



- [Delivery Method](#)
- [What is Machine Learning?](#)
- [Traditional Programming](#)
- [Machine Learning](#)
- [Applications of Machine Learning](#)
- [Types of Machine Learning Problems](#)
- [Classification](#)
- [Regression/ Estimation](#)
- [Affinity Grouping](#)
- [Clustering](#)
- [Typical Process of Machine Learning](#)
- [Python as a Programming Language](#)
- [Popular Python Data Analytics Libraries](#)
- [IDE For Python Programming](#)
- [Comparison – R vs. Python](#)
- [What is in this course?](#)

What is Machine Learning?



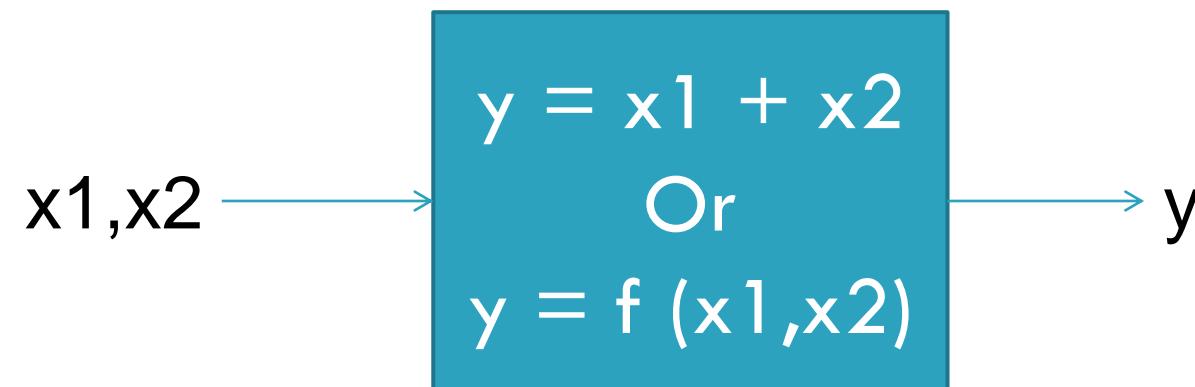
Example: **Child learns his /her mother Language**

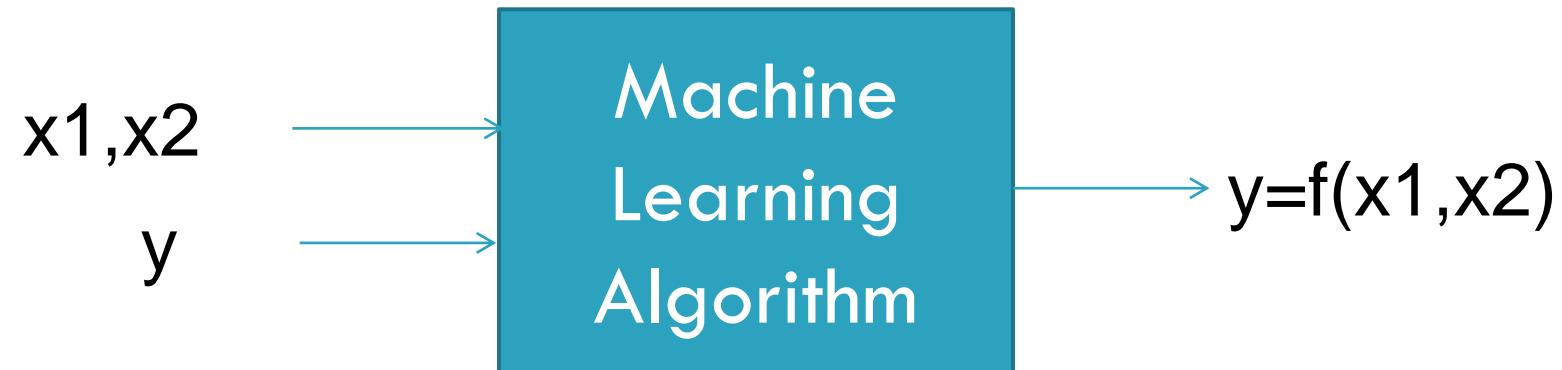
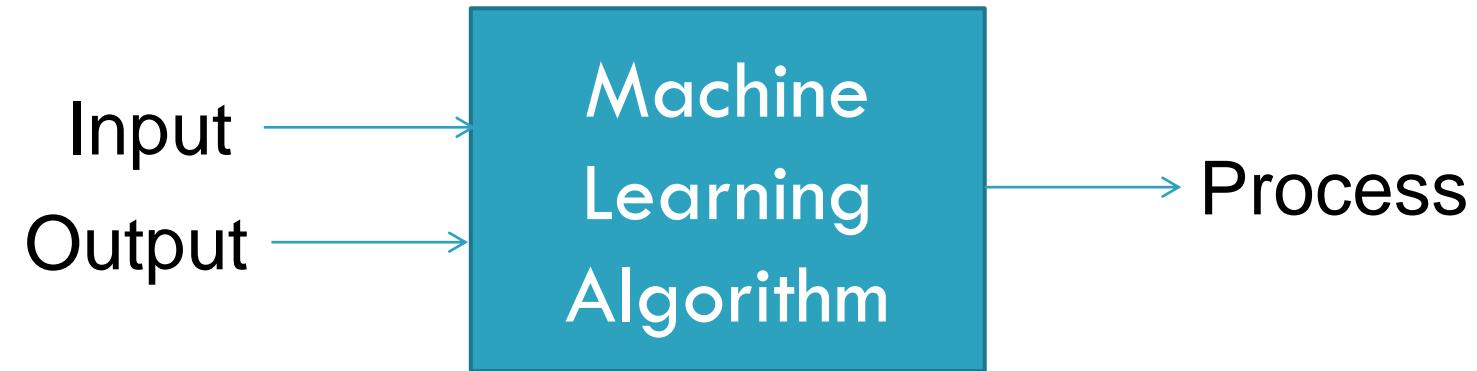
- The simplest example of machine learning is a child's language learning. Normally, children start talking in their mother tongue from a young age. They are neither told the dictionaries of the mother tongue nor are they taught grammar but even from the age of 2-3; they can easily speak and understand their mother tongue.
- The child gradually begins to learn the mother tongue through his mother. And after some time he starts to understand and speak easy sentences. As a child grows up, he starts speaking and understanding difficult words and sentences easily.
- The seed of machine learning is data training. You cannot start machine learning without training data.



x1 (Input)	y (Output)
2	4
3	6
4	8
5	10
6	12
7	14
8	16
9	?

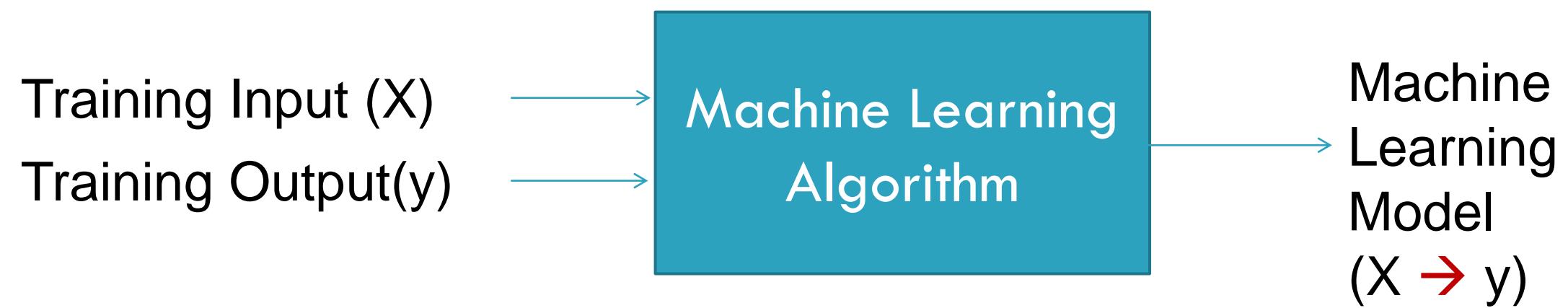
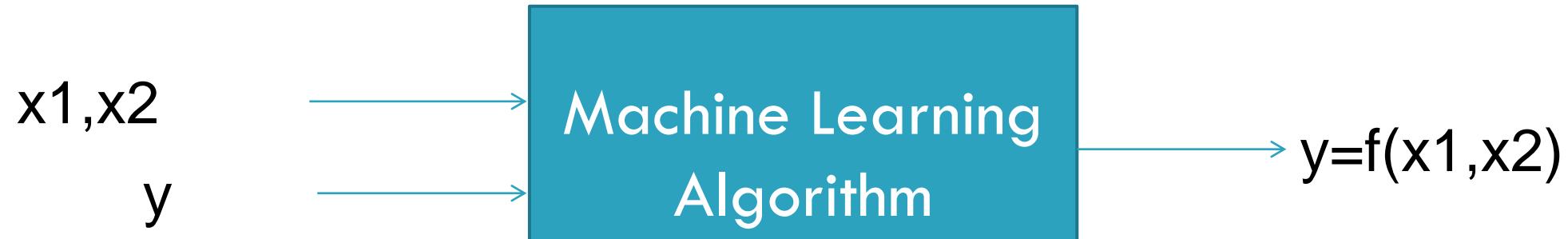
$y = 2 \text{ multiplied by } x_1$





What is the requirement of Machine Learning Problem?

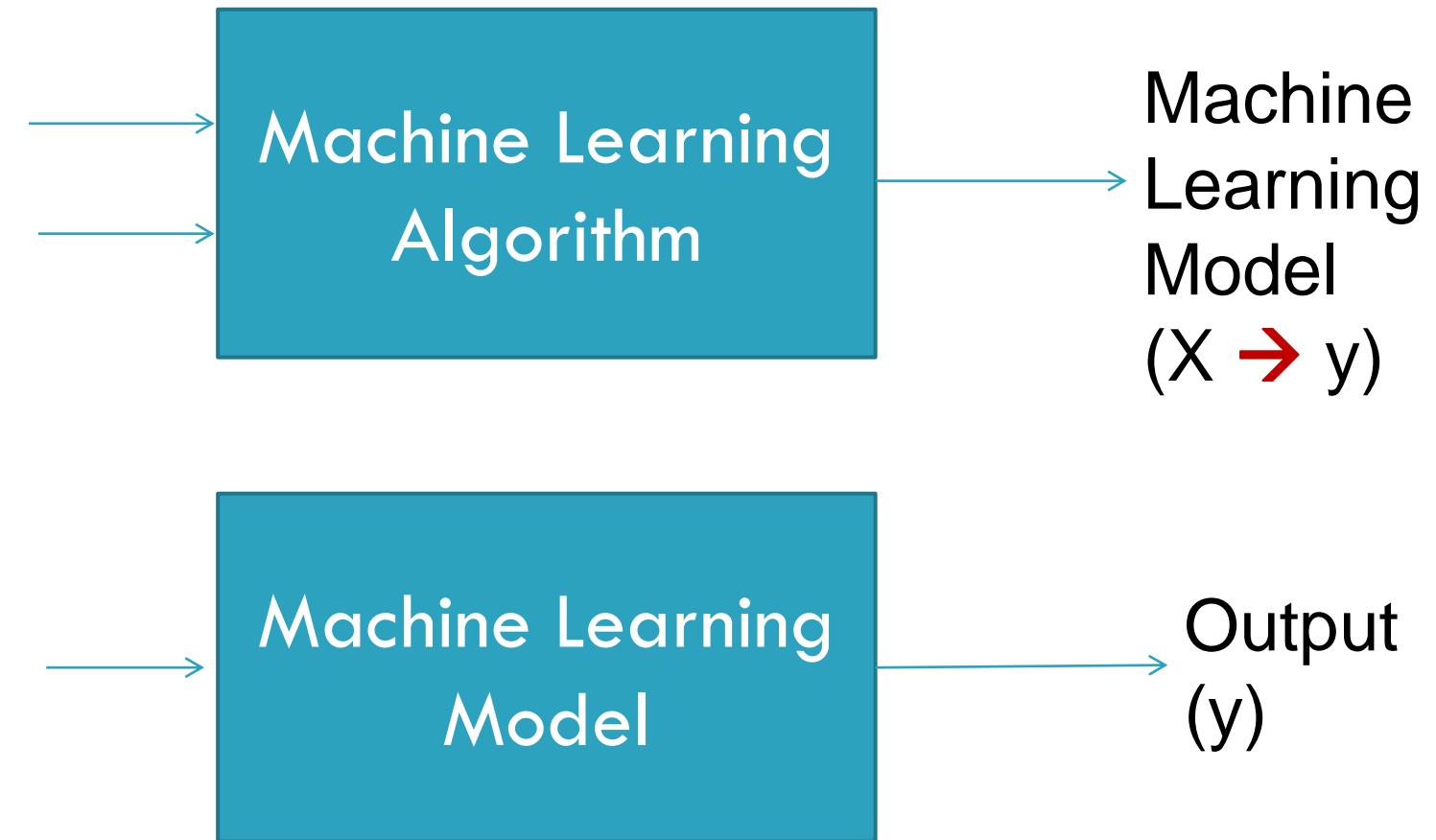




Machine Learning Process



Training Input (X)
Training Output(y)



Applications of Machine Learning



- Social Media: Koo, Facebook, Twitter
- E-commerce App: Flipkart, Amazon etc.
- Government: Railways, Income Tax, etc.
- Banking Application

Types of Machine Learning Problems



- Classification**
- Regression or Estimation**
- Affinity Grouping**
- Clustering**

What kind of problem Machine Learning can solve?



S. No.	Area	bedrooms	bathrooms	stories	parking	price
1	585	3	1	2	no	4200000
2	400	2	1	1	no	3850000
3	306	3	1	1	no	4950000
4	665	3	1	2	yes	6050000
5	636	2	1	1	no	6100000
6	416	3	1	1	yes	6600000
7	388	3	2	2	no	6600000
8	416	3	1	3	no	6900000
9	480	3	1	1	yes	8380000
10	550	3	2	4	yes	8850000
11	720	3	2	1	no	9000000

This is called Regression Problem

Regression/ Estimation



- Calculated approximation of a continuous variable

For Example:

- Estimated household income for loan processing or credit card limit etc.
- Estimated Crop production in an area
- Estimated traffic at a place
- Estimated future requirement for saving/investment
- Admission Prediction

What kind of problem Machine Learning can solve?



S. No.	Area	bedrooms	bathrooms	stories	parking	House Cost
1	585	3	1	2	no	Low
2	400	2	1	1	no	Low
3	306	3	1	1	no	Low
4	665	3	1	2	yes	Medium
5	636	2	1	1	no	Medium
6	416	3	1	1	yes	Medium
7	388	3	2	2	no	Medium
8	416	3	1	3	no	High
9	480	3	1	1	yes	High
10	550	3	2	4	yes	High
11	720	3	2	1	no	High

This is called Classification Problem



- Examining the feature of a newly presented object and assigning it to one of the predefined classes.

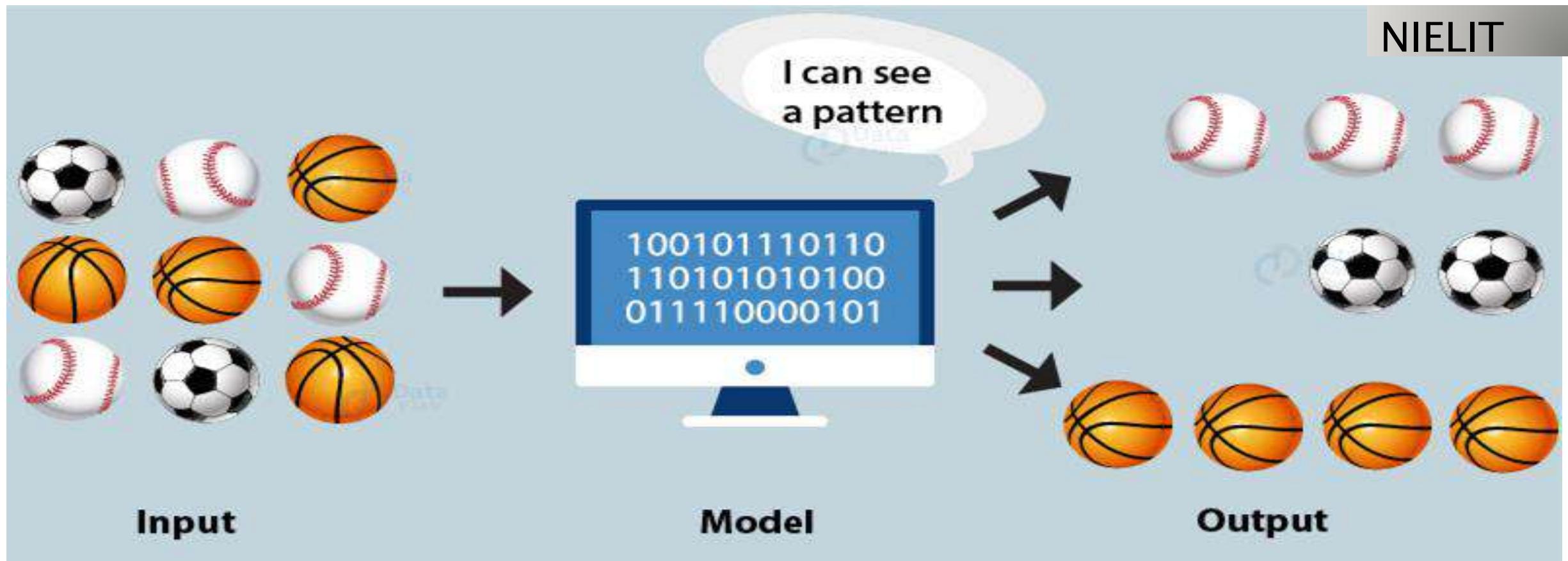
For Example:

- Classifying loan applications as Low, Medium and High Risk
- Assigning product into Categories and sub-categories
- Classifying people as BPL etc.
- Text classification
- Image classification

What kind of problem Machine Learning can solve?



NIELIT



This is called Clustering Problem



- Segmenting heterogeneous group of population into a more homogenous sub groups or clusters.

For Example:

- Customer segmentation according to buying behavior
- Creating cluster of patients with similar symptoms to identify deceases

Affinity Grouping

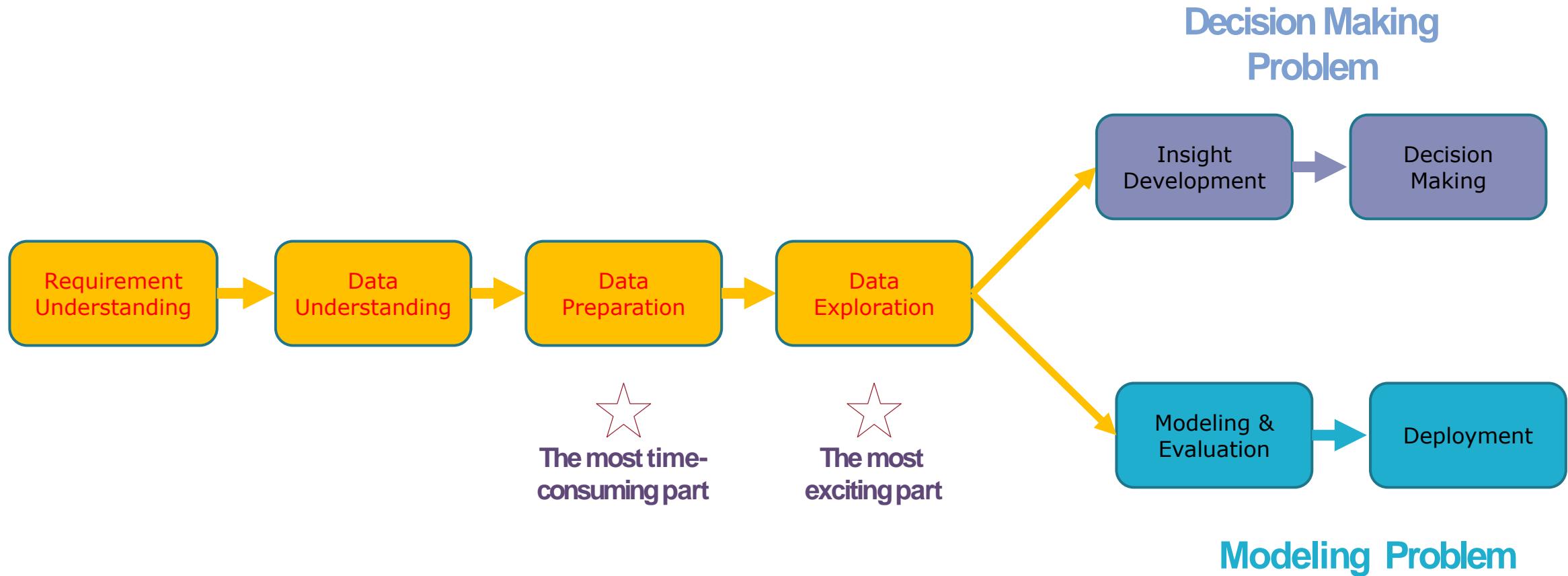


- Which things go together?
- To understand the purchase behaviour of customers
- Market Basket Analysis

For example:

- If someone buys a book on Data Science, it is most likely that he will also buy some book on Python.
- If someone buys Milk, he may buy bread or cornflakes.

Typical Process of Machine Learning



Python as a Programming Language



- Machine Learning
- Web development
- Networking
- Scientific computing
- Data analytics
- Data Visualisation

Python as a Programming Language



The nature of Python makes it a perfect-fit for machine learning:

- Easy to learn
- Readable
- Scalable
- Extensive set of libraries
- Easy integration with other apps
- Active community & ecosystem

Popular Python Data Analytics Libraries



Library	Usage
numpy, scipy	Scientific & technical computing
pandas	Data manipulation & aggregation
mlpy, scikit-learn	Machine learning
theano, tensorflow, keras	Deep learning
statsmodels	Statistical analysis
nltk, gensim	Text processing
networkx	Network analysis & visualization
bokeh, matplotlib, seaborn, plotly	Visualization
beautifulsoup, scrapy	Web scraping

IDE For Python Programming



- Jupyter Notebook
- Spider
- Python IDLE
- Anaconda Distribution: <https://www.anaconda.com/distribution/>

Comparison – R vs. Python



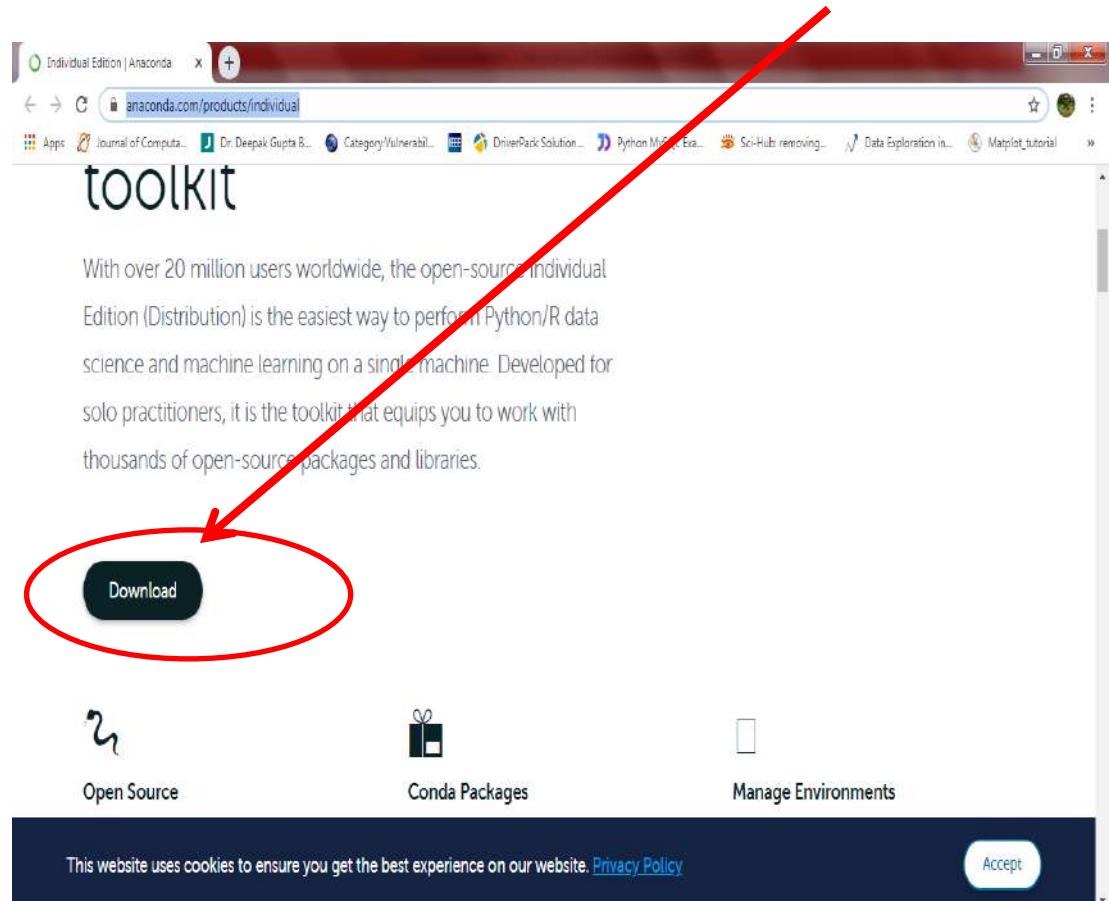
- Comparison between R and Python has been absolutely one of the hottest topics in data science communities:
- R came from the statisticians community, whereas Python came from the computer scientists community
- Python is said to be a challenger against R, but in general it's a tie
- It's up to you to choose the one that best fits your needs

Anaconda Installation

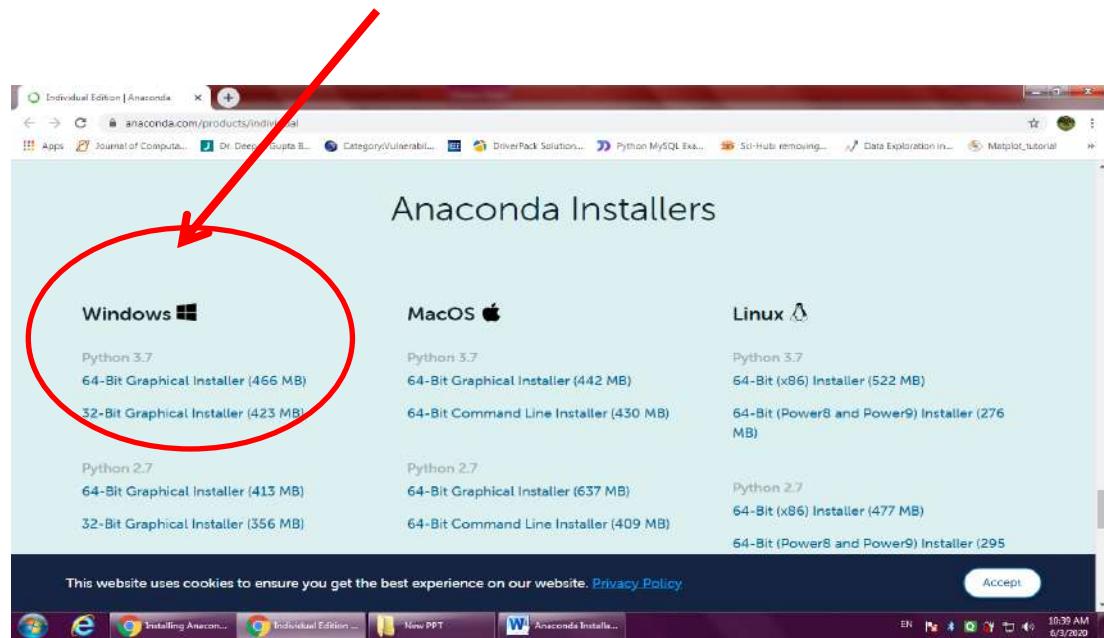
Anaconda is a free and open-source distribution of the Python and R programming languages for scientific computing (data science, machine learning applications, large-scale data processing, predictive analytics, etc.), that aims to simplify package management and deployment.

Installation

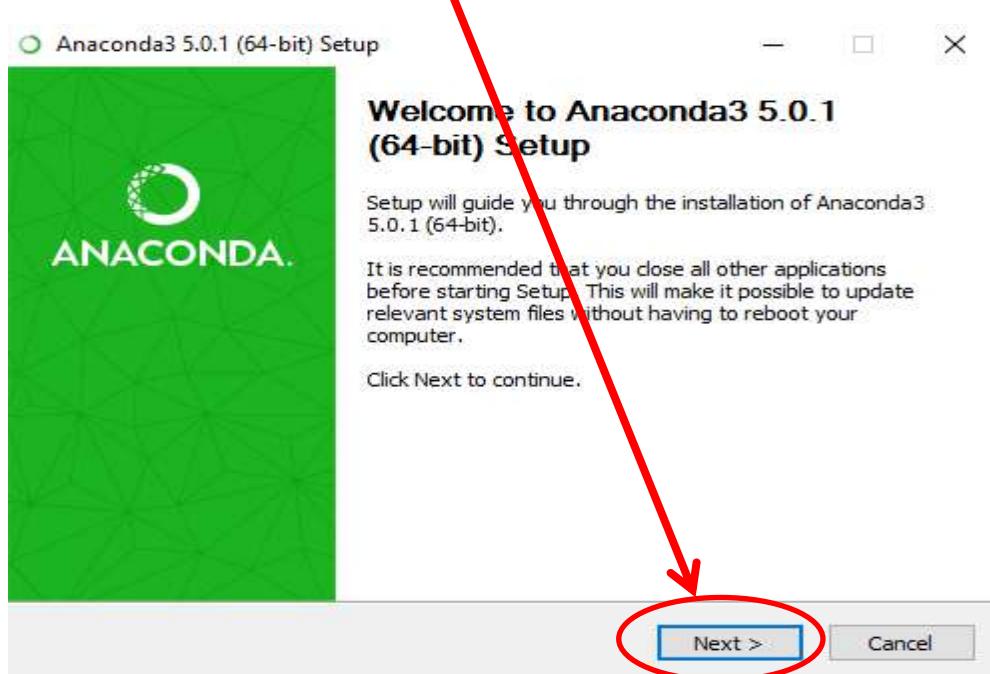
1. Open <https://www.anaconda.com/products/individual> and click download.



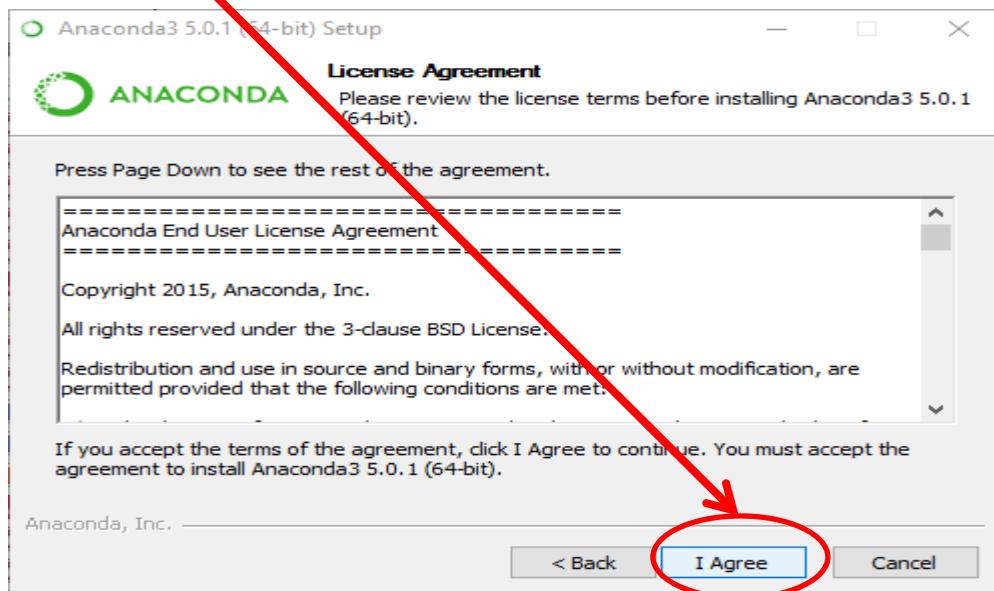
2. In Windows, choose Python 3.7. Click either 64-Bit Graphical Installer or 32-Bit Graphical Installer.
 - a. If your Windows OS is 64 bit, click 64-Bit Graphical Installer.
 - b. If your Windows OS is 32 bit, click 32-Bit Graphical Installer.



