must make some assumptions which constitute the similarity of points and each assumption make different and equally valid clusters.

### **Clustering Algorithms :**

It is the simplest unsupervised learning algorithm that solves clustering problem. K-means algorithm partition n observations into k clusters where each observation belongs to the cluster with the nearest mean serving as a prototype of the cluster.

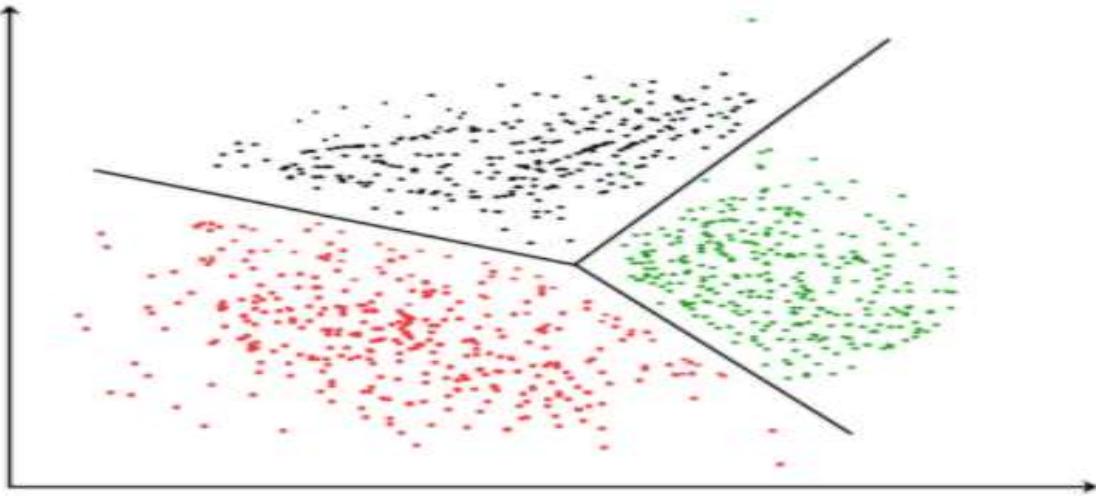


Figure 21.1

- Applications of Clustering in different fields

Marketing : It can be used to characterize & discover customer segments for marketing purposes.

Biology : It can be used for classification among different species of plants and animals.

Libraries : It is used in clustering different books on the basis of topics and information.

Insurance : It is used to acknowledge the customers, their policies and identifying the frauds.

### **Types of Clustering Methods:**

The clustering methods are broadly divided into Hard clustering (datapoint belongs to only one group) and Soft Clustering (data points can belong to another group also). But there are also

---

other various approaches of Clustering exist. Below are the main clustering methods used in Machine learning:

- 1- Partitioning Clustering
  - 2- Density-Based Clustering
  - 3- Distribution Model-Based Clustering
  - 4- Hierarchical Clustering
  - 5- Fuzzy Clustering
- 

## CHAPTER 22

---

### K-Means Clustering

#### 22.1 Overview

K-Means Clustering is an unsupervised learning algorithm that is used to solve the clustering problems in machine learning or data science. In this topic, we will learn what is K-means clustering algorithm, how the algorithm works, along with the Python implementation of k-means clustering.

## What is K-Means Algorithm?

K-Means Clustering is an Unsupervised Learning algorithm, which groups the unlabeled dataset into different clusters. Here K defines the number of pre-defined clusters that need to be created in the process, as if K=2, there will be two clusters, and for K=3, there will be three clusters, and so on.

It allows us to cluster the data into different groups and a convenient way to discover the categories of groups in the unlabeled dataset on its own without the need for any training.

It is a centroid-based algorithm, where each cluster is associated with a centroid. The main aim of this algorithm is to minimize the sum of distances between the data point and their corresponding clusters.

The algorithm takes the unlabeled dataset as input, divides the dataset into k-number of clusters, and repeats the process until it does not find the best clusters. The value of k should be predetermined in this algorithm.

The k-means clustering algorithm mainly performs two tasks:

Determines the best value for K center points or centroids by an iterative process.

Assigns each data point to its closest k-center. Those data points which are near to the particular k-center, create a cluster.

Hence each cluster has datapoints with some commonalities, and it is away from other clusters.

The below diagram explains the working of the K-means Clustering Algorithm:

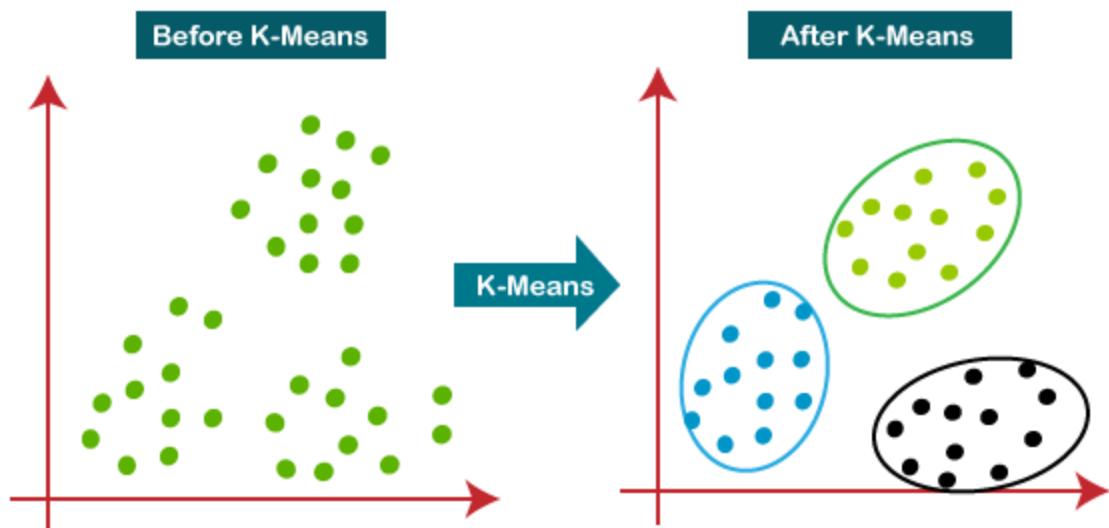


Figure 22.1

next →← prev

**K-Means Clustering Algorithm :**

K-Means Clustering is an unsupervised learning algorithm that is used to solve the clustering problems in machine learning or data science. In this topic, we will learn what is K-means clustering algorithm, how the algorithm works, along with the Python implementation of k-means clustering.

## **What is K-Means Algorithm?**

K-Means Clustering is an Unsupervised Learning algorithm, which groups the unlabeled dataset into different clusters. Here K defines the number of pre-defined clusters that need to be created in the process, as if K=2, there will be two clusters, and for K=3, there will be three clusters, and so on.

It is an iterative algorithm that divides the unlabeled dataset into k different clusters in such a way that each dataset belongs only one group that has similar properties.

It allows us to cluster the data into different groups and a convenient way to discover the categories of groups in the unlabeled dataset on its own without the need for any training.

It is a centroid-based algorithm, where each cluster is associated with a centroid. The main aim of this algorithm is to minimize the sum of distances between the data point and their corresponding clusters.

The algorithm takes the unlabeled dataset as input, divides the dataset into  $k$ -number of clusters, and repeats the process until it does not find the best clusters. The value of  $k$  should be predetermined in this algorithm.

The  $k$ -means clustering algorithm mainly performs two tasks:

Determines the best value for  $K$  center points or centroids by an iterative process.

Assigns each data point to its closest  $k$ -center. Those data points which are near to the particular  $k$ -center, create a cluster.

Hence each cluster has datapoints with some commonalities, and it is away from other clusters.

The below diagram explains the working of the K-means Clustering Algorithm:

## **K-Means Clustering Algorithm**

## How does the K-Means Algorithm Work?

The working of the K-Means algorithm is explained in the below steps:

**Step-1:** Select the number K to decide the number of clusters.

**Step-2:** Select random K points or centroids. (It can be other from the input dataset).

**Step-3:** Assign each data point to their closest centroid, which will form the predefined K clusters.

**Step-4:** Calculate the variance and place a new centroid of each cluster.

**Step-5:** Repeat the third steps, which means reassign each datapoint to the new closest centroid of each cluster.

**Step-6:** If any reassignment occurs, then go to step-4 else go to FINISH.

**Step-7:** The model is ready.

Let's understand the above steps by considering the visual plots:

Suppose we have two variables M1 and M2. The x-y axis scatter plot of these two variables is given below:

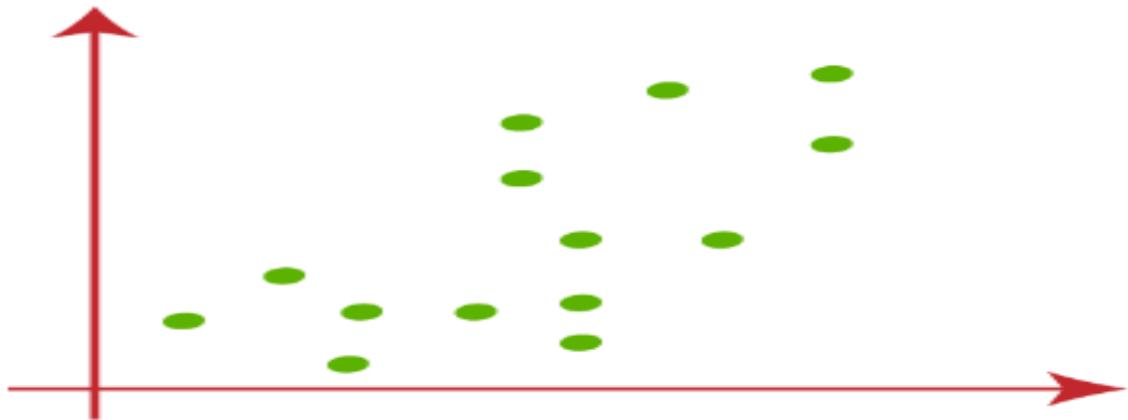


Figure 22.2

- Let's take number  $k$  of clusters, i.e.,  $K=2$ , to identify the dataset and to put them into different clusters. It means here we will try to group these datasets into two different clusters.
- We need to choose some random  $k$  points or centroid to form the cluster. These points can be either the points from the dataset or any other point. So, here we are selecting the below two points as  $k$  points, which are not the part of our dataset. Consider the below image:

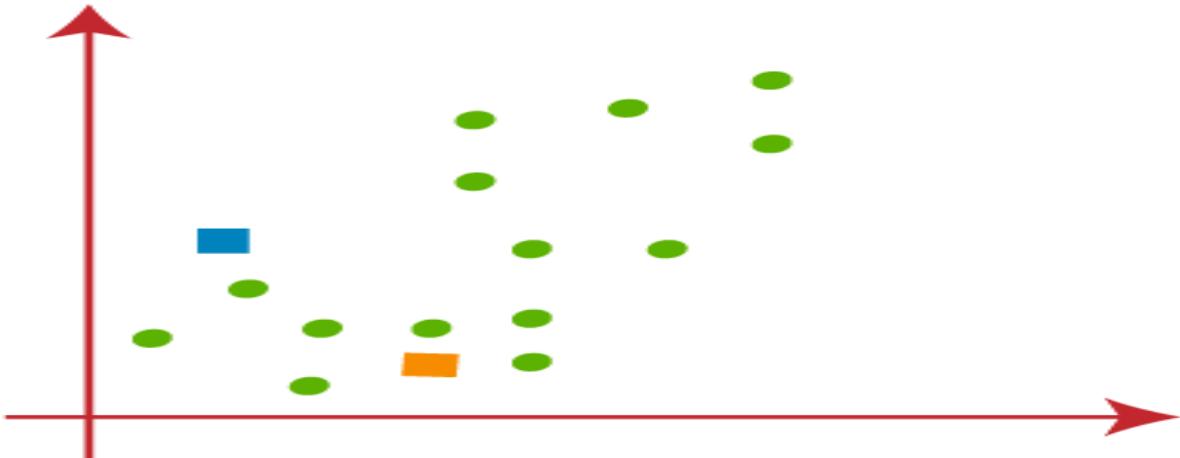


Figure 22.3

Now we will assign each data point of the scatter plot to its closest K-point or centroid. We will compute it by applying some mathematics that we have studied to calculate the distance between two points. So, we will draw a median between both the centroids. Consider the below image:

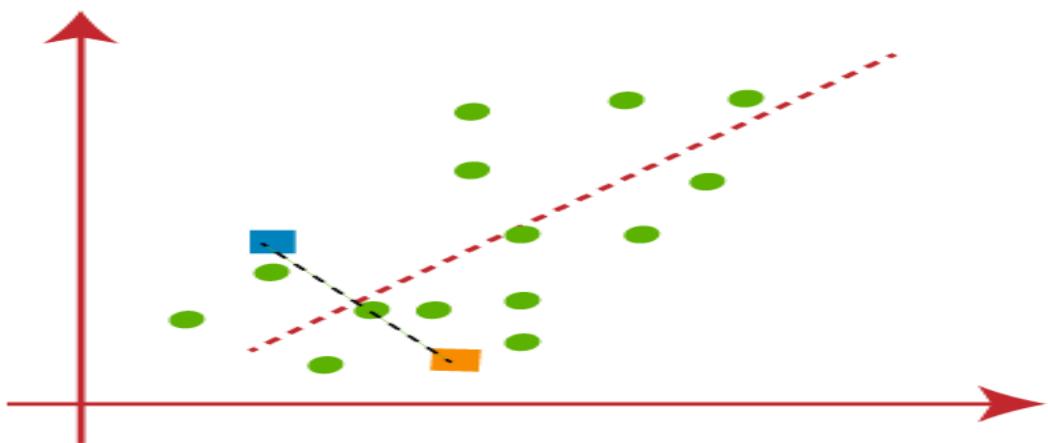


Figure 22.4

From the above image, it is clear that points left side of the line is near to the K1 or blue centroid, and points to the right of the

line are close to the yellow centroid. Let's color them as blue and yellow for clear visualization.

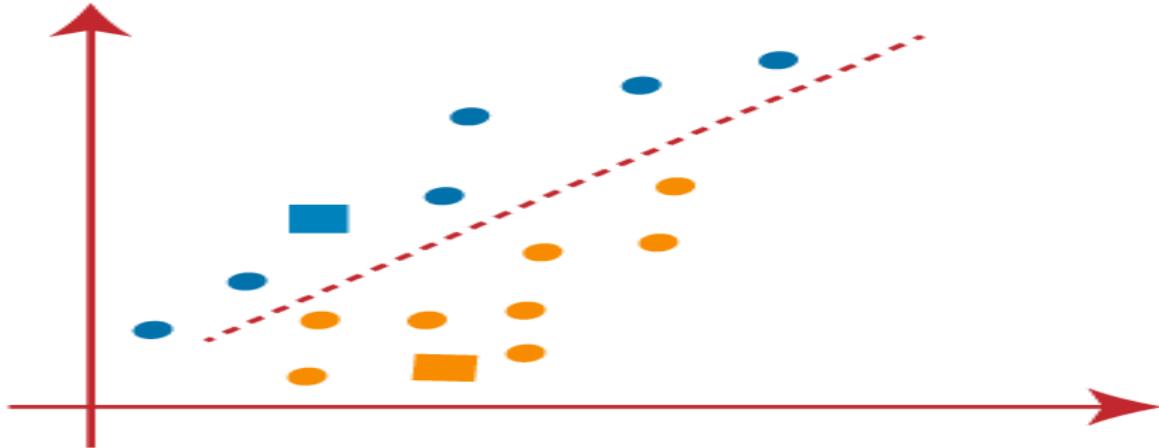


Figure 22.5

As we need to find the closest cluster, so we will repeat the process by choosing a new centroid. To choose the new centroids, we will compute the center of gravity of these centroids, and will find new centroids as below:

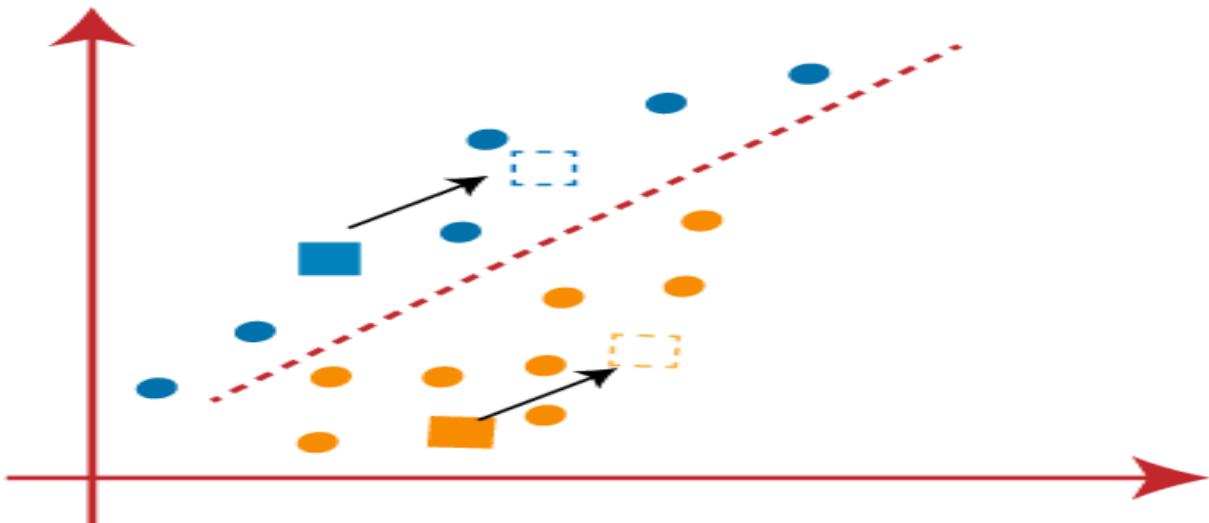


Figure 22.6

Next, we will reassign each datapoint to the new centroid. For this, we will repeat the same process of finding a median line. The median will be like below image:

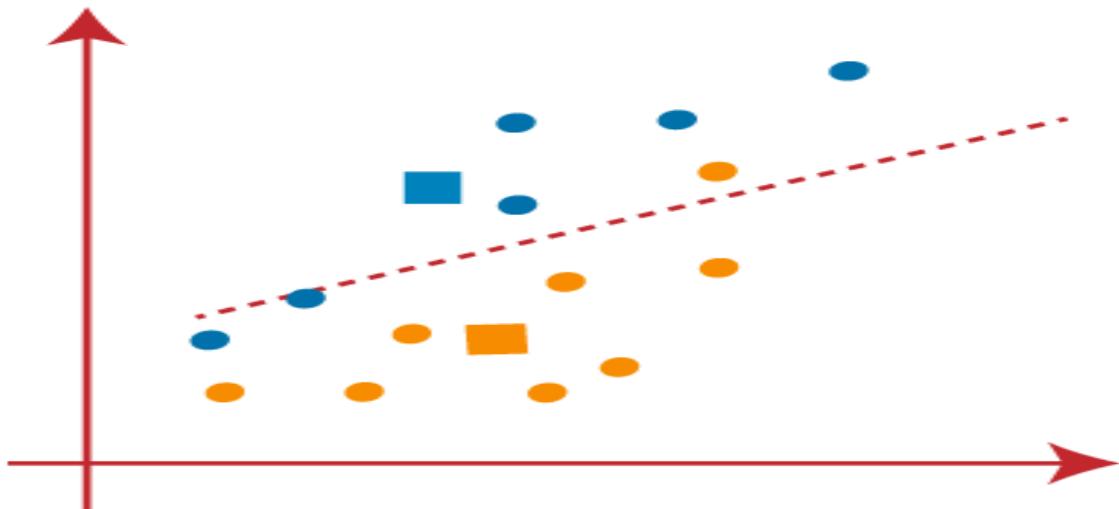


Figure 22.7

From the above image, we can see, one yellow point is on the left side of the line, and two blue points are right to the line. So, these three points will be assigned to new centroids.

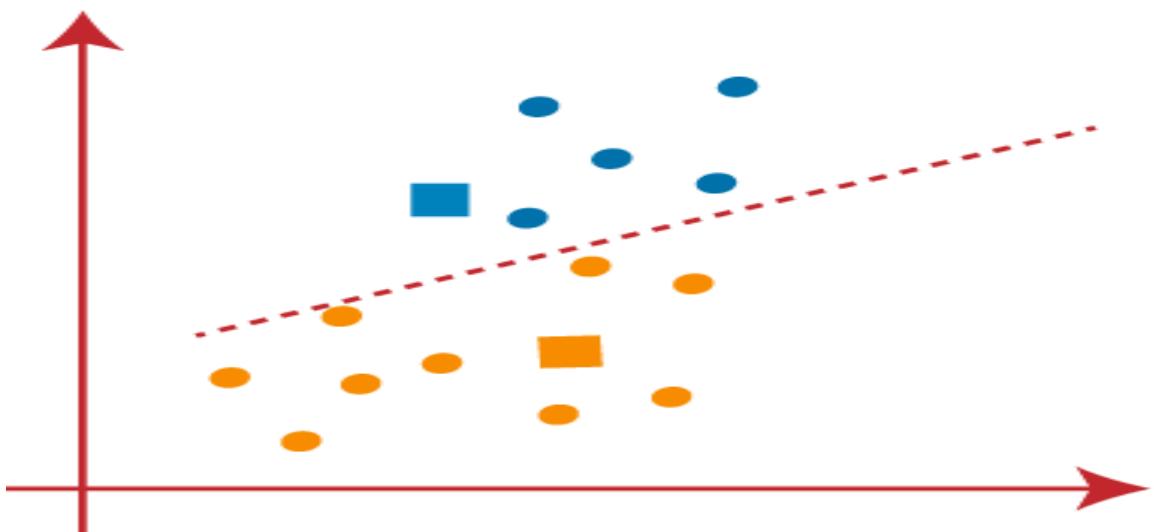


Figure 22.8

As reassignment has taken place, so we will again go to the step-4, which is finding new centroids or K-points.

- We will repeat the process by finding the center of gravity of centroids, so the new centroids will be as shown in the below image:

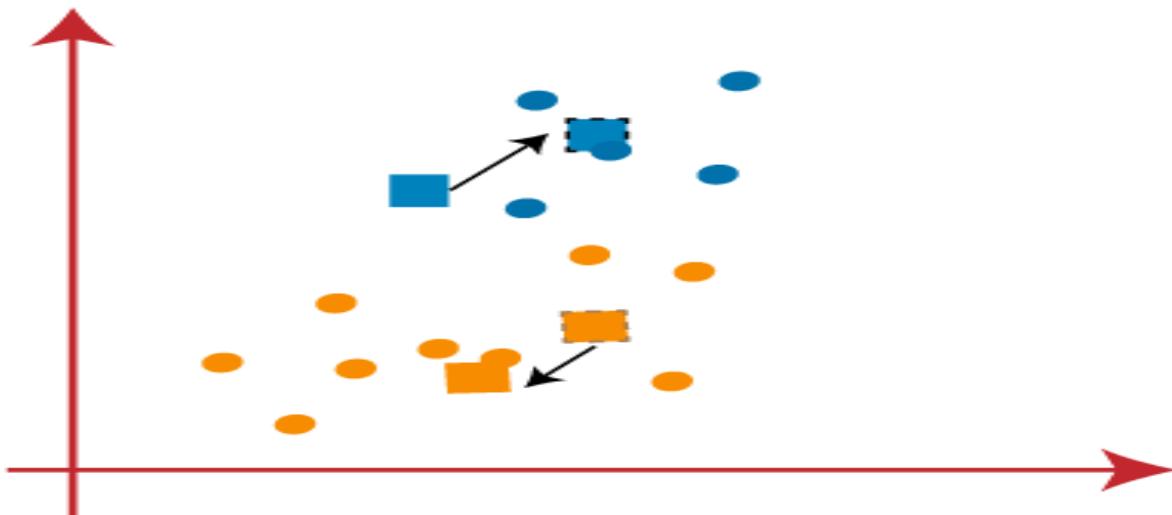


Figure 22.9

As we got the new centroids so again will draw the median line and reassign the data points. So, the image will be:

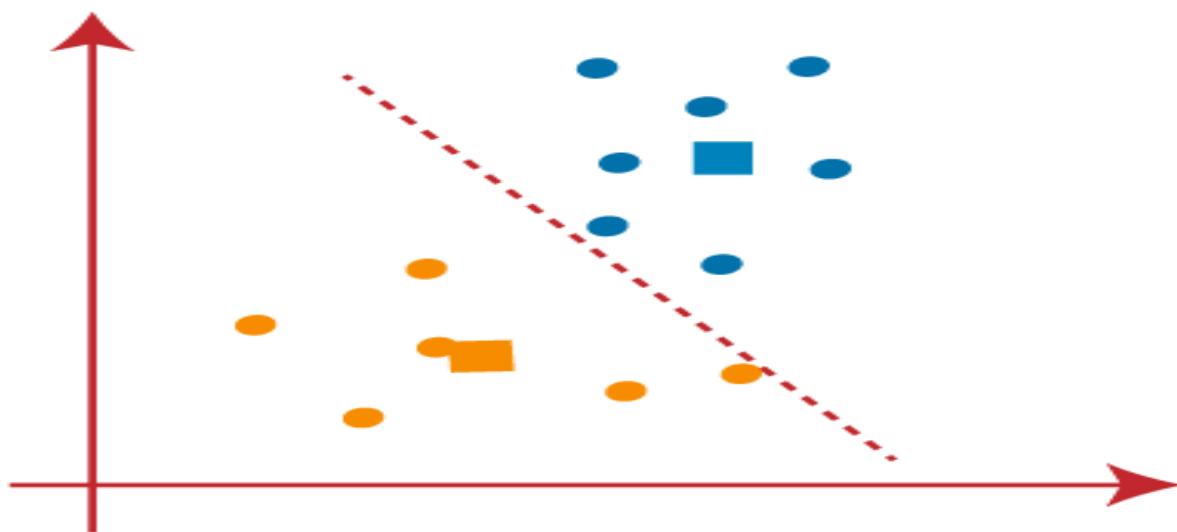


Figure 22.10

We can see in the above image; there are no dissimilar data points on either side of the line, which means our model is formed. Consider the below image:

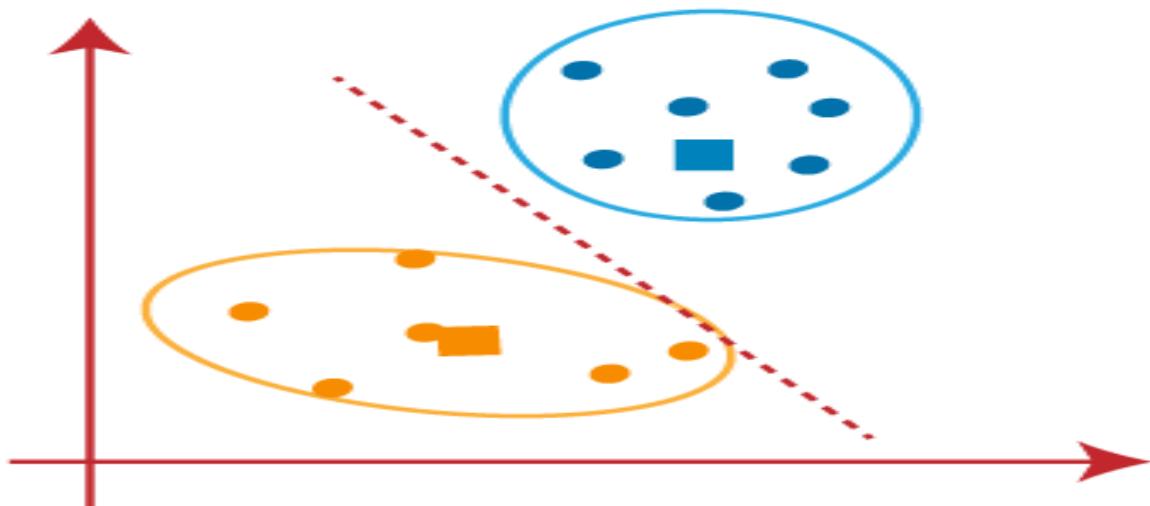


Figure 22.11

As our model is ready, so we can now remove the assumed centroids, and the two final clusters will be as shown in the below image:

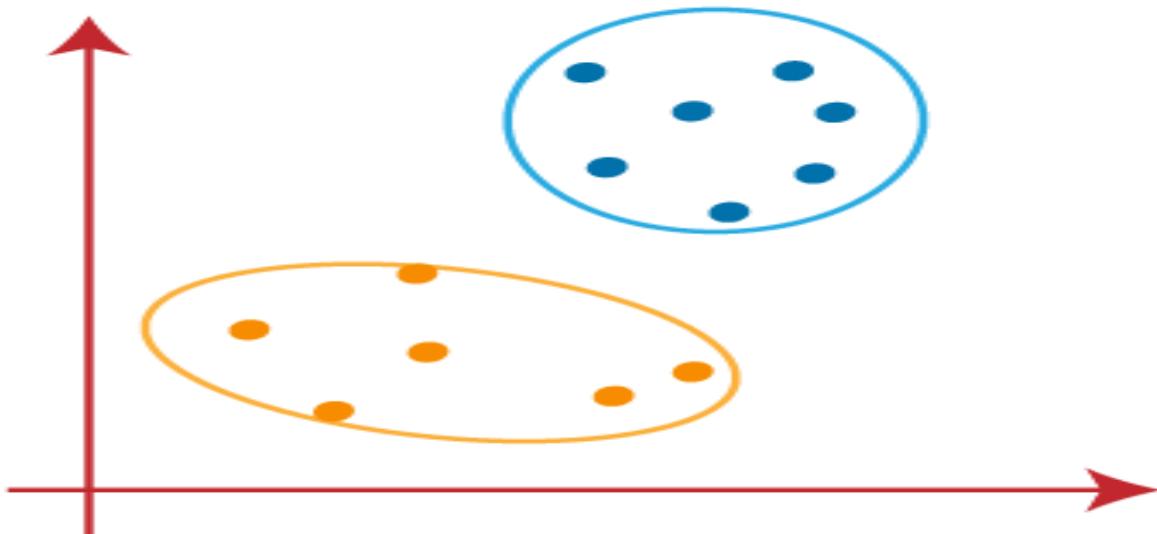


Figure 22.12

The performance of the K-means clustering algorithm depends upon highly efficient clusters that it forms. But choosing the optimal number of clusters is a big task. There are some different ways to find the optimal number of clusters, but here we are discussing the most appropriate method to find the number of clusters or value of K. The method is given below:

### Elbow Method

The Elbow method is one of the most popular ways to find the optimal number of clusters. This method uses the concept of WCSS value. WCSS stands for Within Cluster Sum of Squares, which defines the total variations within a cluster. The formula to calculate the value of WCSS (for 3 clusters) is given below:

$$\text{WCSS} = \sum_{P_i \text{ in Cluster1}} \text{distance}(P_i, C_1)^2 + \sum_{P_i \text{ in Cluster2}} \text{distance}(P_i, C_2)^2 + \sum_{P_i \text{ in Cluster3}} \text{distance}(P_i, C_3)^2$$

Figure 22.13

In the above formula of WCSS,

$\sum_{P_i \text{ in Cluster1}} \text{distance}(P_i, C_1)^2$ : It is the sum of the square of the distances between each data point and its centroid within a cluster1 and the same for the other two terms.