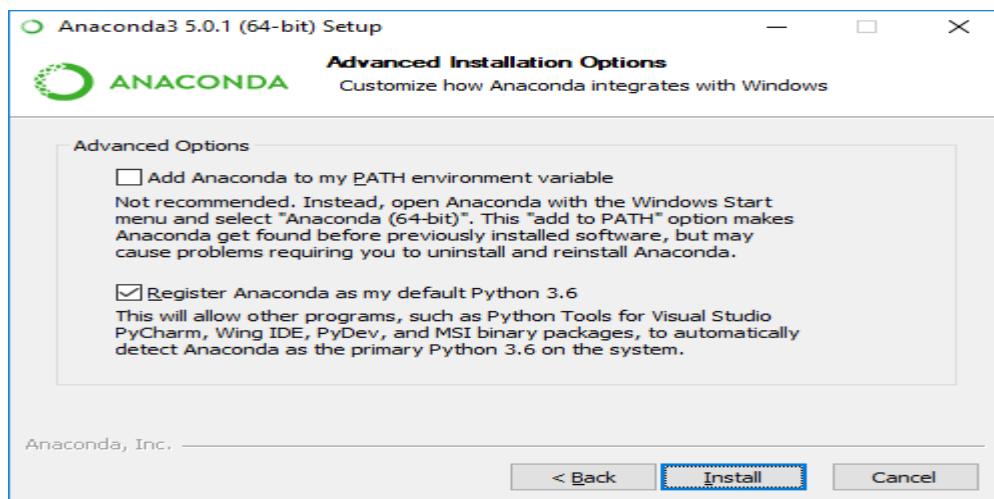
3. Download the .exe Installer.
4. Open and run the installer
Once the download completes, open and run the .exe installer
5. On next screen, click next to begin the installation.



6. Then click on I Agree button to accept the license agreement.

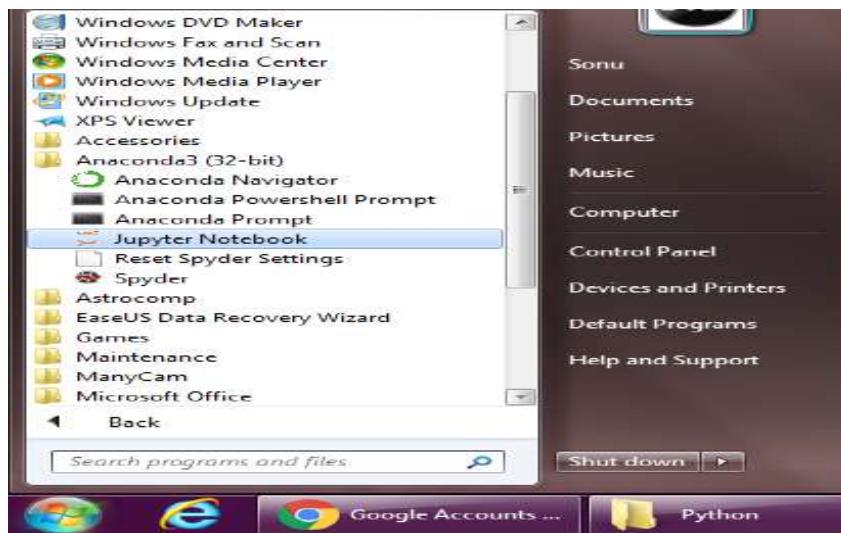


7. At the next screen, it is recommended checking "Add Anaconda to my PATH environment variable". Click on Add Anaconda to my path and then click on Install button.

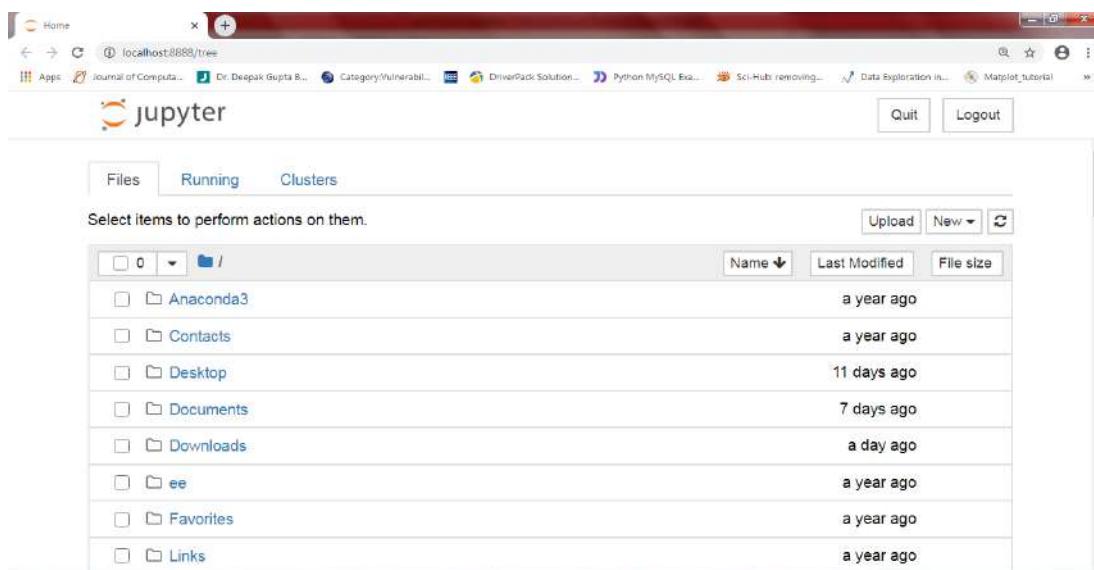


8. Wait, it takes time.

9. After the installation of Anaconda is complete, open Jupyter Notebook.
10. To open Jupyter Notebook, go to Start → All Programs → Anaconda → Jupyter Notebook



11. Jupyter Notebook will be open in your default browser.



Variable in Python



```
x = 5
```

```
y = "John"
```

```
print(x)
```

```
print(y)
```

Variables do not need to be declared with any particular type and can even change type after they have been set.

```
x = 4 # x is of type int
```

```
x = "Sally" # x is now of type str
```

```
print(x)
```

Python Numbers



There are three numeric types in Python:

- int
- float
- complex

Variables of numeric types are created when you assign a value to them:

```
x = 1 # int
```

```
y = 2.8 # float
```

```
z = 1j # complex
```

Python Casting



Integers:

```
x = int(1) # x will be 1
```

```
y = int(2.8) # y will be 2
```

```
z = int("3") # z will be 3
```

FLOATS:

```
x = float(1) # x will be 1.0
```

```
y = float(2.8) # y will be 2.8
```

STRINGS:

```
x = str("s1") # x will be 's1'
```

```
y = str(2) # y will be '2'
```

```
z = str(3.0) # z will be '3.0'
```

Python Program to Add Two Numbers



```
a = input("Enter first number: ")  
b = input("Enter second number: ")
```

```
sum = a + b
```

```
print("sum:", sum)
```

Python Program to Add Two Numbers



```
a = int(input("Enter first number: "))  
b = int(input("Enter second number: "))
```

```
sum = a + b
```

```
print("sum:", sum)
```

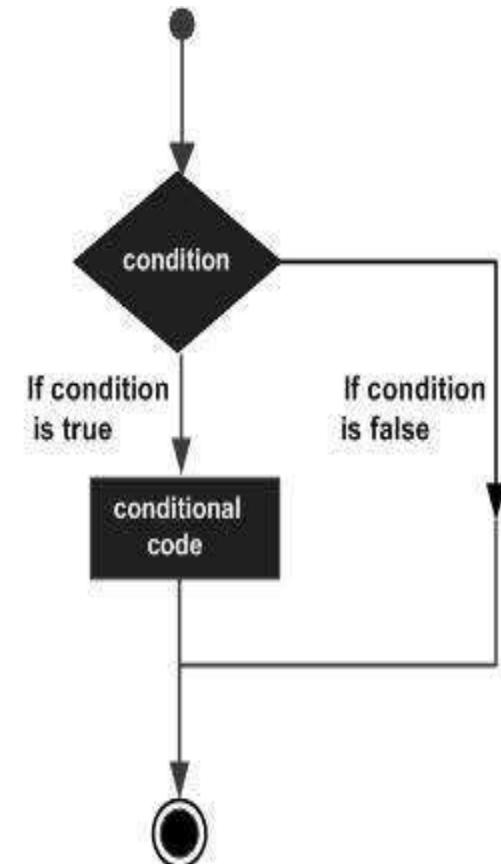
Python - Decision Making



Decision making is anticipation of conditions occurring while execution of the program and specifying actions taken according to the conditions.

Decision structures evaluate multiple expressions which produce TRUE or FALSE as outcome. You need to determine which action to take and which statements to execute if outcome is TRUE or FALSE otherwise.

Following is the general form of a typical decision making structure found in most of the programming languages –



Python IF Statement



It is similar to that of other languages. The if statement contains a logical expression using which data is compared and a decision is made based on the result of the comparison.

Syntax

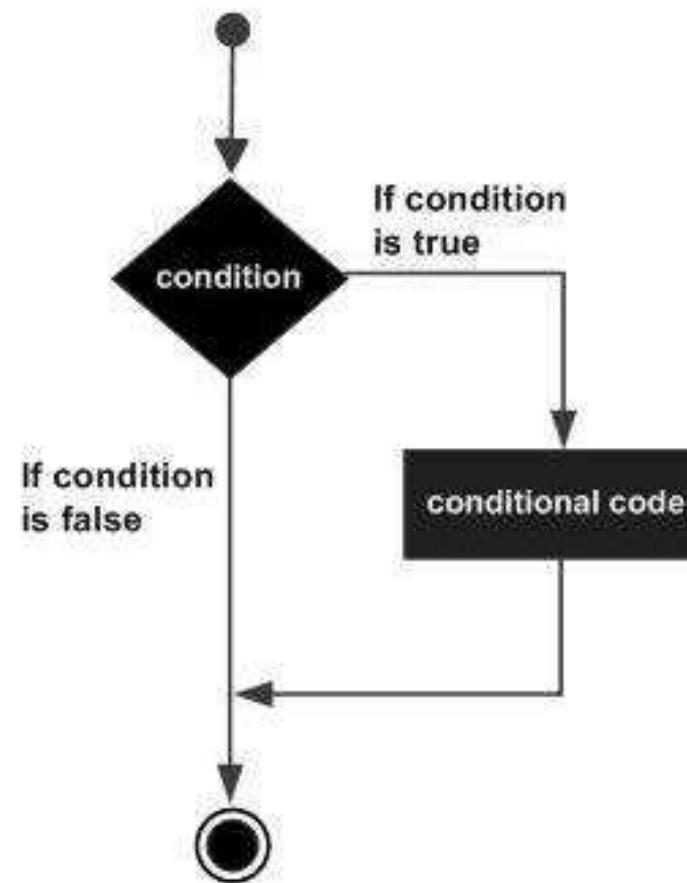
```
if expression:  
    statement(s)
```

If the boolean expression evaluates to TRUE, then the block of statement(s) inside the if statement is executed. If boolean expression evaluates to FALSE, then the first set of code after the end of the if statement(s) is executed.

Python IF Statement



Flow Diagram



Python IF...ELSE Statements



An **else** statement can be combined with an **if** statement. An **else** statement contains the block of code that executes if the conditional expression in the if statement resolves to 0 or a FALSE value.

The **else** statement is an optional statement and there could be at most only one **else** statement following **if**

Syntax

The syntax of the **if...else** statement is –

if expression:

 statement(s)

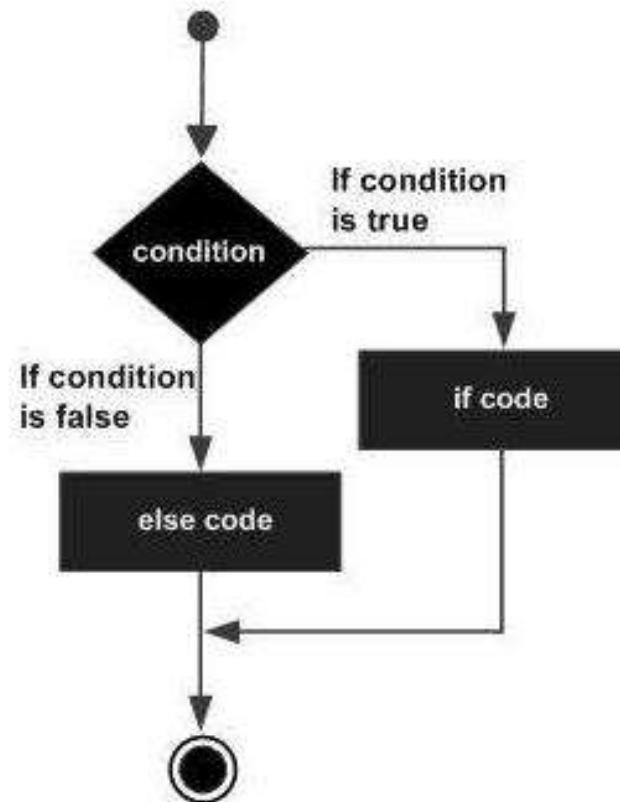
else:

 statement(s)

Python IF...ELSE Statements



Flow Diagram



The **elif** Statement



The **elif** statement allows you to check multiple expressions for TRUE and execute a block of code as soon as one of the conditions evaluates to TRUE.

Similar to the **else**, the **elif** statement is optional. However, unlike **else**, for which there can be at most one statement, there can be an arbitrary number of **elif** statements following an **if**.

syntax

```
if expression1:  
    statement(s)  
elif expression2:  
    statement(s)  
elif expression3:  
    statement(s)  
else:  
    statement(s)
```

Python Nested IF statements



There may be a situation when you want to check for another condition after a condition resolves to true. In such a situation, you can use the nested **if** construct. In a nested **if** construct, you can have an **if...elif...else** construct inside another **if...elif...else** construct.

Syntax

The syntax of the nested **if...elif...else** construct may be –

```
if expression1:  
    statement(s)  
    if expression2:  
        statement(s)  
    elif expression3:  
        statement(s)  
    elif expression4:  
        statement(s)  
    else:  
        statement(s)  
else:  
    statement(s)
```

Python String Manipulation



String literals in python are surrounded by either single quotation marks, or double quotation marks.

'hello' is the same as "hello"

Strings can be output to screen using the print function. For example:
`print("hello")`.

Like many other popular programming languages, strings in Python are arrays of bytes representing unicode characters. However, Python does not have a character data type, a single character is simply a string with a length of 1. Square brackets can be used to access elements of the string.

```
a = "Hello, World!"
```

```
print(a[1])
```

```
b = "Hello, World!"
```

```
print(b[2:5])
```

Operations on String



The `strip()` method removes any whitespace from the beginning or the end:

```
a = "Hello, World!"
```

```
print(a.strip()) # returns "Hello, World!"
```

The `len()` method returns the length of a string:

```
a = "Hello, World!"
```

```
print(len(a))
```

The `lower()` method returns the string in lower case:

```
a = "Hello, World!"
```

```
print(a.lower())
```

Operations on String



The `upper()` method returns the string in upper case:

```
a = "Hello, World!"
```

```
print(a.upper())
```

The `replace()` method replaces a string with another string:

```
a = "Hello, World!"
```

```
print(a.replace("H", "J"))
```

The `split()` method splits the string into substrings if it finds instances of the separator:

```
a = "Hello, World!"
```

```
print(a.split(",")) # returns ['Hello', ' World!'] as array
```

Command-line String Input



Python allows for command line input. That means we are able to ask the user for input.

The following example asks for the user's name, then, by using the `input()` method, the program prints the name to the screen:

Example

```
print("Enter your name:")  
x = input()  
print("Hello, ", x)
```

Updating Strings



You can "update" an existing string by (re)assigning a variable to another string. The new value can be related to its previous value or to a completely different string altogether.
For example –

```
var1 = 'Hello World'  
print ("Updated String :- ", var1[:6] + 'Python')
```

When the above code is executed, it produces the following result –

Updated String :- Hello Python

Escape Characters



Let's say that we want to print the path to the directory C:\nature. Consider what happens when we try to print the path:

```
>>> print ("C:\nature")
```

In the example above, Python interpreted the \n characters as special characters. These characters are not meant to be displayed on the screen; they are used to control the display. In our case, the \n characters were interpreted as a new line.

Escape Characters



The backslash (\) character is used in Python to escape special characters. So in our example, we would use the following code to print the path:

```
>>> print ("C:\\\\nature")  
C:\\nature
```

Escape Characters



List of Escape characters

Backslash notation	Hexadecimal character	Description
\a	0x07	Bell or alert
\b	0x08	Backspace
\cx		Control-x
\C-x		Control-x
\e	0x1b	Escape
\f	0x0c	Formfeed
\M-\C-x		Meta-Control-x

Escape Characters



Backslash notation	Hexadecimal character	Description
\n	0x0a	Newline
\nnn		Octal notation, where n is in the range 0..7
\r	0x0d	Carriage return
\s	0x20	Space
\t	0x09	Tab
\v	0x0b	Vertical tab
\x		Character x

String Special Operators



Operator	Description	Example
+	Concatenation - Adds values on either side of the operator	a = "Hello" and b = "Python" a + b will give HelloPython
*	Repetition - Creates new strings, concatenating multiple copies of the same string	a*2 will give -HelloHello
[]	Slice - Gives the character from the given index	a[1] will give e
[:]	Range Slice - Gives the characters from the given range	a[1:4] will give ell

String Special Operators



Operator	Description	Example
in	Membership - Returns true if a character exists in the given string	H in a will give 1
not in	Membership - Returns true if a character does not exist in the given string	M not in a will give 1

String Formatting Operator



Format Symbol	Conversion
%c	character
%s	string conversion via str() prior to formatting
%i	signed decimal integer
%d	signed decimal integer
%u	unsigned decimal integer
%o	octal integer
%x	hexadecimal integer (lowercase letters)
%X	hexadecimal integer (UPPERcase letters)
%e	exponential notation (with lowercase 'e')
%E	exponential notation (with UPPERcase 'E')
%f	floating point real number
%g	the shorter of %f and %e
%G	the shorter of %f and %E

String Formatting Operator



Example

```
# This prints out "Ram is 20 years old."  
name = "Ram"  
age = 20  
print("%s is %d years old." % (name, age))
```

Triple Quotes



Python's triple quotes comes to the rescue by allowing strings to span multiple lines, including verbatim NEWLINEs, TABs, and any other special characters.

The syntax for triple quotes consists of three consecutive **single or double** quotes.

`para_str = """this is a long string that is made up of several lines and non-printable characters such as TAB (\t) and they will show up that way when displayed. NEWLINEs within the string, whether explicitly given like this within the brackets [\n], or just a NEWLINE within the variable assignment will also show up."""`

```
print para_str
```

Unicode String



Normal strings in Python are stored internally as 8-bit ASCII, while Unicode strings are stored as 16-bit Unicode.

This allows for a more varied set of characters, including special characters from most languages in the world. I'll restrict my treatment of Unicode strings to the following –

```
print(u'Hello, world!')
```

Built-in String Methods



capitalize()

Capitalizes first letter of string

center(width, fillchar)

Returns a space-padded string with the original string centered to a total of width columns.

count(str, beg= 0,end=len(string))

Counts how many times str occurs in string or in a substring of string if starting index beg and ending index end are given.

decode(encoding='UTF-8',errors='strict')

Decodes the string using the codec registered for encoding. encoding defaults to the default string encoding.

encode(encoding='UTF-8',errors='strict')

Returns encoded string version of string; on error, default is to raise a ValueError unless errors is given with 'ignore' or 'replace'.

endswith(suffix, beg=0, end=len(string))

Determines if string or a substring of string (if starting index beg and ending index end are given) ends with suffix; returns true if so and false otherwise.

Built-in String Methods



expandtabs(tabsize=8)

Expands tabs in string to multiple spaces; defaults to 8 spaces per tab if tabsize not provided.

find(str, beg=0 end=len(string))

Determine if str occurs in string or in a substring of string if starting index beg and ending index end are given returns index if found and -1 otherwise.

index(str, beg=0, end=len(string))

Same as find(), but raises an exception if str not found.

isalnum()

Returns true if string has at least 1 character and all characters are alphanumeric and false otherwise.

isalpha()

Returns true if string has at least 1 character and all characters are alphabetic and false otherwise.

isdigit()

Returns true if string contains only digits and false otherwise.

Built-in String Methods



islower()

Returns true if string has at least 1 cased character and all cased characters are in lowercase and false otherwise.

isnumeric()

Returns true if a unicode string contains only numeric characters and false otherwise.

isspace()

Returns true if string contains only whitespace characters and false otherwise.

istitle()

Returns true if string is properly "titlecased" and false otherwise.

isupper()

Returns true if string has at least one cased character and all cased characters are in uppercase and false otherwise.

join(seq)

Merges (concatenates) the string representations of elements in sequence seq into a string, with separator string.

len(string)

Returns the length of the string

Built-in String Methods



ljust(width[, fillchar])

Returns a space-padded string with the original string left-justified to a total of width columns.

lower()

Converts all uppercase letters in string to lowercase.

lstrip()

Removes all leading whitespace in string.

maketrans()

Returns a translation table to be used in translate function.

max(str)

Returns the max alphabetical character from the string str.

min(str)

Returns the min alphabetical character from the string str.

replace(old, new [, max])

Replaces all occurrences of old in string with new or at most max occurrences if max given.

Built-in String Methods



rfind(str, beg=0,end=len(string))

Same as find(), but search backwards in string.

rindex(str, beg=0, end=len(string))

Same as index(), but search backwards in string.

rjust(width[, fillchar])

Returns a space-padded string with the original string right-justified to a total of width columns.

rstrip()

Removes all trailing whitespace of string.

split(str="", num=string.count(str))

Splits string according to delimiter str (space if not provided) and returns list of substrings; split into at most num substrings if given.

splitlines(num=string.count('\n'))

Splits string at all (or num) NEWLINEs and returns a list of each line with NEWLINEs removed.

startswith(str, beg=0,end=len(string))

Determines if string or a substring of string (if starting index beg and ending index end are given) starts with substring str; returns true if so and false otherwise.

Built-in String Methods



strip([chars])

Performs both lstrip() and rstrip() on string.

swapcase()

Inverts case for all letters in string.

title()

Returns "titlecased" version of string, that is, all words begin with uppercase and the rest are lowercase.

translate(table, deletechars="")

Translates string according to translation table str(256 chars), removing those in the del string.

upper()

Converts lowercase letters in string to uppercase.

zfill (width)

Returns original string leftpadded with zeros to a total of width characters; intended for numbers, zfill() retains any sign given (less one zero).

isdecimal()

Returns true if a unicode string contains only decimal characters and false otherwise.

Python Collection



There are four collection data types in the Python programming language:

- **List** - is a collection which is ordered and changeable.
Allows duplicate members.
- **Tuple** - is a collection which is ordered and unchangeable.
Allows duplicate members.
- **Set** - is a collection which is unordered and unindexed. No
duplicate members.
- **Dictionary** -is a collection which is unordered, changeable
and indexed. No duplicate members.

Python Lists



The list is a most versatile data type available in Python which can be written as a list of comma-separated values (items) between square brackets. Important thing about a list is that items in a list need not be of the same type.

Creating a list is as simple as putting different comma-separated values between square brackets. For example –

```
list1 = ['physics', 'chemistry', 1997, 2000]
```

```
list2 = [1, 2, 3, 4, 5 ]
```

```
list3 = ["a", "b", "c", "d"]
```

The list() Constructor



It is also possible to use the list() constructor to make a new list.

Using the list() constructor to make a List:

```
thislist = list(("apple", "banana", "cherry"))
print(thislist)
```

Accessing Values in Lists



To access values in lists, use the square brackets for slicing along with the index or indices to obtain value available at that index.
For example –

```
thislist = ["apple", "banana", "cherry"]  
print(thislist[1])
```

This code show output as “banana”

Loop Through a List



You can loop through the list items by using a for loop:

```
thislist = ["apple", "banana", "cherry"]
```

```
for x in thislist:
```

```
    print(x)
```

Updating Lists



You can update single or multiple elements of lists by giving the slice on the left-hand side of the assignment operator, and you can add to elements in a list with the append() method. For example –

```
thislist = ["apple", "banana", "cherry"]
```

```
thislist[1] = "blackcurrant"
```

```
print(thislist)
```

Check if Item Exists



To determine if a specified item is present in a list use the in keyword:

Check if "apple" is present in the list:

```
thislist = ["apple", "banana", "cherry"]
```

```
if "apple" in thislist:
```

```
    print("Yes, 'apple' is in the fruits list")
```

List Methods



append() Method

The `append()` method appends an element to the end of the list.

Add an element to the fruits list:

```
fruits = ['apple', 'banana', 'cherry']
fruits.append("orange")
```

List Methods



insert() Method

The `insert()` method inserts the specified value at the specified position.

Insert the value "orange" as the second element of the fruit list:

```
fruits = ['apple', 'banana', 'cherry']
```

```
fruits.insert(1, "orange")
```

List Methods



extend() Method

The `extend()` method adds the specified list elements (or any iterable) to the end of the current list.

Add the elements of cars to the fruits list:

```
fruits = ['apple', 'banana', 'cherry']
```

```
cars = ['Ford', 'BMW', 'Volvo']
```

```
fruits.extend(cars)
```

Add a tuple to the fruits list:

```
fruits = ['apple', 'banana', 'cherry']
```

```
points = (1, 4, 5, 9)
```

```
fruits.extend(points)
```

List Methods



pop() Method

The pop() method removes the element at the specified position.

Remove the second element of the fruit list:

```
fruits = ['apple', 'banana', 'cherry']
```

```
fruits.pop(1)
```

If index not specified then delete last node

List Methods



remove() Method

The `remove()` method removes the first occurrence of the element with the specified value.

Remove the "banana" element of the fruit list:

```
fruits = ['apple', 'banana', 'cherry']
```

```
fruits.remove("banana")
```

List Methods



clear() Method

The clear() method removes all the elements from a list.

Remove all elements from the fruits list:

```
fruits = ['apple', 'banana', 'cherry', 'orange']
```

```
fruits.clear()
```

Delete List Elements



To remove a list element, you can use either the `del` statement if you know exactly which element(s) you are deleting or the `remove()` method if you do not know. For example –

```
list1 = ['physics', 'chemistry', 1997, 2000];  
print (list1 )  
del list1[2];  
print ("After deleting value at index 2 : ")  
print (list1)
```

List Methods



reverse() Method

The reverse() method reverses the sorting order of the elements

Reverse the order of the fruit list:

```
fruits = ['apple', 'banana', 'cherry']
```

```
fruits.reverse()
```

Output-['cherry', 'banana', 'apple']

List Methods



sort() Method

The `sort()` method sorts the list ascending by default.

You can also make a function to decide the sorting criteria(s).

Syntax

```
list.sort(reverse=True | False, key=myFunc)
```

Sort the list alphabetically:

```
cars = ['Ford', 'BMW', 'Volvo']
```

```
cars.sort()
```

Parameter	Description
reverse	Optional. <code>reverse=True</code> will sort the list descending. Default is <code>reverse=False</code>
key	Optional. A function to specify the sorting criteria(s)

List Methods



sort() Method

Sort the list by the length of the values:

```
# A function that returns the length of the value:
```

```
def myFunc(e):  
    return len(e)
```

```
cars = ['Ford', 'Mitsubishi', 'BMW', 'VW']
```

```
cars.sort(key=myFunc)
```

Output- ['VW', 'BMW', 'Ford', 'Mitsubishi']

List Methods



sort() Method

Sort a list of dictionaries based on the "year" value of the dictionaries:

```
# A function that returns the 'year' value:
```

```
def myFunc(e):  
    return e['year']
```

```
cars = [  
    {'car': 'Ford', 'year': 2005},  
    {'car': 'Mitsubishi', 'year': 2000},  
    {'car': 'BMW', 'year': 2019},  
    {'car': 'VW', 'year': 2011}  
]  
cars.sort(key=myFunc)
```

List Methods



copy() Method

The `copy()` method returns a copy of the specified list.

Copy the fruits list:

```
fruits = ['apple', 'banana', 'cherry', 'orange']
```

```
x = fruits.copy()
```

List Methods



count() Method

The count() method returns the number of elements with the specified value.

Return the number of times the value "cherry" appears int the fruits list:

```
fruits = ['apple', 'banana', 'cherry']
```

```
x = fruits.count("cherry")
```

Return the number of times the value 9 appears int the list:

```
points = [1, 4, 2, 9, 7, 8, 9, 3, 1]
```

```
x = points.count(9)
```

List Methods



index() Method

The `index()` method returns the position at the first occurrence of the specified value.

What is the position of the value "cherry":

```
fruits = ['apple', 'banana', 'cherry']
x = fruits.index("cherry")
print(x)
```

What is the position of the value 32:

```
fruits = [4, 55, 64, 32, 16, 32]
x = fruits.index(32)
print(x)
```

List Methods



len(list)

It is used to calculate the length of the list.

max(list)

It returns the maximum element of the list.

min(list)

It returns the minimum element of the list.

list(seq)

It converts any sequence to the list.

Python Dictionary



Dictionary in Python is an unordered collection of data values, used to store data values like a map, which unlike other Data Types that hold only single value as an element, Dictionary holds key: value pair. Key value is provided in the dictionary to make it more optimized. Each key-value pair in a Dictionary is separated by a colon :, whereas each key is separated by a ‘comma’.

A Dictionary in Python works similar to the Dictionary in a real world. Keys of a Dictionary must be unique and of immutable data type such as Strings, Integers and tuples, but the key-values can be repeated and be of any type.

Note – Dictionary keys are case sensitive, same name but different cases of Key will be treated distinctly.

Python Dictionary



Create and print a dictionary:

```
thisdict = {  
    "brand": "Ford",  
    "model": "Mustang",  
    "year": 1964  
}  
print(thisdict)
```

Properties of Dictionary Keys



Dictionary values have no restrictions. They can be any arbitrary Python object, either standard objects or user-defined objects. However, same is not true for the keys.

- (a) More than one entry per key not allowed. Which means no duplicate key is allowed. When duplicate keys encountered during assignment, the last assignment wins.
- (b) Keys must be immutable. Which means you can use strings, numbers or tuples as dictionary keys but something like ['key'] is not allowed.

Accessing Items



You can access the items of a dictionary by referring to its key name, inside square brackets:

Example

```
thisdict = {  
    "brand": "Ford",  
    "model": "Mustang",  
    "year": 1964  
}
```

Get the value of the "model" key:

```
x = thisdict["model"]
```

There is also a method called `get()` that will give you the same result:

Get the value of the "model" key:

```
x = thisdict.get("model")
```

Change or Add Elements in a Dictionary



Dictionary are mutable. We can add new items or change the value of existing items using assignment operator.

If the key is already present, value gets updated, else a new key: value pair is added to the dictionary.

```
my_dict = {'name':'Jack', 'age': 26}  
my_dict['age'] = 27  
print(my_dict)  
my_dict['address'] = 'Downtown'  
print(my_dict)
```

Delete or Remove Elements from a Dictionary

We can remove a particular item in a dictionary by using the method `pop()`. This method removes an item with the provided key and returns the value.

The method, `popitem()` can be used to remove and return an arbitrary item (key, value) from the dictionary. All the items can be removed at once using the `clear()` method.

We can also use the `del` keyword to remove individual items or the entire dictionary itself.

Loop Through a Dictionary



You can loop through a dictionary by using a for loop.

When looping through a dictionary, the return value are the keys of the dictionary, but there are methods to return the values as well.

Print all key names in the dictionary, one by one:

```
for x in thisdict:  
    print(x)
```

Print all values in the dictionary, one by one:

```
for x in thisdict:  
    print(thisdict[x])
```

You can also use the values() function to return values of a dictionary:

```
for x in thisdict.values():  
    print(x)
```

Loop through both keys and values, by using the items() function:

```
for x, y in thisdict.items():  
    print(x, y)
```

Check if Key Exists



To determine if a specified key is present in a dictionary use the `in` keyword:

Example

Check if "model" is present in the dictionary:

```
thisdict = {  
    "brand": "Ford",  
    "model": "Mustang",  
    "year": 1964  
}  
if "model" in thisdict:  
    print("Yes, 'model' is one of the keys in the thisdict dictionary")
```

Dictionary Methods



len()

The method **len()** gives the total length of the dictionary. This would be equal to the number of items in the dictionary.

Following is the syntax for **len()** method –

len(dict)

Parameters

dict – This is the dictionary, whose length needs to be calculated.

Return Value

This method returns the length.

Dictionary Methods



str()

The method **str()** produces a printable string representation of a dictionary.

Following is the syntax for **str()** method –

str(dict)

Parameters

dict – This is the dictionary.

Return Value

This method returns string representation.

Dictionary Methods



type()

The method **type()** returns the type of the passed variable. If passed variable is dictionary then it would return a dictionary type.

Following is the syntax for **type()** method –

```
type(dict)
```

Parameters

dict – This is the dictionary.

Return Value

This method returns the type of the passed variable.

Dictionary Methods



clear() Method

The clear() method removes all the elements from a dictionary.

Remove all elements from the car list:

```
car = {  
    "brand": "Ford",  
    "model": "Mustang",  
    "year": 1964  
}  
car.clear()  
print(car)
```

Dictionary Methods



copy() Method

The copy() method returns a copy of the specified dictionary.

Copy the car dictionary:

```
car = {  
    "brand": "Ford",  
    "model": "Mustang",  
    "year": 1964  
}
```

```
x = car.copy()
```

```
print(x)
```

Dictionary Methods



fromkeys() Method

The `fromkeys()` method returns a dictionary with the specified keys and values.

Create a dictionary with 3 keys, all with the value 0:

```
x = ('key1', 'key2', 'key3')  
y = 0
```

```
thisdict = dict.fromkeys(x, y)
```

```
print(thisdict)
```

Dictionary Methods



get() Method

The get() method returns the value of the item with the specified key.

Get the value of the "model" item:

```
car = {  
    "brand": "Ford",  
    "model": "Mustang",  
    "year": 1964  
}
```

```
x = car.get("model")  
print(x)
```

Dictionary Methods



keys() Method

The `keys()` method returns a view object. The view object contains the keys of the dictionary, as a list.

The view object will reflect any changes done to the dictionary, see example below.

Return the keys:

```
car = {  
    "brand": "Ford",  
    "model": "Mustang",  
    "year": 1964  
}  
  
x = car.keys()  
print(x)
```

Dictionary Methods



pop() Method

The pop() method removes the specified item from the dictionary.

Remove "model" from the dictionary:

```
car = {  
    "brand": "Ford",  
    "model": "Mustang",  
    "year": 1964  
}
```

```
car.pop("model")
```

```
print(car)
```

Dictionary Methods



update() Method

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

Insert an item to the dictionary:

```
car = {  
    "brand": "Ford",  
    "model": "Mustang",  
    "year": 1964  
}
```

```
car.update({"color": "White"})  
print(car)
```

Dictionary Methods



values() Method

The values() method returns a view object. The view object contains the values of the dictionary, as a list.

Return the values:

```
car = {  
    "brand": "Ford",  
    "model": "Mustang",  
    "year": 1964  
}  
x = car.values()  
print(x)
```

```
#The extension of jupyter notebook file is .ipynb

#The extenstion of Python file is .py

#Day 2

#Python programming Basics

#You can run a cell in jupyter by clicking or Run or pressing
shift+enter key

print('This is first program in Python')

This is first program in Python

#Declaring a variable

#Variables are declared without using any keyword.

#Syntax: var_name=value

#Ex:

a=90          # variable named 'a' will be of int type

print(a)
print(type(a))

90
<class 'int'>

b=45.78

print(b)
print(type(b))

45.78
<class 'float'>

c='NIELIT'

print(c)
print(type(c))

NIELIT
<class 'str'>

d="Lucknow"
print(type(d))

<class 'str'>
```

*#We can change the value of a previously declared variable
#type of variable will also change accordingly*

```
a=78.56
```

```
print('Value of a:', a)
print('Data Type of a:', type(a))
```

```
Value of a: 78.56
Data Type of a: <class 'float'>
```

```
a='Python'
```

```
print('Value of a:', a)
print('Data Type of a:', type(a))
```

```
Value of a: Python
Data Type of a: <class 'str'>
```

#Rules for variable declaration

#1. variable names may be alphanumeric but can't start with number

```
area123=34.56      #valid
piel=8.9           #valid
```

```
123area=67         #invalid name
```

#2. variable names only contain a-z, A-Z, 0-9 and underscore(_)

```
var1=23  #valid
var_=445 #valid
_var23=900 #valid

_1var=8900 #valid
```

```
var$=90  #invalid
%radius=54 #invalid
```

#3. Multiword variable names must be seperated with '_' or without any space.

#Space is not allowed in multiword variable names

```
areacircle=900    #valid
area_circle=678   #valid
```

```
area circle=89    #invalid
```

#Python is case sensitive language.

```
var=800  
VAR=1500
```

```
print(var)  
print(VAR)
```

```
800  
1500
```

```
print(Var)
```

```
-----  
-----  
NameError                                Traceback (most recent call  
last)  
<ipython-input-15-2388b1543f32> in <module>  
----> 1 print(Var)
```

```
NameError: name 'Var' is not defined
```

```
#car  
#Car  
#caR  
#CaR  
#CAr
```

#Python Data Type

1. Numeric Type
 - a. integer: 1200, 34, 45
 - b. float: 45.62, 65.85, 56.58
 - c. complex: 4+5j
2. Boolean: True, False
3. Sequence Type
 - a. String
 - b. List
 - c. Tuple
 - d. Dictionary
 - e. set

#Operators

*#Arithmetic Operators: +, -, *, /, %*

```
x=10  
y=5
```

```
print('Addition:', x+y)
print('Multiplication:', x*y)
print('Division:', x/y)
print('Substraction:', x-y)
print('Remainder:', x%y)

Addition: 15
Multiplication: 50
Division: 2.0
Substraction: 5
Remainder: 0

#Exponent (**)

p=5
q=3

print(p**q)

125

#floor division(//): it returns the floor value of division operation

r=13
s=5

print('Division:', r/s)
print('Floor Division:', r//s)

Division: 2.6
Floor Division: 2

#floor value

x=3.8 floor value of x=3

y=4.3 floor value of y=4

#Comparison operator

#equal to(==)
#less than(<)
#less than or equal to (<=)
#greator than (>)
#greator than or equal to (>=)
#not equal to (!=)

p=12
q=12
```

```
print(p==q)
```

```
True
```

```
p=12
```

```
q=6
```

```
print(p==q)
```

```
print(p<q)
```

```
print(p<=q)
```

```
print(p>q)
```

```
print(p>=q)
```

```
print(p!=q)
```

```
False
```

```
False
```

```
False
```

```
True
```

```
True
```

```
True
```

#Logical operator

#and

#or

#not

```
a=6
```

```
b=8
```

```
c=10
```

#and

```
print(a>b and a>c)
```

```
print(a<b and a>c)
```

```
print(a<b and a<c)
```

```
False
```

```
False
```

```
True
```

#or

```
a=6
```

```
b=8
```

```
c=10
```

```
print(a>b or a>c)
```

```
print(a<b or a>c)
```

```
print(a<c or a<b)
```

```
False
```

```
True
```

```
True
```

#Logical not

```
p=True
```

```
q=not(p)
```

```
print('value of P=',p)
print('value of P=',q)
```

```
value of P= True
```

```
value of P= False
```

#Logical not

```
p=False
```

```
q=not(p)
```

```
print('value of P=',p)
print('value of Q=',q)
```

```
value of P= False
```

```
value of Q= True
```

#Type casting: process to change the datatype of a variable

#int(): used to cast a variable to integer type

#float(): used to cast a variable to float type

#str(): used to cast a variable to string type

#boolean(): used to cast a variable to boolean type

#complex(): used to cast a variable to complex type

#Ex:

```
w1=45
```

```
print('Value of w1 before casting=', w1)
print('Datatype of w1 before casting=', type(w1))
```

```
#Casting w1 to float type

w1=float(w1)

print('Value of w1 after casting=', w1)
print('Datatype of w1 after casting=', type(w1))

Value of w1 before casting= 45
Datatype of w1 before casting= <class 'int'>
Value of w1 after casting= 45.0
Datatype of w1 after casting= <class 'float'>

w2=56.78

print('Value of w2 before casting=', w2)
print('Datatype of w2 before casting=', type(w2))

#casting w2 to integer type

w2=int(w2)

print('Value of w2 after casting=', w2)
print('Datatype of w2 after casting=', type(w2))

Value of w2 before casting= 56.78
Datatype of w2 before casting= <class 'float'>
Value of w2 after casting= 56
Datatype of w2 after casting= <class 'int'>

w3=50.8958

print('Value of w3 before casting=', w3)
print('Datatype of w3 before casting=', type(w3))

#Ex: casting w3 to string type

w3=str(w3)      #w3="50"

print('Value of w3 after casting=', w3)
print('Datatype of w3 after casting=', type(w3))

Value of w3 before casting= 50.8958
Datatype of w3 before casting= <class 'float'>
Value of w3 after casting= 50.8958
Datatype of w3 after casting= <class 'str'>

#Taking input from keyboard

#input(): used to take input from keyboard
#The default datatype of input values is String
```

```
a=input('Input value of a=')  
  
print('Value of a=', a)  
print('Datatype of a=', type(a))  
  
Input value of a=85.95  
Value of a= 85.95  
Datatype of a= <class 'str'>  
  
a=input('Input value of a=')  
  
print('Value of a=', a)  
print('Datatype of a=', type(a))  
  
Input value of a=ML using Python  
Value of a= ML using Python  
Datatype of a= <class 'str'>  
  
#Type casting is required to input value other than string type  
  
x=input('Enter any Number=')  
x=int(x)  
  
print('Value of x=', x)  
print('Datatype of x=', type(x))  
  
Enter any Number=25  
Value of x= 25  
Datatype of x= <class 'int'>  
  
y=int(input('Enter value of y='))  
  
print('Value of y=', y)  
print('Datatype of y=', type(y))  
  
Enter value of y=85  
Value of y= 85  
Datatype of y= <class 'int'>  
  
#Program to add two numbers using input()  
  
x=int(input('Enter first number='))  
y=int(input('Enter second number='))  
  
z=x+y  
print('sum=', z)  
  
Enter first number=65  
Enter second number=75  
sum= 140
```

```
#Program for area of circle

r=float(input('Enter radius='))
a=3.14*r*r

print('Area of circle=', a)
```

Enter radius=5
Area of circle= 78.5

#Decision Making Statements

#1 If statement

#2 If---else statement

#3 If--Elif--else statemnt

#4 Nested If

#If Statement:

#Syntax:

```
if condition:
    s1
    s2
    .
    .
    sn
```

#Example:

```
data=5
```

```
if(data<=10):
    print('Inside If block')
    print('Data is less than or equal to 10')
```

```
print('Statement after if block')
```

Inside If block
Data is less than or equal to 10
Statement after if block

```
data=15
```

```
if(data<=10):
```

```
print('Inside If block')
print('Data is less than or equal to 10')

print('Statement after if block')
Statement after if block

#If---else statement

n=int(input('Enter any number:'))

if(n%2==0):
    print('Inside If block')
    print('Number is Even')

else:
    print('Inside Else block')
    print('Number is Odd')

Enter any number:17
Inside Else block
Number is Odd

#If---elif--else

#per>=85, Grade=A

#per>=75 and per<85, Grade=B
#per>=60 and per<75, Grade=C
#per>=50 and per<60, Grade=D
#otherwise, Grade=Fail

per=float(input('Enter the percentage of student='))

if(per>=85):
    print('Grade A')

elif(per>=75 and per<85):
    print('Grade B')

elif(per>=60 and per<75):
    print('Grade C')

elif(per>=50 and per<60):
    print('Grade D')
else:
    print('Grade: Fail')
```

```
Enter the percentage of student=45
Grade: Fail
```

#Nested if: if statement within/inside other if statement

#Ex: to check whether a number is positive, negative or zero

```
num=int(input('Enter any number='))

if(num>=0): #Outer If

    if(num==0): #Inner If
        print('Number is Zero')

    else: #else statement of inner if
        print('Number is Positive')

else: #else statement of outer if
    print('Number is Negative')
```

```
Enter any number=-3
Number is Negative
```

#Day 3

#Sequence Data Type

#String: collection of alphabets

#String can be declared using '' , "" or even "*****" / *****

s1='a' #single character string

s2="NIELIT" #multi character string

s3="123" #string

#s4=NIELIT #Not a valid string

```
print(s1)
```

```
print(s2)
```

```
print(s3)
```

```
print(type(s1))
```

```
print(type(s2))
```

```
print(type(s3))
```

a

NIELIT

123

```
<class 'str'>
```

```
<class 'str'>
```

```
<class 'str'>
```

#Indexing

#Logical address of a particular character in string

#Indexing in python starts with 0

#Index is always integer

x="Python"

```
#x[0]=P
```

```
#x[1]=y
```

```
#x[2]=t
```

```
#x[3]=h
```

```
#x[4]=o
```

```
#x[5]=n
```

#We can access the individual character from string using index

```
print('Entire string:', x)
print('First letter in X=', x[0])

print('Second letter in X=', x[1])

print('Fourth letter in x=', x[3])
```

Entire string: Python
First letter in X= P
Second letter in X= y
Fourth letter in x= h

#Python also supports Negative indexing

<i>#Positive Indexing</i>	<i>#Negative Indexing</i>
---------------------------	---------------------------

<i>#x[0]</i>	=P	<i>x[-6]</i>
<i>#x[1]</i>	=y	<i>x[-5]</i>
<i>#x[2]</i>	=t	<i>x[-4]</i>
<i>#x[3]</i>	=h	<i>x[-3]</i>
<i>#x[4]</i>	=o	<i>x[-2]</i>
<i>#x[5]</i>	=n	<i>x[-1]</i>

#We can access the individual elements of string using positive or negative indexing

```
print('First letter of x using positive index=', x[0])
print('First letter of x using negative index=', x[-6])
```

```
print('Last letter of x using positive index=', x[5])
print('Last letter of x using negative index=', x[-1])
```

First letter of x using positive index= P
First letter of x using negative index= P
Last letter of x using positive index= n
Last letter of x using negative index= n

#String slicing

#slice operator (:): used to slice a string/tuple/list

#Syntax: [start_index: end_index]

#start_index: denotes the starting index for slicing. Default is 0.

#end_index: denotes the end index for slicing. Default is last index of string.

```
#end_index is always goes up to end_index-1

st="Machine Learning"

print('Accessing elements/letter from index=2 to index=5:', st[2:6])
print('Accessing elements/letter from index=3 to index=9:', st[3:10])

Accessing elements/letter from index=2 to index=5: chin
Accessing elements/letter from index=3 to index=9: hine Le

#If we omit the start index, then default is 0

print('Accessing elements/letter from index=0 to index=5:', st[0:6])
print('Accessing elements/letter from index=0 to index=5:', st[:6])

Accessing elements/letter from index=0 to index=5: Machin
Accessing elements/letter from index=0 to index=5: Machin

#If we omit the end index, then default is last index of the string

print('Accessing elements/letter from index=8 to index=15:', st[8:16])
print('Accessing elements/letter from index=8 to index=15:', st[8:])

Accessing elements/letter from index=8 to index=15: Learning
Accessing elements/letter from index=8 to index=15: Learning

#String functions

print('Original String:', st)

print('Length of st:', len(st))
print('String in lower case:', st.lower())
print('String in upper case:', st.upper())

Original String: Machine Learning
Length of st: 16
String in lower case: machine learning
String in upper case: MACHINE LEARNING

#Python Lists

#List is the collection of different elements of different datatypes

#List can be declared using [] or list()

#Using []


```

```
data=[23, 45.67, 'Python']

print(data)
print(type(data))

[23, 45.67, 'Python']
<class 'list'>

data2=[12, 34, 56, 67, 89]
print(data2)

[12, 34, 56, 67, 89]

#list(): used to create list in python. It also used to convert/typecast other data as list

b=list((12,34,5,67))
print('List:', b)
print(type(b))

List: [12, 34, 5, 67]
<class 'list'>

#List element can be accessed using indexing
#List supports positive and negative indexing

print(data2)

print('First element of data2 using positive indexing=' , data2[0])
print('First element of data2 using negative indexing=' , data2[-5])

print('Last element of data2 using positive indexing=' , data2[4])
print('Last element of data2 using index indexing=' , data2[-1])

[12, 34, 56, 67, 89]
First element of data2 using postive indexing= 12
First element of data2 using negative indexing= 12
Last element of data2 using postive indexing= 89
Last element of data2 using index indexing= 89

#List also supports Slicing

print("Elements of data2 from index=1 to index=3:" , data2[1:4])
print("Elements of data2 from index=2 to index=4:" , data2[2:5])
print("Elements of data2 from index=0 to index=3:" , data2[:4])
print("Elements of data2 from index=1 to index=end:" , data2[1:])
```

```
Elements of data2 from index=1 to index=3: [34, 56, 67]
Elements of data2 from index=2 to index=4: [56, 67, 89]
Elements of data2 from index=0 to index=3: [12, 34, 56, 67]
Elements of data2 from index=1 to index=end: [34, 56, 67, 89]

#List is mutable data structure

#means we can modify the elements of the list

print(data2)

#Ex: Modifying second element of data2
#Syntax: list_name[index]=new_value

data2[1]=3000

print('Data2 after modification:', data2)
[12, 34, 56, 67, 89]
Data2 after modification: [12, 3000, 56, 67, 89]

#modifying 4th element

data2[3]='ML'

data2
[12, 3000, 56, 'ML', 89]

#modifying elements from index=1 to index=3

data2[1:4]=[1001, 2001, 3001]

data2
[12, 1001, 2001, 3001, 89]

data2
[12, 1001, 2001, 3001, 89]

b
[12, 34, 5, 67]

data2
b

[12, 34, 5, 67]

#Searching in list
```

```
#You can search for any specific data using Membership operator 'in'
course=['ML', "AI", 'IoT', 'Deep Learning', 'Python']

#To search 'Python' in course

if 'Python' in course:
    print('Python course is available')
else:
    print('Python course is not available')

Python course is available

if 'Java' in course:
    print('Java course is available')
else:
    print('Java course is not available')

Java course is not available

course

['ML', 'AI', 'IoT', 'Deep Learning', 'Python']

#List methods

#append(): used to append new data at the end of the list

#Syntax: list_name.append(new_data)

#Adding 'Java' course at the end of course list

course.append('Java')
course

['ML', 'AI', 'IoT', 'Deep Learning', 'Python', 'Java']

#Generally, append() is used to insert single data at the end of the
list

#Ex: adding more than one course in list

course.append(['BBA', 'MBA'])
course

-----
-----
TypeError                                         Traceback (most recent call
last)
<ipython-input-56-16414885317e> in <module>
      3 #Ex: adding more than one course in list
```

```
4
----> 5 course.append('BBA', 'MBA')
6 course

TypeError: append() takes exactly one argument (2 given)
course[6]
['BBA', 'MBA']

#extend(): it also inserts the new data at the end of the list

#Generally, It is used to inset more than one data at the end of the list

#Ex:

course.extend(['BCA', 'MCA'])
course

['ML',
 'AI',
 'IoT',
 'Deep Learning',
 'Python',
 'Java',
 ['BBA', 'MBA'],
 'BCA',
 'MCA']

#insert(): used to insert data at a specific index

#Syntax: list_name.insert(index, new_value)

#EX: insert course='CA' at index=1

course.insert(1, 'CA')
course

['ML',
 'CA',
 'AI',
 'IoT',
 'Deep Learning',
 'Python',
 'Java',
 ['BBA', 'MBA'],
 'BCA',
 'MCA']
```

```
#Deleting the data

#pop(): deletes the data using index
#syntax: list_name.pop(index)

#Ex: to remove data from index=2

course.pop(2)
course

['ML',
 'CA',
 'IoT',
 'Deep Learning',
 'Python',
 'Java',
 ['BBA', 'MBA'],
 'BCA',
 'MCA']

#If index is not given, then pop() will delete the last element

course.pop()
course

['ML', 'CA', 'IoT', 'Deep Learning', 'Python', 'Java', ['BBA', 'MBA'],
 'BCA']

#remove(): deletes the element on the basis of value

#Ex: remove 'Java' from course

course.remove('Java')
course

['ML', 'CA', 'IoT', 'Deep Learning', 'Python', ['BBA', 'MBA'], 'BCA']

#To remove 'Deep Learning'

course.remove('Deep Learning')
course

['ML', 'CA', 'IoT', 'Python', ['BBA', 'MBA'], 'BCA']

#del keyword is also used to delete the data
#data will be deleted using index

#You can use either pop() or del, to delete data using index

#Ex: to delete data at index=2 using del
```

```
del course[2]

#Same as course.pop(2)
course
['ML', 'CA', 'Python', ['BBA', 'MBA'], 'BCA']

#Ex: to delete data at index=3 using del

del course[3]
course
['ML', 'CA', 'Python', 'BCA']

#clear(): deletes the data of the list
#It does not delete the list

course.clear()
course
[]

#To delete data and list using del

del course

course
-----
-----
NameError                               Traceback (most recent call
last)
<ipython-input-67-219d1c244944> in <module>
----> 1 course

NameError: name 'course' is not defined

c=['ML', 'CA', 'Python', 'BBA', 'MBA', 'BCA']
c

['ML', 'CA', 'Python', 'BBA', 'MBA', 'BCA']

#reverse(): reverses the list

c.reverse()
c

['BCA', 'MBA', 'BBA', 'Python', 'CA', 'ML']

#sort()
```

```
#list_name.sort(): sorts the list in ascending order  
  
print('Unsorted list:', c)  
  
c.sort()  
print("Sorted List:", c)  
  
Unsorted list: ['BCA', 'MBA', 'BBA', 'Python', 'CA', 'ML']  
Sorted List: ['BBA', 'BCA', 'CA', 'MBA', 'ML', 'Python']  
  
##list_name.sort(reverse=True): sorts the list in descending order
```

```
c.sort(reverse=True)  
c  
  
['Python', 'ML', 'MBA', 'CA', 'BCA', 'BBA']  
  
r=[21,13,4,17,8]  
r.sort()  
r  
  
[4, 8, 13, 17, 21]
```

```
#index(): to check the index of specific data of the list
```

```
#To check the index of "CA" in list c
```

```
i=c.index('CA')  
  
print('Index of CA is:', i)
```

```
-----  
-----  
ValueError                                Traceback (most recent call  
last)  
<ipython-input-76-e395952c41ee> in <module>  
      3 #To check the index of "CA" in list c  
      4  
----> 5 i=c.index('cA')  
      6  
      7 print('Index of CA is:', i)
```

```
ValueError: 'cA' is not in list
```

```
#To check the index of "BBA" in list c
```

```
i=c.index('BBA')  
  
print('Index of BBA is:', i)  
  
Index of BBA is: 5
```

```
#copy()

a=10
b=a

#In this, no memory will be allocated to b. It will create an
reference of memory a to b

#If we initialize a list to another list using assignment operator(=),
then it will create a reference
#of the list
x=[12,34,56, 'NIELIT']
y=x

#In this case, no seperate memory will be allocated to y.
#This is known as 'Shallow Copy'

print(x)
print(y)

[12, 34, 56, 'NIELIT']
[12, 34, 56, 'NIELIT']

#Modifying the second of x

x[1]=6500

print(x)
print(y)

[12, 6500, 56, 'NIELIT']
[12, 6500, 56, 'NIELIT']

#Modifying third element of y

y[2]='Python'

print(x)
print(y)

[12, 6500, 'Python', 'NIELIT']
[12, 6500, 'Python', 'NIELIT']

#copy(): used to copy the data from one list to another list

#This is known as 'Deep Copy'

#Syntax: new_list=list_name.copy()

z=x.copy()
```

```

#IN this case, separate memory will be allocated to z

print('List x:', x)
print('List z:', z)

List x: [12, 6500, 'Python', 'NIELIT']
List z: [12, 6500, 'Python', 'NIELIT']

#Modifying first element of x

x[0]=1001

print('List x:', x)
print('List z:', z)

List x: [1001, 6500, 'Python', 'NIELIT']
List z: [12, 6500, 'Python', 'NIELIT']

#Modifying fourth element of z

z[3]='Ford'

print('List x:', x)
print('List z:', z)

List x: [1001, 6500, 'Python', 'NIELIT']
List z: [12, 6500, 'Python', 'Ford']

#count(): counts the frequency of any specific data in a list

r1=[14, 12, 25, 12, 25, 12, 10]

print('Frequency of 14 in r1:', r1.count(14))
print('Frequency of 12 in r1:', r1.count(12))

Frequency of 14 in r1: 1
Frequency of 12 in r1: 3

w=[12, 34, 56, 78, 90, 21, 37, 8]

print('Length of w=', len(w))
print('Sum of List w=', sum(w)) #sum() will work only on numeric data
i.e. integer or float

print('Minimum element of List w=', min(w))
print('Maximum element of List w=', max(w))

#min() and max() will work on numeric and text data.

e=['Oppo', 'Samsung', 'Apple']

```

```
print('Min of e:', min(e))
print('Max of e:', max(e))

Length of w= 8
Sum of List w= 336
Minimum element of List w= 8
Maximum element of List w= 90
Min of e: Apple
Max of e: Samsung
```



LOOPS

Python - Loops



A loop statement allows us to execute a statement or group of statements multiple times till a given condition is true.

Python While Loop Statements



A **while** loop statement in Python programming language repeatedly executes a target statement as long as a given condition is true.

Syntax

The syntax of a **while** loop in Python programming language is –
while expression: **statement(s)**

Here, **statement(s)** may be a single statement or a block of statements.
The **condition** may be any expression, and true is any non-zero value.
The loop iterates while the condition is true

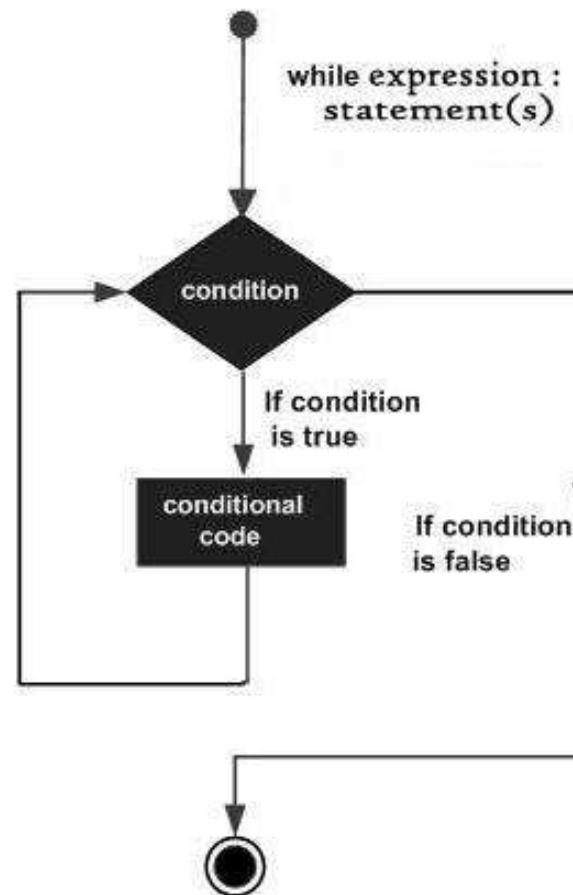
.

When the condition becomes false, program control passes to the line immediately following the loop.

Python While Loop Statements



Flow Diagram



Python While Loop Statements



Example

```
count = 0
while (count < 9):
    print ('The count is:', count)
    count = count + 1
print "Good bye!"
```

The Infinite Loop

```
var = 1
while (var == 1) : # This constructs an infinite loop
    num = input("Enter a number :")
    print("You entered: ", num)
print "Good bye!"
```

Using else Statement with Loops



Python supports to have an **else** statement associated with a **loop** statement.

If the **else** statement is used with a **for** loop, the **else** statement is executed when the loop has exhausted iterating the list.

If the **else** statement is used with a **while** loop, the **else** statement is executed when the condition becomes false.

The while or for loop can be terminated with a **break** statement. In such cases, Else part is ignored. Hence a While loop else part runs if no **break** occurs and the condition is false.

Using else Statement with Loops



The following example illustrates the combination of an else statement with a while statement that prints a number as long as it is less than 5, otherwise else statement gets executed.

Example

```
count = 0
while count < 5:
    print count, " is less than 5"
    count = count + 1
else:
    print count, " is not less than 5"
```

Single Statement Suites



Similar to the **if** statement syntax, if your **while** clause consists only of a single statement, it may be placed on the same line as the while header.

Here is the syntax and example of a **one-line while** clause

—

```
flag = 1
```

```
while (flag): print 'Given flag is really true!'
```

```
print "Good bye!"
```

It is better not try above example because it goes into infinite loop and you need to press CTRL+C keys to exit.

Python for Loop Statements



It has the ability to iterate over the items of any sequence, such as a list or a string.

Syntax

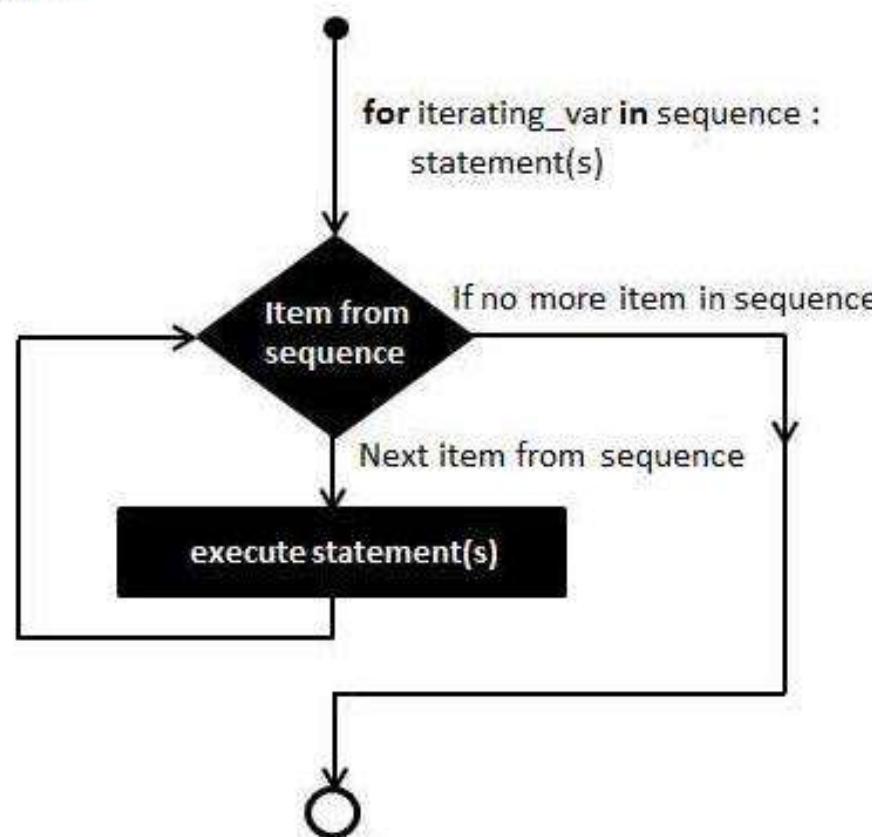
```
for iterating_var in sequence:  
    statements(s)
```

If a sequence contains an expression list, it is evaluated first. Then, the first item in the sequence is assigned to the iterating variable *iterating_var*. Next, the statements block is executed. Each item in the list is assigned to *iterating_var*, and the statement(s) block is executed until the entire sequence is exhausted.

Python for Loop Statements



Flow Diagram



Python for Loop Statements



Example

```
for letter in 'Python':  
    print ('Current Letter :, letter)  
fruits = ['banana', 'apple', 'mango']  
for fruit in fruits:  
    print ('Current fruit :, fruit)  
print ("Good bye!")
```

Iterating by Sequence Index

```
fruits = ['banana', 'apple', 'mango']  
for index in range(len(fruits)):  
    print ('Current fruit :, fruits[index])  
print "Good bye!"
```

Python for Loop Statements



range() Function

The `range()` function returns a sequence of numbers, starting from 0 by default, and increments by 1 (by default), and ends at a specified number.

Ex-

```
x = range(6)
for n in x:
    print(n)
```

Python for Loop Statements



range() Function

Parameter Values

Parameter	Description
start	An integer number specifying at which position to start. Default is 0
stop	An integer number specifying at which position to end.
step	An integer number specifying the incrementation. Default is 1

Python for Loop Statements



range() Function

Example

Create a sequence of numbers from 3 to 5, and print each item in the sequence:

```
x = range(3, 6)
for n in x:
    print(n)
```

Example

Create a sequence of numbers from 3 to 19, but increment by 2 instead of 1:

```
x = range(3, 20, 2)
for n in x:
    print(n)
```

Python nested loops



Python programming language allows to use one loop inside another loop. Following section shows few examples to illustrate the concept.

Syntax

```
for iterating_var in sequence:  
    for iterating_var in sequence:  
        statements(s)  
    statements(s)
```

Python nested loops



Example

The following program uses a nested for loop to find the prime numbers from 2 to 100 –

```
i = 2
while(i < 100):
    j = 2
    while(j <= (i/j)):
        if not(i%j): break
        j = j + 1
    if (j > i/j) : print( i, " is prime")
    i = i + 1
print "Good bye!"
```

Loop Control Statements



break statement Terminates the loop statement and transfers execution to the statement immediately following the loop.

continue statement Causes the loop to skip the remainder of its body and immediately retest its condition prior to reiterating.

Python break statement



```
for i in range(9):
    if i > 3:
        break
    print(i)
```

Python continue statement



```
for i in range(9):
    if i == 3:
        continue
    print(i)
```

Python - Tuples



A tuple is a sequence of immutable Python objects. Tuples are sequences, just like lists. The differences between tuples and lists are, the tuples cannot be changed unlike lists and tuples use parentheses, whereas lists use square brackets.