To measure the distance between data points and centroid, we can use any method such as Euclidean distance or Manhattan distance.

To find the optimal value of clusters, the elbow method follows the below steps:

- It executes the K-means clustering on a given dataset for different K values (ranges from 1-10).
- For each value of K, calculates the WCSS value.
- Plots a curve between calculated WCSS values and the number of clusters K.
- The sharp point of bend or a point of the plot looks like an arm, then that point is considered as the best value of K.

Since the graph shows the sharp bend, which looks like an elbow, hence it is known as the elbow method. The graph for the elbow method looks like the below image:

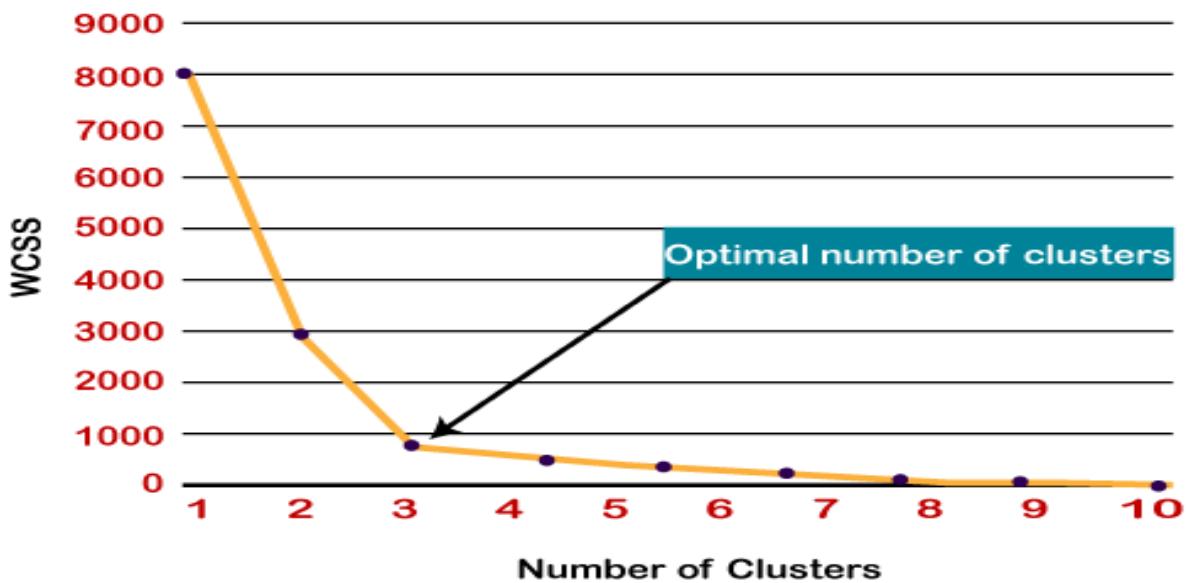


Figure 22.14

- Python Implementation of K-means Clustering Algorithm

In the above section, we have discussed the K-means algorithm, now let's see how it can be implemented using [Python](#).

Before implementation, let's understand what type of problem we will solve here. So, we have a dataset of **Mall\_Customers**, which is the data of customers who visit the mall and spend there.

In the given dataset, we have **Customer\_Id**, **Gender**, **Age**, **Annual Income (\$)**, and **Spending Score** (which is the calculated value of how much a customer has spent in the mall, the more the value, the more he has spent). From this dataset, we need to calculate some patterns, as it is an unsupervised method, so we don't know what to calculate exactly.

The steps to be followed for the implementation are given below:

- 1- **Data Pre-processing**
- 2- **Finding the optimal number of clusters using the elbow method**
- 3- **Training the K-means algorithm on the training dataset**
- 4- **Visualizing the clusters**

## **Step-1: Data pre-processing Step**

The first step will be the data pre-processing, as we did in our earlier topics of Regression and Classification. But for the

clustering problem, it will be different from other models. Let's discuss it:

## Importing Libraries

As we did in previous topics, firstly, we will import the libraries for our model, which is part of data pre-processing. The code is given below:

```
1. # K-Means Clustering
2.
3. # Importing the libraries
4.
5. import numpy as np
6. import matplotlib.pyplot as plt
7. import pandas as pd
```

Figure 22.15

In the above code, the numpy we have imported for the performing mathematics calculation, matplotlib is for plotting the graph, and pandas are for managing the dataset.

## Importing the Dataset:

Next, we will import the dataset that we need to use. So here, we are using the Mall\_Customer\_data.csv dataset. It can be imported using the below code:

```
1. # Importing the dataset
2.
3. dataset = pd.read_csv('Mall_Customers.csv')
```

Figure 22.16

By executing the above lines of code, we will get our dataset in the Spyder IDE. The dataset looks like the below image:

Index	CustomerID	Genre	Age	Annual Income	Spending Score
0	1	Male	19	15	39
1	2	Male	21	15	81
2	3	Female	20	16	6
3	4	Female	23	16	77
4	5	Female	31	17	40
5	6	Female	22	17	76
6	7	Female	35	18	6
7	8	Female	23	18	94
8	9	Male	64	19	3
9	10	Female	30	19	72
10	11	Male	67	19	14

Figure 22.17

From the above dataset, we need to find some patterns in it.

- Extracting Independent Variables

Here we don't need any dependent variable for data pre-processing step as it is a clustering problem, and we have no idea about what to determine. So we will just add a line of code for the matrix of features.

```
1. X = dataset.iloc[:, [3, 4]].values
```

Figure 22.18

Output :

	0	1
0	15	39
1	15	81
2	16	6
3	16	77
4	17	40
5	17	76
6	18	6
7	18	94
8	19	3
9	10	72

Figure 22.19

As we can see, we are extracting only 3<sup>rd</sup> and 4<sup>th</sup> feature. It is because we need a 2d plot to visualize the model, and some features are not required, such as customer\_id.

## Step-2: Finding the optimal number of clusters using the elbow method

In the second step, we will try to find the optimal number of clusters for our clustering problem. So, as discussed above, here we are going to use the elbow method for this purpose.

As we know, the elbow method uses the WCSS concept to draw the plot by plotting WCSS values on the Y-axis and the number of clusters on the X-axis. So we are going to calculate the value

for WCSS for different k values ranging from 1 to 10. Below is the code for it:

In the second step, we will try to find the optimal number of clusters for our clustering problem. So, as discussed above, here we are going to use the elbow method for this purpose.

As we know, the elbow method uses the WCSS concept to draw the plot by plotting WCSS values on the Y-axis and the number of clusters on the X-axis. So we are going to calculate the value for WCSS for different k values ranging from 1 to 10. Below is the code for it:

```
1. # Using the elbow method to find the optimal number of clusters
2.
3. from sklearn.cluster import KMeans
4. wcss = []
5. for i in range(1, 11):
6.     kmeans = KMeans(n_clusters = i, init = 'k-means++', random_state = 42)
7.     kmeans.fit(X)
8.     wcss.append(kmeans.inertia_)
9. plt.plot(range(1, 11), wcss)
10. plt.title('The Elbow Method')
11. plt.xlabel('Number of clusters')
12. plt.ylabel('WCSS')
13. plt.show()
```

Figure 22.20

As we can see in the above code, we have used the **KMeans** class of `sklearn.cluster` library to form the clusters.

Next, we have created the `wcss_list` variable to initialize an empty list, which is used to contain the value of wcss computed for different values of k ranging from 1 to 10.

After that, we have initialized the for loop for the iteration on a different value of k ranging from 1 to 10; since for loop in Python, exclude the outbound limit, so it is taken as 11 to include 10<sup>th</sup> value.

The rest part of the code is similar as we did in earlier topics, as we have fitted the model on a matrix of features and then plotted the graph between the number of clusters and WCSS.

## Output:

After executing the above code, we will get the below output:

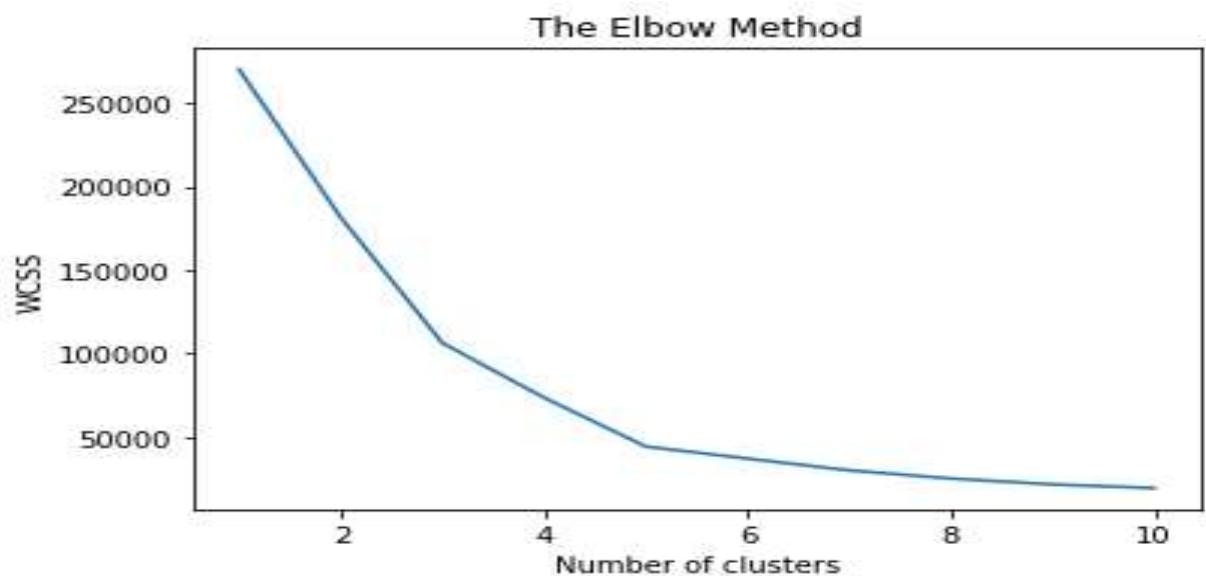
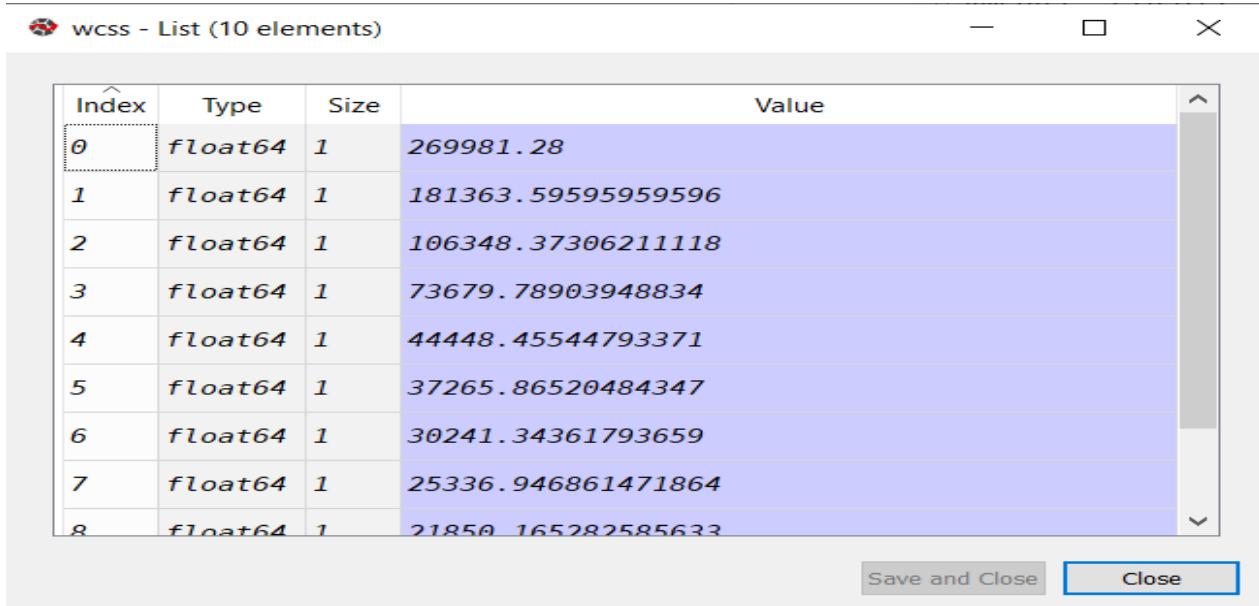


Figure 22.21

From the above plot, we can see the elbow point is at **5**. So the number of clusters here will be **5**.



Index	Type	Size	Value
0	float64	1	269981.28
1	float64	1	181363.59595959596
2	float64	1	106348.37306211118
3	float64	1	73679.78903948834
4	float64	1	44448.45544793371
5	float64	1	37265.86520484347
6	float64	1	30241.34361793659
7	float64	1	25336.946861471864
8	float64	1	21850.165282585633

Figure 22.22

### Step- 3: Training the K-means algorithm on the training dataset

As we have got the number of clusters, so we can now train the model on the dataset.

To train the model, we will use the same two lines of code as we have used in the above section, but here instead of using **i**, we will use **5**, as we know there are **5** clusters that need to be formed. The code is given below:

```
1. # Training the K-Means model on the dataset
2.
3. kmeans = KMeans(n_clusters = 5, init = 'k-means++', random_state = 42)
4. y_kmeans = kmeans.fit_predict(X)
```

Figure 22.23

The first line is the same as above for creating the object of KMeans class.

In the second line of code, we have created the dependent variable **y\_predict** to train the model.

By executing the above lines of code, we will get the **y\_predict** variable. We can check it under **the variable explorer** option in the Spyder IDE. We can now compare the values of **y\_predict** with our original dataset. Consider the below image:



Figure 22.24

From the above image, we can now relate that the CustomerID 1 belongs to a cluster

3(as index starts from 0, hence 2 will be considered as 3), and 2 belongs to cluster 4, and so on.

## Step-4: Visualizing the Clusters

The last step is to visualize the clusters. As we have 5 clusters for our model, so we will visualize each cluster one by one.

To visualize the clusters will use scatter plot using `mtp.scatter()` function of `matplotlib`.

```
1. # Visualising the clusters
2.
3. plt.scatter(X[y_kmeans == 0, 0], X[y_kmeans == 0, 1], s = 100, c = 'red', label = 'Cluster 1')
4. plt.scatter(X[y_kmeans == 1, 0], X[y_kmeans == 1, 1], s = 100, c = 'blue', label = 'Cluster 2')
5. plt.scatter(X[y_kmeans == 2, 0], X[y_kmeans == 2, 1], s = 100, c = 'green', label = 'Cluster 3')
6. plt.scatter(X[y_kmeans == 3, 0], X[y_kmeans == 3, 1], s = 100, c = 'cyan', label = 'Cluster 4')
7. plt.scatter(X[y_kmeans == 4, 0], X[y_kmeans == 4, 1], s = 100, c = 'magenta', label = 'Cluster 5')
8. plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], s = 300, c = 'yellow', label = 'Centroids')
9. plt.title('Clusters of customers')
10. plt.xlabel('Annual Income (k$)')
11. plt.ylabel('Spending Score (1-100)')
12. plt.legend()
13. plt.show()
```

Figure 22.25

In above lines of code, we have written code for each clusters, ranging from 1 to 5. The first coordinate of the `mtp.scatter`, i.e., `x[y_predict == 0, 0]` containing the x value for the showing the matrix of features values, and the `y_predict` is ranging from 0 to 1.

**Output:**



Figure 22.26

The output image is clearly showing the five different clusters with different colors. The clusters are formed between two parameters of the dataset; Annual income of customer and Spending. We can change the colors and labels as per the requirement or choice. We can also observe some points from the above patterns, which are given below:

- **Cluster1** shows the customers with average salary and average spending so we can categorize these customers as **average**.
- Cluster2 shows the customer has a high income but low spending, so we can categorize them as **careful**.
- Cluster3 shows the low income and also low spending so they can be categorized as **sensible**.
- Cluster4 shows the customers with low income with very high spending so they can be categorized as **careless**.
- Cluster5 shows the customers with high income and high spending so they can be categorized as **target**, and these customers can be the most profitable customers for the mall owner.

# CHAPTER 23

---

## Hierarchical Clustering

### 23.1 Overview

Hierarchical clustering is another unsupervised machine learning algorithm, which is used to group the unlabeled datasets into a cluster and also known as hierarchical cluster analysis or HCA.

In this algorithm, we develop the hierarchy of clusters in the form of a tree, and this tree-shaped structure is known as the dendrogram.

Sometimes the results of K-means clustering and hierarchical clustering may look similar, but they both differ depending on how they work. As there is no requirement to predetermine the number of clusters as we did in the K-Means algorithm.

**The hierarchical clustering technique has two approaches:**

**Agglomerative:** Agglomerative is a bottom-up approach, in which the algorithm starts with taking all data points as single clusters and merging them until one cluster is left.

**Divisive:** Divisive algorithm is the reverse of the agglomerative algorithm as it is a top-down approach.

- **Why hierarchical clustering?**

As we already have other clustering algorithms such as K-Means Clustering, then why we need hierarchical clustering? So, as we have seen in the K-means clustering that there are some challenges with this algorithm, which are a predetermined number of clusters, and it always tries to create the clusters of the same size. To solve these two challenges, we can opt for the hierarchical clustering algorithm because, in this algorithm, we don't need to have knowledge about the predefined number of clusters.

In this topic, we will discuss the Agglomerative Hierarchical clustering algorithm.

### **Agglomerative Hierarchical clustering**

The agglomerative hierarchical clustering algorithm is a popular example of HCA. To group the datasets into clusters, it follows the bottom-up approach. It means, this algorithm considers each dataset as a single cluster at the beginning, and then start combining the closest pair of clusters together. It does

this until all the clusters are merged into a single cluster that contains all the datasets.

This hierarchy of clusters is represented in the form of the dendrogram.

- How the Agglomerative Hierarchical clustering Work?

The working of the AHC algorithm can be explained using the below steps:

- **Step-1:** Create each data point as a single cluster. Let's say there are  $N$  data points, so the number of clusters will also be  $N$

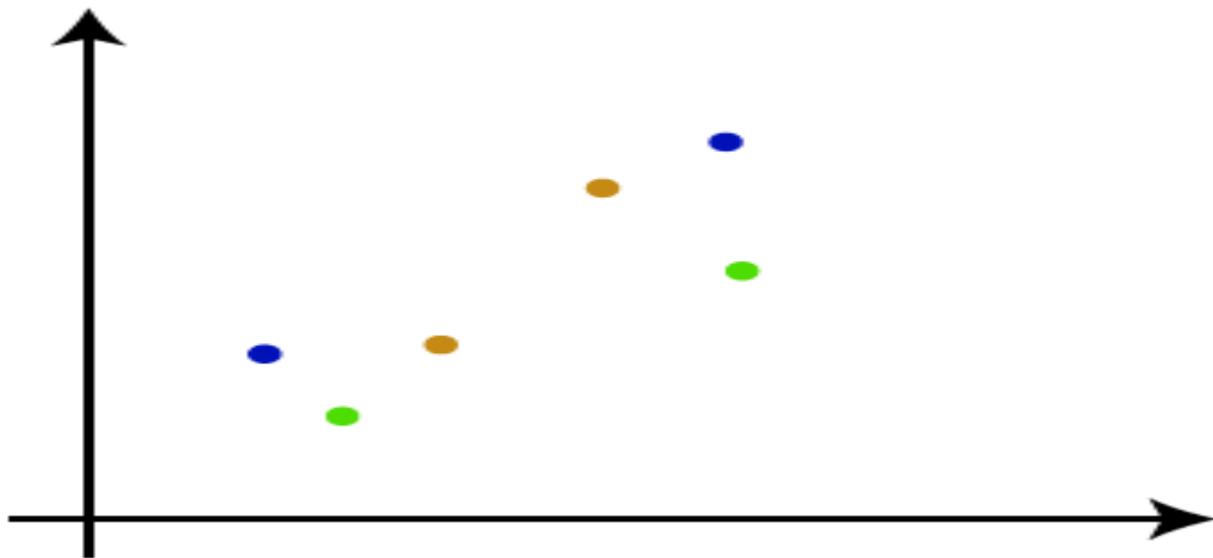


Figure 23.1

**Step-2:** Take two closest data points or clusters and merge them to form one cluster. So, there will now be  $N-1$  clusters.

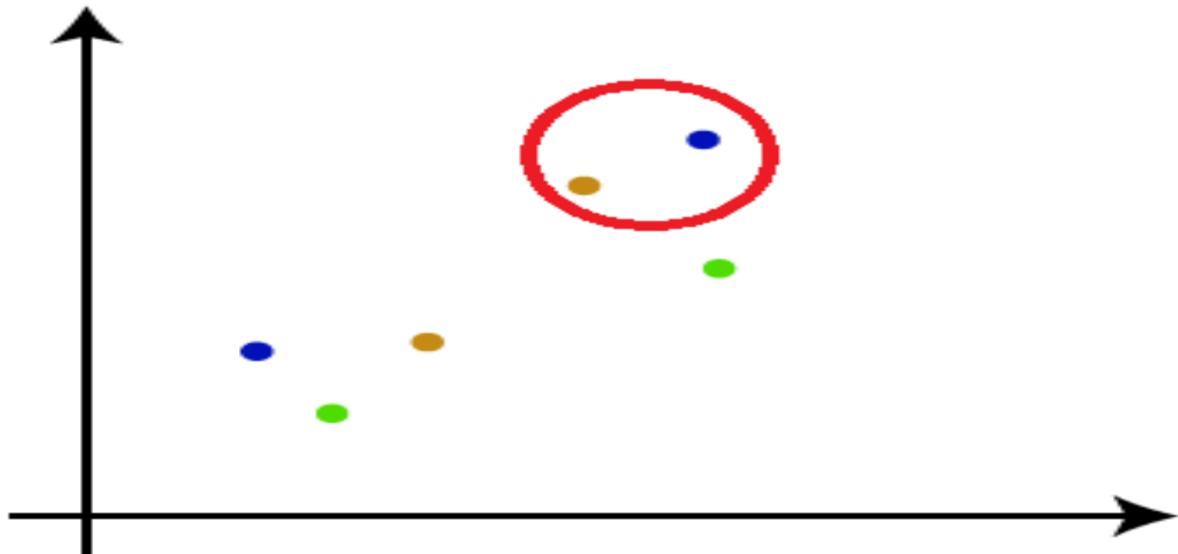


Figure 23.2

**Step-3:** Again, take the two closest clusters and merge them together to form one cluster. There will be  $N-2$  clusters.