Creating a tuple is as simple as putting different comma-separated values. Optionally you can put these comma-separated values between parentheses also. For example –

```
tup1 = ('physics', 'chemistry', 1997, 2000)
tup2 = (1, 2, 3, 4, 5 )
```

Accessing Values in Tuples



To access values in tuple, use the square brackets for slicing along with the index or indices to obtain value available at that index.

For example-

Return the item in position 1:

```
thistuple = ("apple", "banana", "cherry")
print(thistuple[1])
```

Change Tuple Values

Once a tuple is created, you cannot change its values.

Loop Through a Tuple



You can loop through the tuple items by using a for loop.

Iterate through the items and print the values:

```
thistuple = ("apple", "banana", "cherry")
```

```
for x in thistuple:
```

```
    print(x)
```

Check if Item Exists



To determine if a specified item is present in a tuple use the in keyword:

Check if "apple" is present in the tuple:

```
thistuple = ("apple", "banana", "cherry")
```

```
if "apple" in thistuple:
```

```
    print("Yes, 'apple' is in the fruits tuple")
```

Find Tuple Length

To determine how many items a tuple has, use the len() method:

Print the number of items in the tuple:

```
thistuple = ("apple", "banana", "cherry")
```

```
print(len(thistuple))
```

Add Items



Once a tuple is created, you cannot add items to it. Tuples are **unchangeable**.

You cannot add items to a tuple:

```
thistuple = ("apple", "banana", "cherry")
```

```
thistuple[3] = "orange" # This will raise an error  
print(thistuple)
```

Remove Items



Tuples are unchangeable, so you cannot remove items from it, but you can delete the tuple completely:

The del keyword can delete the tuple completely:

```
thistuple = ("apple", "banana", "cherry")
```

```
del thistuple
```

```
print(thistuple) #this will raise an error because the  
tuple no longer exists
```

The tuple() Constructor



It is also possible to use the tuple() constructor to make a tuple.

Using the tuple() method to make a tuple:

```
thistuple = tuple(("apple", "banana", "cherry")) #  
note the double round-brackets  
print(thistuple)
```

Basic Tuples Operations



Tuples respond to the + and * operators much like strings; they mean concatenation and repetition here too, except that the result is a new tuple, not a string.

In fact, tuples respond to all of the general sequence operations we used on strings in the prior chapter –

Python Expression	Results	Description
<code>len((1, 2, 3))</code>	3	Length
<code>(1, 2, 3) + (4, 5, 6)</code>	<code>(1, 2, 3, 4, 5, 6)</code>	Concatenation
<code>('Hi!') * 4</code>	<code>('Hi!', 'Hi!', 'Hi!', 'Hi!')</code>	Repetition
<code>3 in (1, 2, 3)</code>	True	Membership
<code>for x in (1, 2, 3): print x,</code>	1 2 3	Iteration

Indexing, Slicing, and Matrixes



Because tuples are sequences, indexing and slicing work the same way for tuples as they do for strings. Assuming following input –

```
L = ('spam', 'Spam', 'SPAM!')
```

Python Expression	Results	Description
L[2]	'SPAM!'	Offsets start at zero
L[-2]	'Spam'	Negative: count from the right
L[1:]	['Spam', 'SPAM!']	Slicing fetches sections

Built-in Tuple Functions



len() Method

The method **len()** returns the number of elements in the tuple.

Following is the syntax for **len()** method –

len(tuple)

Parameters

tuple – This is a tuple for which number of elements to be counted.

Built-in Tuple Functions



max() Method

The method **max()** returns the elements from the tuple with maximum value.

Following is the syntax for **max()** method –
max(tuple)

Parameters

tuple – This is a tuple from which max valued element to be returned.

Built-in Tuple Functions



min() Method

The method **min()** returns the elements from the tuple with minimum value.

Following is the syntax for min() method –

min(tuple)

Parameters

tuple – This is a tuple from which min valued element to be returned.

Where use tuple



Using tuple instead of list is used in the following scenario.

Using tuple instead of list gives us a clear idea that tuple data is constant and must not be changed.

List Vs Tuple



List	Tuple
The literal syntax of list is shown by the [].	The literal syntax of the tuple is shown by the ().
The List is mutable.	The tuple is immutable.
The List has the variable length.	The tuple has the fixed length.
The list provides more functionality than tuple.	The tuple provides less functionality than the list.
The list is used in the scenario in which we need to store the simple collections with no constraints where the value of the items can be changed.	The tuple is used in the cases where we need to store the read-only collections i.e., the value of the items can not be changed. It can be used as the key inside the dictionary.

Nesting List and tuple



We can store list inside tuple or tuple inside the list up to any number of level.

Lets see an example of how can we store the tuple inside the list.

```
Employees =  
[(101, "Ayush", 22), (102, "john", 29), (103, "james", 45), (104, "Ben", 3  
4)]  
print("----Printing list----");  
for i in Employees:  
    print(i)  
Employees[0] = (110, "David",22)  
print("----Printing list after modification----");  
for i in Employees:  
    print(i)
```

Convert List to Tuple



```
l = [4,5,6]
```

```
t=tuple(l)
```

```
print(t)
```

Convert Tuple to List

```
t = (4,5,6)
```

```
l=list(t)
```

```
print(l)
```

Python - Sets



Mathematically a set is a collection of items not in any particular order. A Python set is similar to this mathematical definition with below additional conditions.

- The elements in the set cannot be duplicates.
- The elements in the set are immutable(cannot be modified) but the set as a whole is mutable.
- There is no index attached to any element in a python set. So they do not support any indexing or slicing operation.

Creating a set



A set is created by using the set() function or placing all the elements within a pair of curly braces.

```
Days=set(["Mon","Tue","Wed","Thu","Fri","Sat","Sun"])
```

```
Months={"Jan","Feb","Mar"}
```

```
Dates={21,22,17}
```

```
print(Days)
```

```
print(Months)
```

```
print(Dates)
```

Accessing Values in a Set



We cannot access individual values in a set. We can only access all the elements together as shown above. But we can also get a list of individual elements by looping through the set.

```
Days=set(["Mon","Tue","Wed","Thu","Fri","Sat","Sun"])
for d in Days:
    print(d)
```

Adding Items to a Set



We can add elements to a set by using `add()` method. Again as discussed there is no specific index attached to the newly added element.

```
Days=set(["Mon","Tue","Wed","Thu","Fri","Sat"])
Days.add("Sun")
print(Days)
```

Removing Item from a Set



Remove Item

We can remove elements from a set by using `discard()` method.

```
thisset = {"apple", "banana", "cherry"}  
thisset.remove("banana")  
print(thisset)
```

Note: If the item to remove does not exist, `remove()` will raise an error.

Removing Item from a Set



Remove the item by using the discard() Method

```
Days=set(["Mon","Tue","Wed","Thu","Fri","Sat"])
Days.discard("Sun")
print(Days)
```

Note: If the item to remove does not exist, **discard()** will NOT raise an error.

Union of Sets



The union operation on two sets produces a new set containing all the distinct elements from both the sets. In the below example the element “Wed” is present in both the sets.

```
DaysA = set(["Mon","Tue","Wed"])
```

```
DaysB = set(["Wed","Thu","Fri","Sat","Sun"])
```

```
AllDays = DaysA | DaysB
```

```
print(AllDays)
```

Intersection of Sets



The intersection operation on two sets produces a new set containing only the common elements from both the sets. In the below example the element “Wed” is present in both the sets.

```
DaysA = set(["Mon","Tue","Wed"])
DaysB = set(["Wed","Thu","Fri","Sat","Sun"])
AllDays = DaysA & DaysB
print(AllDays)
```

Difference of Sets



The difference operation on two sets produces a new set containing only the elements from the first set and none from the second set. In the below example the element “Wed” is present in both the sets so it will not be found in the result set.

```
DaysA = set(["Mon","Tue","Wed"])
DaysB = set(["Wed","Thu","Fri","Sat","Sun"])
AllDays = DaysA - DaysB
print(AllDays)
```

#Day4

#Dictionary: Stores data in key:value pair

#Syntax:

#dictionary_name={key1:value1, key2:value2..... key_n:value_n}

#Each key must be associated with one or more key

```
emp={'id': 1001, 'Name':'Rajat', 'course':'ML'}
```

```
print(emp)
print(type(emp))
```

```
{'id': 1001, 'Name': 'Rajat', 'course': 'ML'}
<class 'dict'>
```

#We can not access the elements of dictionary using default indexing
emp[0]

```
-----
-----
KeyError                                Traceback (most recent call
last)
<ipython-input-5-f01a01ae5811> in <module>
      1 #We can not access the elements of dictionary using default
indexing
----> 2 emp[0]
```

KeyError: 0

#We can access the individual key and its associated values

```
print('Id=', emp['id'])
print('Name=', emp['Name'])
print('Course=', emp['course'])
```

```
Id= 1001
Name= Rajat
Course= ML
```

#We can associate more than one value to any key

```
emp2={'id':[101, 102, 103], 'name':['Rajat', 'Jatin', 'Ravi'], 'ph':
[123, 456, 789]}
emp2
```

```
{'id': [101, 102, 103],
 'name': ['Rajat', 'Jatin', 'Ravi'],
 'ph': [123, 456, 789]}
```

```
print('Id of first employee= ', emp2['id'][0])
print('Name of First Employee= ', emp2['name'][0])
print('Phone number of first Employee= ', emp2['ph'][0])

Id of first employee= 101
Name of First Employee= Rajat
Phone number of first student= 123

print('Id of Second Employee= ', emp2['id'][1])
print('Name of Second Employee= ', emp2['name'][1])
print('Phone number of Second Employee= ', emp2['ph'][1])

Id of Second Employee= 102
Name of Second Employee= Jatin
Phone number of Second Employee= 456

print('Id of Third Employee= ', emp2['id'][2])
print('Name of Third Employee= ', emp2['name'][2])
print('Phone number of Third Employee= ', emp2['ph'][2])

Id of Third Employee= 103
Name of Third Employee= Ravi
Phone number of Third Employee= 789
```

#Dictionary is mutable data structure

#We can modify/update any value of a dictionary

#Ex: updating the id of first employee

```
emp2['id'][0]=1001
```

emp2

```
{'id': [1001, 102, 103],
 'name': ['Rajat', 'Jatin', 'Ravi'],
 'ph': [123, 456, 789]}
```

#Ex: updating name of second employee

```
emp2['name'][1]='Arvind'
emp2
```

```
{'id': [1001, 102, 103],
 'name': ['Rajat', 'Arvind', 'Ravi'],
 'ph': [123, 456, 789]}
```

#Ex: updating phone number of third employee

```
emp2['ph'][2]=9897
```

emp2

```
{'id': [1001, 102, 103],  
 'name': ['Rajat', 'Arvind', 'Ravi'],  
 'ph': [123, 456, 9897]}  
#Adding a new key  
#Adding dept key  
  
emp2['Dept']=['IT', 'Finance', 'Accounts']  
emp2  
  
{'id': [1001, 102, 103],  
 'name': ['Rajat', 'Arvind', 'Ravi'],  
 'ph': [123, 456, 9897],  
 'Dept': ['IT', 'Finance', 'Accounts']}  
  
#Adding 'city' key  
  
emp2['City']= ['LK0', 'KNP', 'DEL']  
emp2  
  
{'id': [1001, 102, 103],  
 'name': ['Rajat', 'Arvind', 'Ravi'],  
 'ph': [123, 456, 9897],  
 'Dept': ['IT', 'Finance', 'Accounts'],  
 'City': ['LK0', 'KNP', 'DEL']}  
  
#keys(): used to view the keys of a dictionary  
  
emp2.keys()  
  
dict_keys(['id', 'name', 'ph', 'Dept', 'City'])  
  
#values(): used to view the values of an dictionary  
  
emp2.values()  
  
dict_values([[1001, 102, 103], ['Rajat', 'Arvind', 'Ravi'], [123, 456, 9897], ['IT', 'Finance', 'Accounts'], ['LK0', 'KNP', 'DEL']])  
  
#Checking key  
  
#to check 'id' key  
  
if 'id' in emp2:  
    print('ID key is available')  
else:  
    print('ID key is not available')  
  
ID key is available
```

```
if 'address' in emp2:  
    print('Address key is available')  
else:  
    print('Address key is not available')
```

Address key is not available

#Looping

#Types

#while loop

#for loop

#while loop
#It is a pre-tested loop

#Syntax:

```
#initialization  
#while(conditon):  
    #stmt  
    #stmt  
  
    #n_stmt  
    #increment/decrement
```

#Ex: Print 'NIELIT Luckow' 5 times using while loop

```
i=0  
while(i<5):  
    print('NIELIT Lucknow')  
    i=i+1
```

NIELIT Lucknow
NIELIT Lucknow
NIELIT Lucknow
NIELIT Lucknow
NIELIT Lucknow

```
i=1  
while(i<6):  
    print('NIELIT Lucknow')  
    i=i+1
```

NIELIT Lucknow
NIELIT Lucknow
NIELIT Lucknow

```
NIELIT Lucknow  
NIELIT Lucknow
```

```
i=1  
while(i<=5):  
    print('NIELIT Lucknow')  
    i=i+1
```

```
NIELIT Lucknow  
NIELIT Lucknow  
NIELIT Lucknow  
NIELIT Lucknow  
NIELIT Lucknow
```

#To print 1 to 5 using while

```
i=1  
while(i<=5):  
    print('I=', i)  
    i=i+1
```

```
I= 1  
I= 2  
I= 3  
I= 4  
I= 5
```

#To print 5 to 1 using while

```
i=5  
while(i>=1):  
    print('I=', i)  
    i=i-1
```

```
I= 5  
I= 4  
I= 3  
I= 2  
I= 1
```

#infinite while loop

```
...  
i=5  
while(i>=1):  
    print('I=', i)  
...  
"\ni=5\nwhile(i>=1):\n    print('I=', i)\n"
```

#else statement with while loop

#Else statement is optional

```
i=1
while(i<=5):
    print('Inside while loop')
    print('I=' , i)
    i=i+1

else:
    print('While loop is executed successfully')
```

```
Inside while loop
I= 1
Inside while loop
I= 2
Inside while loop
I= 3
Inside while loop
I= 4
Inside while loop
I= 5
While loop is executed successfully
```

#Single line while loop

#To print 5 to 1 using while

```
i=5
while(i>=1):
    print('I=' , i)
    i=i-1

I= 5
I= 4
I= 3
I= 2
I= 1
```

#If your while loop executes only one statement, then it can be written in same line.

#Since, above while loop executes only one statement. This while loop can be written as single line
#while loop

```
i=5
while(i>=1):print('I=' , i);i=i-1

I= 5
I= 4
I= 3
```

```
I= 2  
I= 1
```

#For loop

#We can access the individual data of a list/tuple/dictionary sequentially using for loop

#Syntax: for var_name in sequence_var_name:

```
li=[12,34,56,78,90]  
print(li)
```

```
[12, 34, 56, 78, 90]
```

#Access element of li using for loop

```
for a in li:  
    print(a)
```

```
12  
34  
56  
78  
90
```

#Print square of each member of li list

```
for t in li:  
    print(t*t)
```

```
144  
1156  
3136  
6084  
8100
```

```
for t in li:  
    if(t==78):  
        print(t*t)
```

```
6084
```

```
s='Machine Learning'  
print(s)
```

```
Machine Learning
```

```
for j in s:  
    print(j)
```

M
a
c
h
i
n
e

L
e
a
r
n
i
n
g

#range(): it is used to generate the sequence of data within a given range

#Syntax: range([start], stop, [step])

#Here, start and step are optional parameter and stop is required parameter

#start: specifies the starting number of the sequence. Default is zero

#stop: specifies the last number of the range. stop always goes up stop-1

#step: It can be positive or negative. Default +1

#range(1,6): 1,2,3,4,5

#range(5):0,1,2,3,4

#range(1,10,2): 1,3,5,7,9

#range(2,10,3): 2, 5, 8

#range(5,0, -1): 5,4,3,2,1

#range(10,1, -2): 10, 8, 6, 4, 2

#Ex: To print 'NIELIT Lucknow' 5 times using for loop

```
for i in range(1,6):
    print('NIELIT Lucknow')
```

```
NIELIT Lucknow  
NIELIT Lucknow  
NIELIT Lucknow  
NIELIT Lucknow  
NIELIT Lucknow
```

#Ex: To print 'NIELIT Lucknow' 5 times using for loop

```
for i in range(5):  
    print('NIELIT Lucknow')
```

```
NIELIT Lucknow  
NIELIT Lucknow  
NIELIT Lucknow  
NIELIT Lucknow  
NIELIT Lucknow
```

#To print r=1 to 5

```
for r in range(1,6):  
    print('R=' , r)
```

```
R= 1  
R= 2  
R= 3  
R= 4  
R= 5
```

#To print 1 to 10 using for loop

#start=1, stop=11, step=1 (Default)

```
for k in range(1,11):  
    print('K=' , k)
```

```
K= 1  
K= 2  
K= 3  
K= 4  
K= 5  
K= 6  
K= 7  
K= 8  
K= 9  
K= 10
```

#To print 10 to 1 using for loop

#start=10, stop=0, step=-1

```
for k in range(10, 0, -1):  
    print('K=', k)
```

```
K= 10  
K= 9  
K= 8  
K= 7  
K= 6  
K= 5  
K= 4  
K= 3  
K= 2  
K= 1
```

```
#start=15, stop=1, step=-3
```

```
for t in range(15,1,-3):  
    print(t)
```

```
15  
12  
9  
6  
3
```

```
#Printing the table using for loop
```

```
num=int(input('Enter any number='))
```

```
for i in range(1,11):  
    print(i*num)
```

```
Enter any number=5  
5  
10  
15  
20  
25  
30  
35  
40  
45  
50
```

```
#Printing the table using for loop
```

```
num=int(input('Enter any number='))
```

```
for i in range(1,11):  
    print(num, "*", i, '=', num*i)
```

```
Enter any number=3
3 * 1 = 3
3 * 2 = 6
3 * 3 = 9
3 * 4 = 12
3 * 5 = 15
3 * 6 = 18
3 * 7 = 21
3 * 8 = 24
3 * 9 = 27
3 * 10 = 30
```

#WAP to insert 5 element of integer in list from keyboard

```
data=[]
```

```
for i in range(5):
    num=int(input('Enter a number=' ))
    data.append(num)

print('List:', data)
```

```
Enter a number=12
Enter a number=35
Enter a number=96
Enter a number=91
Enter a number=45
List: [12, 35, 96, 91, 45]
```

#Program to append even number in one list and append odd number in another list

```
l1=[10, 8, 13, 21, 17, 19, 24]
```

```
even=[]
odd=[]
```

```
for a in l1:
    if(a%2==0):
        even.append(a)
    else:
        odd.append(a)
```

```
print('Original List:', l1)
print('List of Even numbers=',even)
print('List of Odd numbers=', odd)
```

```
Original List: [10, 8, 13, 21, 17, 19, 24]
List of Even numbers= [10, 8, 24]
List of Odd numbers= [13, 21, 17, 19]
```

#Loop control statement

#break

#continue

#break: immediately breaks/stops the execution of the for/while loop

```
for i in range(5):
    print(i)

print('Outside for loop')

0
1
2
3
4
Outside for loop
```

#Unconditional break

```
for i in range(5):
    print(i)
    break

print('Outside for loop')

0
Outside for loop
```

#Conditional break

```
for i in range(5):
    print(i)
    if(i==3):
        break

print('Outside for loop')

0
1
2
3
Outside for loop
```

#Continue: it skips the execution of for loop

```
for i in range(5):
    print('Value of I=', i)
```

```
print('Inside for loop')
```

```
Value of I= 0  
Inside for loop  
Value of I= 1  
Inside for loop  
Value of I= 2  
Inside for loop  
Value of I= 3  
Inside for loop  
Value of I= 4  
Inside for loop
```

#continue will skip the statements which are written after continue statement in for/while loop

#Unconditional continue

```
for i in range(5):  
    print('Value of I=', i)  
    continue  
    print('Inside for loop')
```

```
print('Hello')
```

```
Value of I= 0  
Value of I= 1  
Value of I= 2  
Value of I= 3  
Value of I= 4
```

#conditional continue

```
for i in range(5):  
    print('Value of I=', i)  
    if(i==2):  
        continue  
    print('Inside for loop')
```

```
Value of I= 0  
Inside for loop  
Value of I= 1  
Inside for loop  
Value of I= 2  
Value of I= 3  
Inside for loop  
Value of I= 4  
Inside for loop
```

```
#Program for prime number

n=int(input('Enter any number:'))

for i in range(2, n):
    if(n%i==0):
        print('Number is Not Prime')
        break
else:
    print('Number is Prime')
```

```
Enter any number:6
Number is Not Prime
```

#Day5

#Tuple: similar to list. It is the collection of diff. elements of diff. datatypes

#Tuples can be declared using

#()

#tuple()

```
t1=(34.56, 56, 'Machine Learning')
print(t1)
print(type(t1))
```

```
(34.56, 56, 'Machine Learning')
<class 'tuple'>
```

#tuple(): you can declare a new tuple or typecast a list to tuple.

```
t2=tuple((12,34,56,78))
print(t2)
print(type(t2))
```

```
(12, 34, 56, 78)
<class 'tuple'>
```

#Accessing elements of a tuple using index

```
marks=(89, 76, 56, 90, 67, 65, 59, 70)
```

```
print(marks)
print('First marks:', marks[0])
print('Second marks:', marks[1])
print('Fifth Marks:', marks[4])
```

```
(89, 76, 56, 90, 67, 65, 59, 70)
First marks: 89
Second marks: 76
Fifth Marks: 67
```

#Slicing

```
print('Element from index=1 to index=4:', marks[1:5])
print('Element from index=3 to index=7:', marks[3:8])
```

```
Element from index=1 to index=4: (76, 56, 90, 67)
Element from index=3 to index=7: (90, 67, 65, 59, 70)
```

#Accessing tuple using for loop

```
for a in marks:  
    print('Marks:', a)  
  
Marks: 89  
Marks: 76  
Marks: 56  
Marks: 90  
Marks: 67  
Marks: 65  
Marks: 59  
Marks: 70  
  
print(marks)  
  
print('Length of marks:', len(marks))  
print('Sum of marks:', sum(marks))  
print('Minimum of marks:', min(marks))  
print('Maximum of marks:', max(marks))
```

(89, 76, 56, 90, 67, 65, 59, 70)
Length of marks: 8
Sum of marks: 572
Minimum of marks: 56
Maximum of marks: 90

#Tuple is immutable data structure.

#We can not insert/update or delete data from tuple

#Updating marks at index=3

marks[3]=98

```
-----  
TypeError Traceback (most recent call  
last)  
<ipython-input-16-34f6968bbca8> in <module>  
      1 #Updating marks at index=3  
      2  
----> 3 marks[3]=98
```

`TypeError: 'tuple' object does not support item assignment`

```
marks.append(78)
```

```
<ipython-input-17-c54ad4cb92a0> in <module>
----> 1 marks.append(78)

AttributeError: 'tuple' object has no attribute 'append'

#You can not delete single element from tuple using index or data

#Deleting second element from marks

del marks[1]

-----
-----
TypeError                                         Traceback (most recent call
last)
<ipython-input-18-a8bf568d0dc0> in <module>
      3 #Deleting second element from marks
      4
----> 5 del marks[1]

TypeError: 'tuple' object doesn't support item deletion

#we can only delete a tuple completely

del marks

marks

-----
-----
NameError                                         Traceback (most recent call
last)
<ipython-input-20-7942852603bd> in <module>
----> 1 marks

NameError: name 'marks' is not defined

#Type casting list to tuple

l1=[12, 34, 56]
t12=tuple(l1)

print(t12)
print(type(t12))

(12, 34, 56)
<class 'tuple'>

#Nesting of tuple with list

li_tup=[(1, 'Ravi'), (2, 'Sachin'), (3, 'Rajesh')]
```

```

print(li_tup)
print(type(li_tup))

[(1, 'Ravi'), (2, 'Sachin'), (3, 'Rajesh')]
<class 'list'>

#If we nest the tuple with list, then we can modify the elements of a tuple

li_tup[0]=(1001, 'Rajat')
li_tup

[(1001, 'Rajat'), (2, 'Sachin'), (3, 'Rajesh')]

data=(1,2,3,4)
data2=list(data)

data2

[1, 2, 3, 4]

data2[0]=1000
data2

[1000, 2, 3, 4]

#Nesting list with tuple

tup_li=([1, 'Rajat'], [2, 'Saurabh'], [3, 'Jatin'])

print(tup_li)
print(type(tup_li))

([1, 'Rajat'], [2, 'Saurabh'], [3, 'Jatin'])
<class 'tuple'>

#If we nest a list into tuple, then modification is not allowed
tup_li[0]=[1001, 'Gaurav']

-----
-----
TypeError                                     Traceback (most recent call
last)
<ipython-input-28-8806b288e296> in <module>
      1 #If we nest a list into tuple, then modification is not
allowed
      2 tup_li[0]=[1001, 'Gaurav']

TypeError: 'tuple' object does not support item assignment

```

```
#set: collection of diff. elements of diff datatypes  
  
#Set can be declared using: set() or {}  
  
#Ex: declaring a set using set()
```

```
s1=set([12, 34, 56.78, 'Python'])  
print(s1)  
print(type(s1))
```

```
{56.78, 34, 12, 'Python'}  
<class 'set'>
```

```
s1  
{12, 34, 56.78, 'Python'}
```

```
#Ex: declaring a set using {}
```

```
s2={'ML', 'AI', 'IoT', 45.67, 89.88}
```

```
print(s2)  
print(type(s2))
```

```
{'AI', 45.67, 89.88, 'IoT', 'ML'}  
<class 'set'>
```

```
#Python set doesn't support indexing
```

```
#We can not access the data from any given index
```

```
s1[0]
```

```
-----  
----  
TypeError                                Traceback (most recent call  
last)  
<ipython-input-32-b041db659b8b> in <module>  
      3 #We can not access the data from any given index  
      4  
----> 5 s1[0]
```

```
TypeError: 'set' object is not subscriptable
```

```
#Access data using for loop  
for s in s1:  
    print(s)
```

```
56.78  
34
```

12
Python

#We can not modify the existing data in a set

```
s1[0]='Language'
```

```
-----
-----
TypeError                                Traceback (most recent call
last)
<ipython-input-34-ffd5c6eeb870> in <module>
      1 #We can not modify the existing data in a set
      2
----> 3 s1[0]='Language'
      4
```

TypeError: 'set' object does not support item assignment

```
print(s2)
```

```
{'AI', 45.67, 89.88, 'IoT', 'ML'}
```

#Set functions

#We can insert new data in a set using: add() or update()

#add(): used to add/insert new data in a set

#using add(), you can only insert single data

#Adding 1200 in s2

```
s2.add(1200)
s2
```

```
{1200, 45.67, 89.88, 'AI', 'IoT', 'ML'}
```

#Adding 'Python' in s2

```
s2.add('Python')
s2
{1200, 45.67, 89.88, 'AI', 'IoT', 'ML', 'Python'}
```

#update(): used to insert/add new data in a set. Data must be given as a sequence type

#You can add/insert more than one data using update()

```
#Adding 'Supervised ML' and 'Unsupervised ML' in s2
```

```
s2.update(['Supervised ML' , 'Unsupervised ML' ])
s2
```

```
{1200,
 45.67,
89.88,
'AI',
'IoT',
'ML',
'Python',
'Supervised ML',
'Unsupervised ML'}
```

```
#We can delete the specific data from a set using
```

```
#discard(): used to remove data from set
#Data is deleted on the basis of values
```

```
#Ex: To remove 'AI'
```

```
s2.discard('AI')
s2
```

```
{1200, 45.67, 89.88, 'IoT', 'ML', 'Python', 'Supervised ML',
'Unsupervised ML'}
```

```
#Ex: Deleting 1200
```

```
s2.discard(1200)
s2
```

```
{45.67, 89.88, 'IoT', 'ML', 'Python', 'Supervised ML', 'Unsupervised
ML'}
```

```
#If data to be deleted is not available, then discard() will not show
any error
```

```
#Ex: deleting 2500
```

```
s2.discard(2500)
```

```
#remove(): used to delete data from set
```

```
#Ex: Delete 'ML'
```

```
s2.remove('ML')
s2
```

```
{45.67, 89.88, 'IoT', 'Python', 'Supervised ML', 'Unsupervised ML'}
```

#If data to be deleted is not available, then remove() will show a error message

#Ex: deleting 2500

```
s2.remove(2500)
```

```
-----
-----
KeyError                                Traceback (most recent call
last)
<ipython-input-43-e67009f9e69f> in <module>
      3 #Ex: deleting 2500
      4
----> 5 s2.remove(2500)
```

KeyError: 2500

#Set operation

#Union: Produce a new set containg all the unique elements from both sets

#Union operation is represented by () Press [shift+|]

```
A={'Mon', 'Tue', 'Wed'}
B={'Mon', 'Tue', 'Thu', 'Fri', 'Sat'}
```

```
C=A|B
```

```
print("Union of Set A and Set B:", C)
```

Union of Set A and Set B: {'Thu', 'Wed', 'Mon', 'Sat', 'Fri', 'Tue'}

#Intersection: It will produce a new set containing only the common elements from both sets

```
i=A&B
```

```
print('Intersection of Set A and Set B:', i)
```

Intersection of Set A and Set B: {'Mon', 'Tue'}

#Difference of two sets

#It will produce a new set containing only the elements from the first set and None from the second set

```
diff=A-B  
print('Difference of Set A and Set B:', diff)  
Difference of Set A and Set B: {'Wed'}
```

Machine Learning Using Python

Day 3: Assignment

1. Write a Python program to input age of three brothers and find youngest brother.
2. Write a Python program to check whether a number is even or odd. Accept the number from the user.
3. Write a Python program to find area of a circle.
4. Write a Python program to swap two numbers and print.
5. Write a Python program to calculate Factorial of a given number.
6. Write a Program to find whether a given number is prime or not.
7. Create a list of 10 elements named marks=[78,87,89,98,67,65,45,56,77,89]. Create another list of 5 elements. Write a Python program to append the new list at the end of the marks list.
8. Write a Python program to interchange first and last elements in a list.
9. Perform the following operations on marks list:
 - a. Insert new marks at index 7 and 10.
 - b. Remove the mark 98 from the list. Remove the element at index 8 and 13.
 - c. Remove all elements from the list.
10. Write a Python program to count occurrences of an element in a list.
11. Write a Python program to sum all elements of a list
12. Write a Python program to print only even numbers in a list.
13. Write a Python program to find average or mean of a list data.
14. Write a Python program to input some data from user and put that into list.
15. Write a Python program to find maximum and minimum element of a list.

FUNCTIONS



A function can be defined as the organized block of reusable code which can be called whenever required. A function is a block of code which only runs when it is called.

- Python allows us to divide a large program into the basic building blocks known as function.
- A function can be called multiple times to provide reusability and modularity to the python program.

Function definition



In python, we can use def keyword to define the function.

Syntax:

```
def function_name(parameter_list):
```

function-definition

return <expression>

- The function block is started with the colon (:)
- All the same level block statements remain at the same indentation.
- A function can accept any number of parameters that must be the same in the definition and function calling.

Example:



```
def hello_world():          #function declaration  
    print("hello world")    #function definition  
hello_world()              #function calling
```

Function parameters

The information into the functions can be passed as the parameters. The parameters are specified in the parentheses.

- A function may have any number of parameters.
- Multiple parameters are separated with a comma.

Function calling



In python, a function must be defined before the function calling otherwise the python interpreter gives an error. Once the function is defined, we can call it. To call the function, use the function name followed by the parentheses.

Example #defining the function

```
def sum(a,b):
```

```
    c=a+b
```

```
    print("Sum=",c)
```

#calling the function

```
sum(13,34)
```

Returning a value



```
def si_intst(p,r,t):  
    si=(p*r*t)/100  
    return si      #calling the function  
  
s=si_intst(20000,8.5,3)  
print("Simple Interest=",s)  
print("Simple Interest=",si_intst(50000,7.5,5))
```

Types of arguments



There may be several types of arguments which can be passed at the time of function calling.

1. Required arguments
2. Keyword arguments
3. Default arguments
4. Variable-length arguments

Required Arguments



The required arguments are required to be passed at the time of function calling with the exact match of their positions in the function call and function definition. If either of the arguments is not provided in the function call, or the position of the arguments is changed, then the python interpreter will show the error.

Example

```
def calculate(a,b):  
    return a+b  
  
calculate(10)  
  
# this causes an error as we are missing a required arguments b.
```

Keyword arguments



Python allows us to call the function with the keyword arguments. This kind of function call will enable us to pass the arguments in the random order.

Example #The function simple_interest(p, t, r) is called with the keyword arguments the order of arguments doesn't matter in this case

```
def simple_interest(p,t,r):
```

```
    return (p*t*r)/100
```

```
print("Simple Interest: ",simple_interest(t=10,r=10,p=1900))
```



- If we provide the different name of arguments at the time of function call, an error will be thrown.

```
simple_interest(20000,rate=7.5, time=6) #error
```

- The python allows us to provide the mix of the required arguments and keyword arguments at the time of function call.

```
simple_interest(20000,t=5,r=6.5)
```

- The required argument must not be given after the keyword argument.

```
simple_interest(20000,r=7.5,6) #error
```

Default Arguments



Python allows us to initialize the arguments at the function definition. If the value of any of the argument is not provided at the time of function call, then the default value for the argument will be used.

```
def printme(name,age=22):
```

```
    print("Name:",name,"nAge:",age)
```

```
printme("Ravi")      # name=Ravi age=22 (default value)
```

```
printme("Sachin", 33) #name =Sachin age=33
```

Variable length Arguments



Variable length argument is a feature that allows a function to receive any number of arguments. However, at the function definition, we have to define the variable with * (star) as *<variable - name >.

Example

```
def printme(*names):
```

```
    print("printing the passed arguments...")
```

```
    for name in names:
```

```
        print(name)
```

```
printme('Prashant', 'Vimal', 'Gaurav', 'Jatin')
```

Example 2:



```
def adder(*num):
```

```
    sum = 0
```

```
    for n in num:
```

```
        sum = sum + n
```

```
    print("Sum:",sum)
```

```
adder(3,5)
```

```
adder(4,5,6,7)
```

```
adder(1,2,3,5,6)
```

Lambda function



- lambda operator or lambda function is used for creating small, one-time and anonymous function objects.
- A lambda function can take any number of arguments, but can only have one expression.
- lambda keyword is used to define a function.

Syntax:

lambda arguments : expression

Example:

add=lambda x,y: x+y

print("Addition:",add(5,6))

Map function



- Used to apply a function on all the elements of specified iterable item.
- map() function returns a list of the results after applying the given function to each item of a given iterable item (list, tuple etc.)

Syntax :

```
map(function, iterable_item)
```

Ex: def multiply2(x):

```
    return x * 2
```

```
list=[1,2,3,4]
```

```
m=[*map(multiply2, list)] # Output [2, 4, 6, 8]
```

```
print((m))
```

Lambda and map



Square1 = [2,5,6,7,4]

Square2= [*map(lambda x:x **2, Square1)]

print([(Square2)])

```
#Python Functions

#define keyword is used to declare a function

#syntax:

#define function_name(parameter_list):
    #definition
    #definition

#Types

#In-built function: print(), input(), tuple(), list(), int(), float()

#User-defined function:

    #Non-parameterized function
    #parameterized function

#Non-parameterized function

def fun1():
    print('This is first function in Python')
    print('Function name is fun1')
    print('Non-parameterized function')

#calling the fun1()

fun1()
This is first function in Python
Function name is fun1
Non-parameterized function

fun1()
This is first function in Python
Function name is fun1
Non-parameterized function

#Parameterized function

#Parameterized function without return type
#Parameterized function with return type

#Parameterized function without return type

#Ex: define a function to calculate area of a circle

def area_circle(r):
```

```

ar=3.14*r*r
print('Area of circle=', ar)

#calling the area_circle()

area_circle(5)
Area of circle= 78.5

#calling the area_circle()
w=9

area_circle(w)
Area of circle= 254.34

-----
-----
NameError                               Traceback (most recent call
last)
<ipython-input-11-f7c5f98cb8e4> in <module>
      3
      4 area_circle(w)
----> 5 print(ar)

NameError: name 'ar' is not defined

#calling the area_circle()
radius=float(input('Enter Redius:'))
area_circle(radius)

Enter Redius:10
Area of circle= 314.0

#can we access the output of area_circle() at the function calling?
#NO

#Parameterized function with return type

#Using return statement, we can return the output of the funtion at
the
#function calling

def area_circle(r):
    ar=3.14*r*r
    return ar      #returning the value of 'ar' variable at the
function calling

#calling area_circle() with return statemnt

```

```
#here, a1 will hold the value of 'ar' which is being returned by the
function
#area_circle
```

```
a1=area_circle(6)
```

```
#Now, the area of circle when radius=6, is available in a1
#I can perform any operation on a1 as per requirement
```

```
print('Area of circle:', a1)
print('Square of area of circle:', a1*a1)
```

```
Area of circle: 113.03999999999999
Square of area of circle: 12778.041599999999
```

```
q=7
ar1=area_circle(q)
```

```
print('Area of circle:', ar1)
```

```
Area of circle: 153.86
```

```
radius=float(input('Enter Redius:'))
```

```
area=area_circle(radius)
print('Area of circle:', area)
```

```
Enter Redius:12
Area of circle: 452.1599999999997
```

```
w=area_circle(3,4)
```

```
-----
-----
TypeError                                Traceback (most recent call
last)
<ipython-input-16-cb039c610200> in <module>
----> 1 w=area_circle(3,4)
```

```
TypeError: area_circle() takes 1 positional argument but 2 were given
```

```
#Function arguments
```

```
#Required argument
```

```
#Ex: Function to calculate Simple Interest
```

```
def si(pr, r,t):
    print('Principal Amount:', pr)
    print('Rate of Interest:', r)
    print('Time:', t)
```

```
    si_inst=(pr*r*t)/100
    return si_inst

#In si(), it is mandatory to give the value of pr, r and t during function calling.

s1=si(34000, 6.7, 10)
print('Simple Interest:', s1)

Principal Amount: 34000
Rate of Interest: 6.7
Time: 10
Simple Interest: 22780.0

s2=si(12000, 6.8)

-----
-----
TypeError                                         Traceback (most recent call
last)
<ipython-input-18-0c637272b773> in <module>
----> 1 s2=si(12000, 6.8)

TypeError: si() missing 1 required positional argument: 't'

#Keyword argument

s3=si(pr=20000, r=6.9, t=5)
print('Simple Interest:', s3)

Principal Amount: 20000
Rate of Interest: 6.9
Time: 5
Simple Interest: 6900.0

#In Keyword argument, you may shuffle the order of the parameters

s4=si(t=8, pr=15000, r=5.6)
print('Simple Interest:', s4)

Principal Amount: 15000
Rate of Interest: 5.6
Time: 8
Simple Interest: 6720.0

#If you use different names, which is not defined in function definition
#Then, it will throw an error

s5=si(time=8, pr=15000, r=5.6)
print('Simple Interest:', s5)
```

```
-----
TypeError                                Traceback (most recent call
last)
<ipython-input-21-c31d68bb66e8> in <module>
      2 #Then, it will throw an error
      3
----> 4 s5=si(time=8, pr=15000, r=5.6)
      5 print('Simple Interest:', s5)
```

TypeError: si() got an unexpected keyword argument 'time'

#We can mix the required argument and keyword argument during function calling

```
s6=si(12000, r=5.6, t=9)
print('Simple Interest:', s6)
```

Principal Amount: 12000
Rate of Interest: 5.6
Time: 9
Simple Interest: 6048.0

#It is mandatory to give/use required argument before using keyword argument

#You can not use required argument after keyword argument

```
s5=si(t=8, 15000, r=9.8)

File "<ipython-input-23-f31750525ce6>", line 3
    s5=si(t=8, 15000, r=9.8)
          ^
```

SyntaxError: positional argument follows keyword argument

#Default argument

```
def data(name, age=18):
    print('Name:', name)
    print('Age:', age)
```

```
data('Amit')
```

Name: Amit
Age: 18

```
data('Rajat', 22)
```

Name: Rajat
Age: 22

#Default argument

```
def si(pr, t, r=5.5):
    print('Principal Amount:', pr)
    print('Rate of Interest:', r)
    print('Time:', t)
    si_inst=(pr*r*t)/100
    return si_inst
```

```
s1=si(14000, 7)
print(s1)
```

```
Principal Amount: 14000
Rate of Interest: 5.5
Time: 7
5390.0
```

```
s1=si(14000, 7, 6.7)
print(s1)
```

```
Principal Amount: 14000
Rate of Interest: 6.7
Time: 7
6566.0
```

```
def total(a,b):
    c=a+b
    print('Total:', c)
```

```
total(12, 34)
```

```
Total: 46
```

#variable length argument

```
def total(*data):
    t=0

    print('Value of data variable:', data)
    print('Type of data variable:', type(data))

    for s in data:
        t=t+s
    print('Total is:', t)
```

#Calling total with 2 argument

```
total(12,3)
```

```
Value of data variable: (12, 3)
Type of data variable: <class 'tuple'>
Total is: 15
```

```
#calling total with 3 argument
total(12,3,5)
```

```
Value of data variable: (12, 3, 5)
Type of data variable: <class 'tuple'>
Total is: 20
```

```
#calling total with 4 argument
total(10,5,5,2)
```

```
Value of data variable: (10, 5, 5, 2)
Type of data variable: <class 'tuple'>
Total is: 22
```

```
def total(*data):
    t=0

    print('Value of data variable:', data)
    print('Type of data variable:', type(data))

    for s in data:
        t=t+s
        print('Total is:', t)
```

```
#calling total with 4 argument
total(10,5,5,2)
```

```
Value of data variable: (10, 5, 5, 2)
Type of data variable: <class 'tuple'>
Total is: 10
Total is: 15
Total is: 20
Total is: 22
```

```
"""

```

Lambda Function

```
def f1(a,b,c,d):
    s1
```

*Since, there is only one statement in f1().
We can declare f1() as lambda function.*

```
def f2(a,b):
    s1
    s2
    s3
```

*Since, there is more than one statement in f2()
We can not declare f2() as lambda function*
"""

#Lambda function can take any no. of argument but may only have one expression/statement

#Lambda function always return the outcome of the arithmetic operation which is being performed.

#Syntax: lambda_function_name=lambda argument: expression/statement

#Ex: Declaring a lambda function to add two numbers

```
add_lambda=lambda x,y:x+y
```

*#Here, add_lambda is the lambda function-name
#x,y are arguments
#x+y is expression/statement*

```
a=add_lambda(10,3)  
print('Addition:', a)
```

Addition: 13

```
p=int(input('Enter First number:'))  
q=int(input('Enter Second number:'))
```

```
a1=add_lambda(p,q)  
print('Addition:', a1)
```

```
Enter First number:25  
Enter Second number:35  
Addition: 60
```

#Calculate area of circle using lambda

```
area_circle=lambda r:3.14*r*r  
  
radius=float(input('Enter Radius='))  
  
r1=area_circle(radius)  
print('Area of circle:', r1)  
  
Enter Radius=8  
Area of circle: 200.96
```

#map function: used to call a function with all the values/data of a tuple/list

#If you want to call any function with all the members of a list/tuple, use map.

#Syntax: var_name=[*map(function_name, list_name/tuple_name)]

```
radius1=[3, 5, 6]
```

#Ex: calling area_circle with each member/data of radius1

```
area1=[*map(area_circle, radius1)]
```

```
print('List of radius:', radius1)
print('Area of circle:', area1)
```

```
List of radius: [3, 5, 6]
Area of circle: [28.25999999999998, 78.5, 113.0399999999999]
```

#map

```
def square(num):
    return num*num
```

```
m1=(1,3,5,7,9)
```

```
result=[*map(square, m1)]
print('Original Tuple:', m1)
print('Square of each data of Tuple:', result)
```

```
Original Tuple: (1, 3, 5, 7, 9)
Square of each data of Tuple: [1, 9, 25, 49, 81]
```

#Lambda and map

#Ex: to calculate cube of each number of list using lambda and map

```
list2=[2,3,4,5,6]
```

```
cube=[*map(lambda x: x*x*x, list2)]
```

```
print('Original List:', list2)
print('Cube of each member of list:', cube)
```

```
Original List: [2, 3, 4, 5, 6]
Cube of each member of list: [8, 27, 64, 125, 216]
```

Machine Learning Using Python

Day 4: Assignment

1. Write a Python function to print area and perimeter of a circle.
2. Write a Python function to check whether a number is even or odd.
3. Write a function to swap two numbers.
4. Write a Python function to calculate simple interest.
5. Write a Python function to calculate the factorial of a number.
6. Write a Python function that takes a list and returns a new list with unique elements of the first list.
7. Write a Python function that takes a number as a parameter and check the number is prime or not.
8. Write a Python function to calculate the average, maximum and minimum salary of 10 employees.
9. Write a Python function using lambda and map to calculate the square of each number in a list.
10. Using a lambda and map function, calculate the sum of all numbers of a list.
11. Using lambda and map calculate the cube of all numbers of a list and find minimum element and maximum element from the resultant list.
12. Calculate average of sum of cube of all numbers of a list by using lambda and map.

OOPs Concepts



Python is an object-oriented programming language. It allows us to develop applications using an Object Oriented approach. In Python, we can easily create and use classes and objects.

- Object
- Class
- Encapsulation
- Method
- Inheritance
- Polymorphism
- Data Abstraction



Object

The object is an entity that has state and behavior. It may be any real-world object like the mouse, keyboard, chair, table, pen, etc. Everything in Python is an object, and almost everything has attributes and methods.

Class

The class can be defined as a collection of objects. It is a logical entity that has some specific attributes and methods. For example: if you have an employee class then it should contain an attribute and method, i.e. an email id, name, age, salary, etc.



Encapsulation

In encapsulation, code and data are wrapped together within a single unit defined as class.

Method

The method is a function that is associated with an object.

Inheritance



It specifies that the child object acquires all the properties and behaviors of the parent object. A class may use all the properties and behavior of another class.

- The new class is known as a derived class or child class.
- The class whose properties are acquired is known as a base class or parent class.
- It provides re-usability of the code.

Polymorphism



Polymorphism contains two words "poly" and "morphs". Poly means many and Morphs means form, shape. By polymorphism, we understand that one task can be performed in different ways. A class may have more than two methods with same name with different parameters.

Data Abstraction

Abstraction is used to hide internal details and show only functionalities. Abstracting something means to give names to things so that the name captures the core of what a function or a whole program does.

Creating classes in python



In python, a class can be created by using the keyword `class` followed by the class name.

Syntax

`class ClassName:`

`#class_members`

A class contains a `class_members` including fields, constructor, function, etc.

Example



class data:

 id=101

 name="Abhishek"

def display(s):

 print("ID=",s.id)

 print("Name",s.name)

The s parameter is a reference to the current instance of the class, and is used to access variables that belong to the class.

It has to be the first parameter of any function in the class.

Creating objects



A class can be instantiated by calling the class using the class name.

Syntax:

<object-name> = <class-name> (<arguments>)

Ex: m=data()

m.display() #calling display function using object of class my.

Access Specifier



Public:

By default, all the variables and member functions of a class are public in a python program.

Ex: `x` #will be public member of the class

Private:

A variable can be made private by prefixing the `__`(double underscore) before the variable name.

ex: `__x` will be the private member of the class.

Protected:

A variable can be made public by prefixing the `_` (single underscore) before the variable name.

Ex: `_x` will be the protected member of the class.

Constructor



A constructor is a special type of method (function) which is used to initialize the members of the class. Constructor definition is executed when we create the object of this class.

Types.

1. Parameterized Constructor
2. Non-parameterized Constructor

Creating the constructor



In python, the method `__init__` is used to create the constructor of the class. This method is called when the object of the class is instantiated.

- We can pass any number of arguments at the time of creating the class object, depending upon `__init__` definition.
- It is mostly used to initialize the class attributes. Every class must have a constructor, even if it simply relies on the default constructor.

Example of Constructor



id=101

name="Rajat"

```
def __init__(a):      #non parameterized constructor
    print("Object is created")
    def display(s):
        print("ID=",s.id)
        print("Name=",s.name)
```

m=my()

m.display()

Parameterized Constructor



```
class emp
    emp_id=0
    emp_name=0
    emp_salary=0
    def __init__(s,i,n,sal) #parameterized constructor
        s.emp_id=i
        s.emp_name=n
        s.emp_salary=sal

def display(s)
    print("Employee ID=",s.emp_id, "\nEmployee
Name=",s.emp_name, "\nSalary=",s.emp_salary)
E=emp(101, "Rahul", 36000)
E.display()
```



Modify Object Properties

The previously initialized variable can be modified as

```
e.salary=45000
```

Delete Object Properties

- The del keyword is used to delete object properties.

```
del e.emp_id    #deletes the value of emp_id
```

- The objects can also be deleted by using del keyword.

```
del e        #deletes the object e
```

Python Inheritance



Inheritance allows us to define a class that inherits all the methods and properties from another class.

- Parent class is the class being inherited from, also called base class.
- Child class is the class that inherits from another class, also called derived class. The child class acquires the properties and can access all the data members and functions defined in the parent class.



Syntax:

```
class class_name(base_class_name):
```

 class definition

- **A class can inherit multiple classes by mentioning all of them inside the bracket.**

```
class class_name(baseclass1, baseclass2,.....):
```

 class definition

Types of Inheritance in Python



Python - Multiple Inheritance

Multiple Inheritance means that you're inheriting the property of multiple classes into one. In case you have two classes, say A and B, and you want to create a new class which inherits the properties of both A and B, then:

class A:

```
# variable of class A  
# functions of class A
```

class B:

```
# variable of class A  
# functions of class A
```

class C(A, B):

```
# class C inheriting property of both class A and B  
# add more properties to class C
```

Multiple inheritance



Python provides us the flexibility to inherit multiple base classes in the child class.

```
class Calculation1:  
    def Summation(self,a,b):  
        return a+b;  
class Calculation2:  
    def Multiplication(self,a,b):  
        return a*b;  
class Derived(Calculation1,Calculation2):  
    def Divide(self,a,b):  
        return a/b;  
d=Derived()  
print(d.Summation(10,20))  
print(d.Multiplication(10,20))  
print(d.Divide(10,20))
```

Types of Inheritance in Python



Python - Multilevel Inheritance

In multilevel inheritance, we inherit the classes at multiple separate levels. We have three classes A, B and C, where A is the super class, B is its sub(child) class and C is the sub class of B.

Here is a simple example, its just to explain you how this looks in code:

class A:

```
# properties of class A
```

class B(A):

```
# class B inheriting property of class A # more properties of class B
```

class C(B):

```
# class C inheriting property of class B
```

```
# thus, class C also inherits properties of class A
```

```
# more properties of class C
```

Multi-Level Inheritance



```
class Animal:  
    def speak(self):  
        print("Animal Speaking")  
class Dog(Animal):  
    def bark(self):  
        print("dog barking")  
class DogChild(Dog):  
    def eat(self):  
        print("Eating bread...")  
d = DogChild()  
d.bark()  
d.speak()  
d.eat()
```

Method Overloading



In Python you can define a method in such a way that there are multiple ways to call it. Depending on the function definition, it can be called with zero, one, two or more parameters. This is known as method overloading.

In the given code, there is a class with one method sayHello(). We rewrite as shown below. The first parameter of this method is set to None, this gives us the option to call it with or without a parameter.

An object is created based on the class, and we call its method using zero and one parameter. To implement method overloading, we call the method sayHello() in two ways:

1. obj.sayHello()
2. obj.sayHello('Rambo')

class Human:

```
def sayHello(self, name=None):
    if name is not None:
        print 'Hello ' + name
    else:
        print 'Hello '
```

```
obj = Human()
```

```
print(obj.sayHello())
```

```
print(obj.sayHello('Rambo'))
```

Method Overriding



We can provide some specific implementation of the parent class method in our child class. When the parent class method is defined in the child class with some specific implementation, then the concept is called method overriding. We may need to perform method overriding in the scenario where the different definition of a parent class method is needed in the child class.

Consider the following example to perform method overriding in python.

```
class Animal:
```

```
    def speak(self):  
        print("speaking")
```

```
class Dog(Animal):
```

```
    def speak(self):  
        print("Barking")
```

```
d=Dog()
```

```
d.speak()
```

Example of method overriding

```
class Bank:  
    def getroi(self):  
        return 10;  
class SBI(Bank):  
    def getroi(self):  
        return 7;  
class ICICI(Bank):  
    def getroi(self):  
        return 8;  
b1=Bank()  
b2=SBI()  
b3=ICICI()  
print("Bank Rate of interest:",b1.getroi());  
print("SBI Rate of interest:",b2.getroi());  
print("ICICI Rate of interest:",b3.getroi());
```



```

#Object oriented programming

class student:
    #declaring data members
    idd=101
    name='Rajat'
    course='ML'

    #declaring methods
    #declaring non-parameterized method

def show_data(d):
    #Here, 'd' is known as class reference variable
    #Each method declared inside class must include this parameter
    #This parameter must be first parameter of the class method
    #Methods may have other parameters as per requirement
    #This parameter is used to access the variable/data members of
the class

    print('Id=', d.idd)
    print('Name=', d.name)
    print('Course=', d.course)

#Object is used to access the public data or methods of the class

#Syntax: object_name=class_name()

#Ex: Creating object of student class

s1=student()

#s1.idd=101
#s1.name='Rajat'
#s1.course='ML'

#Calling show_data() using s1 object
s1.show_data()

#d=s1

Id= 101
Name= Rajat
Course= ML

#creating another object of student

s2=student()

```

```

#s12.idd=101
#s2.name='Rajat'
#s2.course='ML'

s2.show_data()

#d=s2

Id= 101
Name= Rajat
Course= ML

#Declare a method named input_data() to input the value of idd, name,
course

class student:
    #declaring data members
    idd=0
    name=0
    course=0

    #parameterized method
    def input_data(d, i, n, c):
        d.idd=i
        d.name=n
        d.course=c

    def show_data(d):
        print('Id=', d.idd)
        print('Name=', d.name)
        print('Course=', d.course)

#Object creation

st1=student()
st1.input_data(101, 'Amit', 'AI')

#d=st1
#st1.idd=101
#st1.name=Amit
#st1.course=AI

st1.show_data()

Id= 101
Name= Amit
Course= AI

```

```

#creating another object

st2=student()
st2.input_data(102, 'Jatin', 'IoT')
st2.show_data()

Id= 102
Name= Jatin
Course= IoT

#Access specifier

class access:
    i=100

        #i is a public variable
        #Public variables are accessible throughout all the classes of
        your program
        #Public variables are also accessible using object of the class
        #class object can access or modify the value of public variable

    __name='Rajat'

        #__name is a private variable
        #private variables are accessible ONLY through the public methods
        of the class
        #Class object can not access the private variables of the class
        #Private data members are not inherited

    _course='ML'

        #Here, _course is declared as projected member of the class
        #Class object can access the protected members of the class
        #They are exclusively used for inheritance

def view(r):
    print('Id:', r.i)
    print('Name:', r.__name)
    print('Course:', r._course)

a1=access()
a1.view()

Id: 100
Name: Rajat
Course: ML

print('Accessing public member using class object:', a1.i )

```

```

Accessing public member using class object: 100
print('Accessing protected member using class object:', a1._course )
Accessing protected member using class object: ML
print('Accessing private member using class object:', a1.__name)
-----
-----
AttributeError                               Traceback (most recent call
last)
<ipython-input-9-113810172b6a> in <module>
----> 1 print('Accessing private member using class object:', a1.__name)

AttributeError: 'access' object has no attribute '__name'

class data:
    x=9000

    #Non-parameterized constructor

    def __init__(s):
        print('Object of data class is created')

    def show(d):
        print('X=', d.x)

d1=data()
Object of data class is created

#Constructors are special methods of the class which are used to
initialize the data members of the
#class

class student:
    #declaring data members
    idd=0
    name=0
    course=0

    #Parameterized constructor
    def __init__(d, i,n,c):
        d.idd=i
        d.name=n
        d.course=c

    def show_data(s):

```

```
print('Id=', s.id)
print('Name=', s.name)
print('Course=', s.course)

#creating object of student class

#You do not need to call the class constructor
#Class constructor are called when you create object of the class

st1=student(1001, 'Ajit', 'ML')
st1.show_data()

Id= 1001
Name= Ajit
Course= ML

#We can modify the values of public data members using class object

st1.name='Aman'

st1.show_data()

Id= 1001
Name= Aman
Course= ML

#We can delete delete the value of public variable of class using its object

del st1.name
st1.show_data()

Id= 1001
Name= None
Course= ML

#You can delete class object using del

del st1

st1.show_data()
-----
-----
NameError                                 Traceback (most recent call
last)
<ipython-input-15-bec8c6239bd4> in <module>
      3 del st1
      4
----> 5 st1.show_data()
```

```
NameError: name 'st1' is not defined

#Inheritance

#Single Inheritance: a class inherits only one class

class student:
    _id=0      #protected data members
    _name=0    #protected data members

    def show(d):
        print('ID=' , d._id)
        print('Name=' , d._name)

#stud_data class will inherit stud class
class student_data(student):

    _marks=0

    def __init__(s, i,n,m):
        s._id=i
        s._name=n
        s._marks=m

    def show_data(d):
        print('Marks=' , d._marks)

#creating object of student_data class
sd1=student_data(1, 'Rakesh', 79)

#calling show() of student class using object of student_data class
sd1.show()

#calling show_data() using object of student_data class
sd1.show_data()

ID= 1
Name= Rakesh
Marks= 79

sd2=student_data(2, 'Abhishek', 86)
sd2.show()
sd2.show_data()

ID= 2
Name= Abhishek
Marks= 86
```

```

#Multiple Inheritance: A child class may inherit more than one class

class cal1:
    def total(s, a, b):
        return a+b

class cal2:
    def multi(s, x,y):
        return x*y

class cal3(cal1, cal2):
    def divide(d, p,q):
        return p/q

c1=cal3()

print('Calling total() of cal1 using object of cal3:', c1.total(15,
20))
print('Calling multi() of cal2 using object of cal3:', c1.multi(5,4))
print('Calling divide() of cal3:', c1.divide(15,3))

```

Calling total() of cal1 using object of cal3: 35
Calling multi() of cal2 using object of cal3: 20
Calling divide() of cal3: 5.0

#Multilevel inheritance

```

"""
class A

class B inherits class A

class C inherits class B

```

so , this case class C inherits class A indirectly.

We can access the members(data/method) of class A and class B using object of class C

```

"""
#Multilevel inheritance

```

```

class A:
    emp_id=0

    def show_id(s):
        print('Employee ID=', s.emp_id)

```

```

class B(A):
    emp_name=0

    def show_name(s):
        print('Employee Name:', s.emp_name)

class C(B):
    emp_sal=0

    def __init__(s, i, n, sal):
        s.emp_id=i          #c1.emp_id=E101,
        s.emp_name=n        #c1.emp_name=Rajat
        s.emp_sal=sal       #c1.emp_sal=35000

    def show_salary(s):
        print('Employee Salary=', s.emp_sal)

c1=C('E101', 'Rajat', 35000)    # s=c1, i='E101', n='Rajat' sal=35000
c1.show_id()
c1.show_name()
c1.show_salary()

Employee ID= E101
Employee Name: Rajat
Employee Salary= 35000

```

#Method Overloading

#Python does not support method overloading

```

class over:
    def show(s):
        print('Non parameterized show')

    def show(s,i):
        print('Parameterized show')

```

```

o=over()
o.show(12)
o.show()

```

Parameterized show

```

-----
-----
TypeError                                Traceback (most recent call
last)
<ipython-input-8-9e7a2ff0f34b> in <module>
     13 o=over()
     14 o.show(12)

```

```
--> 15 o.show()
```

```
TypeError: show() missing 1 required positional argument: 'i'
```

#Indirect method overloading

```
class overl:
    def overload(s, name='Guest'):
        print('Welcome!! ', name)
```

```
o1=overl()
o1.overload()
o1.overload('jatin')
```

```
Welcome!! Guest
Welcome!! jatin
```

#Method overriding: defining more than one method with same name and similar kind of parameters

#If parent class and child class both have two methods with same name and similar kind of parameters, then it means
#both classes have overridden method

#In this case, the definition of overridden method will be changed in child class.

```
class bank:
    def getroi(s):
        print('Calling getroi() of bank class')
        return 10
```

```
class sbi(bank):
    def getroi(s):
        print('Calling getroi() of sbi class')
        return 8.9
```

#If i call getroi() using object of sbi class, then the getroi() of sbi class will be accessed

```
s=sbi()
print('Rate of Interest:', s.getroi())
```

```
Calling getroi() of sbi class
Rate of Interest: 8.9
```

#If you want to access the getroi() of class bank, you have to create the object of bank class

```
b=bank()  
print('Rate of interest:', b.getroi())
```

```
Calling getroi() of bank class  
Rate of interest: 10
```

Machine Learning Using Python

Day 5: Assignment

1. Write a Python class named Rectangle constructed by a length and width and a method which will compute the area of a rectangle.
2. Write a python class to display student's roll number, Name, marks of 3 subjects using functions. Also display sum and average of their marks using function.
3. Write a python program to delete an object's properties.
4. Write a Python class named Circle constructed by a radius and two methods which will compute the area and the perimeter of a circle.
5. Write a Python program to create the class named data that has three members – id, name and basic_salary. Create a new class named calculation that inherits the class data and calculates the HRA, DA and Gross salary using function. Display the id, name, salary, HRA, DA and gross salary. HRA is 45% of basis salary and DA is 60% of basic salary. Input the values from the user.

Python - Modules



A module allows you to logically organize your Python code. Grouping related code into a module makes the code easier to understand and use. A module is a Python object with arbitrarily named attributes that you can bind and reference.

Simply, a module is a file consisting of Python code. A module can define functions, classes and variables. A module can also include runnable code.

Python - Modules



Create a Module

Save this code in a file named mymodule.py

```
def greeting(name):  
    print("Hello, " + name)
```

Use a Module

Now we can use the module we just created, by using the import statement:

Example

Import the module named mymodule, and call the greeting function:

```
import mymodule  
mymodule.greeting("Jonathan")
```

Import with renaming



We can import a module by renaming it as follows.

```
# import module by renaming it
import math as m
print("The value of pi is", m.pi)
```

We have renamed the math module as m. This can save us typing time in some cases.

Import all names



We can import all names(definitions) from a module using the following construct.

```
# import all names from the standard module math
```

```
from math import *
print("The value of pi is", pi)
```

We imported all the definitions from the math module. This makes all names except those beginnig with an underscore, visible in our scope.

Importing everything with the asterisk (*) symbol is not a good programming practice. This can lead to duplicate definitions for an identifier. It also hampers the readability of our code.

The *from import* Statement



Python's *from* statement lets you import specific attributes from a module.

The module named mymodule has one function and one dictionary:

```
def greeting(name):
    print("Hello, " + name)
```

```
person1 = {
    "name": "John",
    "age": 36,
    "country": "Norway"
}
```

Import only the person1 dictionary from the module:

from mymodule import person1

```
print (person1["age"])
```

Code Snippet illustrating python built-in modules:

```
import math  
print(math.sqrt(25))  
print(math.pi)  
print(math.degrees(2))  
print(math.radians(60))  
print(math.sin(2))  
print(math.cos(0.5))  
print(math.tan(0.23))  
print(math.factorial(4))
```



Packages in Python

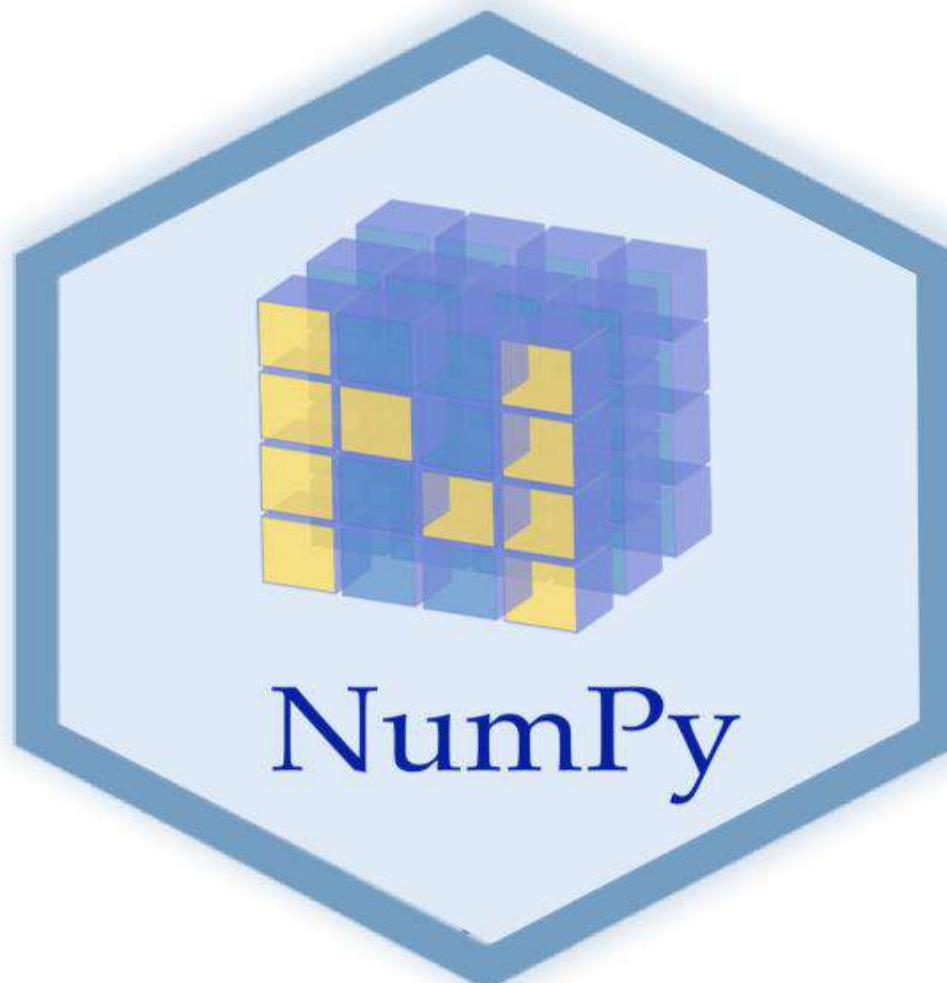


We don't usually store all of our files in our computer in the same location. We use a well-organized hierarchy of directories for easier access.