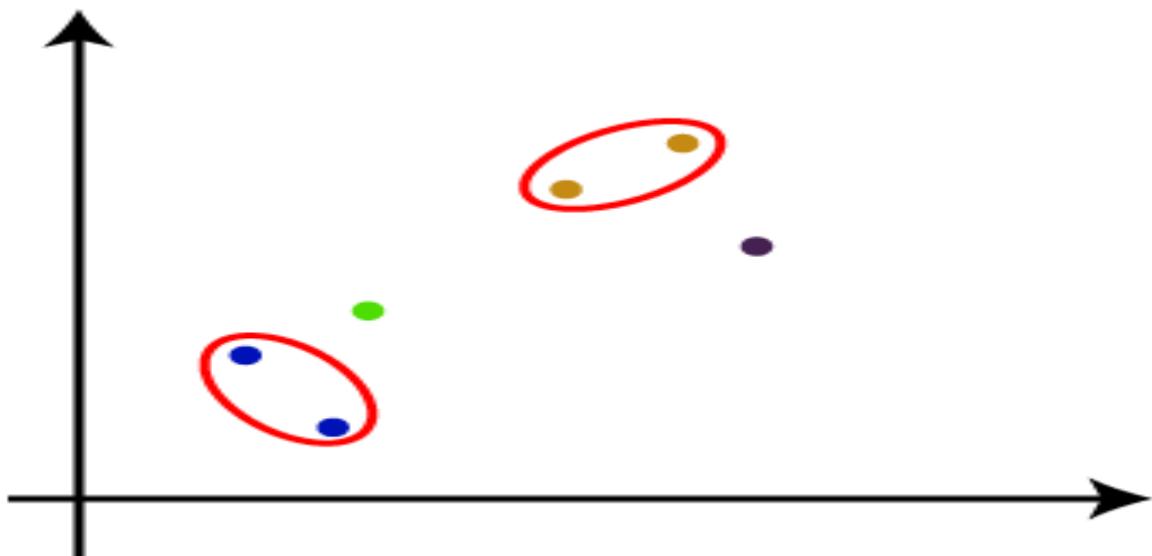
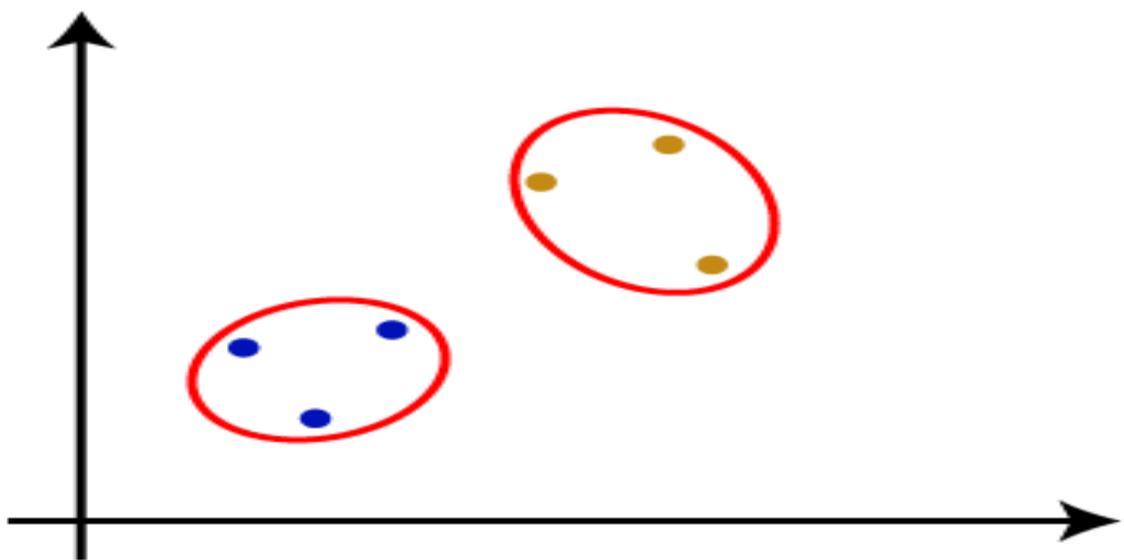
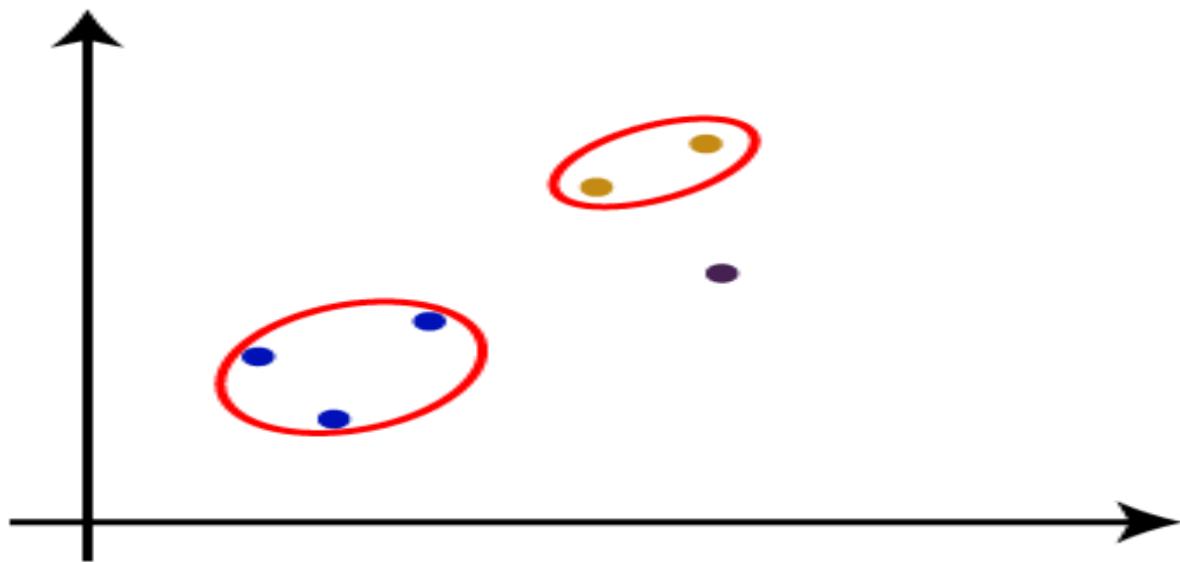


Figure 23.3

**Step-4:** Repeat Step 3 until only one cluster left. So, we will get the following clusters. Consider the below images:



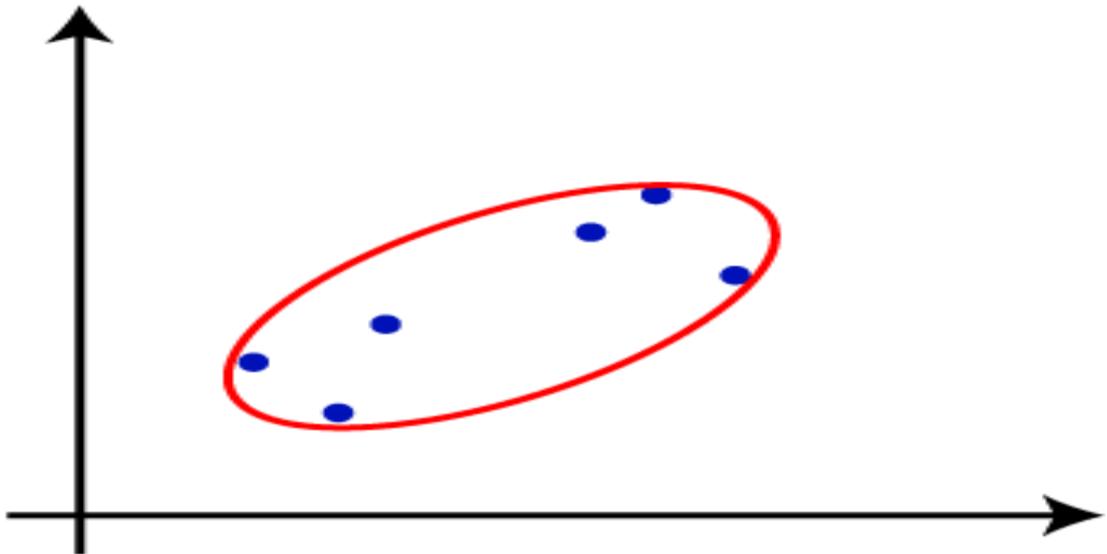


Figure 23.4

- **Step-5:** Once all the clusters are combined into one big cluster, develop the dendrogram to divide the clusters as per the problem.

### Measure for the distance between two clusters

As we have seen, the **closest distance** between the two clusters is crucial for the hierarchical clustering. There are various ways to calculate the distance between two clusters, and these ways decide the rule for clustering. These measures are called **Linkage methods**. Some of the popular linkage methods are given below:

1. **Single Linkage:** It is the Shortest Distance between the closest points of the clusters. Consider the below image:

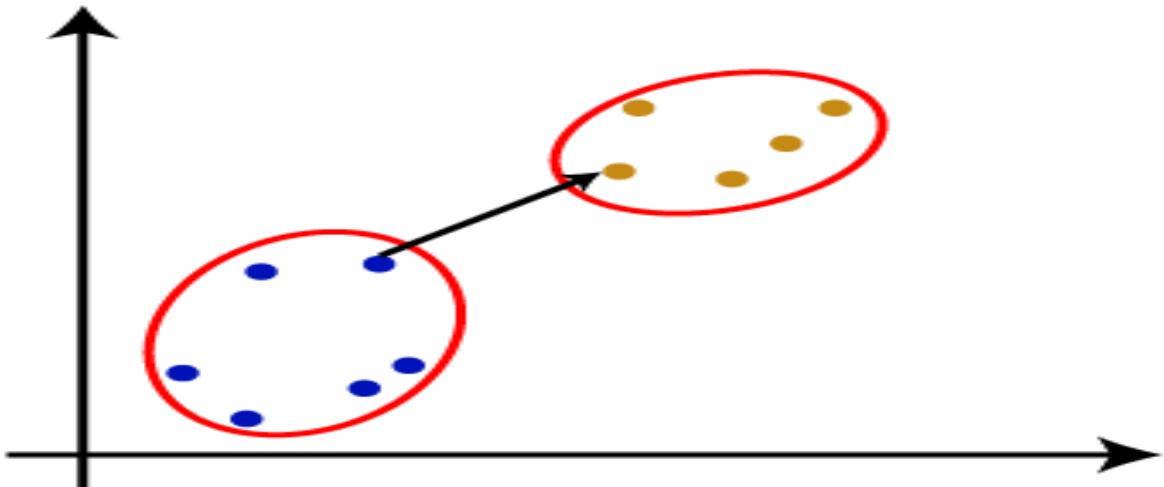


Figure 23.5

**2. Complete Linkage:** It is the farthest distance between the two points of two different clusters. It is one of the popular linkage methods as it forms tighter clusters than single-linkage.

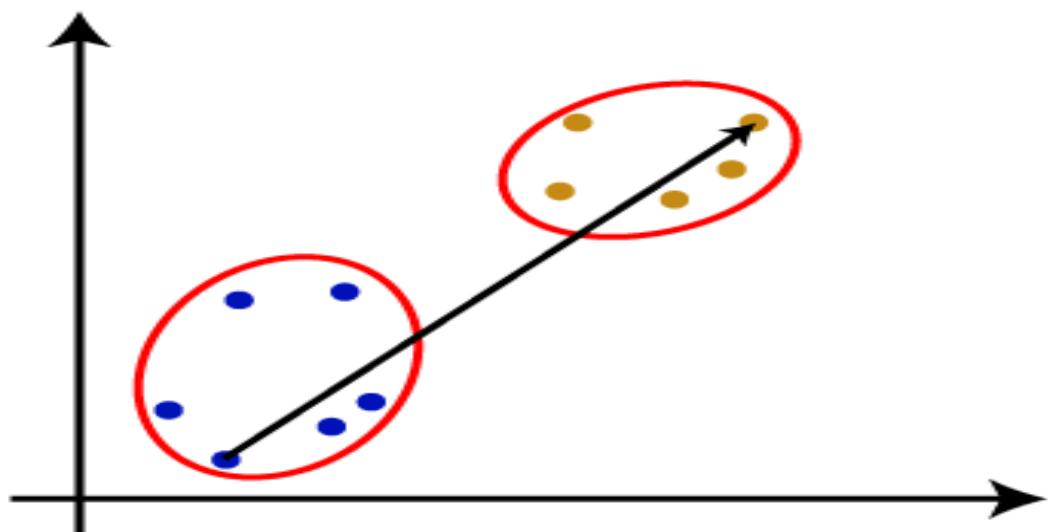


Figure 23.6

**3. Average Linkage:** It is the linkage method in which the distance between each pair of datasets is added up and then divided by the total number of datasets to calculate the average distance between two clusters. It is also one of the most popular linkage methods.

**4. Centroid Linkage:** It is the linkage method in which the distance between the centroid of the clusters is calculated. Consider the below image:

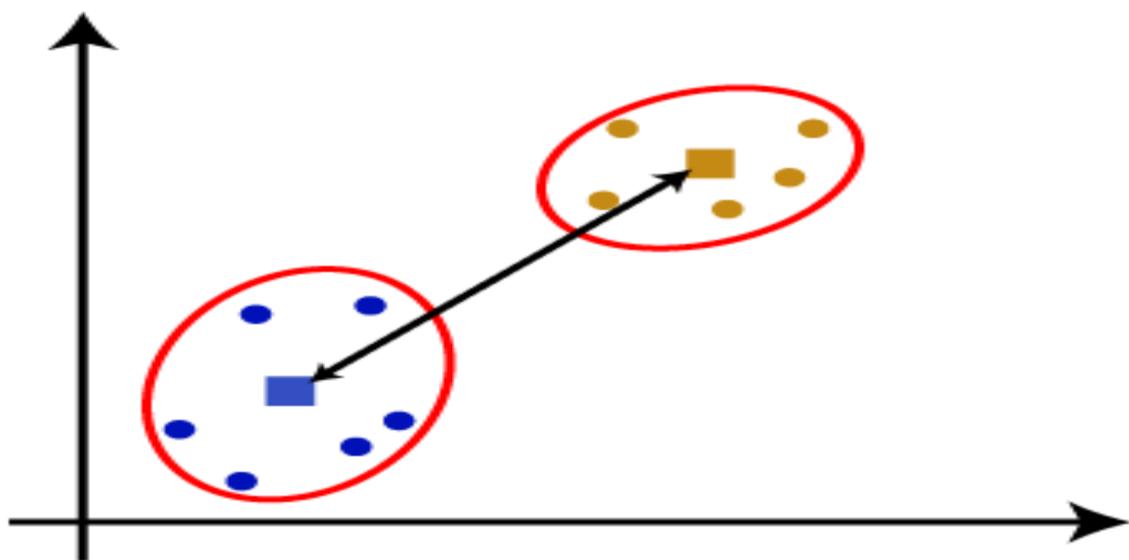


Figure 23.7

From the above-given approaches, we can apply any of them according to the type of problem or business requirement.

### **3. Working of Dendrogram in Hierarchical clustering**

The dendrogram is a tree-like structure that is mainly used to store each step as a memory that the HC algorithm performs. In the dendrogram plot, the Y-axis shows the Euclidean distances between the data points, and the x-axis shows all the data points of the given dataset.

The working of the dendrogram can be explained using the below diagram:

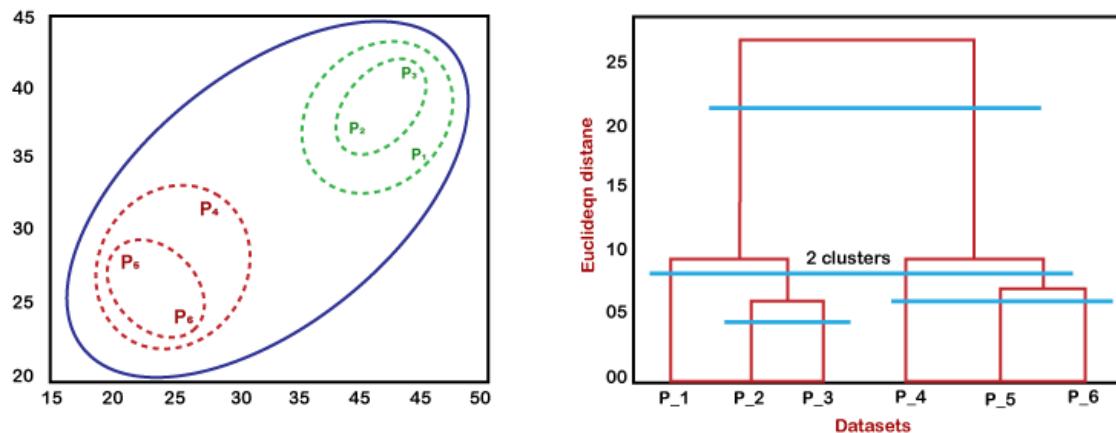


Figure 23.8

In the above diagram, the left part is showing how clusters are created in agglomerative clustering, and the right part is showing the corresponding dendrogram.

- As we have discussed above, firstly, the datapoints P2 and P3 combine together and form a cluster, correspondingly a dendrogram is created, which connects P2 and P3 with a rectangular shape. The height is decided according to the Euclidean distance between the data points.
- In the next step, P5 and P6 form a cluster, and the corresponding dendrogram is created. It is higher than of previous, as the Euclidean distance between P5 and P6 is a little bit greater than the P2 and P3.
- Again, two new dendograms are created that combine P1, P2, and P3 in one dendrogram, and P4, P5, and P6, in another dendrogram.
- At last, the final dendrogram is created that combines all the data points together.

We can cut the dendrogram tree structure at any level as per our requirement.

## Python Implementation of Agglomerative Hierarchical Clustering

Now we will see the practical implementation of the agglomerative hierarchical clustering algorithm using Python. To implement this, we will use the same dataset problem that we have used in the previous topic of K-means clustering so that we can compare both concepts easily.

The dataset is containing the information of customers that have visited a mall for shopping. So, the mall owner wants to find some patterns or some particular behavior of his customers using the dataset information.

Steps for implementation of AHC using Python:

The steps for implementation will be the same as the k-means clustering, except for some changes such as the method to find the number of clusters. Below are the steps:

1. **Data Pre-processing**
2. **Finding the optimal number of clusters using the Dendrogram**
3. **Training the hierarchical clustering model**
4. **Visualizing the clusters**

- **Data Pre-processing Steps:**

In this step, we will import the libraries and datasets for our model.

- **Importing the libraries**

```
◦ # Hierarchical Clustering
◦
◦ # Importing the libraries
◦ import numpy as np
◦ import matplotlib.pyplot as plt
◦ import pandas as pd
◦
◦ # Importing the dataset
◦ dataset = pd.read_csv('Mall_Customers.csv')
```

Figure 23.9

As discussed above, we have imported the same dataset of **Mall\_Customers\_data.csv**, as we did in k-means clustering. Consider the below output:



Index	CustomerID	Genre	Age	Annual Income	Spending Score
0	1	Male	19	15	39
1	2	Male	21	15	81
2	3	Female	20	16	6
3	4	Female	23	16	77
4	5	Female	31	17	40
5	6	Female	22	17	76
6	7	Female	35	18	6
7	8	Female	23	18	94
8	9	Male	64	19	3
9	10	Female	30	19	72
10	11	Male	67	19	14

Figure 23.10

- **Extracting the matrix of features**

Here we will extract only the matrix of features as we don't have any further information about the dependent variable. Code is given below:

```
1. X = dataset.iloc[:, [3, 4]].values
```

Figure 23.11

Output:

	0	1
0	15	39
1	15	81
2	16	6
3	16	77
4	17	40
5	17	76
6	18	6
7	18	94
8	19	3
9	10	72

Figure 23.12

Here we have extracted only 3 and 4 columns as we will use a 2D plot to see the clusters. So, we are considering the Annual income and spending score as the matrix of features.

## Step-2: Finding the optimal number of clusters using the Dendrogram

Now we will find the optimal number of clusters using the Dendrogram for our model. For this, we are going to use **scipy** library as it provides a function that will directly return the dendrogram for our code. Consider the below lines of code:

```
1. # Using the dendrogram to find the optimal number of clusters
2.
3. import scipy.cluster.hierarchy as sch
4. dendrogram = sch.dendrogram(sch.linkage(X, method = 'ward'))
5. plt.title('Dendrogram')
6. plt.xlabel('Customers')
7. plt.ylabel('Euclidean distances')
8. plt.show()
```

Figure 23.13

In the above lines of code, we have imported the **hierarchy** module of **scipy** library. This module provides us a method **shc.dendrogram()**, which takes the **linkage()** as a parameter. The linkage function is used to define the distance between two clusters, so here we have passed the x(matrix of features), and method "ward," the popular method of linkage in hierarchical clustering.

The remaining lines of code are to describe the labels for the dendrogram plot.

### Output:

By executing the above lines of code, we will get the below output:

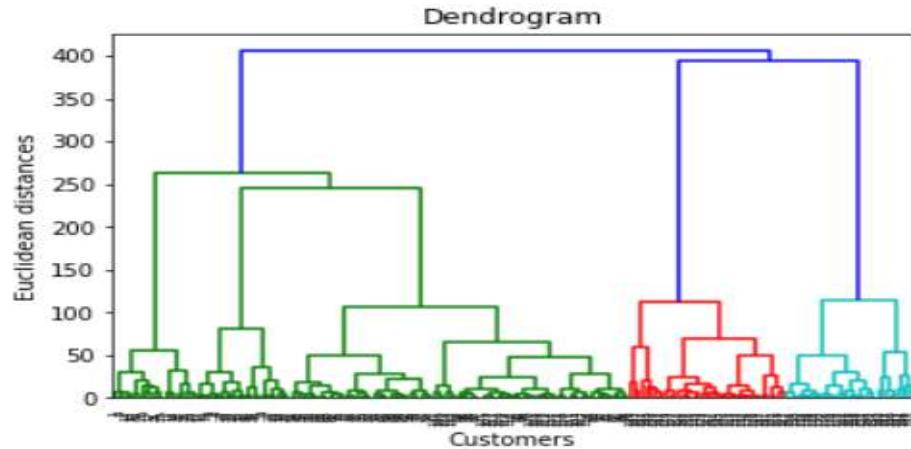


Figure 23.14

Using this Dendrogram, we will now determine the optimal number of clusters for our model. For this, we will find the **maximum vertical distance** that does not cut any horizontal bar. Consider the below diagram:

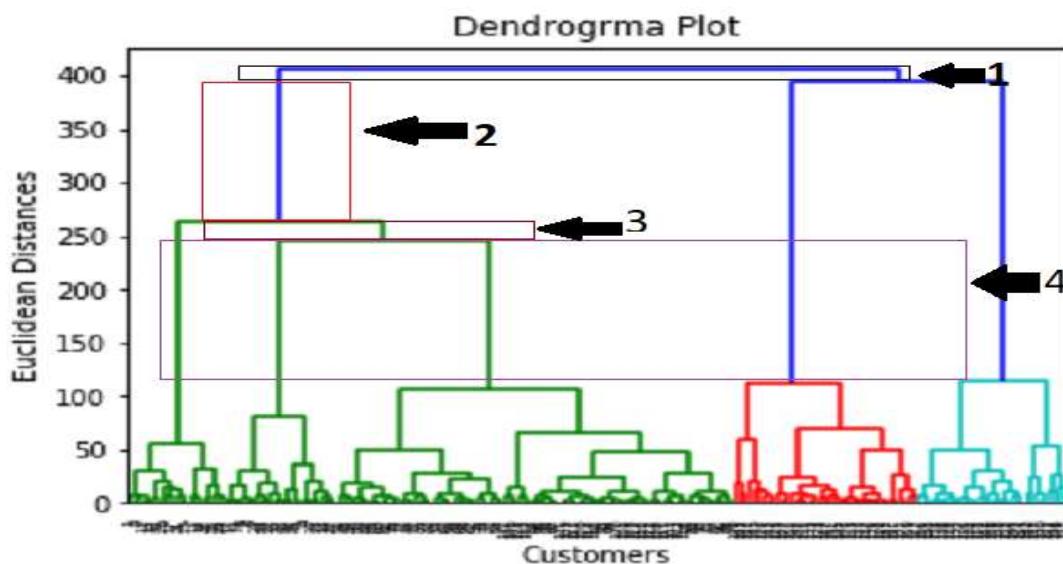


Figure 23.15

In the above diagram, we have shown the vertical distances that are not cutting their horizontal bars. As we can visualize, the 4<sup>th</sup> distance is looking the maximum, so according to this, **the number of clusters will be 5**(the vertical lines in this range). We can also take the 2<sup>nd</sup> number as it approximately equals the 4<sup>th</sup> distance, but we will consider the 5 clusters because the same we calculated in the K-means algorithm.

So, **the optimal number of clusters will be 5**, and we will train the model in the next step, using the same.

### Step-3: Training the hierarchical clustering model

As we know the required optimal number of clusters, we can now train our model. The code is given below:

```
1. # Training the Hierarchical Clustering model on the dataset
2.
3. from sklearn.cluster import AgglomerativeClustering
4. hc = AgglomerativeClustering(n_clusters = 5, affinity = 'euclidean', linkage = 'ward')
5. y_hc = hc.fit_predict(X)
```

Figure 23.16

In the above code, we have imported the **AgglomerativeClustering** class of cluster module of scikit learn library.

Then we have created the object of this class named as **hc**. The AgglomerativeClustering class takes the following parameters:

- **n\_clusters=5**: It defines the number of clusters, and we have taken here 5 because it is the optimal number of clusters.
- **affinity='euclidean'**: It is a metric used to compute the linkage.

- **linkage='ward'**: It defines the linkage criteria, here we have used the "ward" linkage. This method is the popular linkage method that we have already used for creating the Dendrogram. It reduces the variance in each cluster.

In the last line, we have created the dependent variable `y_pred` to fit or train the model. It does train not only the model but also returns the clusters to which each data point belongs.

After executing the above lines of code, if we go through the variable explorer option in our Spyder IDE, we can check the `y_pred` variable. We can compare the original dataset with the `y_pred` variable. Consider the below image:

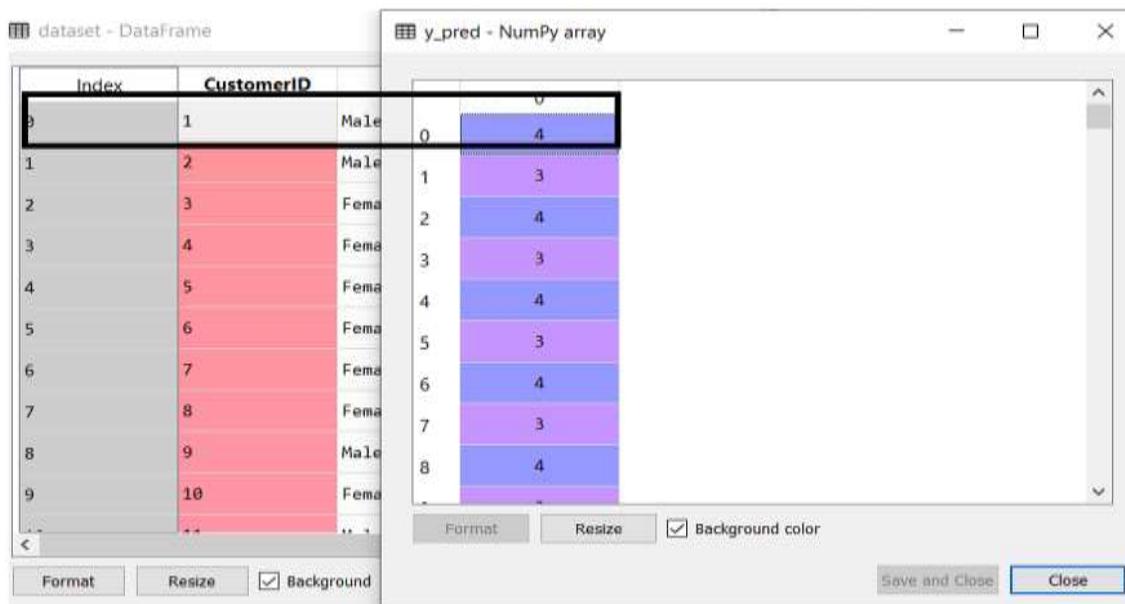


Figure 23.17

As we can see in the above image, the `y_pred` shows the clusters value, which means the customer id 1 belongs to the 5<sup>th</sup> cluster (as indexing starts from 0, so 4 means 5<sup>th</sup> cluster), the customer id 2 belongs to 4<sup>th</sup> cluster, and so on.