Similar files are kept in the same directory, for example, we may keep all the songs in the "music" directory. Analogous to this, Python has packages for directories and modules for files.

As our application program grows larger in size with a lot of modules, we place similar modules in one package and different modules in different packages. This makes a project (program) easy to manage and conceptually clear.

Similar, as a directory can contain sub-directories and files, a Python package can have sub-packages and modules.



What is Numpy



NumPy stands for numeric python which is a python package for the computation and processing of the multidimensional and single dimensional array elements.

Travis Oliphant created NumPy package in 2005 by injecting the features of the ancestor module Numeric into another module Numarray.

Why Numpy



With the revolution of data science, data analysis libraries like NumPy, SciPy, Pandas, etc. have seen a lot of growth. With a much easier syntax than other programming languages, python is the first choice language for the data scientist.

NumPy provides a convenient and efficient way to handle the vast amount of data. NumPy is also very convenient with Matrix operations and data reshaping. NumPy is fast which makes it reasonable to work with a large set of data.

Why Numpy



There are the following advantages of using NumPy for data analysis.

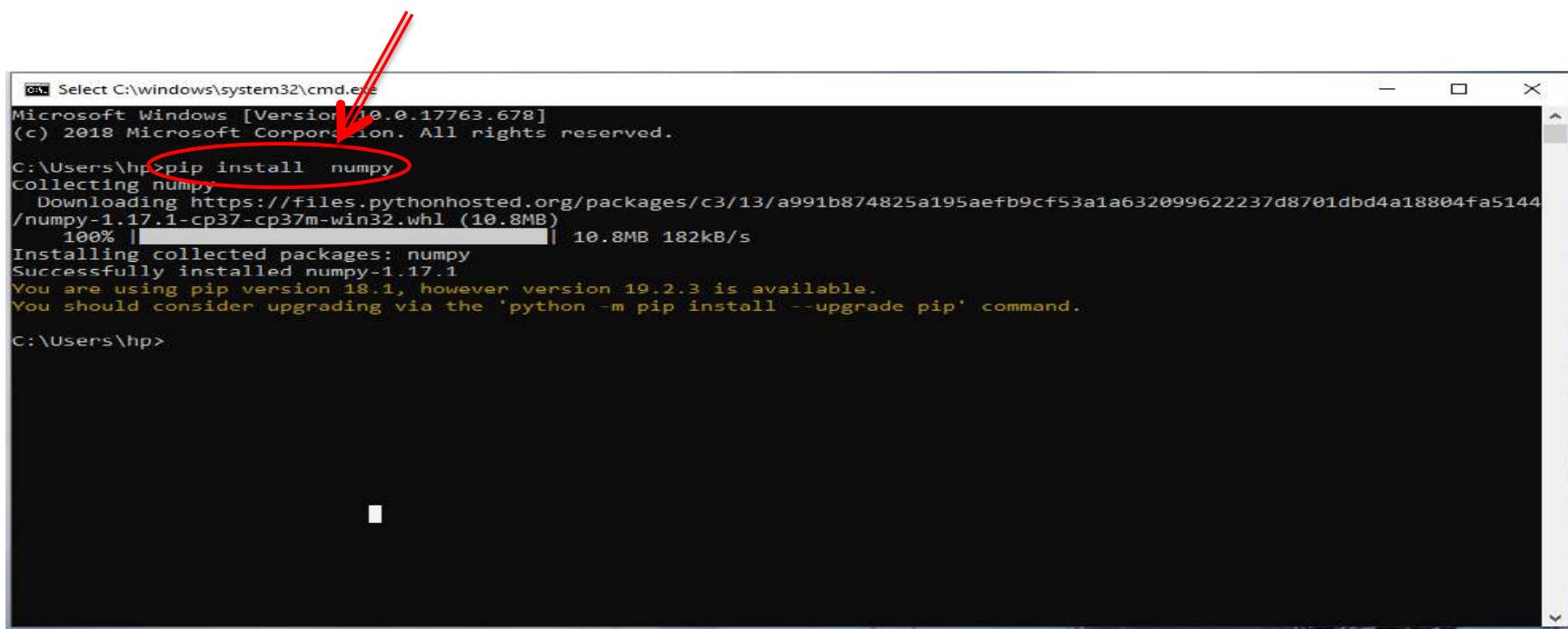
- NumPy performs array-oriented computing.
- It efficiently implements the multidimensional arrays.
- It performs scientific computations.
- It is capable of reshaping the data stored in multidimensional arrays.

Numpy Environment Setup



NumPy doesn't come bundled with Python. We have to install it by using following step

- Open “cmd”
- Type “ pip install numpy” on cmd then press enter.



```
cmd Select C:\windows\system32\cmd.exe
Microsoft Windows [Version 10.0.17763.678]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Users\hp>pip install numpy
Collecting numpy
  Downloading https://files.pythonhosted.org/packages/c3/13/a991b874825a195aefb9cf53a1a632099622237d8701dbd4a18804fa5144
/numpy-1.17.1-cp37-cp37m-win32.whl (10.8MB)
  100% |██████████| 10.8MB 182kB/s
Installing collected packages: numpy
Successfully installed numpy-1.17.1
You are using pip version 18.1, however version 19.2.3 is available.
You should consider upgrading via the 'python -m pip install --upgrade pip' command.

C:\Users\hp>
```

Numpy Ndarray



- Ndarray is the n-dimensional array object defined in the numpy which stores the collection of the similar type of elements. In other words, we can define a ndarray as the collection of the data type (dtype) objects.

- The ndarray object can be accessed by using the **0** based indexing. Each element of the Array object contains the same size in the memory.

Creating Ndarray



To create an array using the list, use the following syntax.

```
import numpy as np  
a = np.array([value1,value2,.....])  
print (a)
```

Creating Ndarray



To create a multi-dimensional array object, use the following syntax.

more than one dimensions

import numpy as np

a = np.array([[values.....], [values.....]])

Finding the size of each array element



The itemsize function is used to get the size of each array item. It returns the number of bytes taken by each array element.

Syntax:

array.itemsize

Finding the size of each array element



Example

```
#finding the size of each item in the array
import numpy as np
a = np.array([[1,2,3]])
print("Each item contains",a.itemsize,"bytes")
```

Output:

Each item contains 4 bytes.

Finding the data type of each array item



To check the data type of each array item, the `dtype` function is used. Consider the following example to check the data type of the array items.

Example

```
#finding the data type of each array item
import numpy as np
a = np.array([[1,2,3]])
print("Each item is of the type",a.dtype)
```

Result: Each item is the type of `int32`

Finding the shape and size of the array



To get the shape and size of the array, the size and shape function associated with the numpy array is used.

- **ndarray.shape():**

Example

```
import numpy as np  
  
a = np.array([[1,2,3,4,5,6,7]])  
  
print("Shape:",a.shape)
```

Output:

Shape: (1, 7)

Finding the shape and size of the array



To get the shape and size of the array, the size and shape function associated with the numpy array is used.

- **ndarray.size():**

Example

```
import numpy as np  
  
a = np.array([[1,2,3,4,5,6,7]])  
  
print("Array Size:",a.size)
```

Output:

Array Size: 7

Reshaping the array objects



By the shape of the array, we mean the number of rows and columns of a multi-dimensional array.

However, the numpy module provides us the way to

reshape the array by changing the number of rows

and columns of the multi-dimensional array.

Reshaping the array objects



The **reshape()** function associated with the ndarray object is used to reshape the array. It accepts the two parameters indicating the row and columns of the new shape of the array

Syntax: numpy.reshape(a, newshape,)

This function helps to get a new shape to an array without changing its data.

Reshaping the array objects



Let's reshape the array given in the following image.

1	2
3	4
5	6



1	2	3
4	5	6

2×3

3×2

Reshaping the array objects



Example

```
import numpy as np
a = np.array([[1,2],[3,4],[5,6]])
print("printing the original array..")
print(a)
a=a.reshape(2,3)
print("printing the reshaped array..")
print(a)
```

Reshaping the array objects



Output:

printing the original array.. [[1 2] [3 4] [5 6]]

printing the reshaped array.. [[1 2 3] [4 5 6]]

Slicing in the Array



- Consider the following example to print a particular element of the array.

Example

```
import numpy as np  
a = np.array([[1,2],[3,4],[5,6]])  
print(a[0,1])  
print(a[2,0])
```

Output:

2 5

The above program prints the 2nd element from the 0th index and 0th element from the 2nd index of the array

Finding the maximum, minimum, and sum of the array elements

The NumPy provides the max(), min(), and sum() functions which are used to find the maximum, minimum, and sum of the array elements respectively.

➤ **max()**

Max function is used to display largest value from a given array list.

```
a = np.array([1,2,3,10,15,4])
```

```
print("The maximum element:",a.max())
```

Finding the maximum, minimum, and sum of the array elements

➤ **min()**

min function is used to display smallest value from a given array list.

```
a = np.array([1,2,3,10,15,4])
```

```
print("The minimum element:",a.min())
```

➤ **sum()**

sum function is used to display total of the values.

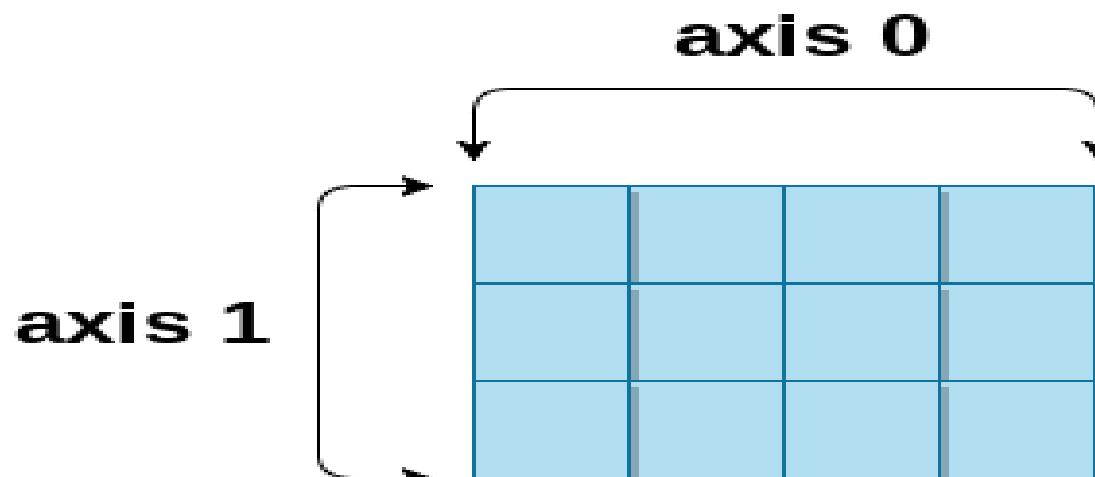
```
a = np.array([1,2,3,10,15,4])
```

```
print("The sum is :",a.sum())
```

NumPy Array Axis



A NumPy multi-dimensional array is represented by the axis where axis-0 represents the columns and axis-1 represents the rows.



NumPy Array

NumPy Array Axis



- To calculate the maximum element among each column.
consider the following example.

Example

```
a = np.array([[1,2,30],[10,15,4]])  
print("The maximum elements of columns:",a.max  
(axis = 0))
```

Output:

maximum elements of columns: [10 15 30] .

Finding square root and standard deviation



- The **sqrt()** and **std()** functions associated with the numpy array are used to find the square root and standard deviation of the array elements respectively. Standard deviation means how much each element of the array varies from the mean value of the numpy array.

Syntax:

numpy.sqrt or std(array_name))

Arithmetic operations on the array



The numpy module allows us to perform the arithmetic operations on multi-dimensional arrays directly.

Example

```
import numpy as np  
  
a = np.array([[1,2,30],[10,15,4]])  
  
b = np.array([[1,2,3],[12, 19, 29]])  
  
print("Sum of array a and b\n",a+b)
```

Array Concatenation



The numpy provides us with the vertical stacking and horizontal stacking which allows us to concatenate two multi-dimensional arrays vertically or horizontally.

Consider the following example.

Example

```
import numpy as np  
a = np.array([[1,2,30],[10,15,4]])  
b = np.array([[1,2,3],[12, 19, 29]])
```

Array Concatenation



```
print("Arrays vertically concatenated\n",np.vstack((a,b)));
```

```
print("Arrays horizontally concatenated\n",np.hstack((a,b)))
```

Output:

Arrays vertically concatenated [[1 2 30] [10 15 4]
[1 2 3] [12 19 29]]

Arrays horizontally concatenated [[1 2 30 1 2 3]
[10 15 4 12 19 29]]

Numpy Array Creation



The ndarray object can be constructed by using the following routines.

- **Numpy.empty**

As the name specifies, The empty routine is used to create an uninitialized array of specified shape and data type.

The syntax is given below.

```
numpy.empty(shape, dtype = float, order = 'C')
```

- **Shape:** The desired shape of the specified array.

Numpy Array Creation



- **dtype:** The data type of the array items. The default is the float.
- **Order:** The default order is the c-style row-major order. It can be set to F for FORTRAN-style column-major order.
- **Example**

```
import numpy as np  
arr = np.empty((3,2), dtype = int)  
print(arr)
```

Numpy Array Creation



Output:

```
[[      140482883954664      36917984]      [  
140482883954648      140482883954648]  
[6497921830368665435 172026472699604272]]
```

Numpy Array Creation



- **NumPy.Zeros:** This routine is used to create the numpy array with the specified shape where each numpy array item is initialized to 0. The syntax is given below.

```
numpy.zeros(shape, dtype = float, order = 'C')
```

It accepts the following parameters.

- **Shape:** The desired shape of the specified array.
- **dtype:** The data type of the array items. The default is the float.

Numpy Array Creation



- **Order:** The default order is the c-style row-major order. It can be set to F for FORTRAN-style column-major order.

Example

```
import numpy as np  
arr = np.zeros((3,2), dtype = int)  
print(arr)
```

Output:

```
[[0 0] [0 0] [0 0]]
```

Numpy Array Creation



➤ NumPy.ones

This routine is used to create the numpy array with the specified shape where each numpy array item is initialized to 1.

The syntax to use this module is given below.

numpy.ones(shape, dtype = none, order = 'C')

All parameter are same as previous order.

Numpy array from existing data



➤ **numpy.asarray**

This routine is used to create an array by using the existing data in the form of lists, or tuples. This routine is useful in the scenario where we need to convert a python sequence into the numpy array object.

Syntax:

```
numpy.asarray(sequence, dtype = None, order = None)
```

Numpy array from existing data



It accepts the following parameters.

- **sequence:** It is the python sequence which is to be converted into the python array.
- **dtype:** It is the data type of each item of the array.
- **order:** It can be set to C or F. The default is C.

Numpy array from existing data



Example:

creating a numpy array using Tuple

```
import numpy as np
```

```
l=(1,2,3,4,5,6,7)
```

```
a = np.asarray(l);
```

```
print(type(a))
```

```
print(a)
```

Output:

```
<class 'numpy.ndarray'> [1 2 3 4 5 6 7]
```

Numpy Arrays within the numerical range

`numpy.arange(start, stop, step, dtype)`

It accepts the following parameters.

- **start:** The starting of an interval. The default is 0.
- **stop:** represents the value at which the interval ends excluding this value.
- **step:** The number by which the interval values change.
- **dtype:** the data type of the numpy array items.

group

NumPy Broadcasting



- In Mathematical operations, we may need to consider the arrays of different shapes. NumPy can perform such operations where the array of different shapes are involved.

- For example, if we consider the matrix multiplication operation, if the shape of the two matrices is the same then this operation will be easily performed. However, we may also need to operate if the shape is not similar.

NumPy Broadcasting



Consider the following example to multiply two arrays.

Example

```
import numpy as np  
a = np.array([1,2,3,4,5,6,7])  
b = np.array([2,4,6,8,10,12,14])  
c = a*b;  
print(c) //result [ 2  8 18 32 50 72 98]
```

NumPy Broadcasting



- In the last example, we can see that the shapes of the two arrays are not similar and therefore they cannot be multiplied together. NumPy can perform such operation by using the concept of broadcasting.
- In broadcasting, the smaller array is broadcast to the larger array to make their shapes compatible with each other.

NumPy Broadcasting

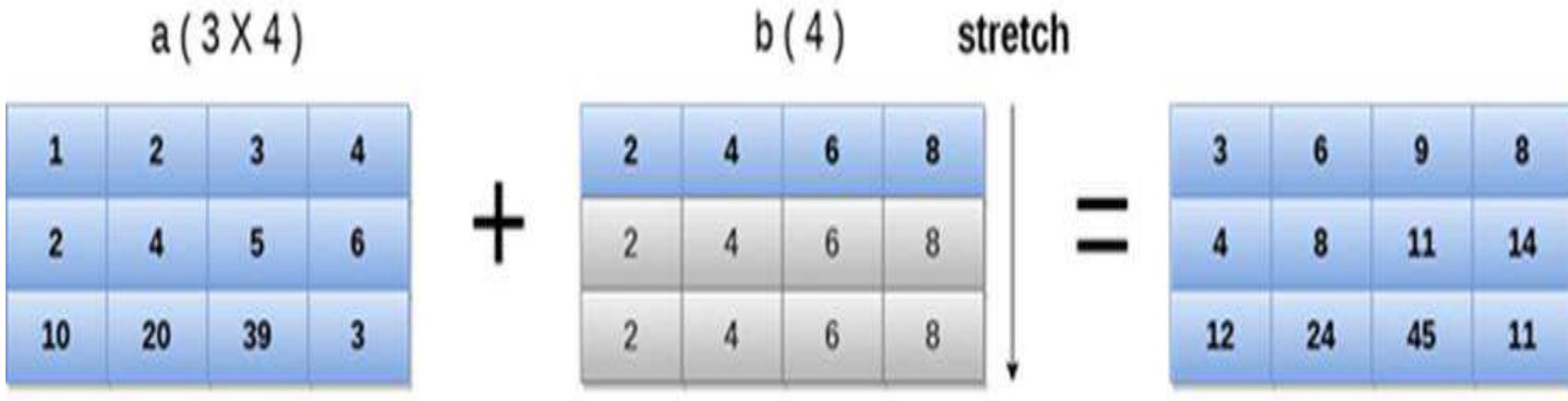


Example

```
import numpy as np  
  
a = np.array([[1,2,3,4],[2,4,5,6],[10,20,39,3]])  
  
b = np.array([2,4,6,8])  
  
print("\nAdding arrays a and b ..")  
  
c = a + b;  
  
print(c)
```

Output: Adding arrays a and b ..[[3 6 9 12] [4
8 11 14][12 24 45 11]]

NumPy Broadcasting



a = [[1, 2, 3, 4], [2, 4, 5, 6], [10, 20, 39, 3]]

b = [[2, 4, 6, 8]]

NumPy Array Iteration



NumPy provides an iterator object, i.e., **nditer** which can be used to iterate over the given array using python standard Iterator interface.

Consider the following example.

Example

```
import numpy as np  
a = np.array([[1,2,3,4],[2,4,5,6],[10,20,39,3]])  
print("Printing array:")
```

NumPy Array Iteration



```
print(a);
print("Iterating over the array:")
for x in np.nditer(a):
    print(x,end=' ')
```

Output:

- Printing array: [[1 2 3 4] [2 4 5 6] [10 20 39 3]]
Iterating over the array: 1 2 3 4 2 4 5 6 10 20 39
3

Order of Iteration



As we know, there are two ways of storing values into the numpy arrays:

- F-style order
- C-style order

Rounding Functions



numpy.around()

This function returns a decimal value rounded to a desired position of the decimal.

Syntax: `numpy.around(num, decimals)`

num: It is the input number.

decimals: It is the number of decimals which to which the number is to be rounded.

The default value is 0. If this value is negative, then the decimal will be moved to the left.



```
import numpy as np  
  
arr = np.array([12.202, 90.23120, 123.020, 23.202])  
  
print("printing the original array values:",end = " ")  
  
print(arr)  
  
print("Array values rounded off to 2 decimal position",np.around(arr, 2))  
  
print("Array values rounded off to -1 decimal position",np.around(arr, -1))
```

The numpy.floor() function



This function is used to return the floor value of the input data which is the largest integer not greater than the input value.

Example

```
import numpy as np  
  
arr = np.array([12.202, 90.23120, 123.020, 23.202])  
  
print(np.floor(arr))
```

The numpy.ceil() function



This function is used to return the ceiling value of the array values which is the smallest integer value greater than the array element.

Example

```
import numpy as np  
  
arr = np.array([12.202, 90.23120, 123.020, 23.202])  
  
print(np.ceil(arr))
```

Numpy statistical functions



The `numpy.amin()` and `numpy.amax()`

These functions are used to find the minimum and maximum of the array elements along the specified axis respectively.

Syntax: `amin(array_name, axis)`

`amax(array_name, axis)`

The axis will be

1: for row wise operation

0: for column wise operation

Example:



```
a = np.array([[2,10,20],[80,43,31],[22,90,18]])  
  
print("The original array:\n")  
  
print(a)  
  
print("\nThe minimum element among the columns of  
array",np.amin(a,0))  
  
print("The maximum element among the columns of  
array",np.amax(a,0))  
  
  
  
  
print("\nThe minimum element among the rows of array",np.amin(a,1))  
  
print("The maximum element among the rows of array",np.amax(a,1))
```

numpy.ptp() function



The name of the function numpy.ptp() is derived from the name peak-to-peak. It is used to return the range of values along an axis.

Syntax: `numpy.ptp(array, axis)`

Range=max value-min value

Example

```
import numpy as np  
  
a = np.array([[2,10,20],[80,43,31],[22,43,10]])  
  
print("Original array:\n",a)  
  
print("Row wise range:",np.ptp(a,1))  
  
print("Column wise range:",np.ptp(a,0))
```



The **numpy.median()** function:

Median is defined as the value that is used to separate the higher range of data sample with a lower range of data sample. The Median is the "middle" of a sorted list of numbers.

numpy.median(): used to calculate the median of the multi-dimensional or one-dimensional arrays.

Syntax: **numpy.median(array_name, axis)**

The **numpy.mean()** function:

The mean can be calculated by adding all the items of the arrays dividing by the number of array elements.

Syntax: **numpy.mean(array_name, axis)**

NumPy Sorting and Searching



Numpy provides a variety of functions for sorting and searching. There are various sorting algorithms like quicksort, merge sort and heapsort which is implemented using the numpy.sort() function.

SN	Algorithm
1	Quick Sort
2	Merge Sort
3	Heap Sort

Syntax: `numpy.sort(a, axis, kind, order)`



Parameter	Description
input	It represents the input array which is to be sorted.
axis	0- column wise operation 1- row wise operation Default: row wise
kind	It represents the type of sorting algorithm. The default is quick sort.
order	It represents the filed according to which the array is to be sorted in the case if the array contains the fields.



Example:

```
a = np.array([[2,45,20],[80,43,31],[22,90,18]])
```

```
print(np.sort(a))      #row wise sorting
```

```
Print("Column wise Sorting", np.sort(a,0))
```

```
#Python module

#Once you save a file with .py extension, it becomes module

#you can import a module in your python code and you can access all
the code written inside that module.

#You can create a python source code file or module using jupyter
notebook.
#You have to click on new button and click on Text file.
#In opened text file, write your python code and save the file with
filename.py extension

#Here, I can access the methods which are defined in module.py file
#First, import that module in jupyter

#syntax: import module_name/python_source_code_file_name

import module

#Now, I can call the function area_circle() of module.py file here

r1=module.area_circle(4)
print('Area of circle:', r1)

r2=module.area_circle(8)
print('Area of circle:', r2)

rect1=module.area_rect(12, 12)
print('Area of Rectangle:', rect1)

#If you import a module using import statement, then all the
methods/code is available in your program

#but, if you want to import some specific function/code of the module,
you can do it.

#Ex: To import only area_rect() of module.py

from module import area_rect

r4=area_rect(10,12)
print('Area of rectangle:', r4)

Area of rectangle: 120
```

```
#Numpy

import numpy as np

#Creating 1-D Ndarray

#numpy.array(): used to create 1-D or 2-D Ndarray

arr=np.array([12, 13, 14, 15, 16])
print(arr)
print(type(arr))

[12 13 14 15 16]
<class 'numpy.ndarray'>

#Creating 2-d array of size row=2 and col=3

arr2=np.array([[2,4,6], [10,12,14]])
print(arr2)

print(type(arr2))

[[ 2  4  6]
 [10 12 14]]
<class 'numpy.ndarray'>

#To view shape of the array

print('Shape of arr:', arr.shape)
print('Shape of arr2:', arr2.shape)

#For 2-D array
print('No. of rows in arr2:', arr2.shape[0])
print('No. of columns in arr2:', arr2.shape[1])

Shape of arr: (5,)
Shape of arr2: (2, 3)
No. of rows in arr2: 2
No. of columns in arr2: 3

#To view the size of the array
#It returns the total number of elements of the Ndarray
print('Size of arr:', arr.size)
print('Size of arr2:', arr2.size)

Size of arr: 5
Size of arr2: 6

#To view the datatype of the Ndarray
```

```

print('Datatype of arr:', arr.dtype)
print('Datatype of arr2:', arr2.dtype)

Datatype of arr: int32
Datatype of arr2: int32

a3=np.array([23.45, 67.889, 69.87, 45.67])
a3.dtype

dtype('float64')

#To view the size of each element of Ndarray

print('Size of each element of arr:', arr.itemsize, 'byte')
print('Size of each element of arr2:', arr2.itemsize, 'byte')
print('Size of each element of a3:', a3.itemsize, 'byte')

Size of each element of arr: 4 byte
Size of each element of arr2: 4 byte
Size of each element of a3: 8 byte

#Reshaping the Ndarray

#reshape(): used to reshape Ndarray

#Syntax: array_name.reshape(row, col)

print('arr2 before reshaping')
print(arr2)

#Ex: Reshape arr2 to row=3 and col=2

arr2=arr2.reshape(3,2)

print('arr2 after reshaping')
print(arr2)

arr2 before reshaping
[[ 2  4  6]
 [10 12 14]]
arr2 after reshaping
[[ 2  4]
 [ 6 10]
 [12 14]]

#Ndarray Indexing

print(arr)
print('First element of arr:', arr[0])
print('Third element of arr:', arr[2])
print('Fourth element of arr:', arr[3])

```

```
[12 13 14 15 16]
First element of arr: 12
Third element of arr: 14
Fourth element of arr: 15

print(arr2)
print('First row of arr2:', arr2[0])

print('First element of first row:', arr2[0][1])
print('Second element of second row:', arr2[1][1])
print('First element of third row:', arr2[2][0])

[[ 2  4]
 [ 6 10]
 [12 14]]
First row of arr2: [2 4]
First element of first row: 4
Second element of second row: 10
First element of third row: 12
```

#Ndarray slicing

```
print(arr)
print('Element from index=1 to index=3:', arr[1:4])
print('Element from index=0 to index=2:', arr[:3])
print('Element from index=2 to index=4:', arr[2:])

[12 13 14 15 16]
Element from index=1 to index=3: [13 14 15]
Element from index=0 to index=2: [12 13 14]
Element from index=2 to index=4: [14 15 16]
```

#Numpy functions for 1-D Array

```
a1=np.array([10, 12, 15, 8, 3, 89, 65, 41])
print('Array:', a1)

print('Minimum element of a1:', a1.min())
print('Maximum element of a1:', a1.max())
print('Sum of a1:', a1.sum())

Array: [10 12 15  8  3 89 65 41]
Minimum element of a1: 3
Maximum element of a1: 89
Sum of a1: 243
```

#2-D array of row=3 and col=3

```
a2=np.array([[12, 4, 10], [20,14,8], [10, 5, 15]])
a2
```

```

array([[12,  4, 10],
       [20, 14,  8],
       [10,  5, 15]])

print('Minimum element of 2-D array a2:', a2.min())
print('Maximum element of 2-D array a2:', a2.max())
print('Sum of 2-D array a2:', a2.sum())

Minimum element of 2-D array a2: 4
Maximum element of 2-D array a2: 20
Sum of 2-D array a2: 98

#Computing row wise minimum element in a2
#Use axis=1, to perform any operation row wise in 2-D array

print('Minimum element of each row in a2:', a2.min(axis=1))

Minimum element of each row in a2: [4 8 5]

row_min=a2.min(axis=1)
print('Minimum element of each row in a2:', row_min)

print('Minimum element of First row in a2:', row_min[0])
print('Minimum element of Second row in a2:', row_min[1])
print('Minimum element of Third row in a2:', row_min[2])

Minimum element of each row in a2: [4 8 5]
Minimum element of First row in a2: 4
Minimum element of Second row in a2: 8
Minimum element of Third row in a2: 5

#Computing row wise maximum element in a2
#Use axis=1, to perform any operation row wise in 2-D array

print('Minimum element of each row in a2:', a2.max(axis=1))

Minimum element of each row in a2: [12 20 15]

row_max=a2.max(axis=1)

print('Maximum element of each row in a2:', row_max)
print('Maximum element of First row in a2:', row_max[0])
print('Maximum element of Second row in a2:', row_max[1])
print('Maximum element of Third row in a2:', row_max[2])

Maximum element of each row in a2: [12 20 15]
Maximum element of First row in a2: 12
Maximum element of Second row in a2: 20
Maximum element of Third row in a2: 15

```

```

#Computing column wise minimum element in a2
#Use axis=0, to perform any operation column wise in 2-D array

print('Minimum element of each column in a2:', a2.min(axis=0))
Minimum element of each column in a2: [10  4  8]

col_min=a2.min(axis=0)

print('Minimum element of each column in a2:', col_min)

print('Minimum element of First column in a2:', col_min[0])
print('Minimum element of Second column in a2:', col_min[1])
print('Minimum element of Third column in a2:', col_min[2])

Minimum element of each column in a2: [10  4  8]
Minimum element of First column in a2: 10
Minimum element of Second column in a2: 4
Minimum element of Third column in a2: 8

#Computing column wise maximum element in a2
#Use axis=0, to perform any operation column wise in 2-D array

print('Max element of each column in a2:', a2.max(axis=0))
Max element of each column in a2: [20 14 15]

col_max=a2.max(axis=0)

print('Maximum element of each column in a2:', col_max)

print('Maximum element of First column in a2:', col_max[0])
print('Maximum element of Second column in a2:', col_max[1])
print('Maximum element of Third column in a2:', col_max[2])

Maximum element of each column in a2: [20 14 15]
Maximum element of First column in a2: 20
Maximum element of Second column in a2: 14
Maximum element of Third column in a2: 15

print('Row wise sum of a2:', a2.sum(axis=1))

Row wise sum of a2: [26 42 30]

print('Column wise sum of a2:', a2.sum(axis=0))

Column wise sum of a2: [42 23 33]

print(a1)

[10 12 15  8  3 89 65 41]

```

```
#std(): to obtain the standard deviation of the numpy Ndarray  
  
print('Standard deviation of a1: ', a1.std())  
Standard deviation of a1:  29.554769073704502  
  
print(a2)  
[[12  4 10]  
 [20 14  8]  
 [10  5 15]]  
  
print('Standard deviation of a2: ', a2.std())  
Standard deviation of a2:  4.747969026493666
```

Machine Learning Using Python

Day 6: Assignment

1. Create two array of size (3, 3) and print their sum and multiplication.
2. Create an array of size (3, 4) and reshape to (4, 3).
3. Create an array for a given range of elements between 2 to 40 step by 3.
4. Create an evenly spaced numpy array of 10 values between 5 to 35
5. Create an array of size 10 and calculate square root and standard deviation.

Pandas



- Pandas is an open-source Python Library providing high-performance data manipulation and analysis tool using its powerful data structures.
- The name Pandas is derived from the word Panel Data – an Econometrics from Multidimensional data.



Pandas is well suited for many different kinds of data:

- Tabular data with heterogeneously-typed columns, as in an SQL table or Excel spreadsheet.
- Ordered and unordered time series data.
- Arbitrary matrix data (homogeneously typed or heterogeneous) with row and column labels.
- Any other form of observational / statistical data sets.