## Step-4: Visualizing the clusters

As we have trained our model successfully, now we can visualize the clusters corresponding to the dataset.

Here we will use the same lines of code as we did in k-means clustering, except one change. Here we will not plot the centroid that we did in k-means, because here we have used dendrogram to determine the optimal number of clusters. The code is given below:

```
1. # Visualising the clusters
2.
3. plt.scatter(X[y_hc == 0, 0], X[y_hc == 0, 1], s = 100, c = 'red', label = 'Cluster 1')
4. plt.scatter(X[y_hc == 1, 0], X[y_hc == 1, 1], s = 100, c = 'blue', label = 'Cluster 2')
5. plt.scatter(X[y_hc == 2, 0], X[y_hc == 2, 1], s = 100, c = 'green', label = 'Cluster 3')
6. plt.scatter(X[y_hc == 3, 0], X[y_hc == 3, 1], s = 100, c = 'cyan', label = 'Cluster 4')
7. plt.scatter(X[y_hc == 4, 0], X[y_hc == 4, 1], s = 100, c = 'magenta', label
8.   = 'Cluster 5')
9. plt.title('Clusters of customers')
10. plt.xlabel('Annual Income (k$)')
11. plt.ylabel('Spending Score (1-100)')
12. plt.legend()
13. plt.show()
```

Figure 23.18

Output:

By executing the above lines of code, we will get the below output:

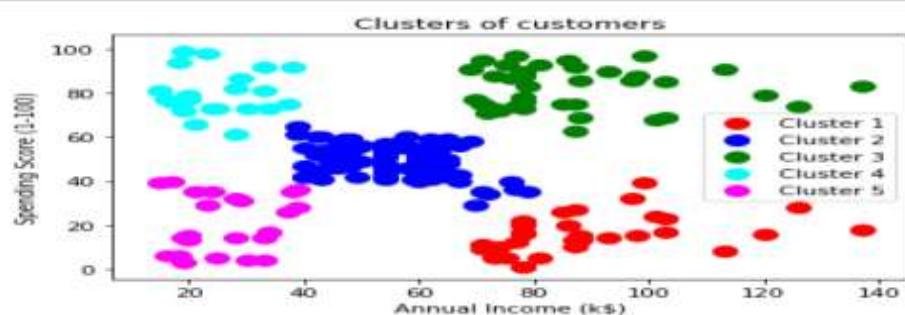


Figure 23.19

---

# CHAPTER 24

---

## Association Rule Learning

### 24.1 Overview

Association rule learning is a type of unsupervised learning technique that checks for the dependency of one data item on another data item and maps accordingly so that it can be more profitable. It tries to find some interesting relations or associations among the variables of dataset. It is based on different rules to discover the interesting relations between variables in the database.

The association rule learning is one of the very important concepts of machine learning, and it is employed in Market Basket analysis, Web usage mining, continuous production, etc. Here market basket analysis is a technique used by the various big retailer to discover the associations between items. We can understand it by taking an example of a supermarket, as in a supermarket, all products that are purchased together are put together.

Association rule learning can be divided into three types of algorithms:

- 1- Apriori
- 2- Eclat

How does Association Rule Learning work?

Association rule learning works on the concept of If and Else Statement, such as if A then B.

- Association Rule Learning

Here the If element is called antecedent, and then statement is called as Consequent. These types of relationships where we can find out some association or relation between two items is known as single cardinality. It is all about creating rules, and if the number of items increases, then cardinality also increases accordingly. So, to measure the associations between thousands of data items, there are several metrics. These metrics are given below:

Support

Confidence

Lift

- Types of Association Rule Learning

Association rule learning can be divided into three algorithms:

### Apriori Algorithm

This algorithm uses frequent datasets to generate association rules. It is designed to work on the databases that contain transactions. This algorithm uses a breadth-first search and Hash Tree to calculate the itemset efficiently.

It is mainly used for market basket analysis and helps to understand the products that can be bought together. It can also be used in the healthcare field to find drug reactions for patients.

### Eclat Algorithm

Eclat algorithm stands for Equivalence Class Transformation. This algorithm uses a depth-first search technique to find frequent itemsets in a transaction database. It performs faster execution than Apriori Algorithm.

---

---

## CHAPTER 25

---

### Apriori

#### 25.1 Overview

The Apriori algorithm uses frequent itemsets to generate association rules, and it is designed to work on the databases that contain transactions. With the help of these association rule, it determines how strongly or how weakly two objects are connected. This algorithm uses a breadth-first search and Hash Tree to calculate the itemset associations efficiently. It is the iterative process for finding the frequent itemsets from the large dataset.

This algorithm was given by the R. Agrawal and Srikant in the year 1994. It is mainly used for market basket analysis and helps to find those products that can be bought together. It can also be used in the healthcare field to find drug reactions for patients.

## 25.2 Steps for Apriori Algorithm

Below are the steps for the apriori algorithm:

**Step-1:** Determine the support of itemsets in the transactional database, and select the minimum support and confidence.

**Step-2:** Take all supports in the transaction with higher support value than the minimum or selected support value.

**Step-3:** Find all the rules of these subsets that have higher confidence value than the threshold or minimum confidence.

**Step-4:** Sort the rules as the decreasing order of lift.

## 25.3 Advantages of Apriori Algorithm

This is easy to understand algorithm

The join and prune steps of the algorithm can be easily implemented on large datasets.

Disadvantages of Apriori Algorithm

The apriori algorithm works slow compared to other algorithms.

The overall performance can be reduced as it scans the database for multiple times.

The time complexity and space complexity of the apriori algorithm is  $O(2D)$ , which is very high. Here D represents the horizontal width present in the database.

## 25.4 Python Implementation of Apriori Algorithm

Now we will see the practical implementation of the Apriori Algorithm. To implement this, we have a problem of a retailer, who wants to find the association between his shop's product, so that he can provide an offer of "Buy this and Get that" to his customers.

The retailer has a dataset information that contains a list of transactions made by his customer. In the dataset, each row shows the products purchased by customers or transactions made by the customer. To solve this problem, we will perform the below steps:

## 25.5 Data Pre-processing

Training the Apriori model on the dataset

Visualizing the results

### 1. Data Pre-processing Step:

The first step is data pre-processing step. Under this, first, we will perform the importing of the libraries. The code for this is given below:

## Importing the libraries:

Before importing the libraries, we will use the below line of code to install the apyori package to use further, as Spyder IDE does not contain it:

```
pip install apyori
```

Figure 25.1

Below is the code to implement the libraries that will be used for different tasks of the model:

```
1. # Apriori
2.
3. # Importing the libraries
4. import numpy as np
5. import matplotlib.pyplot as plt
6. import pandas as pd
```

Figure 25.2

## Output:

dataset - DataFrame

Index	0	1	2
0	shrimp	almonds	avocado
1	burgers	meatballs	eggs
2	chutney	nan	nan
3	turkey	avocado	nan
4	mineral water	milk	energy bar
5	low fat yogurt	nan	nan
6	whole wheat pasta	french fries	nan
7	soup	light cream	shallot
8	frozen vegetables	spaghetti	green tea
9	french fries	nan	nan
10	eggs	pet food	nan

Format Resize  Background  Column min/ Save and Close Close

Figure 25.3

### Importing the dataset:

Now, we will import the dataset for our apriori model. To import the dataset, there will be some changes here. All the rows of the dataset are showing different transactions made by the customers. The first row is the transaction done by the first customer, which means there is no particular name for each column and have their own individual value or product details(See the dataset given below after the code). So, we need to mention here in our code that there is no header specified. The code is given below:

```

1. # Data Preprocessing
2. dataset = pd.read_csv('Market_Basket_Optimisation.csv', header = None)
3. transactions = []
4. for i in range(0, 7501):
5.     transactions.append([str(dataset.values[i,j]) for j in range(0, 20)])

```

Figure 25.4

In the above code, the first line is showing importing the dataset into pandas format. The second line of the code is used because the `apriori()` that we will use for training our model takes the dataset in the format of the list of the transactions. So, we have created an empty list of the transaction. This list will contain all the itemsets from 0 to 7500. Here we have taken 7501 because, in Python, the last index is not considered.

The dataset looks like the below image:

Figure 25.5

## 2. Training the Apriori Model on the dataset

To train the model, we will use the **apriori** function that will be imported from the **apyroi** package. This function will return the **rules** to train the model on the dataset. Consider the below code:

```
1. # Training the Apriori model on the dataset
2. from apyori import apriori
```

```
3. rules = apriori(transactions, min_support =0.003 , min_confidence =0.2 , min_lift =3 ,  
min_length =2 )
```

Figure 25.6

In the above code, the first line is to import the apriori function. In the second line, the apriori function returns the output as the rules. It takes the following parameters:

- **transactions**: A list of transactions.
- **min\_support**= To set the minimum support float value. Here we have used 0.003 that is calculated by taking 3 transactions per customer each week to the total number of transactions.
- **min\_confidence**: To set the minimum confidence value. Here we have taken 0.2. It can be changed as per the business problem.
- **min\_lift**= To set the minimum lift value.
- **min\_length**= It takes the minimum number of products for the association.
- **max\_length** = It takes the maximum number of products for the association.

### 3. Visualizing the result

Now we will visualize the output for our apriori model. Here we will follow some more steps, which are given below:

- **Displaying the result of the rules occurred from the apriori function**

```
1. # Visualising the results
2. results = list(rules)
```

Figure 25.7

By executing the above lines of code, we will get the 9 rules. Consider the below output:

## Output:

Index	Type	Size	Value
0	RelationRecord	3	RelationRecord object of apyori module
1	RelationRecord	3	RelationRecord object of apyori module
2	RelationRecord	3	RelationRecord object of apyori module
3	RelationRecord	3	RelationRecord object of apyori module
4	RelationRecord	3	RelationRecord object of apyori module
5	RelationRecord	3	RelationRecord object of apyori module
6	RelationRecord	3	RelationRecord object of apyori module
7	RelationRecord	3	RelationRecord object of apyori module
8	RelationRecord	3	RelationRecord object of apyori module
9	RelationRecord	3	RelationRecord object of apyori module
10	RelationRecord	3	RelationRecord object of apyori module
11	RelationRecord	3	RelationRecord object of apyori module
12	RelationRecord	3	RelationRecord object of apyori module
...	...	...	...

Figure 25.8

As we can see, the above output is in the form that is not easily understandable. So, we will print all the rules in a suitable format.

- Visualizing the rule, support, confidence, lift in more clear way:

```

1. for item in results:
2.     pair = item[0]
3.     items = [x for x in pair]
4.     print("Rule: " + items[0] + " -> " + items[1])
5.
6.     print("Support: " + str(item[1]))
7.     print("Confidence: " + str(item[2][0][2]))
8.     print("Lift: " + str(item[2][0][3]))
9.     print("=====")

```

Figure 25.9

## Output:

By executing the above lines of code, we will get the below output:



```

Confidence: 0.28380751879699247
Lift: 3.0825089038385434
=====
Rule: milk -> soup
Support: 0.0030662578322890282
Confidence: 0.21904761904761905
Lift: 4.335293378565146
=====
Rule: milk -> whole wheat pasta
Support: 0.003999466737768298
Confidence: 0.4545454545454546
Lift: 3.5077628133183696
=====
Rule: olive oil -> soup
Support: 0.005199306759098787
Confidence: 0.22543352601156072
Lift: 3.4230301186492245
=====
Rule: olive oil -> whole wheat pasta
Support: 0.0038661511798426876
Confidence: 0.4027777777777778
Lift: 6.115862573099416
=====
Rule: nan -> olive oil
Support: 0.007998933475536596
Confidence: 0.2714932126696833
Lift: 4.122410097642296
=====
Rule: nan -> pasta
Support: 0.005065991201173177
Confidence: 0.3220338983050847
Lift: 4.506672147735896
=====
```

Activate Windows  
Go to Settings to activate Windows.

Figure 25.10

# CHAPTER 26

## Eclat

### 26.1 Overview

Eclat Algorithm is abbreviated as Equivalence class clustering and bottom up lattice transversal algorithm. It is an algorithm for finding frequent item sets in a transaction or database. It is one of the best methods of Association Rule Learning. Which means Eclat algorithm is used to generate frequent item sets in a database.

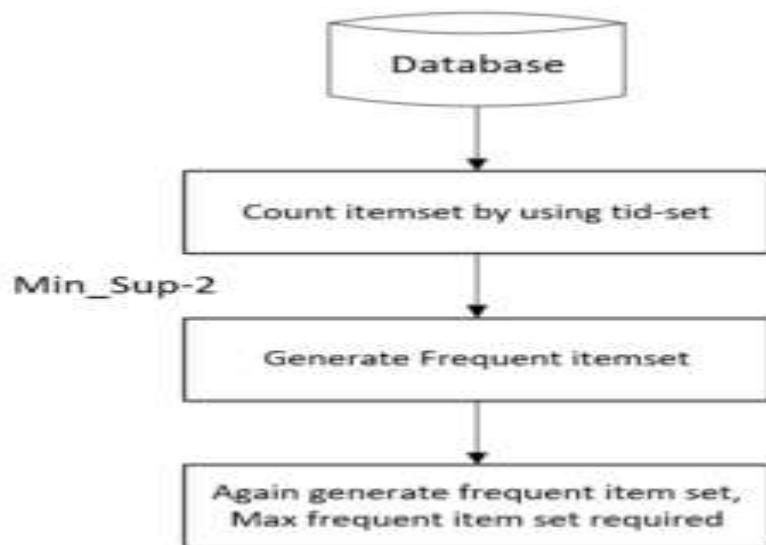


Figure 26.1

Eclat algorithm uses a Depth first search for discovering frequent item sets, whereas Apriori algorithm uses breadth first search. It represents the data in vertical manner unlike Apriori algorithm which represents data in horizontal pattern. This vertical pattern of Eclat algorithm making it into faster algorithm compared to Apriori algorithm. Hence, Eclat algorithm is more efficient and scalable version of the Association Rule Learning.

Generally, Transaction Id sets which are also called as tidsets are used to calculate the value of Support value of a dataset and also avoiding the generation of subsets which does not exist in the prefix tree. In the first call of function, all single items or data are used along with their respective tidsets. Then the function is called recursive, in each recursive call, each item in tidsets pair is verified and combined with other item in tidsets pairs. This process is repeated until no candidate item in tidsets pairs can be combined.

The input given to this Eclat algorithm is a transaction database and a threshold value which is in the range of 0 to 100.

A transaction database is a set of transaction values where each transaction is a set of items. It is important to note that an item should not be appear more than once in the same transaction and also the items are assumed to be sorted by lexicographical order in a transaction.

Each frequent itemset is marked with its corresponding support value. The support of an itemset is given by number of times the itemset appears in the transaction database.

The given transaction data should be a Boolean matrix where for each cell (i, j), the value denotes that whether the

$j^{\text{th}}$  item is included in the  $i^{\text{th}}$  transaction or not. Here, 1 means true and 0 means false.

Now, we have to call the function for the first time and arrange each item with its tidset in a tabular column. We have to call this function iteratively till no more item-tidset pairs can be combined.

An example for Eclat Algorithm is given below :

- Both Apriori and FP-growth use **horizontal data format**
- Alternatively data can also be represented in **vertical format**

TID	Items
1	Bread,Butter,Jam
2	Butter,Coke
3	Butter,Milk
4	Bread,Butter,Coke
5	Bread,Milk
6	Butter,Milk
7	Bread,Milk
8	Bread,Butter,Milk,Jam
9	Bread,Butter,Milk

Item Set	TID set
Bread	1,4,5,7,8,9
Butter	1,2,3,4,6,8,9
Milk	3,5,6,7,8,9
Coke	2,4
Jam	1,8

Figure 26.2

Finally, we can determine the frequent items or data in the transaction sets or database by using this Eclat algorithm.

## Advantages

- 
1. Since the Eclat algorithm uses a Depth-First Search approach, it consumes less memory than the Apriori algorithm.
  2. The Eclat algorithm is naturally faster compared to the Apriori algorithm.
  3. The Eclat algorithm does not involve in the repeated scanning of the data in order to calculate the individual support values.
  4. This algorithm is better suited for small and medium datasets where as Apriori algorithm is used for large datasets.
  5. Eclat algorithm scans the currently generated dataset unlike Apriori which scans the original dataset.

## **Disadvantages**

Intermediate Tidsets which are created in Eclat algorithm consumes more space in memory.

---

---

---

# CHAPTER 27

---

## Reinforcement Learning

### 27.1 Overview

Reinforcement Learning is a feedback-based Machine learning technique in which an agent learns to behave in an environment by performing the actions and seeing the results of actions. For each good action, the agent gets positive feedback, and for each bad action, the agent gets negative feedback or penalty.

- In Reinforcement Learning, the agent learns automatically using feedbacks without any labeled data, unlike supervised learning.
- Since there is no labeled data, so the agent is bound to learn by its experience only.
- RL solves a specific type of problem where decision making is sequential, and the goal is long-term, such as **game-playing, robotics**, etc.
- The agent interacts with the environment and explores it by itself. The primary goal of an agent in reinforcement learning is to improve the performance by getting the maximum positive rewards.

- The agent learns with the process of hit and trial, and based on the experience, it learns to perform the task in a better way. Hence, we can say that "Reinforcement learning is a type of machine learning method where an intelligent agent (computer program) interacts with the environment and learns to act within that." How a Robotic dog learns the movement of his arms is an example of Reinforcement learning.

## 27.2 Terms used in Reinforcement Learning

- **Agent()**: An entity that can perceive/explore the environment and act upon it.
- **Environment()**: A situation in which an agent is present or surrounded by. In RL, we assume the stochastic environment, which means it is random in nature.
- **Action()**: Actions are the moves taken by an agent within the environment.
- **State()**: State is a situation returned by the environment after each action taken by the agent.
- **Reward()**: A feedback returned to the agent from the environment to evaluate the action of the agent.
- **Policy()**: Policy is a strategy applied by the agent for the next action based on the current state.

- Value(): It is expected long-term return with the discount factor and opposite to the short-term reward.
- Q-value(): It is mostly similar to the value, but it takes one additional parameter as a current action (a).

## 27.3 Reinforcement Learning Applications



Figure 27.1

1- Robotics:

---

RL is used in Robot navigation, Robo-soccer, walking, juggling, etc.

2-Control:

RL can be used for adaptive control such as Factory processes, admission control in telecommunication, and Helicopter pilot is an example of reinforcement learning.

3-Game Playing:

RL can be used in Game playing such as tic-tac-toe, chess, etc.

4-Chemistry:

RL can be used for optimizing the chemical reactions.

5-Business:

RL is now used for business strategy planning.

6-Manufacturing:

In various automobile manufacturing companies, the robots use deep reinforcement learning to pick goods and put them in some containers.

7-Finance Sector:

The RL is currently used in the finance sector for evaluating trading strategies.

---

---

---

## CHAPTER 28

---

# Upper Confidence Bound (UCB)

### 28.1 Overview

You can find the best option by using an A/B test. Where you can show a different ad to a user and take his feedback, that is whether the user clicks or not. And summing up all the feedback, you can find the best ad to display. But wait! There is a problem. In A/B test you are incurring more cost and time by doing exploration and exploitation separately. So we need to combine exploration and exploitation and get the result as soon as possible. We do not have enough time and money to actually do this exploitation, so we will do this during the campaign.

The Upper Confidence Bound algorithm is the kind of algorithm that helps us to perform exploitation and exploration together.

### How the UCB Algorithm Works?

At the start of the campaign, we don't know what is the best arm or ad. So we cannot distinguish or discriminate any arm or ad. So the UCB algorithm assumes they all have the same observed average value. Then the algorithm creates confidence bound for each arm or ad. So it randomly picks any of the arms or ads. Then two things can happen- the user clicks the ad or the arm

gives reward or does not. Let's say the ad did not have a click or the arm was a failure. So the observed average of the ad or arm will go down. And the confidence bound will also go down. If it had a click, the observed average would go up and the confidence bound also goes up. By exploiting the best one we are decreasing the confidence bound. As we are adding more and more samples, the probability of other arms or ad doing well is also going up. This is the fundamental concept of Upper Confidence Bound algorithm.

## 28.2 Implementing Upper Confidence Bound (UCB):

We will implement the whole algorithm in Python. First of all, we need to import some essential libraries.

```
1. # Upper Confidence Bound (UCB)
2.
3. # Importing the libraries
4. import numpy as np
5. import matplotlib.pyplot as plt
6. import pandas as pd
```

Figure 28.1

Now, let's import the dataset :

```
1. # Importing the dataset
2. dataset = pd.read_csv('Ads_CTR_Optimisation.csv')
```

Figure 28.2

Output :

Index	Ad 1	Ad 2	Ad 3	Ad 4	Ad 5	Ad 6	Ad 7
0	1	0	0	0	1	0	0
1	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0
3	0	1	0	0	0	0	0
4	0	0	0	0	0	0	0
5	1	1	0	0	0	0	0
6	0	0	0	1	0	0	0
7	1	1	0	0	1	0	0
8	0	0	0	0	0	0	0
9	0	0	1	0	0	0	0
10	0	0	0	0	0	0	0

Figure 28.3

Now, implement the UCB algorithm:

```

1. # Implementing UCB
2. import math
3. N = 10000
4. d = 10
5. ads_selected = []
6. numbers_of_selections = [0] * d
7. sums_of_rewards = [0] * d
8. total_reward = 0
9. for n in range(0, N):
10.     ad = 0
11.     max_upper_bound = 0
12.     for i in range(0, d):
13.         if (numbers_of_selections[i] > 0):
14.             average_reward = sums_of_rewards[i] / numbers_of_selections[i]
15.             delta_i = math.sqrt(3/2 * math.log(n + 1) / numbers_of_selections[i])
16.             upper_bound = average_reward + delta_i
17.         else:
18.             upper_bound = 1e400
19.         if upper_bound > max_upper_bound:
20.             max_upper_bound = upper_bound
21.             ad = i
22.     ads_selected.append(ad)
23.     numbers_of_selections[ad] = numbers_of_selections[ad] + 1
24.     reward = dataset.values[n, ad]
25.     sums_of_rewards[ad] = sums_of_rewards[ad] + reward
26.     total_reward = total_reward + reward

```

Figure 28.4

**Code Explanation:**

Here, three vectors are used.

Index	Type	Size	Value
0	int	1	0
1	int	1	1
2	int	1	2
3	int	1	3
4	int	1	4
5	int	1	5
6	int	1	6
7	int	1	7
8	int	1	8

Index	Type	Size	Value
0	int	1	705
1	int	1	387
2	int	1	186
3	int	1	345
4	int	1	6323
5	int	1	150
6	int	1	292
7	int	1	1170
8	int	1	256

Index	Type	Size	Value
0	int64	1	120
1	int64	1	47
2	int64	1	7
3	int64	1	38
4	int64	1	1675
5	int64	1	1
6	int64	1	27
7	int64	1	236
8	int64	1	20

Figure 28.5

Visualizing the Result :

Now, it's time to see what we get from our model. For this, we will visualize the output.

```
1. # Visualising the results
2.
3. plt.hist(ads_selected)
4. plt.title('Histogram of ads selections')
5. plt.xlabel('Ads')
6. plt.ylabel('Number of times each ad was selected')
7. plt.show()
```

Figure 28.6

## Output:

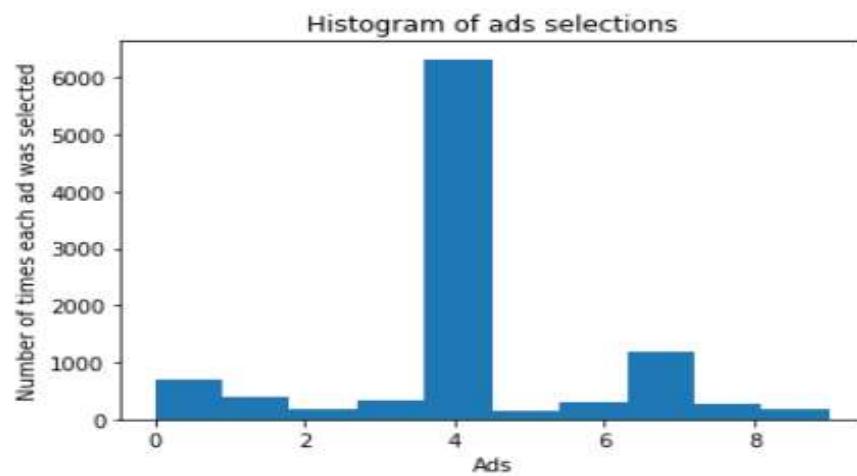


Figure 28.7

From the above visualization, we can see that the fifth ad got the highest click. So our model advice us to place the fourth version of the ad to the user for getting the highest number of clicks!

---

# CHAPTER 29

---

## Thompson Sampling

### 29.1 Overview

Thompson Sampling is an algorithm that follows exploration and exploitation to maximize the cumulative rewards obtained by performing an action. Thompson Sampling is also sometimes referred to as Posterior Sampling or Probability Matching.

An action is performed multiple times which is called exploration and based on the results obtained from the actions, either rewards or penalties, further actions are performed with the goal to maximize the reward which is called exploitation. In other words, new choices are explored to maximize rewards while exploiting the already explored choices.

### 29.2 Implementing Thompson Sampling With Python:

Let us consider 5 bandits or slot machines. Initially, we do not have any data on how the probability of success is distributed among the machines. So we start by exploring the machines one by one.

1. Data Pre-processing
2. Implementing Thompson Sampling

### 3. Visualizing the clusters

#### 1. Data Pre-processing Steps:

In this step, we will import the libraries and datasets for our model.

- Importing the libraries

```
• # Thompson Sampling
•
• # Importing the libraries
• import numpy as np
• import matplotlib.pyplot as plt
• import pandas as pd
```

Figure 29.1

- Importing the dataset

```
1. # Importing the dataset
2. dataset = pd.read_csv('Ads_CTR_Optimisation.csv')
3.
```

Figure 29.2

#### Output:

dataset - DataFrame

Index	Ad 1	Ad 2	Ad 3	Ad 4	Ad 5	Ad 6	Ad 7
0	1	0	0	0	1	0	0
1	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0
3	0	1	0	0	0	0	0
4	0	0	0	0	0	0	0
5	1	1	0	0	0	0	0
6	0	0	0	1	0	0	0
7	1	1	0	0	1	0	0
8	0	0	0	0	0	0	0
9	0	0	1	0	0	0	0
10	0	0	0	0	0	0	0

Format Resize  Background  Column min/ Save and Close Close

Figure 29.3

## 2-Implementing Thompson Sampling

```

1. # Implementing Thompson Sampling
2. import random
3. N = 10000
4. d = 10
5. ads_selected = []
6. numbers_of_rewards_1 = [0] * d
7. numbers_of_rewards_0 = [0] * d
8. total_reward = 0
9. for n in range(0, N):
10.     ad = 0
11.     max_random = 0
12.     for i in range(0, d):
13.         random_beta = random.betavariate(numbers_of_rewards_1[i] + 1, numbers_of_rewards_0[i] + 1)
14.         if random_beta > max_random:
15.             max_random = random_beta
16.             ad = i
17.     ads_selected.append(ad)
18.     reward = dataset.values[n, ad]
19.     if reward == 1:
20.         numbers_of_rewards_1[ad] = numbers_of_rewards_1[ad] + 1
21.     else:
22.         numbers_of_rewards_0[ad] = numbers_of_rewards_0[ad] + 1
23.     total_reward = total_reward + reward

```

Figure 29.4

## Output :

numbers\_of\_rewards\_0 - List (10 elements)

Index	Type	Size	Value
0	int	1	67
1	int	1	64
2	int	1	24
3	int	1	52
4	int	1	6862
5	int	1	33
6	int	1	37
7	int	1	167
8	int	1	53

Save and Close Close

numbers\_of\_rewards\_1 - List (10 elements)

Index	Type	Size	Value
0	int	1	10
1	int	1	8
2	int	1	0
3	int	1	5
4	int	1	2542
5	int	1	1
6	int	1	2
7	int	1	41
8	int	1	6

Save and Close Close

Name	Type	Size	Value
N	int	1	10000
ad	int	1	4
ads_selected	list	10000	[2, 3, 2, 6, 4, 1, 1, 7, 1, 9, ...]
d	int	1	10
dataset	DataFrame	(10000, 10)	Column names: Ad 1, Ad 2, Ad 3, Ad 4, Ad 5, Ad 6, Ad 7, Ad 8, Ad 9, Ad ...
i	int	1	9
max_random	float	1	0.27467975548429313
n	int	1	9999
numbers_of_rewards_0	list	10	[67, 64, 24, 52, 6862, 33, 37, 167, 53, 26]
numbers_of_rewards_1	list	10	[10, 8, 0, 5, 2542, 1, 2, 41, 6, 0]
random_beta	float	1	0.01962150414202515
reward	int64	1	0
total_reward	int64	1	2615

Figure 29.5

### 3-Visualizing the result – Histogram

```
1. # Visualising the results - Histogram
2.
3. plt.hist(ads_selected)
4. plt.title('Histogram of ads selections')
5. plt.xlabel('Ads')
6. plt.ylabel('Number of times each ad was selected')
7. plt.show()
```

Figure 29.6

### Output:

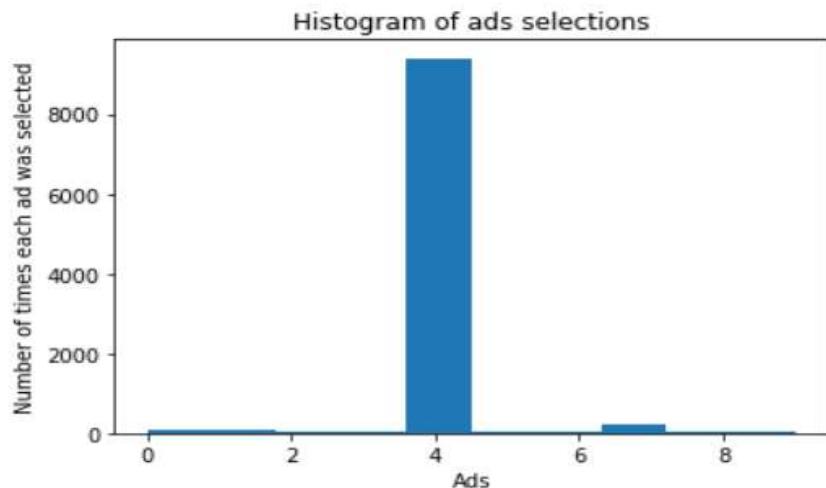


Figure 29.7

---

# CHAPTER 30

---

## Natural Language Processing

### 30.1 Overview

Natural Language Processing (or NLP), which is a part of Computer Science, Human language, and Artificial Intelligence. It is the technology that is used by machines to understand, analyse, manipulate, and interpret human's languages. It helps developers to organize knowledge for performing tasks such as translation, automatic summarization, Named Entity Recognition (NER), speech recognition, relationship extraction, and topic segmentation.

You can also use NLP on a text review to predict if the review is a good one or a bad one. You can use NLP on an article to predict some categories of the articles you are trying to segment. You can use NLP on a book to predict the genre of the book. And it can go further.

You can use NLP to build a machine translator or a speech recognition system, and in that last example you use classification algorithms to classify language. Speaking of classification algorithms, most of NLP algorithms are

classification models, and they include Logistic Regression, Naive Bayes, CART which is a model based on decision trees, Maximum Entropy again related to Decision Trees, Hidden Markov Models which are models based on Markov processes.

## 30.2 Main NLP library example

we will talk about natural language processing (NLP) using Python. This NLP tutorial will use Python NLTK library. NLTK is a popular Python library which is used for NLP.

### 30.2.1 Natural Language Toolkit

NLTK is a leading platform for building Python programs to work with human language data. It provides easy-to-use interfaces to over 50 corpora and lexical resources such as WordNet, along with a suite of text processing libraries for classification, tokenization, stemming, tagging, parsing, and semantic reasoning, wrappers for industrial-strength NLP libraries, and an active discussion forum.

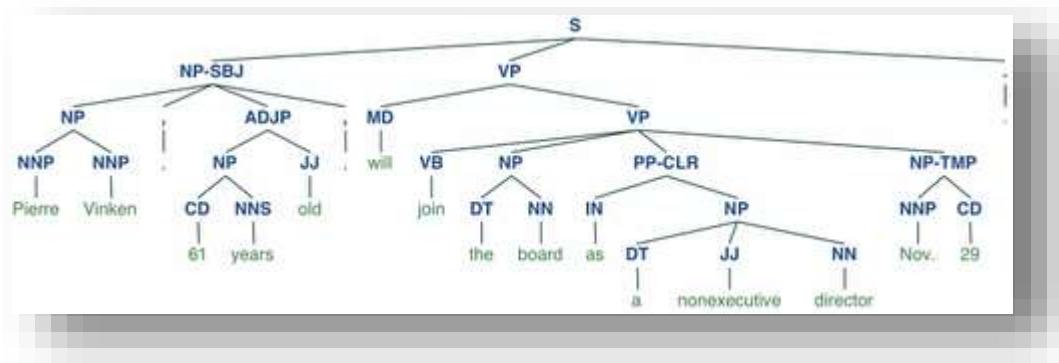


Figure 30.1

### 30.2.2 SpaCy

### 30.2.3 Stanford NLP

#### 30.2.4 OpenNLP

### 30.3 Natural Language Processing in Python

#### 30.3.1 Import libraries and dataset

```
1. # Natural Language Processing
2.
3. # Importing the libraries
4. import numpy as np
5. import matplotlib.pyplot as plt
6. import pandas as pd
7.
8. # Importing the dataset
9. dataset = pd.read_csv('Restaurant_Reviews.tsv', delimiter = '\t', quoting = 3)
```

Figure 30.2

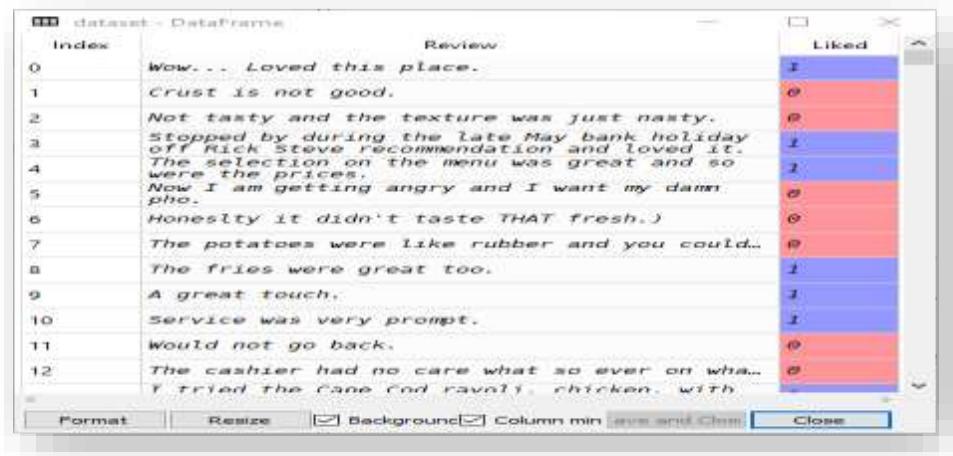


Figure 30.3

#### 30.3.2 Cleaning the texts

```

1. # Cleaning the texts
2. import re
3. import nltk
4. nltk.download('stopwords')
5. from nltk.corpus import stopwords
6. from nltk.stem.porter import PorterStemmer
7. corpus = []
8. for i in range(0, 1000):
9.     review = re.sub('[^a-zA-Z]', ' ', dataset['Review'][i])
10.    review = review.lower()
11.    review = review.split()

```

Figure 30.4

review - List (29 elements)

Index	Type	Size	Value
0	str	1	then
1	str	1	as
2	str	1	if
3	str	1	i
4	str	1	hadn
5	str	1	t
6	str	1	wasted
7	str	1	enough
8	str	1	of
9	str	1	my
10	str	1	life

Save and Close Close

Figure 30.5

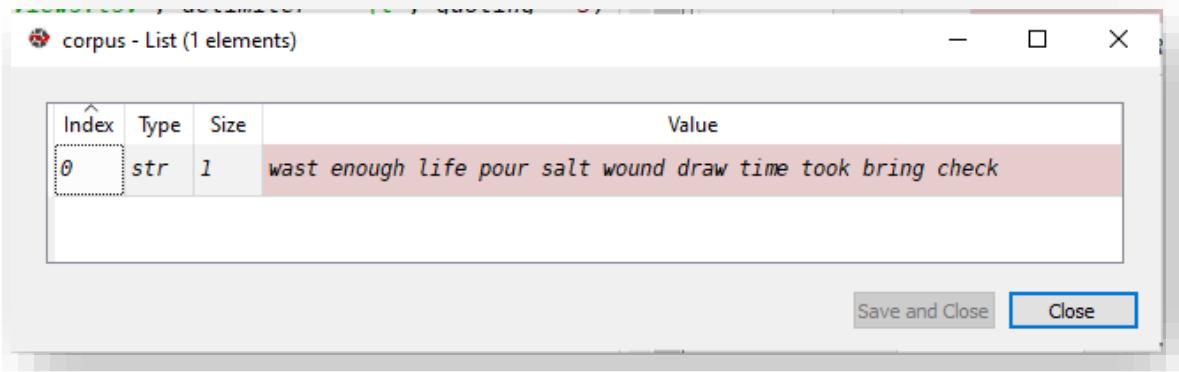
```

1. ps = PorterStemmer()
2. review = [ps.stem(word) for word in review if not word in set(stopwords.words('english'))]

```

```
3. review = ' '.join(review)
4. corpus.append(review)
```

Figure 30.6



Index	Type	Size	Value
0	str	1	wast enough life pour salt wound draw time took bring check

Figure 30.7

### 30.3.2 Creating the Bag of Words model

```
1. # Creating the Bag of Words model
2. from sklearn.feature_extraction.text import CountVectorizer
3. cv = CountVectorizer(max_features = 1500)
4. X = cv.fit_transform(corpus).toarray()
5. y = dataset.iloc[:, 1].values
```

Figure 30.8

corpus - List (1000 elements)

Index	Type	Size	Value
0	str	1	wow love place
1	str	1	crust good
2	str	1	tasti textur nasti
3	str	1	stop late may bank holiday rick steve recommend love
4	str	1	select menu great price
5	str	1	get angri want damn pho
6	str	1	honeslti tast fresh
7	str	1	potato like rubber could tell made ahead time kept warmer
8	str	1	fri great
9	str	1	great touch
10	str	1	servic prompt

Save and Close Close

Figure 30.9

### 30.3.2 Splitting the dataset into the Training set and Test set

```
1. # Splitting the dataset into the Training set and Test set
2. from sklearn.model_selection import train_test_split
3. X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.20, random_state = 0)
```

Figure 30.10

### 30.3.3 Training the Naive Bayes model on the Training set

```
1. from sklearn.naive_bayes import GaussianNB
2. classifier = GaussianNB()
3. classifier.fit(X_train, y_train)
```

Figure 30.11

### 30.3.4 Predicting the Test set results

```
1. # Predicting the Test set results
2. y_pred = classifier.predict(X_test)
```

Figure 30.12



The image shows two side-by-side data grid windows. Both windows are titled 'y\_pred - NumPy array' and 'y\_test - NumPy array'. Each window has a vertical list of indices from 6 to 18 on the left and a horizontal list of indices from 0 to 8 at the top. The cells in the grids are colored blue for 0 and red for 1. In the 'y\_pred' window, the values are: (6,0) blue, (6,1) red, (7,0) red, (7,1) blue, (8,0) blue, (8,1) red, (9,0) red, (9,1) blue, (10,0) red, (10,1) blue, (11,0) blue, (11,1) red, (12,0) blue, (12,1) red, (13,0) red, (13,1) blue, (14,0) blue, (14,1) red, (15,0) red, (15,1) blue, (16,0) blue, (16,1) red, (17,0) red, (17,1) blue, (18,0) red. In the 'y\_test' window, the values are: (6,0) blue, (6,1) red, (7,0) red, (7,1) blue, (8,0) red, (8,1) blue, (9,0) blue, (9,1) red, (10,0) blue, (10,1) red, (11,0) blue, (11,1) red, (12,0) red, (12,1) blue, (13,0) blue, (13,1) red, (14,0) blue, (14,1) red, (15,0) red, (15,1) blue, (16,0) blue, (16,1) red, (17,0) red, (17,1) blue, (18,0) red. The 'y\_pred' window has a 'Close' button highlighted in blue, while the 'y\_test' window has a 'Save and Close' button highlighted in blue.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
6	blue	red					blue	red					blue	red					
7		red						blue						blue					
8	blue								red						blue				
9		red							blue							blue			
10		red							blue								blue		
11	blue									red								blue	
12		red								blue									blue
13		blue									red								
14			red								blue								
15			blue									red							
16			blue									red							
17				red								blue							
18					red								blue						

Figure 30.13

### 30.3.4 Making the Confusion Matrix

```
1. # Making the Confusion Matrix
2. from sklearn.metrics import confusion_matrix
3. cm = confusion_matrix(y_test, y_pred)
4. print(cm)
```

Figure 30.14

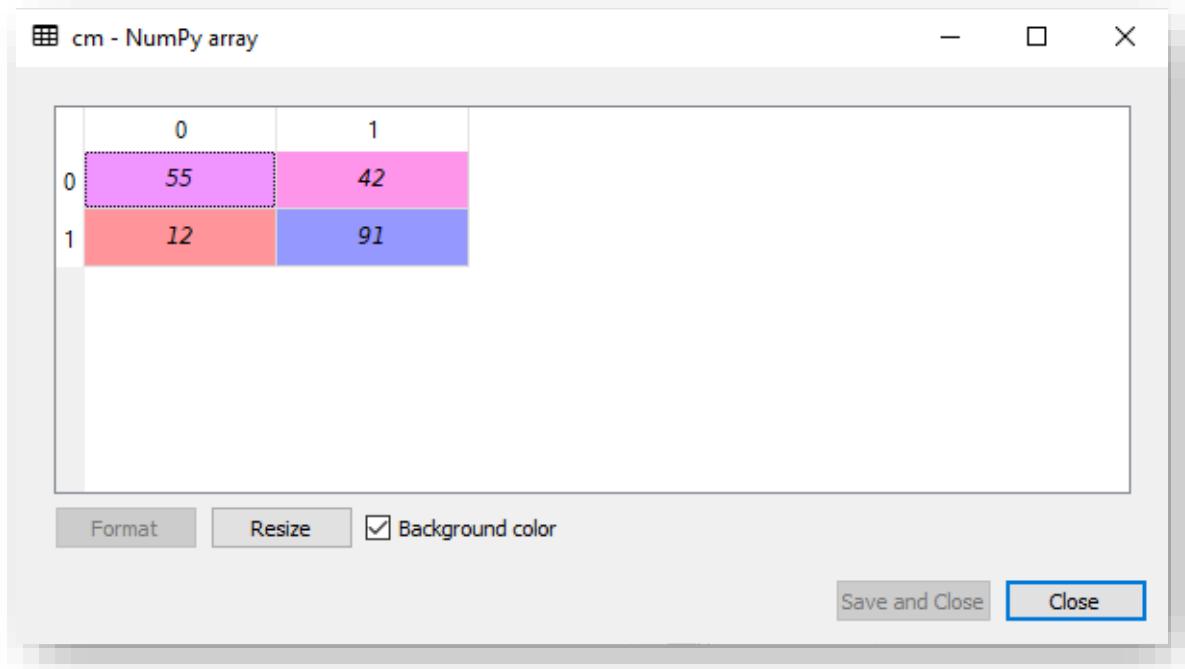


Figure 30.15

---

# CHAPTER 31

---

## Deep Learning

### 31.1 Overview

Deep learning is a machine learning technique that teaches computers to do what comes naturally to humans: learn by example. Deep learning is a key technology behind driverless cars, enabling them to recognize a stop sign, or to distinguish a pedestrian from a lamppost. It is the key to voice control in consumer devices like phones, tablets, TVs, and hands-free speakers. Deep learning is getting lots of attention lately and for good reason. It's achieving results that were not possible before.

In deep learning, a computer model learns to perform classification tasks directly from images, text, or sound. Deep learning models can achieve state-of-the-art accuracy, sometimes exceeding human-level performance. Models are trained by using a large set of labeled data and neural network architectures that contain many layers.

## 31.2 How Deep Learning Works

Most deep learning methods use neural network architectures, which is why deep learning models are often referred to as deep neural networks.

The term “deep” usually refers to the number of hidden layers in the neural network. Traditional neural networks only contain 2-3 hidden layers, while deep networks can have as many as 150.

Deep learning models are trained by using large sets of labeled data and neural network architectures that learn features directly from the data without the need for manual feature extraction.

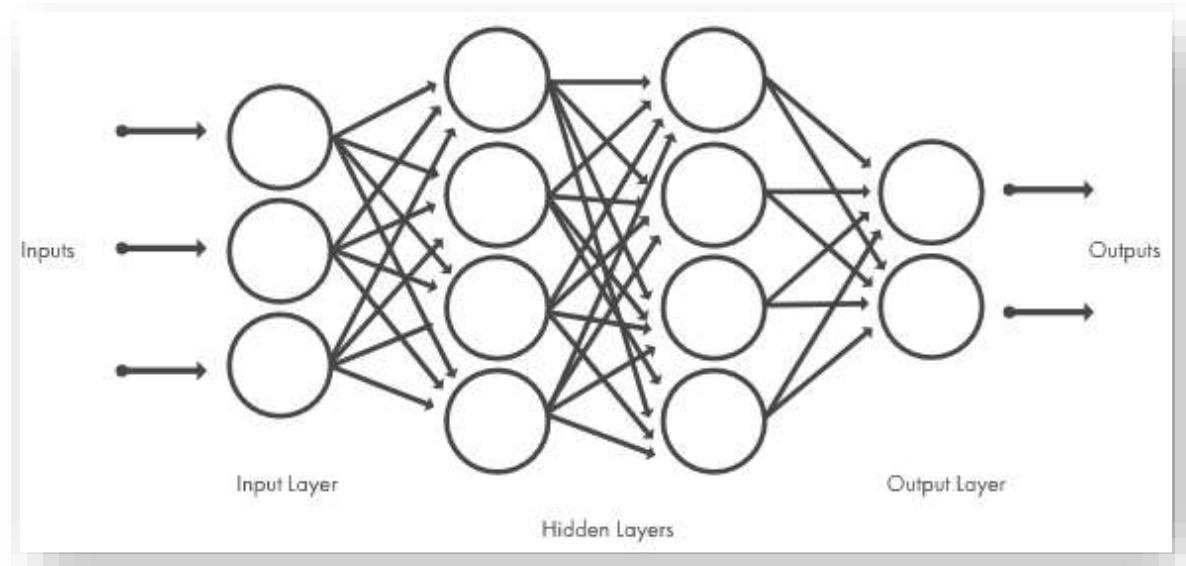


Figure 31.1

One of the most popular types of deep neural networks is known as convolutional neural networks (CNN or ConvNet). A CNN convolves learned features with input data, and uses 2D convolutional layers, making this architecture well suited to processing 2D data, such as images.

CNNs eliminate the need for manual feature extraction, so you do not need to identify features used to classify images. The CNN works by extracting features directly from images. The relevant features are not pretrained; they are learned while the network trains on a collection of images. This automated feature extraction makes deep learning models highly accurate for computer vision tasks such as object classification.

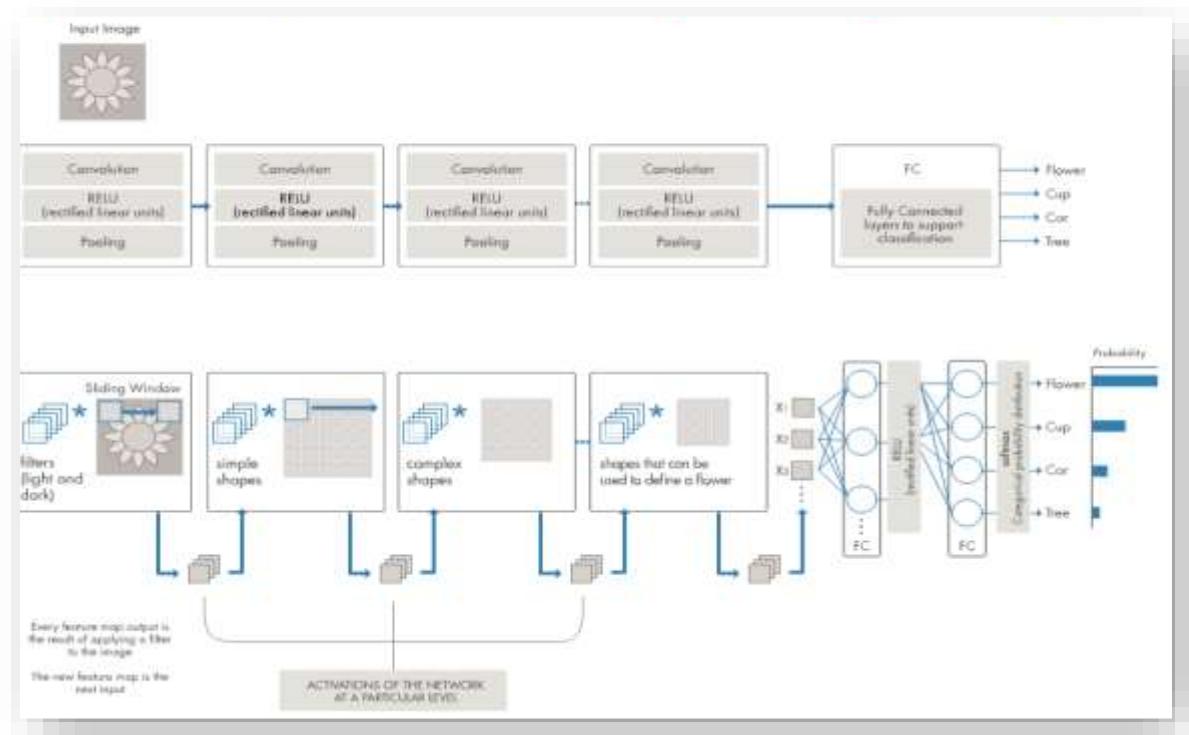


Figure 31.2

### 31.3 Geoffrey Hinton



Figure 31.3

Geoffrey Hinton designs machine learning algorithms. His aim is to discover a learning procedure that is efficient at finding complex structure in large, high-dimensional datasets and to show that this is how the brain learns to see. He was one of the researchers who introduced the backpropagation algorithm and the first to use backpropagation for learning word embeddings. His other contributions to neural network research include Boltzmann machines, distributed representations, time-delay neural nets, mixtures of experts, variational learning, products of experts and deep belief nets. His research group in Toronto made major breakthroughs in deep learning that revolutionized speech recognition and object classification.

Geoffrey received his BA in Experimental Psychology from Cambridge in 1970 and his PhD in Artificial Intelligence from Edinburgh in 1978. He did postdoctoral work at Sussex University and the University of California San Diego and spent five years as a faculty member in the Computer Science department at Carnegie-Mellon University. He then became a fellow of the Canadian Institute for Advanced Research and moved to the Department of Computer Science at the University of Toronto. From 1998 until 2001 he set up the Gatsby Computational Neuroscience Unit at University College London and then returned to the University of Toronto. From 2004 until 2013 he was the director of the program on “Neural Computation and Adaptive Perception,” funded by the Canadian Institute for Advanced Research. In 2013, Google acquired Hinton’s neural networks startup, DNNresearch, which developed out of his research at U of T. He is a Vice President and Engineering Fellow at Google where he manages Brain Team Toronto, which is a new part of the Google Brain Team.

In this part, you will understand and learn how to implement the following Deep Learning models:

1. Artificial Neural Networks for a Business Problem
2. Convolutional Neural Networks for a Computer Vision task

---

# CHAPTER 32

---

## Artificial Neural Network (ANN)

### 32.1 Overview

An artificial neural network (ANN) is the piece of a computing system designed to simulate the way the human brain analyzes and processes information. It is the foundation of artificial intelligence (AI) and solves problems that would prove impossible or difficult by human or statistical standards. ANNs have self-learning capabilities that enable them to produce better results as more data becomes available

We will learn in this section :

### 32.2 The Neuron

An artificial neuron is a mathematical function conceived as a model of biological neurons, a neural network. Artificial neurons are elementary units in an artificial neural network. The artificial neuron receives one or more inputs (representing dendrites) and sums them to produce an output.

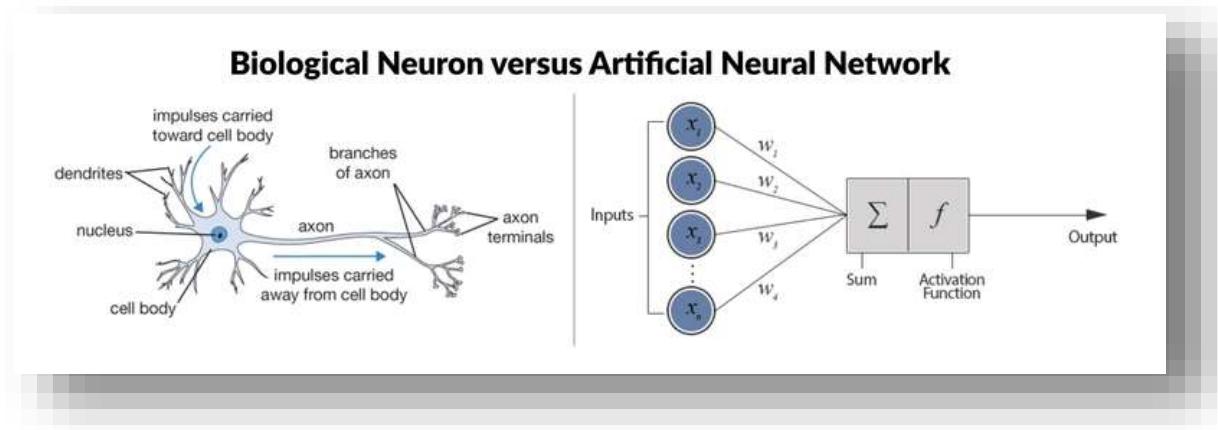


Figure 32.1

There are a few basic similarities between neurons and transistors. They both serve as the basic unit of information processing in their respective domains; they both have inputs and outputs; and they both connect with their neighbors. However, there are drastic differences between neurons and transistors as well. Transistors are simple logic gates generally connected to no more than three other transistors. Neurons, by contrast, are highly complex organic structures connected to roughly 10,000 other neurons. Naturally, this rich network of connections gives neurons an enormous advantage over transistors when it comes to performing cognitive feats that require thousands of parallel connections. For decades, engineers and developers have envisioned ways to capitalize on this advantage by making computers and applications operate more like brains. Finally, their ideas have made their way into the mainstream. Although transistors themselves will not look like neurons anytime soon, some of the AI software they run can now mimic basic neural processing, and it's only getting more sophisticated.