Features



- Columns can be inserted and deleted from DataFrame and higher dimensional objects.
- Flexible reshaping and pivoting of data sets.
- Robust IO tools for loading data from flat files (CSV and delimited), Excel files, databases etc.
- Label-based slicing, indexing and subsetting of large data sets.

Python Pandas - Environment Setup



Standard Python distribution doesn't come bundled with Pandas module. A lightweight alternative is to install Pandas using popular Python package installer, pip.

Syntax: **pip install pandas**

Note: If you install Anaconda Python package, Pandas will be installed by default

Pandas data structures



Data Structure	Description
Series	1D labeled homogeneous array, size immutable.
Data Frames	General 2D labeled, size-mutable tabular structure with potentially heterogeneously typed columns.

Series



Series is a one-dimensional array like structure with homogeneous data.

Key Points

- Homogeneous data
- Dimension is Immutable
- Values of Data Mutable

Features



- A pandas series is a one dimensional labeled numpy array.
- Extracting a single column from a dataframe returns a series.
- We can access the elements of the series using slicing and indexing.
- The indexing on a series also starts from 0.

pandas.Series



A pandas Series can be created using the following constructor –

Sr. No	Parameter & Description
1	Data: data takes various forms like ndarray, list, constants
2	Index: Index values must be unique and hashable, same length as data. Default np.arange(n) if no index is passed.
3	Dtype: dtype is for data type. If None, data type will be inferred
4	Copy: Copy data. Default False



A series can be created using various inputs like –

- Array
- Dictionary
- Scalar value or constant

Create an Empty Series



A basic series, which can be created is an Empty Series.

Example

```
#import the pandas library and aliasing as pd
import pandas as pd
s = pd.Series()
print(s)
```

Example:



```
S=pd.Series([1,2,3,4,5])
```

```
print (S[0])
```

```
print (S[2])
```

```
print(S.values)
```

```
print(S.index)
```

```
print (list(S.index))
```



The index can be specified explicitly.

```
Import numpy as np
```

```
data = np.array(['Ravi','Sahil','Rahul','Ankit'])
```

```
s = pd.Series(data,index=[100,101,102,103])
```

```
print(s)
```

Note:

Numerical index values will overwrite the default indexing.



The index need not be an integer; it can also be a string.

Example:

```
S=pd.Series([1,2,3,4,5], index=['a','b','c','d','e'])
```

```
print(S[0])
```

```
print(S['a'])
```

Note:

String index values will not overwrite the default index.

Accessing Data from Series with Position



Data in the series can be accessed similar to that in an ndarray.

Retrieve the first element:

```
print(s[0])
```

```
print(s['a']) #same as above
```

Retrieve the first three elements in the Series:

```
#retrieve the first three element
```

```
print(s[:3])
```



Adding one element to the series:

```
s['f']=7
```

Adding more than one element to the series: use append().

```
new=Series([8,9,10],index=['g','h','i'])
```

```
s=s.append(new)
```



Deleting elements from a series

Syntax: `s=s.drop([index])`

Example `s=s.drop(['g'])`

Deleting more than one element from a series

`s=s.drop(['i','j'])`

Data Frame



A Data frame is a two-dimensional data structure, i.e., data is aligned in a tabular form in rows and columns.

Features of Data Frame

- columns are of different types
- Size – Mutable
- Labeled axes (rows and columns)
- Can Perform Arithmetic operations on rows and columns

pandas.DataFrame



A pandas DataFrame can be created using constructor –

`pandas.DataFrame(data, index, columns, dtype, copy)`

Sr.No	Parameter & Description
data	data takes various forms like ndarray, series, map, lists, dict, constants and also another DataFrame.
index	For the row labels
columns	For column labels
dtype	Data type of each column.
copy	This command (or whatever it is) is used for copying of data, if the default is False.

Create DataFrame



A pandas DataFrame can be created using various inputs like –

- Lists
- dictionary
- Series
- Numpy ndarrays
- Another DataFrame

Create an Empty DataFrame



A basic DataFrame, which can be created is an Empty Dataframe.

Example

```
#import the pandas library and aliasing as pd  
import pandas as pd  
  
df = pd.DataFrame()  
  
print df
```

Create a DataFrame from Lists



The DataFrame can be created using a single list or a list of lists.

```
import pandas as pd
```

```
data = [1,2,3,4,5]
```

```
df = pd.DataFrame(data)
```

```
print (df)
```

Specifying column name in a data frame.



```
import pandas as pd
```

```
data = [['Ravi',10],['Mohan',12],['Alok',13]]
```

```
df=pd.DataFrame(data,columns=['Name','Age'], dtype=np.float)
```

```
print(df)
```

Create a DataFrame from Dictionary



```
import pandas as pd
```

```
d={'Item': ['A', 'B', 'C', 'D', 'E'], 'Price': [600, 756, 587, 859, 990]}
```

```
frame1=pd.DataFrame(d)
```

```
print(frame1)
```

```
print(frame1['Item'][2]) # printing 3rd item
```

```
print(frame1['Price']) # printing values of Price column
```

Addition of Rows



Add new rows to a DataFrame using the append function. This function will append the rows at the end.

Example

```
f2 = pd.DataFrame([['G', 500]], columns=['Item','Price'])  
frame1=frame1.append(f2)
```



Changing names of columns in a DataFrame

rename()

Used to rename the columns of a data frame.

dataframe.rename(columns={'Old_Name':'New_Name',...}, inplace=True)

```
frame1.rename(columns={'Item':'Products','Price':'Base_price'}, inplace=True)
```



Column Addition

Adding a new column to an existing DataFrame object with column label by passing new series

```
print ("Adding a new column by passing as Series:")
```

```
frame1['Unit']=pd.Series([30,50,80,55,90])
```

```
print(frame1)
```



#Adding a new column using the existing columns in Data Frame

```
frame1['Total_Price']=frame1['Unit']*frame1['Base_price']
```

```
print(frame1)
```



insert(): Insert column into DataFrame at specified location.

insert(loc, column, value, allow_duplicates=False)

```
new_val=['NewDelhi','Mumbai','Chennai',
'Hyderabad','Lucknow']
```

```
frame1.insert(2,column='Place',value=new_val)
```

```
print(frame1)
```

Row Selection



loc():

Access a group of rows and columns by label(s).

Syntax:

```
dataframe.loc[r_label, c_label]
```

To access Multiple Row and column label

```
dataframe.loc[[r_label1,           r_label2..],      [c_label1,  
c_label2,...]]
```



Accessing rows where Item ==A

```
print(frame1.loc[frame1['Item']=='A'])
```

Accessing rows where Price>700

```
print(frame3.loc[frame3['Price']>700])
```

Checking multiple values in column

```
print(frame3.loc[frame3['Item'].isin(['A','D'])])
```



iloc():

Rows can be selected by passing integer location to iloc() function.

Syntax:

```
dataframe.iloc[row_index, col_index]
```

To access multiple rows and columns

```
dataframe.iloc[[r_index1,r_index2..],  
[c_index1,c_index2..]]
```

Slice Rows



Multiple rows can be selected using ‘:’ operator.

Example:

#shows the all records for 4th column.

```
print(frame1.iloc[:,3])
```

#shows the all records for Price column.

```
print(frame1.loc[:,['Price']])
```

```
print(frame1.iloc[:,[1]]) #same as above using iloc
```

Column Deletion



Columns can be deleted from the data frames.

using del function

Syntax:

```
del dataframe_name['Col_Name']
```

Example:

```
del frame1['Unit']
```

```
print(frame1)
```

Deletion of Rows



Use index label to delete or drop rows from a DataFrame. If label is duplicated, then multiple rows will be dropped.

Drop rows with label 0

```
Frame1.drop(0,inplace=True) #deletes first row
```

I/O Tools



File Formats

Python can work with the following file formats:

- Comma-separated values (CSV)
- SQL
- XLSX
- Plain Text (txt)
- XML
- HTML
- Images
- PDF
- DOCX

Comma Separated Values(CSV)



CSV is a simple file format used to store tabular data, such as a spreadsheet or database. It has the file extension .csv.

Reading CSV files in pandas

`read_csv()`: used to read a comma-separated values (csv) file into DataFrame.

Syntax: `df=pandas.read_csv(location)`

Example

```
d=pd.read_csv('comma_sep_val.csv')
```

Reading directly from location

```
df=pd.read_csv('E:\\Python\\Pandas\\comma_sep_val.csv')
```

Reading Text file



#if contents are separated with comma

```
dat1=pd.read_csv('E:\\Python\\Pandas\\data.csv',sep=',',  
header=0)  
  
print(dat1)
```

#if contents are separated with tab

```
dat1=pd.read_csv('E:\\Python\\Pandas\\table.csv',sep='\t'  
,header=0)  
  
print(dat1)
```

Handling Missing values other than NaN



Missing values are specified with NaN. Python will recognize only NaNs as missing, any other missing values such as space, .(dot) will not be recognized by the Python.

na_values - handles non NaN values in a DataFrame.

For example: the data.txt file contains the text Missing in place of NaN. Replace Missing with NaN as

```
dat3=pd.read_csv('E:\\Python\\Pandas\\data.txt',sep=',',na_values  
=[' Missing','NULL'])
```



Data preparation



Converting names of the columns into list

```
dat1.columns.tolist()
```

Remove space from column names

```
dat1.columns=[column.replace(" ", "_") for  
column in dat1.columns]
```

Checking missing Values



isnull(): used to check missing values in a data frame. It returns Boolean values which are True for NaN values.

checking entire data frame

```
print(dat3.isnull())
```

checking Age column only

```
print(dat3['Age'].isnull())
```



Counting missing values from each column

syntax:

```
dataframe_name.isnull().sum()
```

Syntax: df.isnull().sum()

Replacing missing values



dropna(): drop Rows/Columns with Null values.

By default, axis=0, i.e., if any value within a row is NA then the whole row is excluded.



#drops all rows that contains missing data.

`dat3.dropna()`

#drops columns that contains missing data

`dat3.dropna(axis=1)`



fillna(): used to fill missing values with specific criterion. Criterion may be

A. For numerical values

The simplest method is to replace the missing numerical values with mean.

```
data['Unit_Price'].fillna(data[' Unit_Price '].mean(),  
inplace=True)
```

B. For Categorical values



Count the occurrences of each category and replace the missing values with high frequency values.

- **Count frequency of each category**

```
Dataframe['column_name'].value_counts()
```

- **Replace the missing values with highest frequency value**

```
Dataframe.fillna('value', inplace=True)
```

Pandas Statistical Functions



Method	Description
count()	Number of non-null observations
sum()	Sum of values
mean()	Mean of values
median()	Arithmetic median of values
min()	Minimum
max()	Maximum
std()	Bessel-corrected sample standard deviation

mean()



Returns the average value.

`dataframe_name.mean()`

median()

Calculate the median of the values.

`dataframe_name.median()`

Summarizing Data



describe() : computes a summary of statistics pertaining to the DataFrame columns.

```
dataframe_name.describe(include)
```

'include' is the argument which is used to pass necessary information regarding what columns need to be considered for summarizing. The default is 'number'.

- object – Summarizes String columns
- number – Summarizes Numeric columns
- all – Summarizes all columns together i.e. numbers and strings



info(): used to get a concise summary of the dataframe.

Ex: `dataframe_name.info()`

The summary includes list of all columns with their data types and the number of non-null values in each column.

#DataFrame

```
import pandas as pd  
#Creating empty dataframe
```

```
df=pd.DataFrame()  
print(df)  
print(type(df))  
  
Empty DataFrame  
Columns: []  
Index: []  
<class 'pandas.core.frame.DataFrame'>
```

```
df1=pd.DataFrame([1,2,3,4,5])  
df1
```

```
    0  
0  1  
1  2  
2  3  
3  4  
4  5
```

#DataFrame using List

```
data=[[1,'Saurabh'], [2, 'Abhi'], [3, 'Jatin']]  
df2=pd.DataFrame(data)  
df2
```

```
    0      1  
0  1  Saurabh  
1  2      Abhi  
2  3      Jatin
```

#Giving names to columns

```
df3=pd.DataFrame(data, columns=['Emp_ID', 'Name'])  
df3
```

```
    Emp_ID      Name  
0        1  Saurabh  
1        2      Abhi  
2        3      Jatin
```

#Creating DataFrame using dictionary

```
sales={'Goods':['A', 'B', 'C', 'D', 'E'], 'Price':[150,285,325,450,525]}
```

```
df_sales=pd.DataFrame(sales)
df_sales

   Goods  Price
0      A    150
1      B    285
2      C    325
3      D    450
4      E    525

#Adding new row in a DataFrame

new=pd.DataFrame([['F', 480], ['G', 650]], columns=['Goods', 'Price'])
new

   Goods  Price
0      F    480
1      G    650

df_sales

   Goods  Price
0      A    150
1      B    285
2      C    325
3      D    450
4      E    525

#append(): used to append the data at the end of the dataframe

#Ex: Appending the values of new dataframe at the end of df_sales

df_sales=df_sales.append(new)
df_sales

   Goods  Price
0      A    150
1      B    285
2      C    325
3      D    450
4      E    525
0      F    480
1      G    650

#Use ignore_index=True, to append the values of new dataframe in df_sales with
#ignoring the index of new dataframe

df_sales=df_sales.append(new, ignore_index=True)
df_sales
```

```
Goods  Price
0      A    150
1      B    285
2      C    325
3      D    450
4      E    525
5      F    480
6      G    650
7      F    480
8      G    650
```

#Data updation

#Chinging the price of 2nd Good

```
df_sales['Price'][1]=375
df_sales
```

```
C:\Users\hp\Anaconda3\lib\site-packages\ipykernel_launcher.py:5:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame
```

See the caveats in the documentation:

https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy

```
Goods  Price
0      A    150
1      B    375
2      C    325
3      D    450
4      E    525
5      F    480
6      G    650
7      F    480
8      G    650
```

```
df3=df_sales.rename(columns={'Goods':'Product_name',
'Price':'Base_price'})
df3
```

```
Product_name  Base_price
0            A        150
1            B        375
2            C        325
3            D        450
4            E        525
5            F        480
6            G        650
```

```
7          F    480
8          G    650
```

```
df_sales
```

```
   Goods  Price
0      A    150
1      B    375
2      C    325
3      D    450
4      E    525
5      F    480
6      G    650
7      F    480
8      G    650
```

#Changing the column names

#rename(): used to rename the columns of dataframe

#Syntax: df_name.rename(columns={'Old_col_name1':'New_col_name1', 'Old_col_name2':'New_col_name2'...})

#Ex: Rename 'Goods' to 'Product_name' and 'Price' to 'Base_price'

```
df_sales.rename(columns={'Goods':'Product_name',
'Price':'Base_price'}, inplace=True)
df_sales
```

```
   Product_name  Base_price
0              A        150
1              B        375
2              C        325
3              D        450
4              E        525
5              F        480
6              G        650
7              F        480
8              G        650
```

#Column Insertion

#We can insert new column at the end of the dataframe as a Series

#Ex: Adding new column at the end of the dataframe names 'Unit'

```
df_sales['Unit']=pd.Series([12, 10, 15, 16, 18, 20, 21, 23, 25])
df_sales
```

```
   Product_name  Base_price  Unit
0              A        150     12
```

```

1      B    375    10
2      C    325    15
3      D    450    16
4      E    525    18
5      F    480    20
6      G    650    21
7      F    480    23
8      G    650    25

```

#Inserting new column at specific location

#insert()

#Syntax:

```
#df_name.insert(index, column='new_col_name', value=new_value)
```

#Ex: Inserting a new column named 'City' at the index=1

```
city=['LK0', 'KNP', 'GKP', 'LMP', 'DEL', 'MUM', 'KNP', 'LK0', 'KNP']
```

```
df_sales.insert(1, column='City', value=city)
df_sales
```

	Product_name	City	Base_price	Unit
0		A LK0	150	12
1		B KNP	375	10
2		C GKP	325	15
3		D LMP	450	16
4		E DEL	525	18
5		F MUM	480	20
6		G KNP	650	21
7		F LK0	480	23
8		G KNP	650	25

#Adding a new column using existing columns of the dataframe

#Adding a new column named 'Total_price'

```
df_sales['Total_price']=df_sales['Base_price']*df_sales['Unit']
df_sales
```

	Product_name	City	Base_price	Unit	Total_price
0		A LK0	150	12	1800
1		B KNP	375	10	3750
2		C GKP	325	15	4875
3		D LMP	450	16	7200
4		E DEL	525	18	9450
5		F MUM	480	20	9600
6		G KNP	650	21	13650

```
7          F  LK0      480     23    11040
8          G  KNP      650     25    16250
```

#Data Access

#to view specific column

```
df_sales['Product_name']
```

```
0    A
1    B
2    C
3    D
4    E
5    F
6    G
7    F
8    G
```

Name: Product_name, dtype: object

```
df_sales.Unit
```

```
0    12
1    10
2    15
3    16
4    18
5    20
6    21
7    23
8    25
```

Name: Unit, dtype: int64

```
df_sales.Product_name
```

```
0    A
1    B
2    C
3    D
4    E
5    F
6    G
7    F
8    G
```

Name: Product_name, dtype: object

#Accessing data in DataFrame

#loc(): access the data from a dataframe using column/row label

```

#Accessing data where Product_name='A'

df_sales.loc[df_sales.Product_name=='A']

   Product_name  City  Base_price  Unit  Total_price
0              A    LKO        150     12       1800

#Accessing data where City='LK0'

df_sales.loc[df_sales.City=='LK0']

   Product_name  City  Base_price  Unit  Total_price
0              A    LKO        150     12       1800
7              F    LKO        480     23      11040

#Access data where unit>=15

df_sales.loc[df_sales.Unit>=15]

   Product_name  City  Base_price  Unit  Total_price
2              C    GKP        325     15       4875
3              D    LMP        450     16       7200
4              E    DEL        525     18       9450
5              F    MUM        480     20       9600
6              G    KNP        650     21      13650
7              F    LKO        480     23      11040
8              G    KNP        650     25      16250

#Access data where unit>=20

df_sales.loc[df_sales.Unit>=20]

   Product_name  City  Base_price  Unit  Total_price
5              F    MUM        480     20       9600
6              G    KNP        650     21      13650
7              F    LKO        480     23      11040
8              G    KNP        650     25      16250

#Access data where unit<=20

df_sales.loc[df_sales.Unit<=20]

   Product_name  City  Base_price  Unit  Total_price
0              A    LKO        150     12       1800
1              B    KNP        375     10      3750
2              C    GKP        325     15       4875
3              D    LMP        450     16       7200
4              E    DEL        525     18       9450
5              F    MUM        480     20       9600

```

```
#Access data where Total_price>4000
```

```
df_sales.loc[df_sales.Total_price>4000]
```

	Product_name	City	Base_price	Unit	Total_price
2		C GKP	325	15	4875
3		D LMP	450	16	7200
4		E DEL	525	18	9450
5		F MUM	480	20	9600
6		G KNP	650	21	13650
7		F LKO	480	23	11040
8		G KNP	650	25	16250

```
#Access data where Total_price<5000
```

```
df_sales.loc[df_sales.Total_price<5000]
```

	Product_name	City	Base_price	Unit	Total_price
0		A LKO	150	12	1800
1		B KNP	375	10	3750
2		C GKP	325	15	4875

```
#Access data where City='LKO' or 'GKP'
```

```
df_sales.loc[df_sales.City.isin(['LKO', 'GKP'])]
```

	Product_name	City	Base_price	Unit	Total_price
0		A LKO	150	12	1800
2		C GKP	325	15	4875
7		F LKO	480	23	11040

```
df_sales
```

	Product_name	City	Base_price	Unit	Total_price
0		A LKO	150	12	1800
1		B KNP	375	10	3750
2		C GKP	325	15	4875
3		D LMP	450	16	7200
4		E DEL	525	18	9450
5		F MUM	480	20	9600
6		G KNP	650	21	13650
7		F LKO	480	23	11040
8		G KNP	650	25	16250

```
#iloc(): used to access the data from a dataframe using integer based indexing
```

```
#Syntax: df_name.iloc[start_row_index:end_row_index,  
start_col_index:end_col_index]
```

```
#Ex: To access all records from row index=2 to 5 and col index= 1 to 3
```

```
df_sales.iloc[2:6, 1:4]
```

```
City  Base_price  Unit  
2    GKP        325    15  
3    LMP        450    16  
4    DEL        525    18  
5    MUM        480    20
```

#Ex: To access all records from row=1 to 3 and col=2 to end

```
df_sales.iloc[1:4, 2:]
```

```
Base_price  Unit  Total_price  
1          375    10      3750  
2          325    15      4875  
3          450    16      7200
```

```
df_sales.iloc[:, 1:3]
```

```
City  Base_price  
0    LK0        150  
1    KNP        375  
2    GKP        325  
3    LMP        450  
4    DEL        525  
5    MUM        480  
6    KNP        650  
7    LK0        480  
8    KNP        650
```

```
df_sales.iloc[1:5, :]
```

```
Product_name  City  Base_price  Unit  Total_price  
1            B    KNP        375    10      3750  
2            C    GKP        325    15      4875  
3            D    LMP        450    16      7200  
4            E    DEL        525    18      9450
```

#Column deletion

#To delete 'City' column

```
del df_sales['City']  
df_sales
```

```
Product_name  Base_price  Unit  Total_price  
0            A        150    12      1800  
1            B        375    10      3750  
2            C        325    15      4875  
3            D        450    16      7200  
4            E        525    18      9450
```

```
5          F      480     20      9600
6          G      650     21     13650
7          F      480     23     11040
8          G      650     25     16250
```

#Row deletion

#drop(): to delete the row from the dataframe

#To delete the row at index=2

```
df_sales.drop(2, inplace=True)
df_sales
```

```
   Product_name  Base_price  Unit  Total_price
0            A        150    12       1800
1            B        375    10       3750
3            D        450    16       7200
4            E        525    18       9450
5            F        480    20      9600
6            G        650    21     13650
7            F        480    23     11040
8            G        650    25     16250
```

```
#Reading csv file
```

```
import pandas as pd
```

```
#1. Upload the file in jupyter and then read
```

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

```
      Row ID      Order ID   Order Date   Ship Date      Ship  
Mode \  
0      1  CA-2013-152156  11/9/2013  11/12/2013  Second Class  
1      2  CA-2013-152156  11/9/2013  11/12/2013  Second Class  
2      3  CA-2013-138688  6/13/2013   6/17/2013  Second Class  
3      4  US-2012-108966  10/11/2012  10/18/2012 Standard Class  
4      5  US-2012-108966  10/11/2012  10/18/2012 Standard Class  
...    ...    ...    ...    ...  
9989  9990  CA-2011-110422  1/22/2011  1/24/2011  Second Class  
9990  9991  CA-2014-121258  2/27/2014   3/4/2014 Standard Class  
9991  9992  CA-2014-121258  2/27/2014   3/4/2014 Standard Class  
9992  9993  CA-2014-121258  2/27/2014   3/4/2014 Standard Class  
9993  9994  CA-2014-119914  5/5/2014   5/10/2014  Second Class
```

```
      Customer ID      Customer Name      Segment      Country  
City \  
0      CG-12520      Claire Gute  Consumer  United States  
Henderson  
1      CG-12520      Claire Gute  Consumer  United States  
Henderson  
2      DV-13045      Darrin Van Huff Corporate United States  Los  
Angeles  
3      SO-20335      Sean O'Donnell Consumer  United States  Fort  
Lauderdale  
4      SO-20335      Sean O'Donnell Consumer  United States  Fort  
Lauderdale  
...    ...    ...    ...  
9989  TB-21400  Tom Boeckenhauer  Consumer  United States
```

Miami					
9990	DB-13060	Dave Brooks	Consumer	United States	
Costa Mesa					
9991	DB-13060	Dave Brooks	Consumer	United States	
Costa Mesa					
9992	DB-13060	Dave Brooks	Consumer	United States	
Costa Mesa					
9993	CC-12220	Chris Cortes	Consumer	United States	
Westminster					

Category \	Postal Code	Region	Product ID	Category Sub-
0 ... Bookcases	42420	South	FUR-B0-10001798	Furniture
1 ... Chairs	42420	South	FUR-CH-10000454	Furniture
2 ... Labels	90036	West	OFF-LA-10000240	Office Supplies
3 ... Tables	33311	South	FUR-TA-10000577	Furniture
4 ... Storage	33311	South	OFF-ST-10000760	Office Supplies
...
9989 ... Furnishings	33180	South	FUR-FU-10001889	Furniture
9990 ... Furnishings	92627	West	FUR-FU-10000747	Furniture
9991 ... Phones	92627	West	TEC-PH-10003645	Technology
9992 ... Paper	92627	West	OFF-PA-10004041	Office Supplies
9993 ... Appliances	92683	West	OFF-AP-10002684	Office Supplies

Quantity \	Product Name	Sales
0	Bush Somerset Collection Bookcase	261.9600
1	Hon Deluxe Fabric Upholstered Stacking Chairs,...	731.9400
2	Self-Adhesive Address Labels for Typewriters b...	14.6200
3	Bretford CR4500 Series Slim Rectangular Table	957.5775
4	Eldon Fold 'N Roll Cart System	22.3680
5
9989	Ultra Door Pull Handle	25.2480

```

3
9990 Tenex Bl-RE Series Chair Mats for Low Pile Car... 91.9600
2
9991                               Aastra 57i VoIP phone 258.5760
2
9992 It's Hot Message Books with Stickers, 2 3/4" x 5" 29.6000
4
9993 Acco 7-Outlet Masterpiece Power Center, Wihtou... 243.1600
2

      Discount    Profit
0        0.00   41.9136
1        0.00  219.5820
2        0.00    6.8714
3        0.45 -383.0310
4        0.20    2.5164
...
9989        0.20    4.1028
9990        0.00   15.6332
9991        0.20   19.3932
9992        0.00   13.3200
9993        0.00   72.9480

```

[9994 rows x 21 columns]

```

type(df)
pandas.core.frame.DataFrame

#2. Reading directly from location

```

#Ex: Reading p1.csv file whose physical/actual location is D:\BigData_All_content\csv.

```
df2=pd.read_csv('D:\\BigData_All_content\\csv\\p1.csv')
```

```
#df2=pd.read_csv(r'D:\\BigData_All_content\\csv\\p1.csv')
```

df2

	Name	Salary	Country
0	Rahul	40000.0	India
1	Ravi	32000.0	USA
2	Saurabh	45000.0	Italy
3	Mahesh	54000.0	India
4	Rishabh	72000.0	India
5	Rajat	NaN	USA
6	Abhishek	92000.0	Italy
7	Jatin	55000.0	India
8	Gaurav	35000.0	Italy

```
9      Ritik  48000.0    USA
10     Mani  34000.0    NaN
11     Neha  32100.0    USA
12     Naman  35600.0   Italy
13    Pradeep  43800.0   Italy
```

```
df3=pd.read_csv(r'D:\BigData_All_content\csv\p1.csv')
df3
```

```
      Name  Salary Country
0    Rahul  40000.0  India
1    Ravi  32000.0    USA
2  Saurabh  45000.0  Italy
3   Mahesh  54000.0  India
4  Rishabh  72000.0  India
5   Rajat     NaN    USA
6  Abhishek  92000.0  Italy
7   Jatin  55000.0  India
8   Gaurav  35000.0  Italy
9    Ritik  48000.0    USA
10   Mani  34000.0    NaN
11   Neha  32100.0    USA
12   Naman  35600.0   Italy
13  Pradeep  43800.0   Italy
```

#Reading the txt file whose contents are seperated with ','

```
df_data=pd.read_csv(r'D:\BigData_All_content\csv\data.txt', sep=',')
df_data
```

```
    Age Customer  Income Location
0    30         A    1200      A1
1    32         B    1333      B1
2    33         C    1400      C1
```

#Reading the txt file whose contents are seperated with 'tab'

```
df_table=pd.read_csv(r'D:\BigData_All_content\csv\table.txt', sep='\t')
df_table
```

```
    Age Customer name  Income Location
0    30         A  1200.0      A1
1    32         B  1333.0      A2
2    33         #    NaN        #
3    34         c    NaN        A4
```

#Handling missing values in your data

```
#Missing values can be given as 'Empty cell/data' in a file
#Or Missing values may be represented by any specific symbol
```

```
#By default, read_csv() only considers 'Empty cell/data' as Missing values
```

```
#Ex: IN table.txt, missing values are given as Empty cell/data and '#' symbol
```

```
df_table1=pd.read_csv(r'D:\BigData_All_content\csv\table.txt', sep='\t', na_values='#')
```

```
df_table1
```

```
   Age Customer name Income Location
0    30           A  1200.0      A1
1    32           B  1333.0      A2
2    33          NaN     NaN      NaN
3    34           c     NaN      A4
```

```
#Reading Sales_record.csv file
```

```
#Missing data is represented by using '?' symbol
```

```
df_sales=pd.read_csv(r'D:\BigData_All_content\csv\Sales_record.csv', na_values='?')
```

```
df_sales
```

	Region	Country
Item Type \ 0 Baby Food	Australia and Oceania	Tuvalu
1 Cereal	Central America and the Caribbean	Grenada
2 Supplies	Europe	Russia Office
3 Fruits	Sub-Saharan Africa	Sao Tome and Principe
4 Supplies	Sub-Saharan Africa	Rwanda Office
..
93	Sub-Saharan Africa	Mali
Nan		
94 Fruits	Asia	Malaysia
95 Vegetables	Sub-Saharan Africa	Sierra Leone
96 Personal Care	North America	Mexico
97 Household	Sub-Saharan Africa	Mozambique

	Units Sold	Unit Price	Unit Cost	Total Revenue	Total Cost	\
0	9925.0	255.28	159.42	2533654.00	1582243.50	
1	2804.0	205.70	117.11	576782.80	328376.44	
2	1779.0	NaN	524.96	1158502.59	933903.84	
3	8102.0	9.33	6.92	75591.66	56065.84	
4	5062.0	651.21	524.96	3296425.02	2657347.52	
..
93	888.0	NaN	35.84	97040.64	31825.92	
94	6267.0	9.33	6.92	58471.11	43367.64	
95	1485.0	154.06	90.93	228779.10	135031.05	
96	NaN	81.73	56.67	471336.91	326815.89	
97	5367.0	668.27	502.54	3586605.09	2697132.18	

	Total Profit	Sales Channel
0	951410.50	Offline
1	248406.36	Online
2	224598.75	Offline
3	19525.82	Offline
4	639077.50	Offline
..
93	65214.72	Online
94	15103.47	Offline
95	93748.05	Offline
96	144521.02	Offline
97	889472.91	Offline

[98 rows x 10 columns]

```
#Ex: If missing data is represented by using '@' and '$'
#df_sales=pd.read_csv(r'D:\BigData_All_content\csv\Sales_record.csv',
na_values=['@', '$'])
```

Machine Learning Using Python

Day 7: Assignment

1. Write a Python program to create and display a series of data using Pandas module.
2. Create a pandas series of 10 elements and specify their index as 101 to 110.
3. Print bottom 5 elements of the series created in question 2.
4. Insert 3 new elements in above series on index 111, 112 and 113.
5. Delete the elements at index- 103, 104,107,111 in above list.

Machine Learning Using Python

Day 8: Assignment

1. Write a Pandas program to create and display a DataFrame from a specified dictionary data which has the index labels. Sample Python dictionary data and list labels:

- a. exam_data = {'name': ['Ankita', 'Dia', 'Kapil', 'Jayesh', 'Esha', 'Mayank', 'Ravi', 'Lata', 'Kamal', 'Jatin'],
- b. 'score': [12.5, 9, 16.5, 15, 9, 20, 14.5, 17.5, 8, 19],
- c. 'attempts': [1, 3, 2, 3, 2, 3, 1, 1, 2, 1],
- d. 'qualify': ['yes', 'no', 'yes', 'no', 'no', 'yes', 'yes', 'no', 'no', 'yes']}

2. Create a data frame using dictionary.

- a. Dictionary ('id':[P101,P102,P103,P104,P105], 'Price':[256, 340, 540, 260, 470])
- b. Print the price of product id – p102.
- c. Print values of Price column.
- d. Rename the column id to Product_Id and Price to Base_Price.

3. Create a new data frame with three columns – Product_Name, Cost, Sales.

- a. Add 10 values in data frame.
- b. Add a new column named quantity with 10 values.
- c. Add a new column named: Profit and total_profit and fill values.
- d. Insert a new column named location after Product_Name column with 10 cities.