### 32.3 Activation Functions

In a neural network, numeric data points, called inputs, are fed into the neurons in the input layer. Each neuron has a weight, and multiplying the input number with the weight gives the output of the neuron, which is transferred to the next layer.

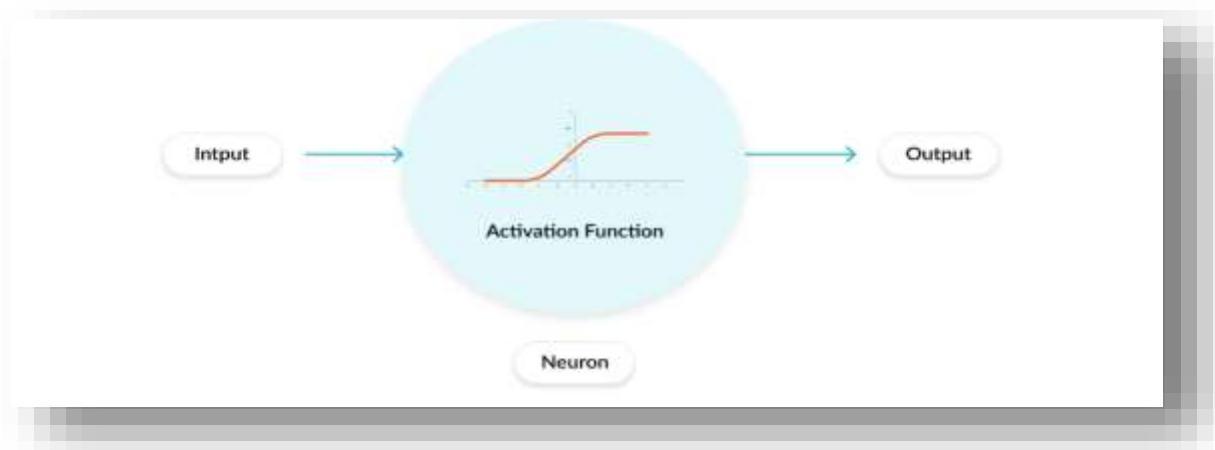


Figure 32.2

Increasingly, neural networks use non-linear activation functions, which can help the network learn complex data, compute and learn almost any function representing a question, and provide accurate predictions.

ANNs depend highly on activation functions, which allow them to follow a non-linear model and learn data very quickly. Here are a few of the most popular ones that you can choose to implement:

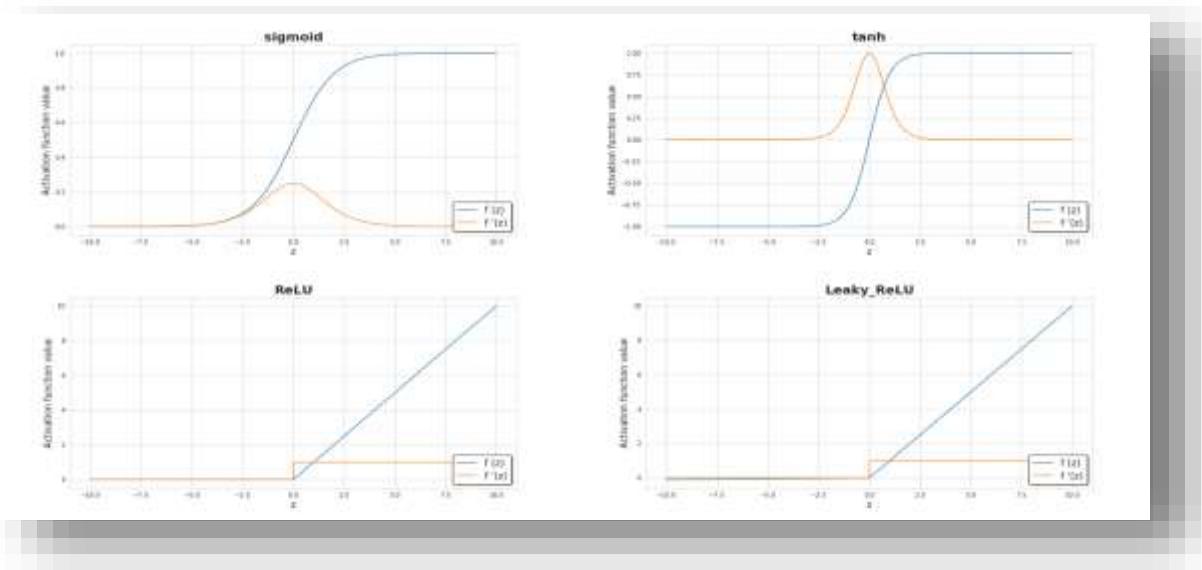


Figure 32.3

## 32.4 Gradient Descent

gradient descent is by far the most popular optimization strategy used in machine learning and deep learning at the moment. It is used when training data models, can be combined with every algorithm and is easy to understand and implement. Everyone working with machine learning should understand its concept. We'll walk through how gradient descent works, what types of it are used today, and its advantages and tradeoffs.

### What is a gradient?

A gradient simply measures the change in all weights with regard to the change in error. You can also think of a gradient as the slope of a function. The higher the gradient, the steeper the slope and the faster a model can learn. But if the slope is

zero, the model stops learning. In mathematical terms, a gradient is a partial derivative with respect to its inputs.

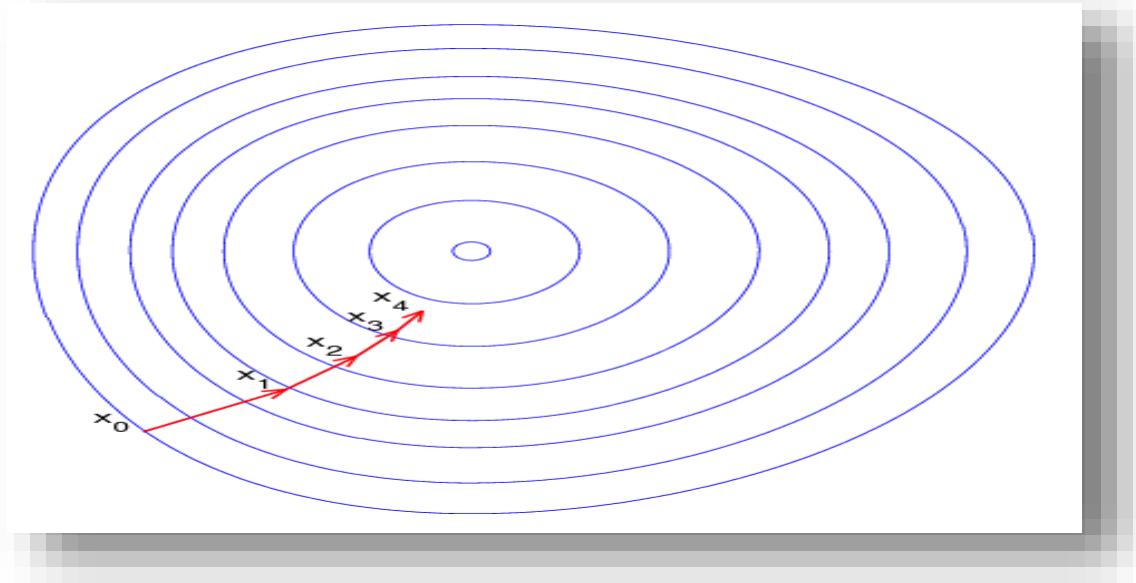


Figure 32.4

the image illustrates our hill from a top-down view and the red arrows are the steps of our climber. Think of a gradient in this context as a vector that contains the direction of the steepest step the blindfolded man can take and also how long that step should be.

## Importance of the learning rate

How big the steps are gradient descent takes into the direction of the local minimum are determined by the learning rate, which figures out how fast or slow we will move towards the optimal weights.

For gradient descent to reach the local minimum we must set the learning rate to an appropriate value, which is neither too low nor too high. This is important because if the steps it takes are too big, it may not reach the local minimum because it bounces back and forth between the convex function of gradient descent (see left image below). If we set the learning rate to a very small value, gradient descent will eventually reach the local minimum but that may take a while (see the right image).

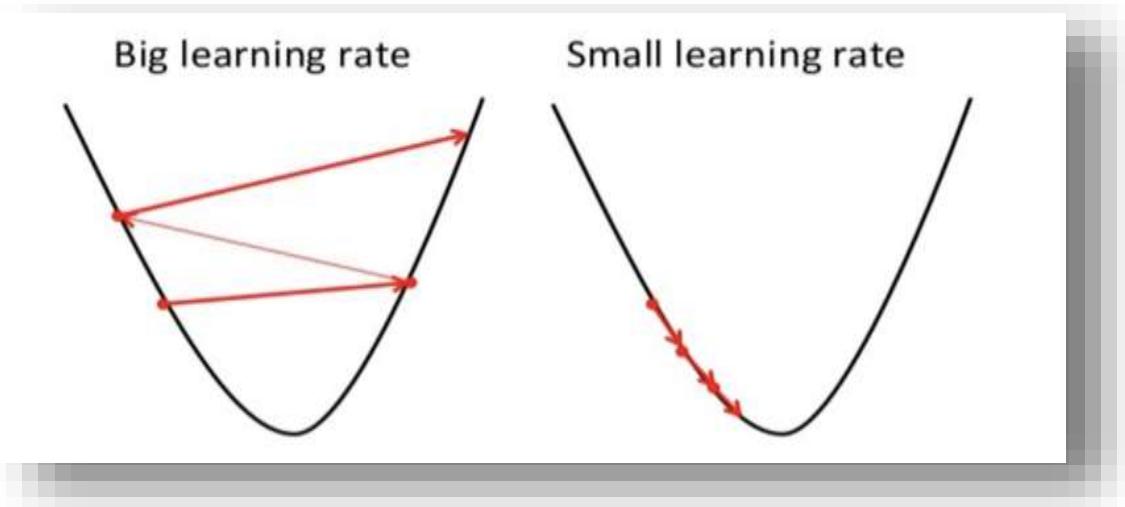


Figure 32.5

So, the learning rate should never be too high or too low for this reason. You can check if your learning rate is doing well by plotting it on a graph.

## 32.5 Backpropagation

In machine learning, gradient descent and backpropagation often appear at the same time, and sometimes they can replace each other. So what is the relationship between them? In fact, we can consider backpropagation as a subset of gradient descent, which is the implementation of gradient descent in multi-layer neural networks. Since the same training rule recursively exists in each layer of the neural network, we can calculate the contribution of each weight to the total error inversely from the output layer to the input layer, which is so-called backpropagation.

The modern neural networks are typically composed of multiple layers, each layer contains multiple neurons and one bias (Note: For ease of reading, the term "bias" in this article sometimes refers to the constant nodes in a neural network, such as here. Sometimes, it refers to the weight connecting a constant node and a neuron), and they are connected by the weights. The data flows forward from the input layer to the output layer. And then, (in the case of supervised machine learning) we compare the predicted results with the real results and calculate the total error by the loss function. In fact, the process of minimizing the total error is the process of finding the minima of the loss function.

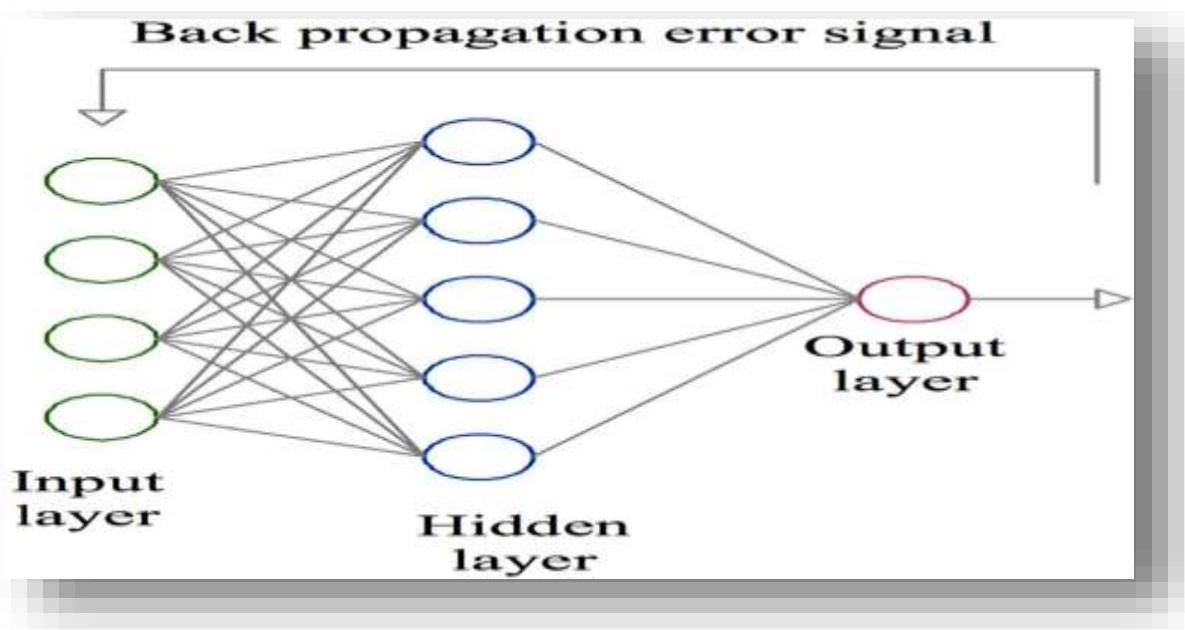


Figure 32.6

We can explain the backpropagation algorithm via this link:

<https://playground.tensorflow.org/#activation=tanh&batchSize=10&dataset=circle&regDataset=reg-plane&learningRate=0.03&regularizationRate=0&noise=0&networkShape=4,2&seed=0.35549&showTestData=false&discretize=false&percTrainData=50&x=true&y=true&xTimesY=false&xSquared=false&ySquared=false&cosX=false&sinX=false&cosY=false&sinY=false&collectStats=false&problem=classification&initZero=false&hideText=false>

## Training the ANN with stochastic gradient descent

**STEP 1:** Randomly initialise the weights to small numbers close to 0 (but not 0).  
**STEP 2:** Input the first observation of your dataset in the input layer, each feature in one input node.  
**STEP 3:** Forward-Propagation: from left to right, the neurons are activated in a way that the impact of each neuron's activation is limited by the weights. Propagate the activations until getting the predicted result  $y$ .  
**STEP 4:** Compare the predicted result to the actual result. Measure the generated error.  
**STEP 5:** Back-Propagation: from right to left, the error is back-propagated. Update the weights according to how much they are responsible for the error. The learning rate decides by how much we update the weights.  
**STEP 6:** Repeat Steps 1 to 5 and update the weights after each observation (Reinforcement Learning). Or: Repeat Steps 1 to 5 but update the weights only after a batch of observations (Batch Learning).  
**STEP 7:** When the whole training set passed through the ANN, that makes an epoch. Redo more epochs.

Figure 32.7

## 32.6 ANN in Python

### Importing the libraries

```
1. # Artificial Neural Network
2.
3. # Importing the libraries
4. import numpy as np
5. import pandas as pd
6. import matplotlib as plt
```

Figure 32.8

## Part 1 - Data Preprocessing

## 1. Importing the dataset

```
1. # Importing the dataset
2. dataset = pd.read_csv('Churn_Modelling.csv')
3. X = dataset.iloc[:, 3:13].values
4. y = dataset.iloc[:, 13].values
```

Figure 32.9

## 2. Encoding categorical data

```
1. # Encoding categorical data
2. # Label Encoding the "Gender" column
3.
4. from sklearn.preprocessing import LabelEncoder , OneHotEncoder
5.
6. lebalencoder_X_1 = LabelEncoder()
7. X[:, 1] = lebalencoder_X_1.fit_transform(X[:, 1])
8.
9. lebalencoder_X_1 = LabelEncoder()
10.X[:, 2] = lebalencoder_X_1.fit_transform(X[:, 2])
11.
```

Figure 32.10

## 3. Taking care of dummy variable trap

```
1. #Taking care of dummy variable trap
2. onehotencoder = OneHotEncoder(categorical_features = [1])
3. X = onehotencoder.fit_transform(X).toarray()
```

Figure 32.11

#### 4. Remove one of 3 dummy variables for the country

```
1. X = X[:,1:]
```

Figure 32.12

#### 5. Splitting the dataset into the Training set and Test set

```
1. # Splitting the dataset into the Training set and Test set
2. from sklearn.model_selection import train_test_split
3. X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_state = 0)
```

Figure 32.13

#### 6. Feature Scaling

```
1. # Feature Scaling
2. from sklearn.preprocessing import StandardScaler
3. sc = StandardScaler()
4. X_train = sc.fit_transform(X_train)
5. X_test = sc.transform(X_test)
```

Figure 32.14

### Part 2 - Building the ANN

#### 1. Importing the keras libraries and package

```
1. # Importing the keras libraries and package
2. import Keras
3. from keras.models import Sequential
4. from keras.layers import Dense
```

Figure 32.15

## 2. Initialize our model

```
1. # Initialising the ANN
2. Classifier = Sequential()
```

Figure 32.16

## 3. Full Connection layer

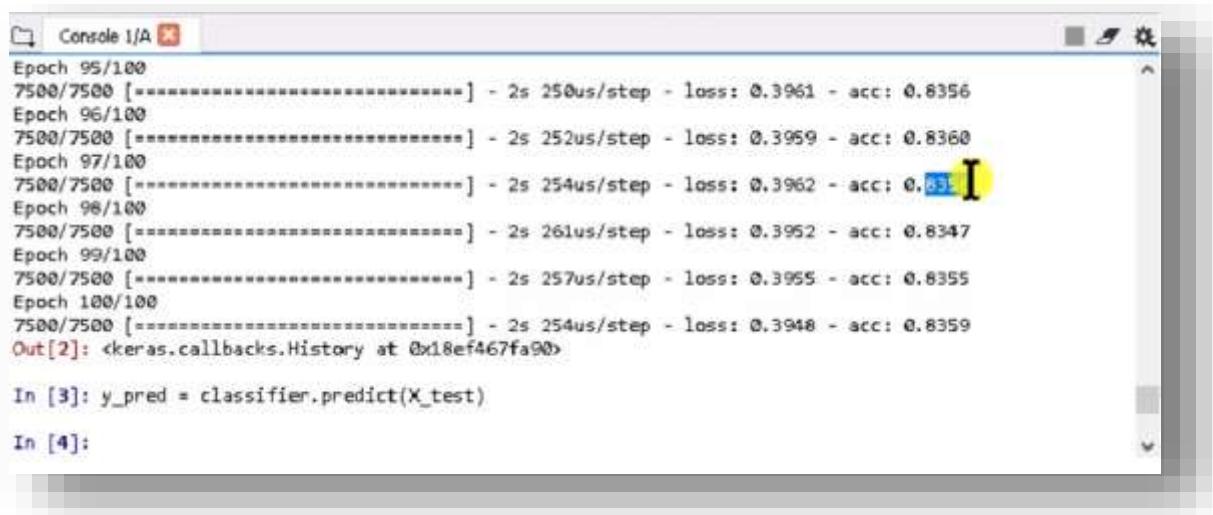
```
1. # Adding the input layer and the first hidden layer
2. Classifier.add(Dense(output_dim = 6,init = "uniform", activation='relu' , input_dim = 11))
3.
4. # Adding the second hidden layer
5. Classifier.add(Dense(output_dim = 6,init = "uniform", activation='relu'))
6.
7. # Adding the output layer
8. Classifier.add(Dense(output_dim = 1,init = "uniform", activation='sigmoid' ))
```

Figure 32.17

## 4. Training the ANN

```
1. # Compiling the ANN
2. Classifier.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics = ['accuracy'])
3.
4. # Training the ANN on the Training set
5. from sklearn.metrics import confusion_matrix
6. Classifier.fit(X_train, y_train, batch_size = 10, epochs = 100)
```

Figure 32.18



Console I/A

```
Epoch 95/100
7500/7500 [=====] - 2s 250us/step - loss: 0.3961 - acc: 0.8356
Epoch 96/100
7500/7500 [=====] - 2s 252us/step - loss: 0.3959 - acc: 0.8360
Epoch 97/100
7500/7500 [=====] - 2s 254us/step - loss: 0.3962 - acc: 0.8352
Epoch 98/100
7500/7500 [=====] - 2s 261us/step - loss: 0.3952 - acc: 0.8347
Epoch 99/100
7500/7500 [=====] - 2s 257us/step - loss: 0.3955 - acc: 0.8355
Epoch 100/100
7500/7500 [=====] - 2s 254us/step - loss: 0.3948 - acc: 0.8359
Out[2]: <keras.callbacks.History at 0x18ef467fa90>

In [3]: y_pred = classifier.predict(X_test)

In [4]:
```

Figure 32.19

## Part 3 - Making the predictions and evaluating the model

### 1. Predicting the Test set results

```
1. # Predicting the Test set results
2. y_pred = Classifier.predict(X_test)
3. y_pred = (y_pred > 0.5)
```

Figure 32.20

### 2. Making the Confusion Matrix

```
1. # Making the Confusion Matrix
2. from sklearn.metrics import confusion_matrix
3. cm = confusion_matrix(y_test, y_pred)
```

Figure 32.21

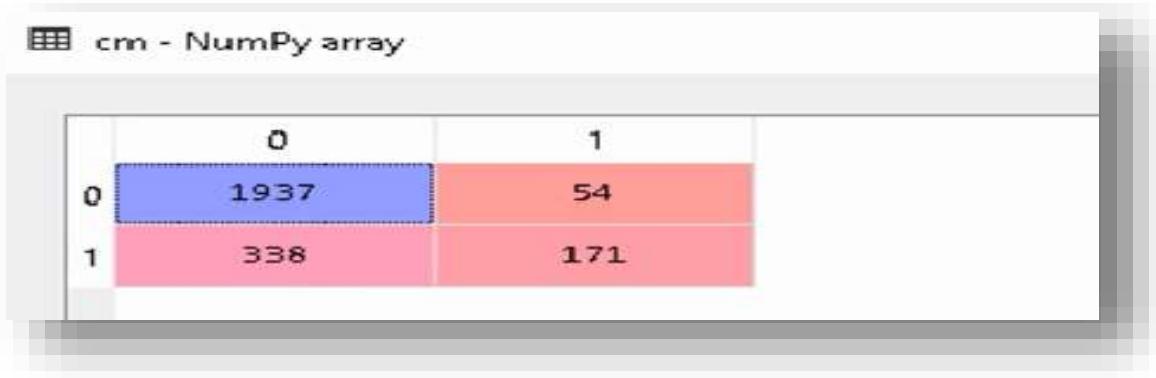


Figure 32.22

---

# CHAPTER 33

---

## Convolutional Neural Networks (CNN)

### 33.1 Overview

Convolutional neural networks. Sounds like a weird combination of biology and math with a little CS sprinkled in, but these networks have been some of the most influential innovations in the field of computer vision. 2012 was the first year that neural nets grew to prominence as Alex Krizhevsky used them to win that year's ImageNet competition (basically, the annual Olympics of computer vision), dropping the classification error record from 26% to 15%, an astounding improvement at the time. Ever since then, a host of companies have been using deep learning at the core of their services. Facebook uses neural nets for their automatic tagging algorithms, Google for their photo search, Amazon for their product recommendations, Pinterest for their home feed personalization, and Instagram for their search infrastructure.

Image classification is the task of taking an input image and outputting a class (a cat, dog, etc) or a probability of classes that best describes the image. For humans, this task of recognition is one of the first skills we learn from the moment we are born and is one that comes naturally and effortlessly as adults.

Without even thinking twice, we're able to quickly and seamlessly identify the environment we are in as well as the objects that surround us. When we see an image or just when we look at the world around us, most of the time we can immediately characterize the scene and give each object a label, all without even consciously noticing. These skills of being able to quickly recognize patterns, generalize from prior knowledge, and adapt to different image environments are ones that we do not share with our fellow machines.

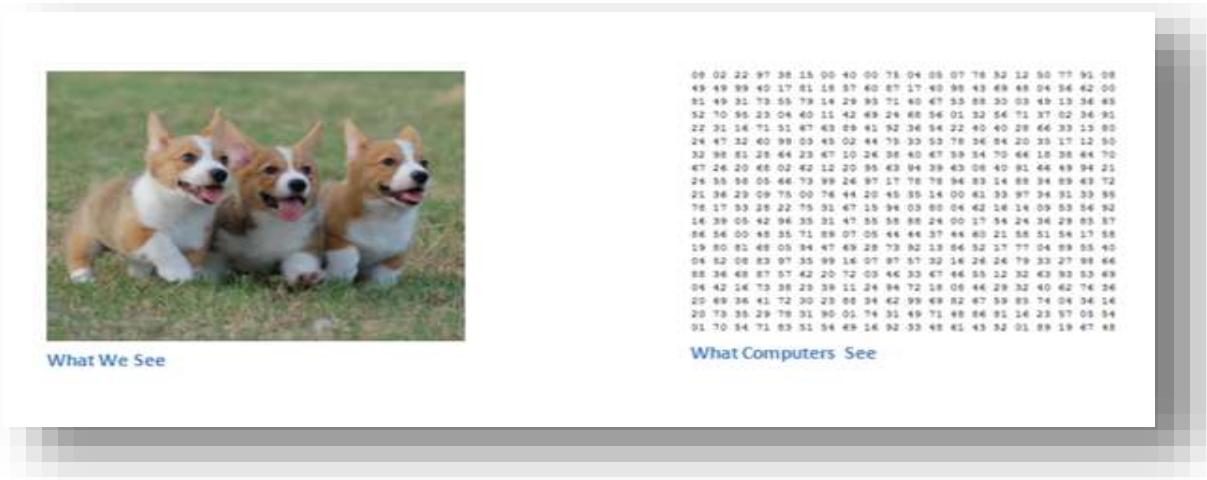


Figure 33.1

## 33.2 Input and Output

When a computer sees an image (takes an image as input), it will see an array of pixel values. Depending on the resolution and size of the image, it will see a  $32 \times 32 \times 3$  array of numbers (The 3 refers to RGB values). Just to drive home the point, let's say we have a color image in JPG form and its size is  $480 \times 480$ . The representative array will be  $480 \times 480 \times 3$ . Each of these numbers is given a value from 0 to 255 which describes the pixel intensity

at that point. These numbers, while meaningless to us when we perform image classification, are the only inputs available to the computer. The idea is that you give the computer this array of numbers and it will output numbers that describe the probability of the image being a certain class (.80 for a cat, .15 for a dog, .05 for a bird, etc).

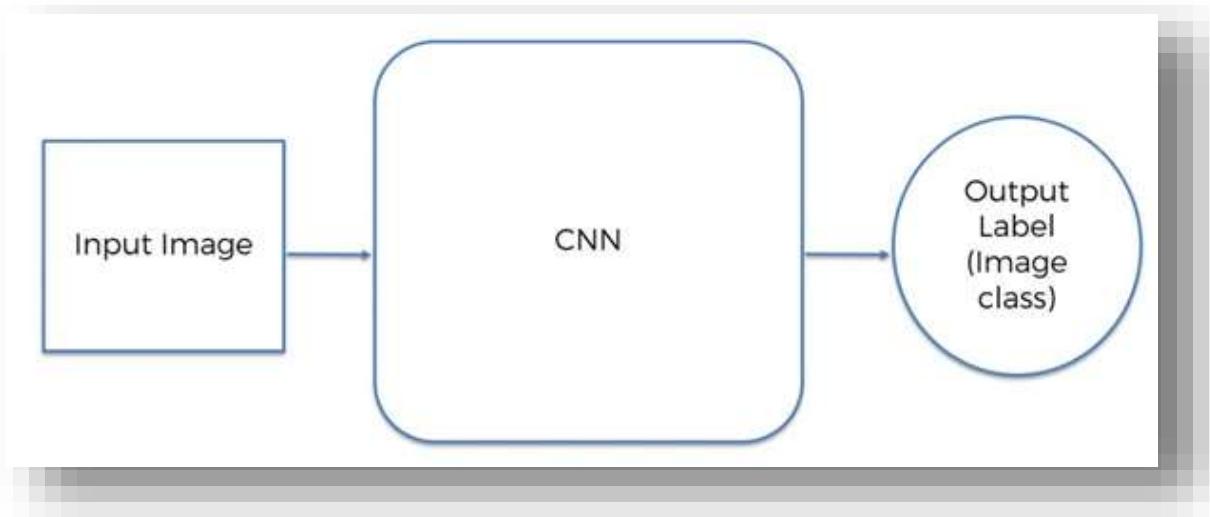


Figure 33.2

### 33.3 Structure of Convolutional Neural Network.

A more detailed overview of what CNNs do would be that you take the image, pass it through a series of convolutional, nonlinear, pooling (downsampling), and fully connected layers, and get an output. As we said earlier, the output can be a single class or a probability of classes that best describes the image. Now, the hard part understands what each of these layers does. So let's get into the most important one.

**STEP-1:** Convolution

**STEP-2:** Max Pooling

**STEP-3:** Flattening

**STEP-4:** Full Connection

## Convolution

ConvNets derive their name from the “Convolution Operation”. The Convolution in case of ConvNet is to extract features from the input images. Convolution preserves the spatial relationships between pixels by learning image features using Small Square of input data. As we discussed above, every image can be considered as a matrix of the pixel value. Consider a 5\*5 image whose pixel values are only 0 & 1 (note that for a grayscale image, pixel values range from 0 to 255, where pixel values are only 0 & 1).

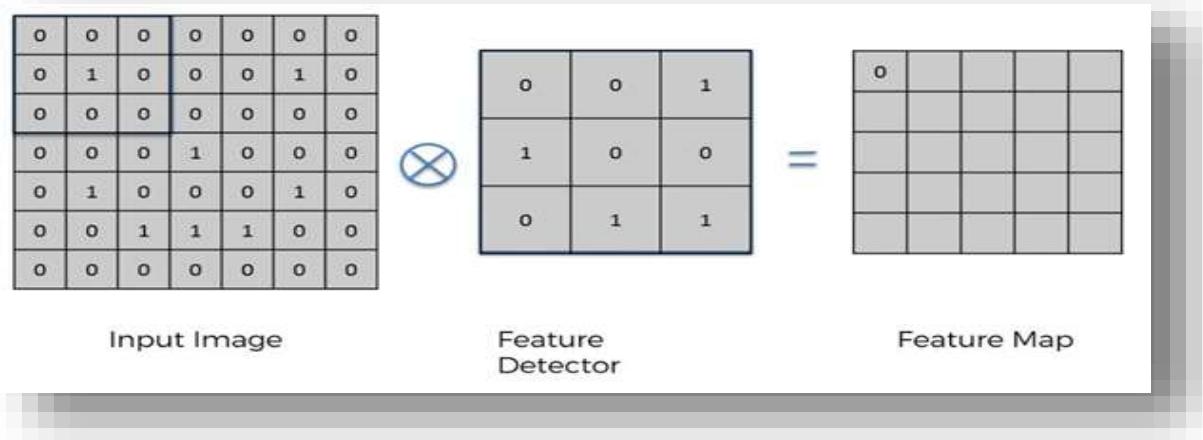


Figure 33.3

Similarly, Feature Detector Detect the Every single part of the input image and then result from the show in Feature map which base on the match of feature Detector of the input image.

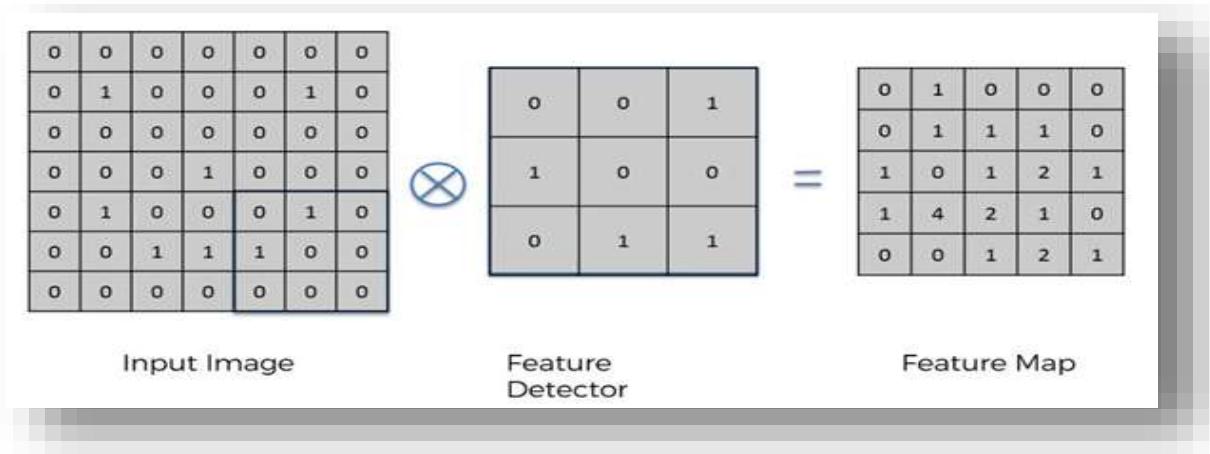


Figure 33.4

CNN terminology, the  $3 \times 3$  matrix is called a ‘filter’ or ‘kernel’ or ‘feature detector’ and the matrix formed by sliding the filter over the image and computing the dot product is called the ‘Convolved Feature’ or ‘Activation Map’ or the ‘Feature Map’. It is important to note that filters act as feature detectors from the original input image.

## ReLU Layer

An additional operation called ReLU has been used after every Convolution operation. ReLU stands for the Rectified Linear Unit and is a non-linear operation. Its output is given by:

Output = Max(zero, Input)

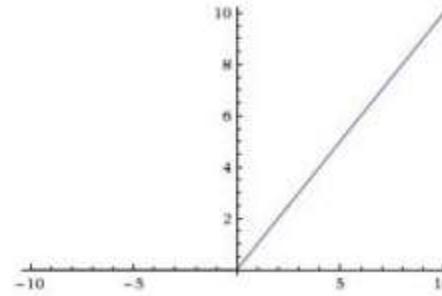


Figure 33.5

ReLU is an element-wise operation (applied per pixel) and replaces all negative pixel values in the feature map by zero. The purpose of ReLU is to introduce non-linearity in our ConvNet since most of the real-world data we would want our ConvNet to learn would be non-linear (Convolution is a linear operation — element-wise matrix multiplication and addition, so we account for non-linearity by introducing a non-linear function like ReLU). As shown below image we apply the Relu operation and he replaces all the negative numbers by 0.

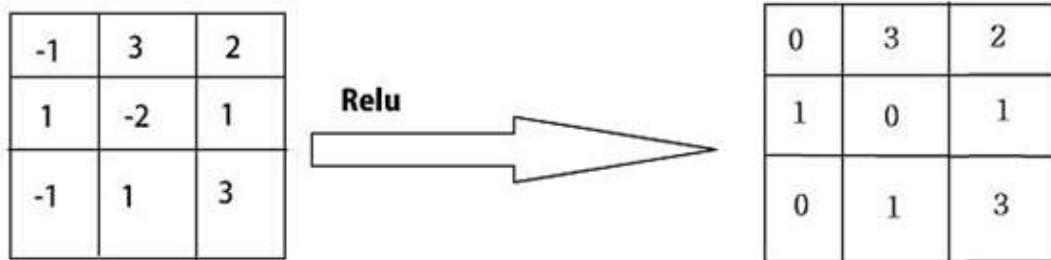


Figure 33.6

## Max Pooling

Spatial Pooling (also called subsampling or downsampling) reduces the dimensionality of each feature map but retains the most important information. Spatial Pooling can be of different types: Max, Average, Sum, etc.

In the case of Max Pooling, we define a spatial neighborhood (for example, a  $2 \times 2$  window) and take the largest element from the rectified feature map within that window. Instead of taking the largest element we could also take the average (Average Pooling) or sum of all elements in that window. In practice, Max Pooling has been shown to work better.

Below shows an example of Max Pooling operation on a Rectified Feature map (obtained after convolution + ReLU operation) by using a  $2 \times 2$  window.

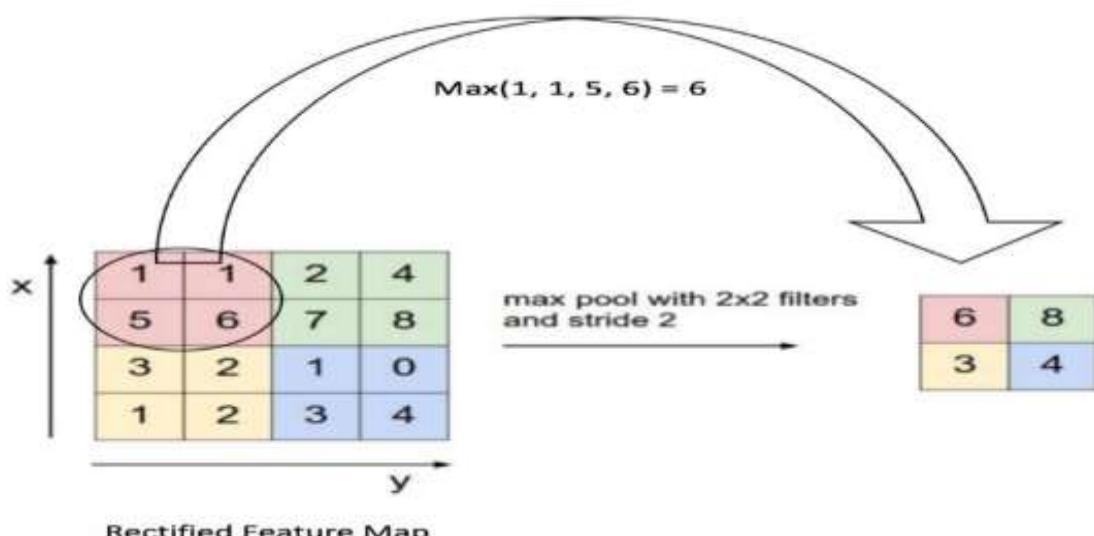


Figure 33.7

## Flattening

Flattening is the process of converting all the resultant 2-dimensional arrays into a single long continuous linear vector.



Figure 33.8

The process of building a CNN involves four major steps

1. Convolution
2. Pooling
3. Flattening
4. Full Connection

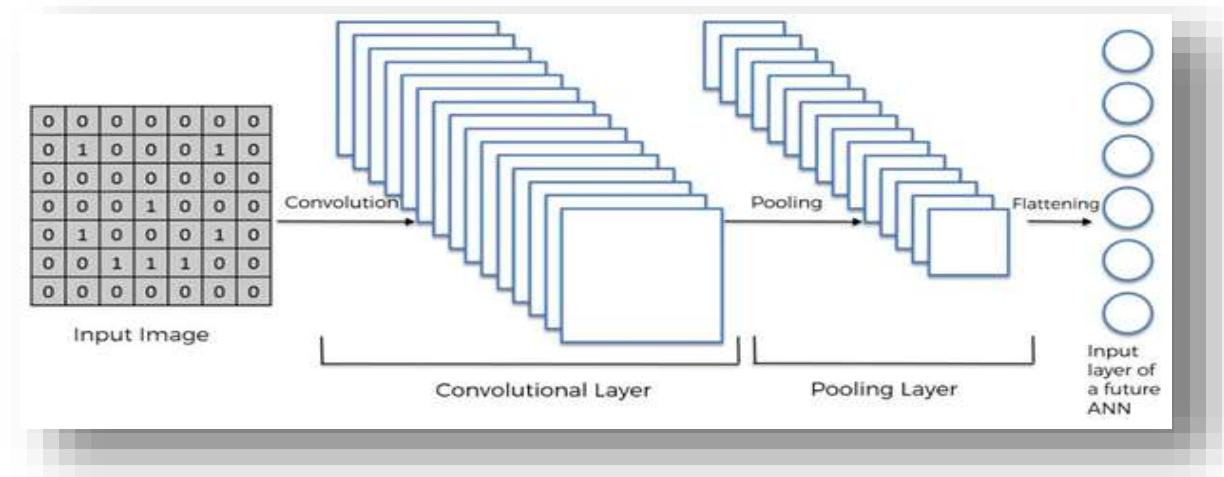


Figure 33.9

So Flattening is become the input of Artificial Neural Network which is used for the **backpropagation** Method.

## Full Connection

The Fully Connected layer is a traditional Multi-Layer Perceptron that uses a softmax activation function in the output layer (other classifiers like SVM can also be used, but will stick to softmax in this post). The term “Fully Connected” implies that every neuron in the previous layer is connected to every neuron on the next layer.

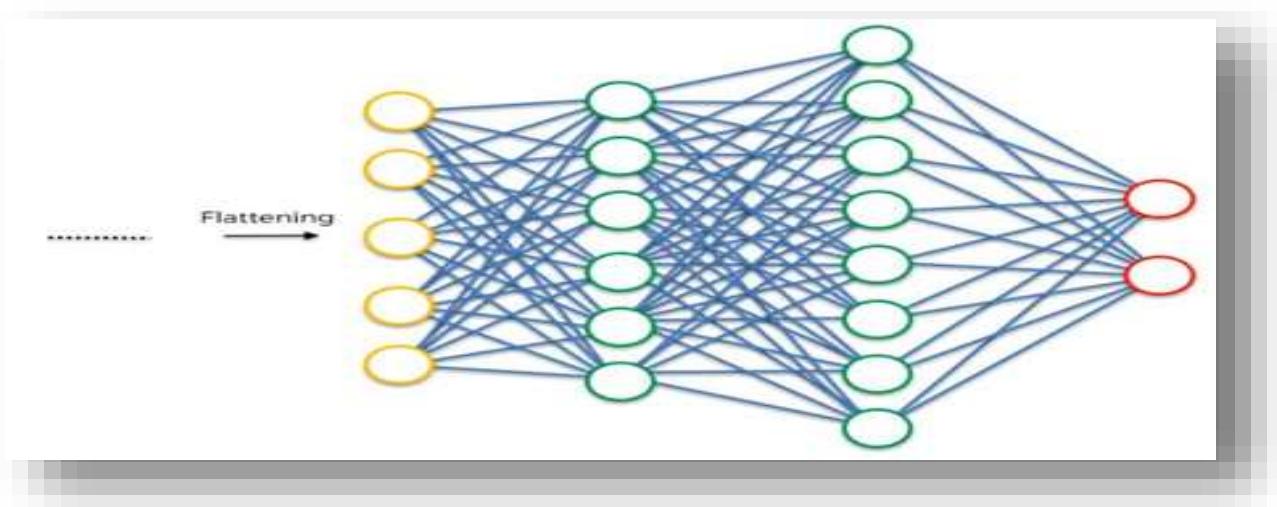


Figure 33.10

The output from the convolutional and pooling layers represent high-level features of the input image. The purpose of the Fully Connected layer is to use these features for classifying the input image into various classes based on the training dataset. For example, the image classification task we set out to perform has four possible outputs as shown in Figure below

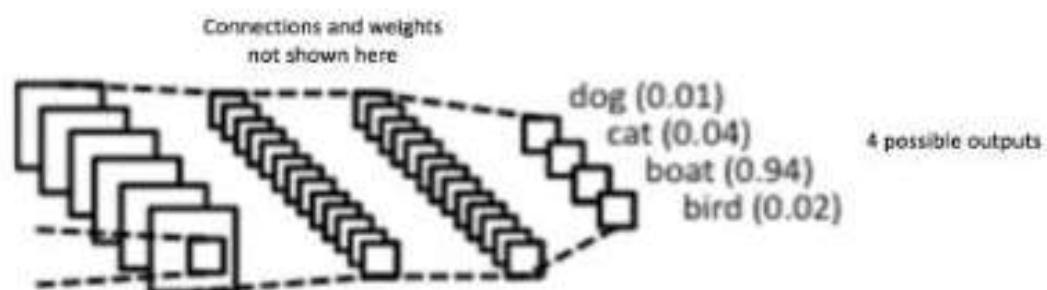


Figure 33.11

Apart from classification, adding a fully-connected layer is also a (usually) cheap way of learning non-linear combinations of these features. Most of the features from convolutional and pooling layers may be good for the classification task, but combinations of those features might be even better. The sum of output probabilities from the Fully Connected Layer is 1. This is ensured by using the Softmax as the activation function in the output layer of the Fully Connected Layer. The Softmax function takes a vector of arbitrary real-valued scores and squashes it to a vector of values between zero and one that sums to one.

## Putting it all together Training using Backpropagation

As discussed above the convolution + Pooling layers act as Features extractor from the input image while a fully connected layer acts as the classifier.

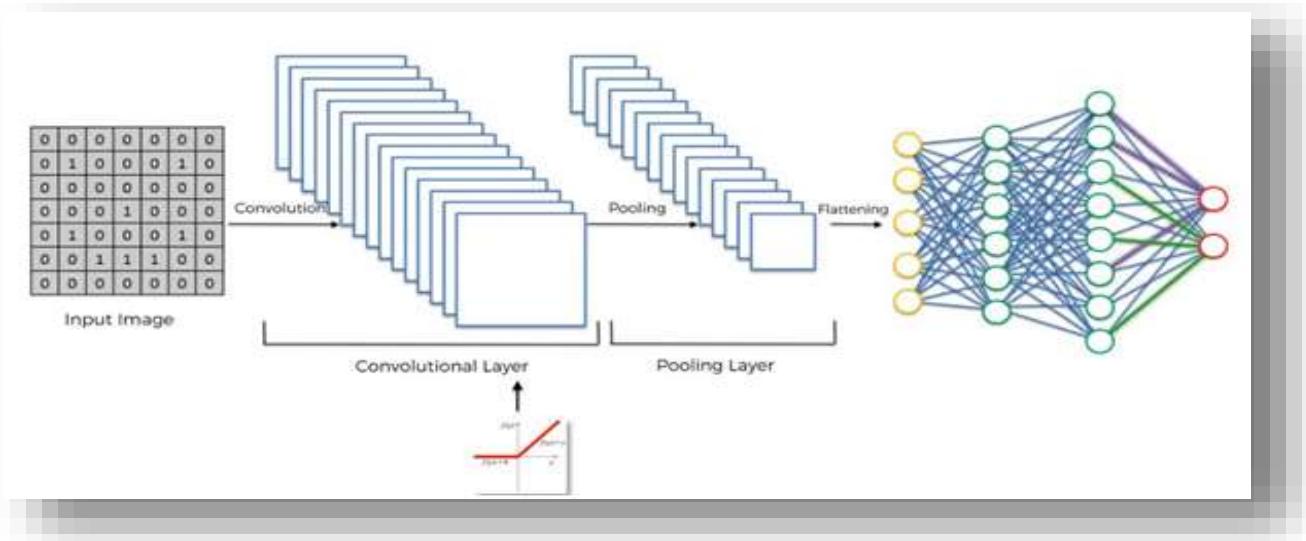


Figure 33.12

Note that in Figure below, since the input image is a boat, the target probability is 1 for Boat class and 0 for the other three classes, i.e.

Input Image = Boat

Target Vector = [0, 0, 1, 0]

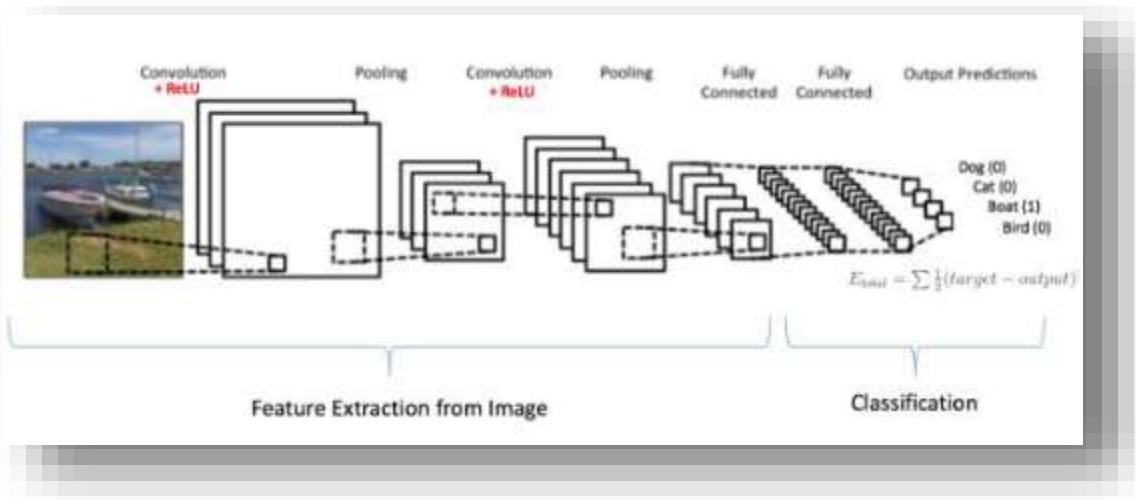


Figure 33.13

## 33.4 CNN in Python

### Importing the keras libraries and package

```
1. # Convolutional Neural Network
2.
3. # Importing the keras libraries and package
4. from keras.models import Sequential
5. from keras.layers import Convolutional2D
```

```
6. from keras.layers import MaxPooling2D  
7. from keras.layers import Flatten  
8. from keras.layers import Dense
```

Figure 33.14

## Part 1 - Building the CNN

### 1. Initialize our model

```
1. # Initialising the CNN  
2. Classifier = Sequential()
```

Figure 33.15

### 2. Convolution Step

```
1. # Step 1 - Convolution  
2. Classifier.add(Convolutional2D(32, 3,3, input_shape=(64, 64, 3), activation="relu"))
```

Figure 33.16

### 3. Pooling Step

```
1. # Step 2 - Pooling  
2. Classifier.add(MaxPool2D( pool_size = (2,2)))
```

Figure 33.17

### 4. Add a second convolution layer and pooling layer

```
1. # Adding a second convolutional layer  
2. Classifier.add(Convolutional2D(32,3,3 activation="relu"))  
3. Classifier.add(MaxPool2D( pool_size= (2,2)))
```

Figure 33.18

## 5. Flatten Step

```
1. # Step 3 - Flattening
2. Classifier.add(Flatten())
```

Figure 33.19

## 6. Full Connection layer

```
1. # Step 4 - Full Connection
2. Classifier.add(Dense(output_dim = 128, activation='relu')) #Hidden Layer
3.
4. Classifier.add(Dense(output_dim = 1, activation='sigmoid')) #Output Layer
```

Figure 33.20

## 7. Compiling the CNN

```
1. # Compiling the CNN
2. Classifier.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics =
 ['accuracy'])
```

Figure 33.21

## Part 2: Image Preprocessing

### 1. Import the library

```
1. from keras.preprocessing.image import ImageDataGenerator
```

Figure 33.22

## Image Augmentation

## 2. Train\_datagen

```
1. # Generating images for the Training set
2. train_datagen = ImageDataGenerator(rescale = 1./255,
3.                                     shear_range = 0.2,
4.                                     zoom_range = 0.2,
5.                                     horizontal_flip = True)
```

Figure 33.23

## 3. Test\_datagen

```
1. test_datagen = ImageDataGenerator(rescale = 1./255)
```

Figure 33.24

## 4. Training set & Test set

```
1. # Creating the Training set
2. training_set = train_datagen.flow_from_directory('dataset/training_set',
3.                                                 target_size = (64, 64),
4.                                                 batch_size = 32,
5.                                                 class_mode = 'binary')
```

Figure 33.25

Found 8000 images belonging to 2 classes

```
1. # Creating the Test set
2. test_set = test_datagen.flow_from_directory('dataset/test_set',
3.                                                 target_size = (64, 64),
4.                                                 batch_size = 32,
5.                                                 class_mode = 'binary')
```

Figure 33.26

Found 2000 images belonging to 2 classes

## 5. Fit the CNN model

```
classifier.fit_generator(train_set,
                         steps_per_epoch=8000,
                         epochs=1,
                         validation_data=test_set,
                         validation_steps=2000)

Epoch 1/1
8000/8000 [=====] - 4038s 505ms/step - loss: 0.4053 - acc: 0.8062 - val_loss:
894 - val_acc: 0.8164
```

Figure 33.27

---

# CHAPTER 34

---

## Dimensionality Reduction

### 34.1 Overview

Remember in Part 3 - Classification, we worked with datasets composed of only two independent variables. We did for two reasons:

1. Because we needed two dimensions to visualize better how Machine Learning models worked (by plotting the prediction regions and the prediction boundary for each model).
2. Because whatever is the original number of our independent variables, we can often end up with two independent variables by applying an appropriate Dimensionality Reduction technique.

There are two types of Dimensionality Reduction techniques:

1. Feature Selection
2. Feature Extraction

Feature Selection techniques are Backward Elimination, Forward Selection, Bidirectional Elimination, Score

Comparison and more. We covered these techniques in Part 2 - Regression.

In this part we will cover the following Feature Extraction techniques:

1. Principal Component Analysis (PCA)
2. Linear Discriminant Analysis (LDA)
3. Kernel PCA
4. Quadratic Discriminant Analysis (QDA)

## 34.2 What is dimensionality reduction

In statistics, machine learning, and information theory, dimensionality reduction or dimension reduction is the process of reduction of n-dimensions to a k-dimensions where  $k << n$ .

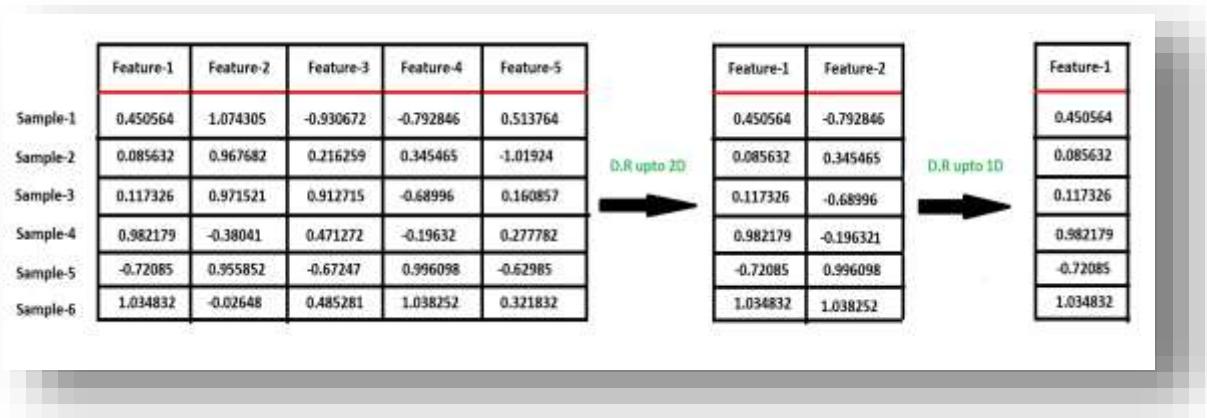


Figure 34.1

### **34.3 Visualization of Data:**

Data visualization brings easiness in understanding and increases effectiveness. The human mind learns fast from visuals than that of text and tables. It is applied to a large population, for e.g., one can remember dialogues and scenes of a movie which he might have watched years before, on the other hand, it is difficult for him to recall the subject he recently read.

Now-a-days we have a good number of tools for data visualization tools, which are fast and effective. Data visualization creates a better selling strategy. Data visualization boosts the ability to process information in an easy and faster way to compare and make conclusions out of it.

Let's have a look at the data of various dimensions.

#### **1. 1Dimensional Data**

Here, we often call the dimensions as features . As an example we have taken an 1D array and started plotting the values on a number line.

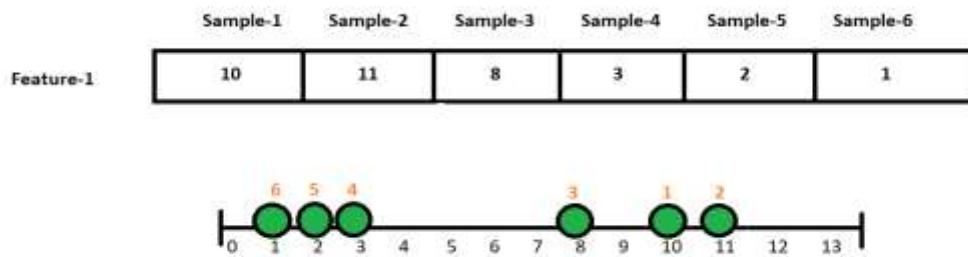


Figure 34.2

## 2. 2Dimensional Data

Now we have taken a 2D array and started plotting the data on X and Y axis which are orthogonal to each other.

	Sample-1	Sample-2	Sample-3	Sample-4	Sample-5	Sample-6
Feature-1	10	11	8	3	2	1
Feature-2	6	4	5	3	2.8	1

Figure 34.3

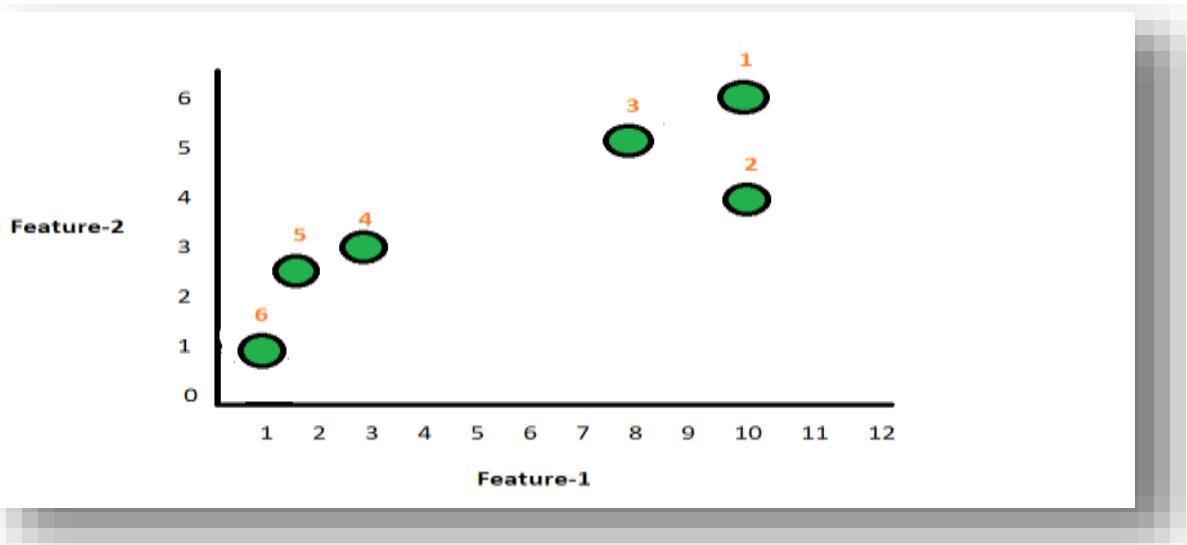


Figure 34.4

### 3. 3Dimensional Data

Now we will work with 3D arrays and let's plot it on X, Y and Z axis

	Sample-1	Sample-2	Sample-3	Sample-4	Sample-5	Sample-6
Feature-1	10	11	8	3	2	1
Feature-2	6	4	5	3	2.8	1
Feature-3	12	9	10	2.5	1.3	2

Figure 34.5

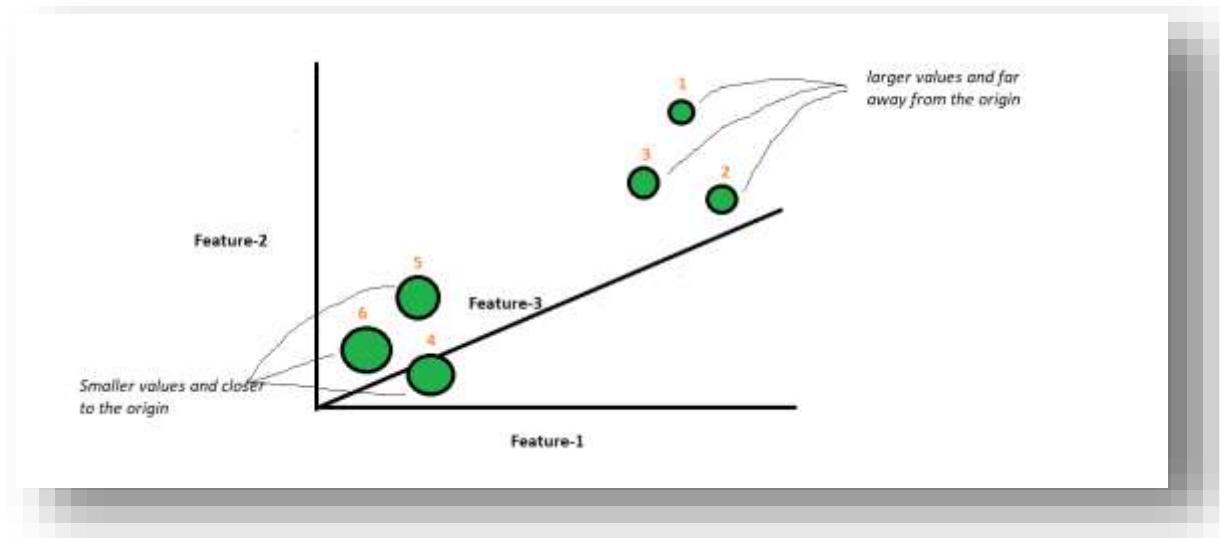


Figure 34.6

We can see that, as the dimensions get increased the visualization of data is getting difficult for us.

#### 4. n-Dimensional Data

Now for N-D data we need N dimensions which we can no longer visualize it. So for visualization of any data having more than 3D, we will reduce it to 2 or 3 dimensions using technique called dimensionality reduction.

### 34.4 Components of Dimensionality Reduction

There are two major components of dimensionality reduction which will be discussed in detail here.

#### 1. Feature Selection

Most of the times the features are not relevant to our problem. For example, we are training a model for predicting the heights of people and we have data with features( weights, color, moles, marital status, gender). We can see that the features like color, moles and marital status are not linked with the heights of people i.e., irrelevant to our problem of finding heights of people. Hence we need to come up with a solution of finding features which are most useful for our task. We can achieve this by following ways:

1. Filter Strategy: Strategy to gain more information on the data.
2. Wrapper Strategy: Basing on the model accuracy we will select features.
3. Embedded Strategy: Basing on model prediction errors, we will take a decision whether to keep or remove the selected features.

## 2. Feature Projection

Feature Projection also know as Feature Extraction is used to transform the data in high dimensional space to low dimensional space. The data transformation can be done in both linear and non linear.

For linear transformation we have principal component analysis(PCA), Linear Discriminant Analysis(LDA) and for non-linear transformations we apply T-SNE.

## 34.5 Principal Component Analysis (PCA)

PCA is mostly used as a tool in exploratory data analysis (EDA) and for making predictive models. It is often used to visualize genetic distance and relatedness between populations. PCA can be done by eigenvalue decomposition of a data covariance (or

correlation) matrix or singular value decomposition of a data matrix.

## 1. Working methodology of PCA

For better understanding of principle working of PCA let's take 2D data.

	Sample-1	Sample-2	Sample-3	Sample-4	Sample-5	Sample-6
Feature-1	10	11	8	3	2	1
Feature-2	6	4	5	3	2.8	1

Figure 34.7: 2D representation of data.

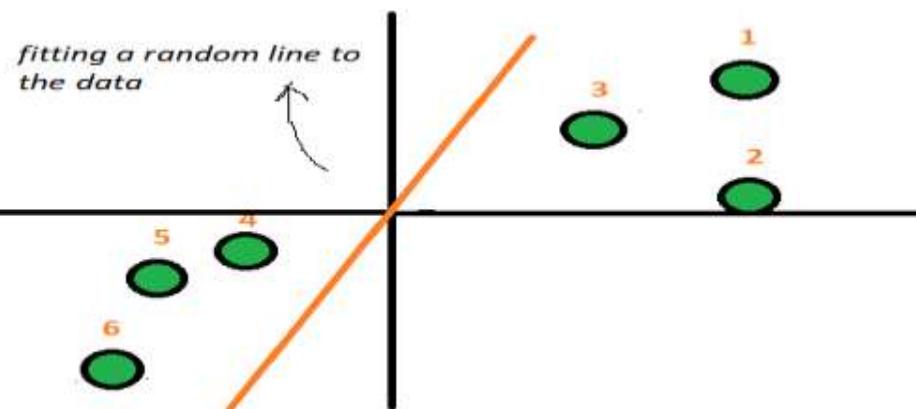


Figure 34.8

1. Firstly we will normalize the data such that the average value shifts to the origin and all the data lie in a unit square.

2. Now we will try to fit a line to the data. For that we will try out with a random line. Now we will rotate the line until it fits best to the data.

Ultimately we end up with the following fit (high degree of fit) which explains the maximum variance of a feature.

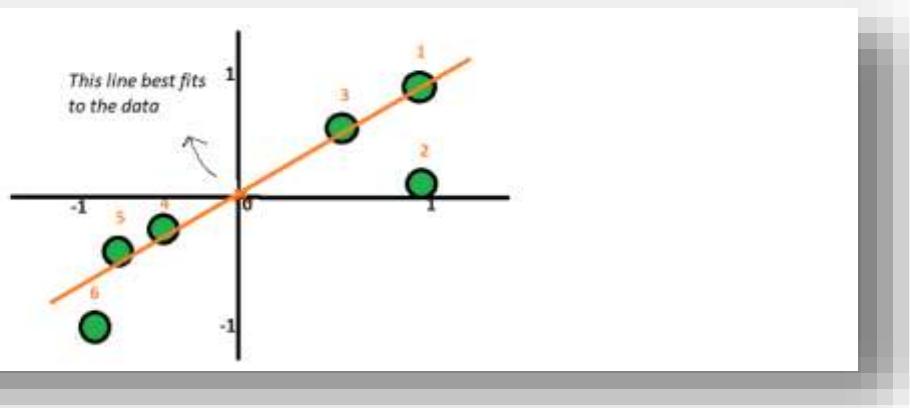


Figure 34.9

## 34.6 Linear Discriminant Analysis(LDA)

LDA is the most commonly used as dimensionality reduction technique in the preprocessing step in machine learning and in statistics, pattern recognition. While the goal of this algorithm is to project a dataset to a lower dimensional space with a separability of the categories in order to avoid overfitting and to reduce the computational power of the machines.

### 1. Working of LDA

Both PCA and LDA are linear reduction techniques but unlike PCA, LDA focuses on maximizing the separability of two groups.

LDA uses features to create a new axes and tries to project the data onto a new axes in a way to maximize the separation of the two categories or groups. This is why LDA is a Supervised learning algorithm since it makes use of target values to find the new axes. PCA tries to find the components that maximizes the variance, while on the other hand LDA tries to find the new axes that

### 1. Maximizes the separability of the categories

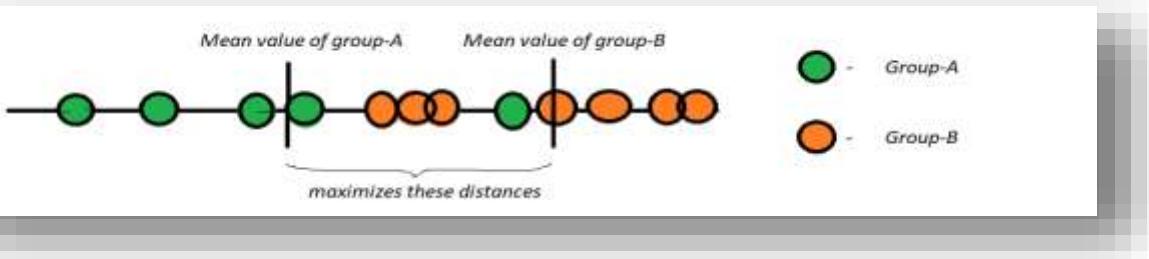


Figure 34.10

### 2. Minimizes the variance among categories.

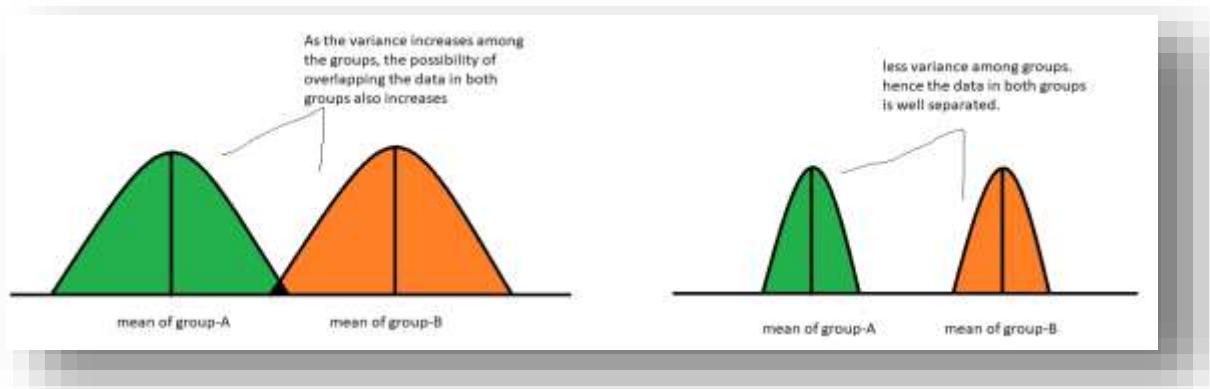


Figure 34.11

By minimizing the variance, we can well separate the clusters of individual groups. Hence it is as important as maximizing the mean values of groups.

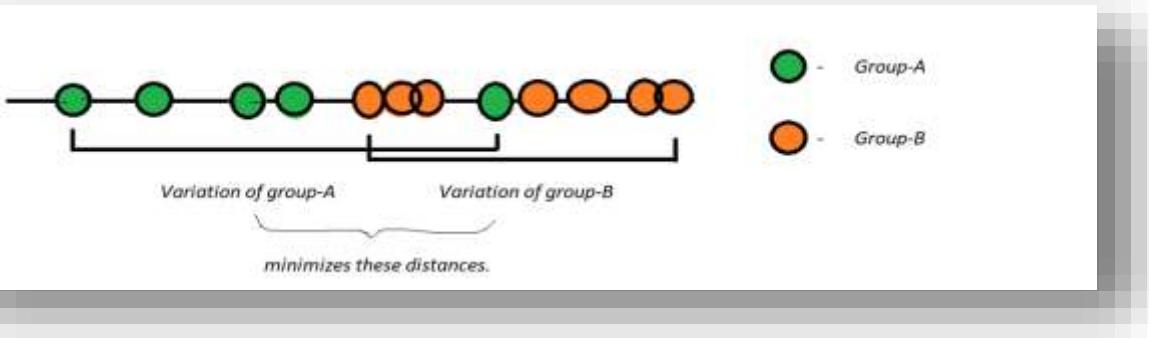


Figure 34.12

Consider the data which has more than 2 groups in such cases LDA finds the mean value of whole data and the center among the individual groups, now it tries to maximize the distance from the center mean value to the individual group mean value. For better understanding look at the following data with 3 categories.

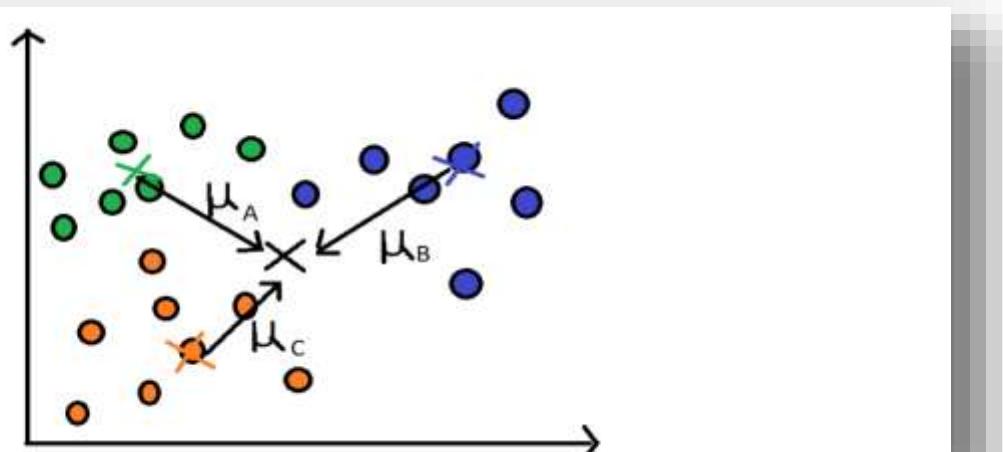


Figure 34.13

Now we can find a plane that best separates the three groups.

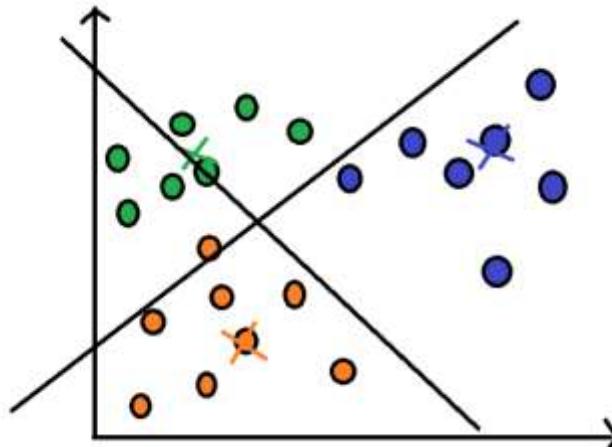


Figure 34.14

## 34.7 LDA in Python

### 1. Importing the keras libraries and package

```
1. # Linear Discriminant Analysis (LDA)
2.
3. # Importing the libraries
4. import numpy as np
5. import matplotlib.pyplot as plt
6. import pandas as pd
```

Figure 34.15

### 2. Importing the dataset

```
1. # Importing the dataset
2. dataset = pd.read_csv('Wine.csv')
```

```
3. X = dataset.iloc[:, :-1].values
4. y = dataset.iloc[:, -1].values
```

Figure 34.16

### 3. Feature Scaling

```
1. # Feature Scaling
2. from sklearn.preprocessing import StandardScaler
3. sc = StandardScaler()
4. X = sc.fit_transform(X)
```

Figure 34.17

### 4. Splitting the dataset into the Training set and Test set

```
1. from sklearn.model_selection import train_test_split
2. X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 0)
```

Figure 34.18

### 5. Applying LDA

```
1. from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA
2. lda = LDA(n_components = 2)
3. X_train = lda.fit_transform(X_train, y_train)
4. X_test = lda.transform(X_test)
```

Figure 34.19

### 6. Training the Logistic Regression model on the Training set and Test set results

```
1. #Training the Logistic Regression model on the Training set
2. from sklearn.linear_model import LogisticRegression
3. classifier = LogisticRegression(random_state = 0)
4. classifier.fit(X_train, y_train)
```

```
5.  
6. # Predicting the Test set results  
7. y_pred = classifier.predict(X_test)
```

Figure 34.20

## 7. Making the Confusion Matrix

```
1. # Making the Confusion Matrix  
2. from sklearn.metrics import confusion_matrix  
3. cm = confusion_matrix(y_test, y_pred)
```

Figure 34.21

## 8. Visualising the Training set results

```
1. # Visualising the Training set results  
2. from matplotlib.colors import ListedColormap  
3. X_set, y_set = X_train, y_train  
4. X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 1, stop = X_set[:, 0].max() + 1, step = 0.01),  
5.                      np.arange(start = X_set[:, 1].min() - 1, stop = X_set[:, 1].max() + 1, step = 0.01))  
6. plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape),  
7.                 alpha = 0.75, cmap = ListedColormap(('red', 'green', 'blue')))  
8. plt.xlim(X1.min(), X1.max())  
9. plt.ylim(X2.min(), X2.max())  
10. for i, j in enumerate(np.unique(y_set)):  
11.     plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],  
12.                 c = ListedColormap(('red', 'green', 'blue'))(i), label = j)  
13. plt.title('Logistic Regression (Training set)')  
14. plt.xlabel('LD1')  
15. plt.ylabel('LD2')  
16. plt.legend()  
17. plt.show()
```

Figure 34.22

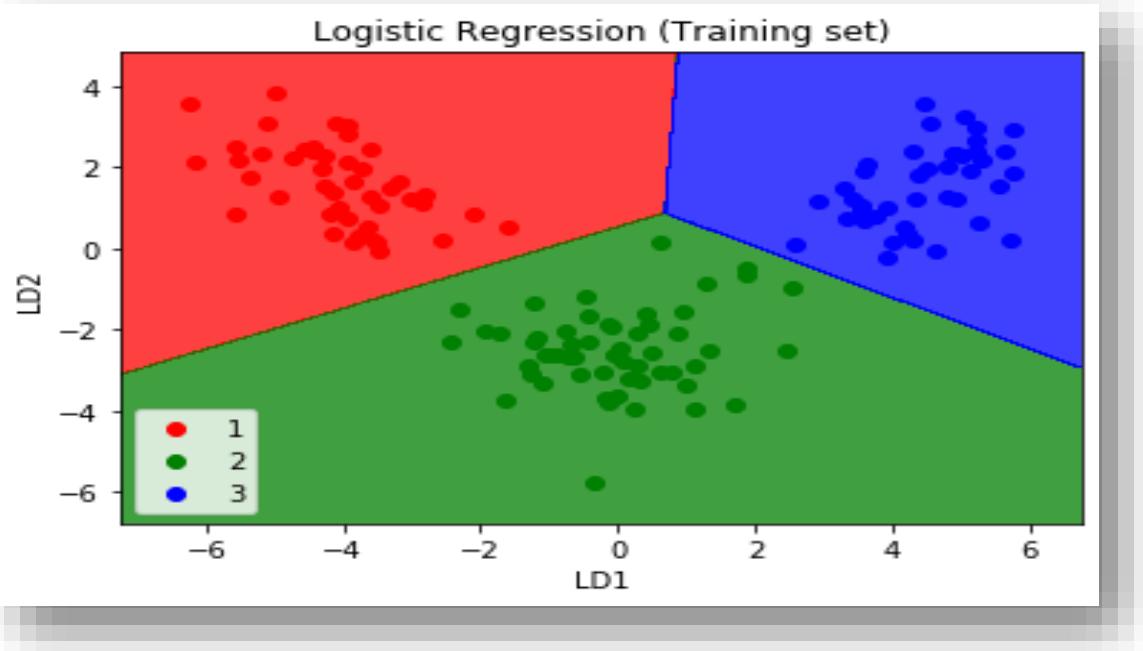


Figure 34.23

## 9. Visualising the Test set results

```

1. # Visualising the Test set results
2. from matplotlib.colors import ListedColormap
3. X_set, y_set = X_test, y_test
4. X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 1, stop = X_set[:, 0].max() + 1, step = 0.01),
5.                      np.arange(start = X_set[:, 1].min() - 1, stop = X_set[:, 1].max() + 1, step = 0.01))
6. plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape),
7.                 alpha = 0.75, cmap = ListedColormap(('red', 'green', 'blue')))
8. plt.xlim(X1.min(), X1.max())
9. plt.ylim(X2.min(), X2.max())
10. for i, j in enumerate(np.unique(y_set)):
11.     plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
12.                 c = ListedColormap(('red', 'green', 'blue'))(i), label = j)
13. plt.title('Logistic Regression (Test set)')
14. plt.xlabel('LD1')
15. plt.ylabel('LD2')
16. plt.legend()
17. plt.show()

```

Figure 34.24

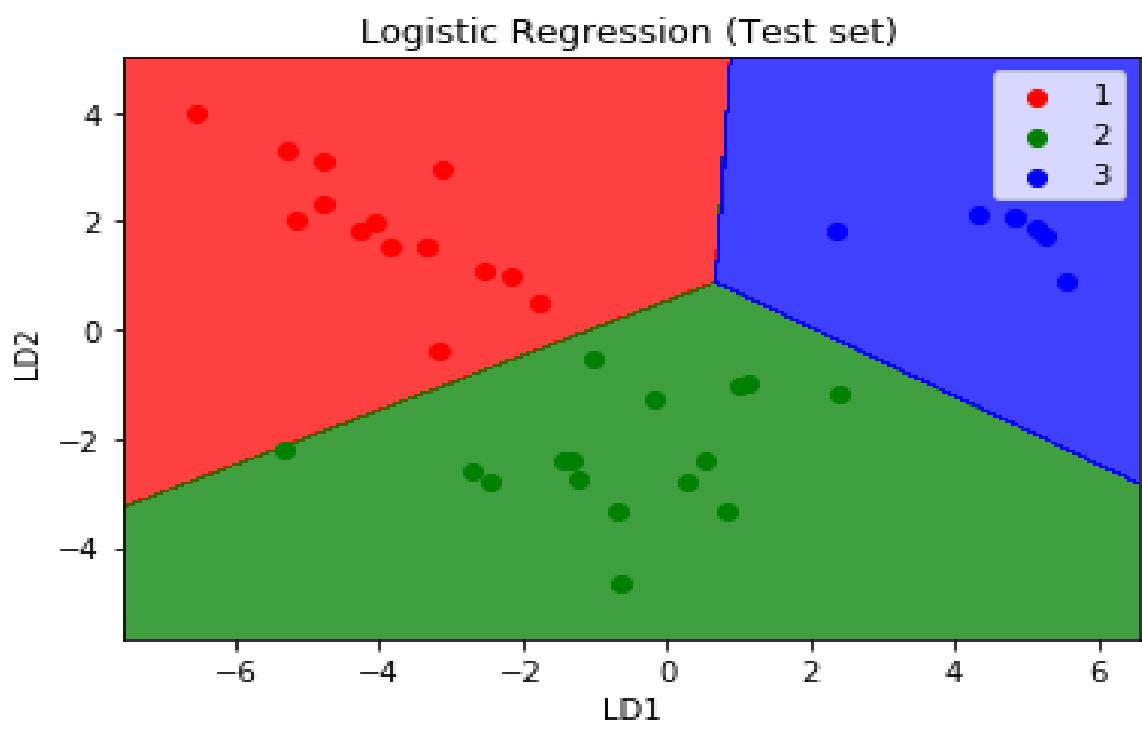


Figure 34.25

---

---

**F**inally, we can only say that we have presented our opinion and made our thoughts on this book so that we may have succeeded in writing and expressing it. Where we were able to get to the extent of the importance of these topics and talked about their vitality. If we had more time, we would have used many other pages, so that we can cover many topics related to this book in all its aspects.



Copyright © 2020-2021, Jordan