(New Delhi, Lucknow, Kolkata, Lucknow, New Delhi, Bengaluru, Chennai, Chennai, Kolkata, Bengaluru)

Machine Learning Using Python

Day 9: Assignment

1. Create a data frame:

```
'Name': ['Ankit', 'Amit', 'Aishwarya', 'Priyanka', 'Priya', 'Shaurya' ],  
'Age': [21, 19, 20, 16, 17, 21],  
'Stream': ['Math', 'Commerce', 'Science', 'Math', 'Commerce',  
'Science'],  
'Percentage': [88, 92, 95, 70, 65, 78]}
```

- A. Insert a new row – Name: Sahil, Age-23, Stream- Commerce, Percentage-88.
- B. selecting rows where percentage is >80
- C. Selecting all the rows from the given dataframe in which ‘Stream’ is Commerce and science.
- D. Selecting all the rows from the given dataframe in which ‘Age’ is greater than 18.
- E. Print sum of age and percentage only.

- 2. Get the first 3 rows of above DataFrame.
- 3. Write a Pandas program to count the number of rows and columns of a DataFrame.
- 4. Print mean of Age and Percentage.
- 5. Print Minimum age



- [Data Preprocessing](#)
- [Why Data Preprocessing](#)
- [Data Preprocessing Process](#)
- [Data Preprocessing Steps](#)
- [Step 1: Import Libraries](#)
- [Step 2: Loading the dataset](#)
- [Step 3: Identify Independent and Dependent Variables](#)
- [Independent Variables](#)
- [Dependent Variable](#)
- [Step 4: Taking care of Missing Data in Dataset](#)
- [Replacing missing values](#)
- [Step 5: Encoding Categorical Data](#)
- [Label Encoding](#)
- [One Hot Encoding](#)
- [Step 6: Feature Scaling](#)
- [Step 7: Splitting the Dataset into Training set and Test Set](#)

3 Data Preprocessing



- In any machine learning process, data preprocessing is the step in which data is transformed or encoded so that the machine can process the data easily. The features of data can now be easily interpreted by machine learning algorithms.
- Pre-processing refers to the transformations applied to our data before feeding it to the algorithm.
- It is a technique that is used to convert the raw data into a clean data set. In other words, whenever the data is gathered from different sources it is collected in raw format which is not feasible for the analysis.

Why Data Preprocessing



- For achieving better results from the applied model in Machine Learning projects, the format of the data has to be in a proper manner. Some specified Machine Learning model needs information in a specified format.

For example, Random Forest algorithm does not support null values, therefore to execute random forest algorithm null values have to be managed from the original raw data set.

- Another aspect is that data set should be formatted in such a way that more than one Machine Learning and Deep Learning algorithms are executed in one data set, and best out of them is chosen.
- The data has to be in proper format and any missing values must be processed before applying the Machine Learning algorithms.

Data Preprocessing Process



- Formatting the data to make it suitable for ML (structured format).
- Cleaning the data to remove incomplete variables.
- Sampling the data further to reduce running times for algorithms and memory requirements.
- Selecting data objects and attributes for the analysis.
- Creating/changing the attributes.

Data Preprocessing Steps



The rituals programmers usually perform data pre processing in 7 simple steps.

- Step 1: Importing the libraries
- Step 2: Loading the Dataset
- Step 3: Identify independent and dependent feature
- Step 4: Handling of Missing Data
- Step 5: Handling of Categorical Data
- Step 6: Feature Scaling
- Step 7: Splitting the dataset into training and testing datasets

Step 1: Import Libraries



- First step is usually importing the libraries that will be needed in the program. A library is essentially a collection of modules that can be called and used. Built-in functions are defined in libraries which can be used by the programmer.

For example, importing the library pandas and assigning alias as pd.

```
import pandas as pd
```

Step 2: Loading the dataset



- Load the dataset into pandas data frame using `read_csv()` function. The `read_csv()` function reads comma separated values (csv) dataset into pandas dataframe.

```
import pandas as pd
```

```
dataset = pd.read_csv('Data_for_preprocessing.csv')
```

Step 3: Identify Independent and Dependent Variables



- The next step of data preprocessing is to identify independent and dependent variables from the dataset.
- All the features of any dataset are not important for Machine Learning algorithm.
- Classification of dependent and independent feature is very important in Machine Learning.

Independent Variables



- Independent variables (also referred to as Features) are the input for a process that is being analyzes.
- Usually independent features/variables are also known as input features/variables and represented as X.

A	B	C	D
Country	Age	Salary	Purchased
France	44	72000	No
Spain	27	48000	Yes
Germany	30	54000	No
Spain	NaN	NaN	No
Germany	40	1000	Yes
France	35	58000	Yes
Spain	78	52000	No
France	NaN	79000	Yes
Germany	50	83000	No
France	37	NaN	Yes



Independent Feature



- For example, in Data_for_preprocessing dataset, the features such as Country, Age and Salary is known as independent features because they are not dependent to Purchased feature.
- They must be extracted before starting Machine Learning process.
- They can be extracted from the dataset as follows:

X=dataset.drop(['Purchased'], axis=1)

Dropping the ‘Purchased’ feature from the dataset and initializing the remaining features to X. Here, axis=1 means dropping the column named ‘Purchased’ from the dataset.

Dependent Variable



- Dependent variables/features are the output of the process.
- Dependent features/variables are also known as output feature/variable and represented as y.

	A	B	C	D
1	Country	Age	Salary	Purchased
2	France	44	72000	No
3	Spain	27	48000	Yes
4	Germany	30	54000	No
5	Spain	NaN	NaN	No
6	Germany	40	1000	Yes
7	France	35	58000	Yes
8	Spain	78	52000	No
9	France	NaN	79000	Yes
10	Germany	50	83000	No
11	France	37	NaN	Yes



Dependent/Output
Variable/Feature



- The result (whether a user purchased or not) is the dependent variable.
- It must be extracted before starting Machine Learning Process.
- It can be extracted as follows:

```
y=dataset['Purchased']
```

Now, the ‘Purchased’ column of dataset will be assigned to y.

Step 4: Taking care of Missing Data in Dataset



- In Python, specifically Pandas, NumPy and Scikit-Learn, missing values are represented as NaN.
- Values with a NaN value are ignored from operations like sum, count, etc.
- Missing values are specified with NaN. Python will recognize only NaNs as missing.
- Any other missing values such as space, .(dot), *, \$ or # will not be recognized by the Python as missing values.
- Missing values other than NaN are handled by **na_values** parameter of **read_csv()**.



- **na_values** - handles non NaN values in a DataFrame.

For example: In the `Data_for_preprocessing.csv` file, the missing values are represented by '#'.
The '#' can be replaced with NaN as

```
dataset=pd.read_csv('Data_for_preprocessing.csv', na_values=[' #','NULL'])
```

- Here, `na_values=[' #','NULL']` specifies that the # and NULL values are treated as NaN.
- We can specify any symbol as missing value in `na_values`. The symbol depends upon the dataset being used.

Checking Missing Values



- ❑ **isnull()**: isnull() function is used to check missing values in a data frame. It returns Boolean values which are True for NaN values.

- ❑ **Checking entire data frame**

```
print(dataset.isnull())
```

- ❑ **Checking Age column only**

```
print(dataset['Age'].isnull())
```

- ❑ **Counting missing values from each column**

```
print(dataset.isnull().sum())
```

Replacing missing values



- **A Simple Option:** Drop Columns or rows with Missing Values

dropna(): dropna() function is used to drop Rows/Columns with NaN values.

- **To drop columns with missing values:**

```
X=X.dropna(axis=1)
```

Now, the column which has NaN values will be dropped from the X dataframe.

- **To drop rows with missing values:**

```
X=X.dropna()
```

Now, all the rows with NaN values are dropped from the X dataframe.

Replacing missing values



- **A Better Option: Imputation**
- **The Imputer() class can take a few parameters —**
 - ▣ **missing_values:** The missing_values placeholder which has to be imputed. By default is NaN.
 - ▣ **strategy :** The data which will replace the NaN values from the dataset. The strategy argument can take the values – ‘mean’(default), ‘median’, ‘most_frequent’.
 - ▣ **axis :** We can either assign it 0 or 1. 0 to impute along columns and 1 to impute along rows.
- **Imputer works on numbers, not strings.**

Replacing Numerical Values



- For numerical values, the simplest method is to replace the missing numerical values with mean.

```
from sklearn.preprocessing import Imputer  
  
fill_NaN = Imputer(missing_values='NaN', strategy='mean', axis=0)  
  
X[['Age','Salary']] = fill_NaN.fit_transform(X[['Age','Salary']])  
  
print (X)
```

- Note:** Since ‘Age’ and ‘Salary’ column contains numerical values. So the missing values of ‘Age’ and ‘Salary’ column is replaced by their mean.

Replacing Categorical Values



- For Categorical values, count the occurrences of each category and replace the missing values with high frequency values.
- Count frequency of each category

```
#Imputing missing values of categorical column 'Country'
```

```
#Counting frequency of each category in 'Country' Column using value_counts()
```

```
X['Country'].value_counts()
```

Output:

```
France    4
```

```
Spain     3
```

```
Germany   1
```



- Replace the missing values with highest frequency value

Output suggests that the most frequent value is 'France'. So replace the NaN values of 'Country' Column with 'France'.

#Replacing the NaN values with 'France'

```
X['Country'].fillna('France', inplace=True)
```

- Checking missing values again,

```
X.isnull().sum()
```

Output:

Country	0
Age	0
Salary	0

Step 5: Encoding Categorical Data



- Machine learning algorithms require numerical inputs.
- Categorical data are variables that contain label values rather than numeric values.
- The number of possible values is often limited to a fixed set.
- Machine learning algorithms cannot work with variables in text form.
- Categorical values must be transformed into numeric values to work with machine learning algorithm.

Encoding Categorical Data



Categorical values can be transformed in to numeric values by :

- Label Encoding
- One Hot Encoding



LabelEncoder:

- Encode target labels with value between 0 and n_classes-1.
- This transformer should be used to encode target values, i.e. y, and not the input X.
- **Example:**

```
from sklearn.preprocessing import LabelEncoder  
  
lb_encode = LabelEncoder()  
  
# Encode labels in column 'Country'.  
  
X['Country']= lb_encode.fit_transform(X['Country'])  
  
print(X.head())
```

 Output:

```
from sklearn.preprocessing import LabelEncoder  
  
label_encoder = LabelEncoder()  
  
# Encode Labels in column 'Country'.  
x['Country']= label_encoder.fit_transform(x['Country'])  
print(x)
```

	Country	Age	Salary
0	0	44.000	72000.000000
1	2	27.000	48000.000000
2	1	30.000	54000.000000
3	2	42.625	52571.428571
4	0	40.000	1000.000000
5	0	35.000	58000.000000
6	2	78.000	52000.000000
7	0	42.625	52571.428571
8	0	50.000	83000.000000
9	0	37.000	52571.428571

Limitation of Label Encoding



- Label encoding convert the data in machine readable form, but it assigns a unique number(starting from 0) to each class of data.
- This may lead to the generation of priority issue in training of data sets. A label with high value may be considered to have high priority than a label having lower value.
- For example, on Label Encoding ‘Country’ column, let France is replaced with 0 , Germany is replaced with 1 and Spain is replaced with 2.
- With this, it can be interpreted that Spain have high priority than Germany and France while training the model. But actually there is no such priority relation between these countries.
- This can be overcome by the concept of **One-Hot Encoding**.

One Hot Encoding



- The technique to convert categorical values into a numerical vector is known as one hot encoding.
- It refers to splitting the column which contains numerical categorical data to many columns depending on the number of categories present in that column. Each column contains “0” or “1” corresponding to which column it has been placed.
- The resulting vector will have only one element equal to 1 and the rest will be 0.

For example, In given dataset ‘Country’ column contains categorical data. So, ‘Country’ column must be converted into numerical values before starting Machine Learning Process.



get_dummies(): Used to encode categorical values into numerical values.

Syntax: get_dummies(dataframe)

Example: X=get_dummies(X)

```
X=pd.get_dummies(X)  
X
```

	Age	Salary	Country_Frace	Country_France	Country_Germany	Country_Spain
0	44.000	72000.000000	0	1	0	0
1	27.000	48000.000000	0	0	0	1
2	30.000	54000.000000	0	0	1	0
3	42.625	52571.428571	0	0	0	1
4	40.000	1000.000000	1	0	0	0

Step 6: Feature Scaling



- Real world dataset contains features that highly vary in magnitudes, units, and range.
- Differences in the scales across input variables may increase the difficulty of the problem being modelled. An example of this is that large input values (e.g. a spread of hundreds or thousands of units) can result in a model that learns large weight values.
- A model with large weight values is often unstable, meaning that it may suffer from poor performance during learning and sensitivity to input values resulting in higher generalization error.
- **Feature Scaling or Standardization** is a step of Data Pre Processing which is applied to independent variables or features of data.
- It basically helps to normalise the data within a particular range. Sometimes, it also helps in speeding up the calculations in an algorithm.



Normalization or Standardization

- Feature Scaling means scaling features to the same scale.
- Normalization scales features between 0 and 1, retaining their proportional range to each other.

Normalization

$$x' = \frac{x - \min(x)}{\max(x) - \min(x)}$$

new value original value

- Standardization scales features to have a mean (μ) of 0 and standard deviation (σ) of 1.

Standardization

$$x' = \frac{x - \mu}{\sigma}$$

new value original value

mean standard deviation



- StandardScaler performs the task of Standardization. Usually a dataset contains variables that are different in scale. For e.g. an Employee dataset will contain AGE column with values on scale 20-70 and SALARY column with values on scale 10000-80000.
- As these two columns are different in scale, they are Standardized to have common scale while building machine learning model.
- **Scaling is done for numerical values only. Categorical values are not scaled.**

Example:

```
from sklearn.preprocessing import StandardScaler  
  
scaler = StandardScaler()  
  
#scaling 'Age' and 'Salary' Column only  
  
X[['Age','Salary']] = scaler.fit_transform(X[['Age','Salary']])
```

MinMaxScaler



- Transform features by scaling each feature to a given range.
- This estimator scales and translates each feature individually such that it is in the given range on the training set, e.g. between zero and one.

Example:

```
from sklearn.preprocessing import MinMaxScaler  
  
scalerX = MinMaxScaler(feature_range=(0, 1))  
  
X[['Age','Salary']] = scalerX.fit_transform(X[['Age','Salary']])  
  
X
```

Step 7: Splitting the Dataset into Training set and Test Set



- One important aspect of all machine learning models is to determine their accuracy. Now, in order to determine their accuracy, one can train the model using the given dataset and then predict the response values for the same dataset using that model and hence, find the accuracy of the model.
- A better option is to split our data into two parts: first one for training our machine learning model, and second one for testing our model.
- Train the model on the training set.
- Test the model on the testing set, and evaluate how well our model did.



- **train_test_split**: splits the data into two sets: train and test.
- It returns four datasets: X_train, X_test, y_train, y_test.

Parameters:

- **test_size**: This parameter decides the size of the data that has to be split as the test dataset. This is given as a fraction. For example, if you pass 0.8 as the value, the dataset will be split 80% as the test dataset.
- **random_state**: Here you pass an integer, which will act as the seed for the random number generator during the split.



- Example:
- Now X and y is ready. Spilt the data in two parts: train data and test data as:

```
from sklearn.model_selection import train_test_split
```

```
X_train,X_test,y_train,y_test =train_test_split(X,y,test_size = 0.20, random_state=42)
```

- The 80% of data will be assigned as training data and remaining 20% of data will be assigned as testing data.

#Data Preprocessing

```
import pandas as pd
```

#Step 1: Reading the file

```
df=pd.read_csv(r'D:\BigData_All_content\csv\ Data_for_preprocessing.csv', na_values='#')  
df
```

	Sr. No.	Country	Age	Salary	Purchased
0	1	France	44.0	72000.0	No
1	2	Spain	27.0	48000.0	Yes
2	3	Germany	30.0	54000.0	No
3	4	Spain	NaN	NaN	No
4	5	NaN	40.0	NaN	Yes
5	6	France	35.0	58000.0	Yes
6	7	Spain	78.0	52000.0	No
7	8	France	NaN	NaN	Yes
8	9	NaN	50.0	83000.0	No
9	10	France	37.0	NaN	Yes

#describe(): To view the description of the data

```
df.describe()
```

#By default, it will show the description of numerical features/columns only

	Sr. No.	Age	Salary
count	10.00000	8.000000	6.000000
mean	5.50000	42.625000	61166.666667
std	3.02765	16.070714	13511.723305
min	1.00000	27.000000	48000.000000
25%	3.25000	33.750000	52500.000000
50%	5.50000	38.500000	56000.000000
75%	7.75000	45.500000	68500.000000
max	10.00000	78.000000	83000.000000

#To view the description of categorical data, use include=object

```
df.describe(include='object')
```

	Country	Purchased
count	8	10
unique	3	2
top	France	No
freq	4	5

#To view the description of categorical and numerical features, use include='all'

```
df.describe(include='all')
```

	Sr. No.	Country	Age	Salary	Purchased
count	10.00000	8	8.000000	6.000000	10
unique	NaN	3	NaN	NaN	2
top	NaN	France	NaN	NaN	No
freq	NaN	4	NaN	NaN	5
mean	5.50000	NaN	42.625000	61166.666667	NaN
std	3.02765	NaN	16.070714	13511.723305	NaN
min	1.00000	NaN	27.000000	48000.000000	NaN
25%	3.25000	NaN	33.750000	52500.000000	NaN
50%	5.50000	NaN	38.500000	56000.000000	NaN
75%	7.75000	NaN	45.500000	68500.000000	NaN
max	10.00000	NaN	78.000000	83000.000000	NaN

#To view the columns of the dataframe/data

```
df.columns
```

```
Index(['Sr. No.', 'Country', 'Age', 'Salary', 'Purchased'],  
      dtype='object')
```

#Remove unnecessary columns/features from the data

#Ex: Removing 'Sr No' column

```
del df['Sr. No.']

df
```

	Country	Age	Salary	Purchased
0	France	44.0	72000.0	No
1	Spain	27.0	48000.0	Yes
2	Germany	30.0	54000.0	No
3	Spain	NaN	NaN	No
4	NaN	40.0	NaN	Yes
5	France	35.0	58000.0	Yes
6	Spain	78.0	52000.0	No
7	France	NaN	NaN	Yes
8	NaN	50.0	83000.0	No
9	France	37.0	NaN	Yes

#Step 2: Identify independent/input features/columns/variables

#In this data, Independent/input features are: 'Country', 'Age' and 'Salary'

#Dropping the 'Purchased' feature/column from the dataset

```
X=df.drop(['Purchased'], axis=1)
X
```

```
   Country    Age    Salary
0   France  44.0  72000.0
1   Spain   27.0  48000.0
2  Germany  30.0  54000.0
3   Spain     NaN      NaN
4      NaN  40.0      NaN
5   France  35.0  58000.0
6   Spain   78.0  52000.0
7  France     NaN      NaN
8      NaN  50.0  83000.0
9   France  37.0      NaN
```

#Identify dependent/outcome feature/variable/column

```
y=df['Purchased']
y
```

```
0    No
1   Yes
2    No
3    No
4   Yes
5   Yes
6    No
7   Yes
8    No
9   Yes
Name: Purchased, dtype: object
```

X.Salary

```
0    72000.0
1    48000.0
2    54000.0
3      NaN
4      NaN
5    58000.0
6    52000.0
7      NaN
8    83000.0
9      NaN
Name: Salary, dtype: float64
```

#Identify missing values in X

#isnull(): used to identify the missing values in a dataframe

#EX: Identifying missing data in X

```
#This code is very useful to identify missing values in each column of  
the X
```

```
X.isnull().sum()
```

```
Country      2  
Age         2  
Salary      4  
dtype: int64
```

```
#It is mandatory in ML, there is no missing values in X  
(Independent/input feature/column)
```

```
#Filling the missing values
```

```
#To check datatype of each feature/column of input/independent variable  
X  
X.dtypes
```

```
Country      object  
Age        float64  
Salary      float64  
dtype: object
```

```
#Filling missing data in Numerical column/features
```

```
#Ex: Fill missing values in 'Age' column/feature
```

```
#1. Obtain mean value of 'Age'
```

```
import numpy as np
```

```
mean_age=np.mean(X.Age)  
print('Mean of Age Feature:', mean_age)
```

```
#Filling misssing values in 'Age' with this mean value
```

```
#fillna(): used to fill missing values in a feature/column with  
specific value
```

```
X.Age.fillna(mean_age, inplace=True)  
X.Age
```

```
Mean of Age Feature: 42.625
```

0	44.000
1	27.000

```
2    30.000
3    42.625
4    40.000
5    35.000
6    78.000
7    42.625
8    50.000
9    37.000
Name: Age, dtype: float64
```

```
X.isnull().sum()
```

```
Country    2
Age        0
Salary     4
dtype: int64
```

#Filling missing values in 'Salary' column/feature

```
X.Salary.fillna(np.mean(X.Salary), inplace=True)
X.Salary
```

```
0    72000.000000
1    48000.000000
2    54000.000000
3    61166.666667
4    61166.666667
5    58000.000000
6    52000.000000
7    61166.666667
8    83000.000000
9    61166.666667
Name: Salary, dtype: float64
```

```
X.isnull().sum()
```

```
Country    2
Age        0
Salary     0
dtype: int64
```

#Filling missing data in 'Categorical' feature/column

#Replace the missing values with highest frequency data

#1. Count frequency of each class/category in categorical feature/column

```
#use value_counts()
```

```
X.Country.value_counts()
```

```
France      4
Spain       3
Germany    1
Name: Country, dtype: int64

#Replace the missing values in 'Country' with data that has highest frequency i.e. 'France'

X.Country.fillna('France', inplace=True)
X.Country

0    France
1    Spain
2  Germany
3    Spain
4    France
5    France
6    Spain
7    France
8    France
9    France
Name: Country, dtype: object

X.isnull().sum()

Country      0
Age          0
Salary       0
dtype: int64

#ML algo requires the independent/input feature/columns to be numeric only

#Transforming categorical feature/column in to numerical one

#Encoding categorical feature into numerical one

#1. Label Encoding
#2. One-hot Encoding

X.Country.value_counts()

France      6
Spain       3
Germany    1
Name: Country, dtype: int64

.....
Working of Label Encoding

Step 1 and step 2 is performed by fit()
```

Step 1: Identify the unique categories

1. France
2. Spain
3. Germany

Step 2: sort the unique categories in ascending order and assign a unique number to each category starting with 0

France	0
Germany	1
Spain	2

Step 3 is performed by transform()

Step 3: Encode the data of categorical feature with these numbers

0	France	0
1	Spain	2
2	Germany	1
3	Spain	2
4	France	0
5	France	0
6	Spain	2
7	France	0
8	France	0
9	France	0
...		

```
from sklearn.preprocessing import LabelEncoder  
  
lb=LabelEncoder()  
  
X.Country=lb.fit_transform(X.Country)  
X.Country  
  
0    0  
1    2  
2    1  
3    2  
4    0  
5    0  
6    2  
7    0  
8    0  
9    0  
Name: Country, dtype: int32
```

```
"""
0    France
1    Spain
2  Germany
3    Spain
4    France
5    France
6    Spain
7    France
8    France
9    France
```

#Working of One-hot encoding

1. Identify unique categories and sort them in ascending order

```
France
Germany
Spain
```

2. It will create one additional column for each category with syntax:
colname_categoryname

		Country_France	Country_Germany	Country_Spain
0	France	1	0	0
1	Spain	0	0	1
2	Germany	0	1	0
3	Spain	0	0	1
4	France	1	0	0
5	France	1	0	0
6	Spain	0	0	1
7	France	1	0	0
8	France	1	0	0
9	France	1	0	0

```
"""
```

```
X1=df.drop('Purchased', axis=1)
X1
```

	Country	Age	Salary
0	France	44.0	72000.0
1	Spain	27.0	48000.0
2	Germany	30.0	54000.0
3	Spain	NaN	NaN
4	NaN	40.0	NaN
5	France	35.0	58000.0
6	Spain	78.0	52000.0
7	France	NaN	NaN

```
8      NaN  50.0  83000.0
9  France  37.0       NaN
```

```
X1.Salary.fillna(np.mean(X1.Salary), inplace=True)
X1.Age.fillna(np.mean(X1.Age), inplace=True)
X1.Country.fillna('France', inplace=True)
X1.isnull().sum()
```

```
Country      0
Age         0
Salary      0
dtype: int64
```

#One-hot Encoding

```
#get_dummies()
```

```
X1=pd.get_dummies(X1)
X1
```

```
          Age      Salary  Country_France  Country_Germany
Country_Spain
0   44.000  72000.000000           1             0
0
1   27.000  48000.000000           0             0
1
2   30.000  54000.000000           0             1
0
3   42.625  61166.666667           0             0
1
4   40.000  61166.666667           1             0
0
5   35.000  58000.000000           1             0
0
6   78.000  52000.000000           0             0
1
7   42.625  61166.666667           1             0
0
8   50.000  83000.000000           1             0
0
9   37.000  61166.666667           1             0
0
```

#Feature Scaling of numerical columns

```
#Standard scaling= original_value -
mean_value_of_feature/standard_daviation_of_feature
```

```
first_age=X1.Age[0]
```

```

#These steps are performed by fit()

#obtain the mean of Age

mean_age=np.mean(X1.Age)

#obtain the standard deviation of Age
std_age=np.std(X1.Age)

#This step is performed by transform() for all data of 'Age'

new_first_age=(first_age-mean_age)/std_age

print('Original First value of Age:', first_age)
print('First value of Age after Standard Scaling:', new_first_age)

Original First value of age: 44.0
First value of age after Standard Scaling: 0.10226299281040578

#Scaling 'Age' using in-built function

from sklearn.preprocessing import StandardScaler

sc=StandardScaler()

X1[['Age']] = sc.fit_transform(X1[['Age']])

X1

```

	Age	Salary	Country_France	Country_Germany
Country_Spain				
0	0.102263	72000.000000	1	0
1	-1.162079	48000.000000	0	0
2	-0.938960	54000.000000	0	1
3	0.000000	61166.666667	0	0
4	-0.195229	61166.666667	1	0
5	-0.567095	58000.000000	1	0
6	2.630948	52000.000000	0	0
7	0.000000	61166.666667	1	0
8	0.548502	83000.000000	1	0

```
0  
9 -0.418349  61166.666667  
0
```

#MinMaxScaler

```
#new_value=(old_value - min_value(x))/(max(x)-min(x))
```

X.Age

```
0    44.000  
1    27.000  
2    30.000  
3    42.625  
4    40.000  
5    35.000  
6    78.000  
7    42.625  
8    50.000  
9    37.000  
Name: Age, dtype: float64
```

#Scaling the Age feature of X using MinMaxScaler

```
first_age=X.Age[0]  
  
min_age=min(X.Age)  
max_age=max(X.Age)  
  
new_age=(first_age-min_age)/(max_age-min_age)  
  
print('First value of Age before scaling:',first_age)  
print('First value of Age after scaling using MinMaxScaler:', new_age)
```

```
First value of Age before scaling: 44.0  
First value of Age after scaling using MinMaxScaler:  
0.3333333333333333
```

```
from sklearn.preprocessing import MinMaxScaler
```

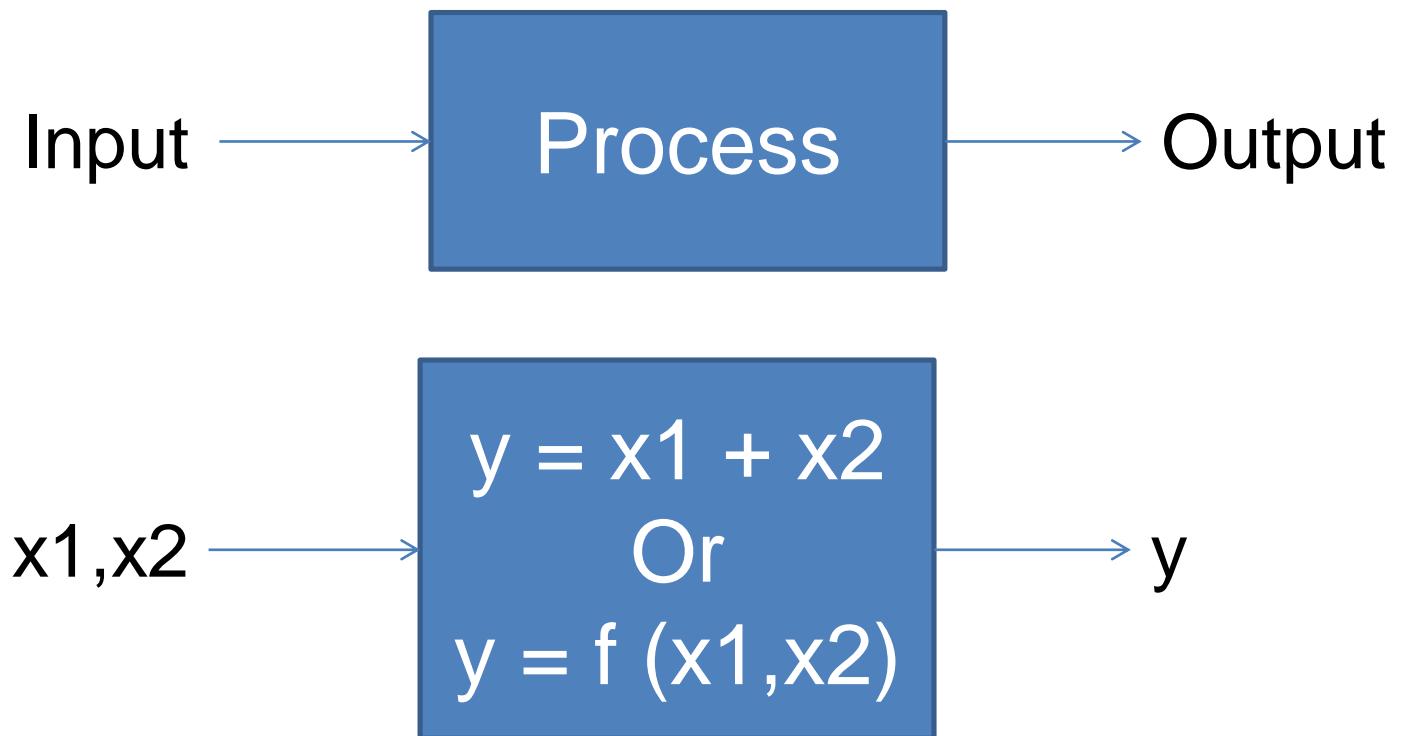
```
mxscale=MinMaxScaler(feature_range=(0,1))  
X[['Age']] = mxscale.fit_transform(X[['Age']])  
X.Age
```

```
0    0.333333  
1    0.000000  
2    0.058824  
3    0.306373  
4    0.254902  
5    0.156863
```

```
6    1.000000
7    0.306373
8    0.450980
9    0.196078
```

```
Name: Age, dtype: float64
```

Traditional Programming



Machine Learning

x1 (Input)	y (Output)
2	4
3	6
4	8
5	10
6	12
7	14
8	16

$$y = 2 \text{ multiplied by } x_1$$

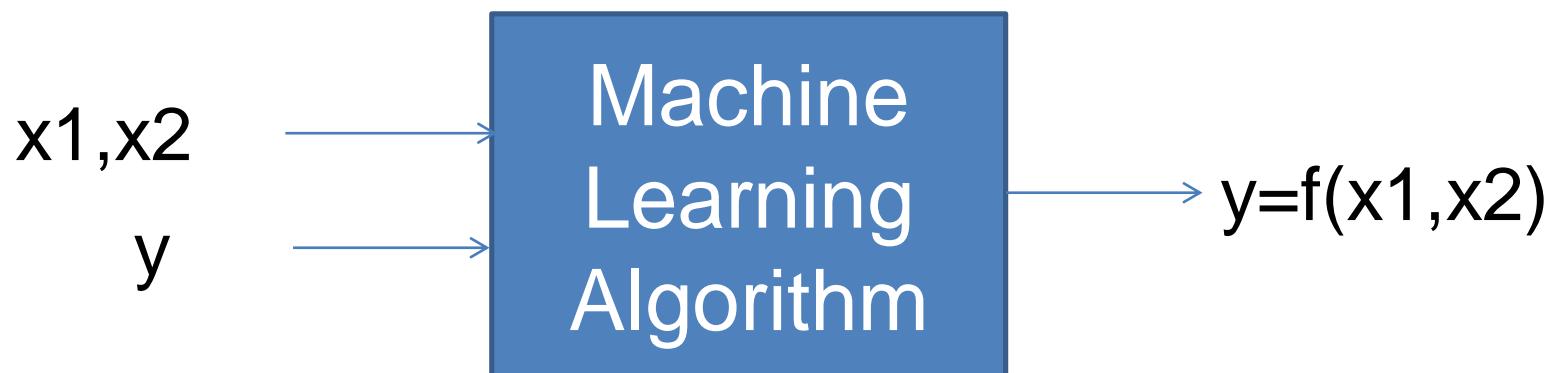
Machine Learning

x1 (Input)	x2(Input)	y (Output)
5	12	17
11	-5	6
2	6	8
14	-4	10
8	4	12
4	10	14
7	9	16

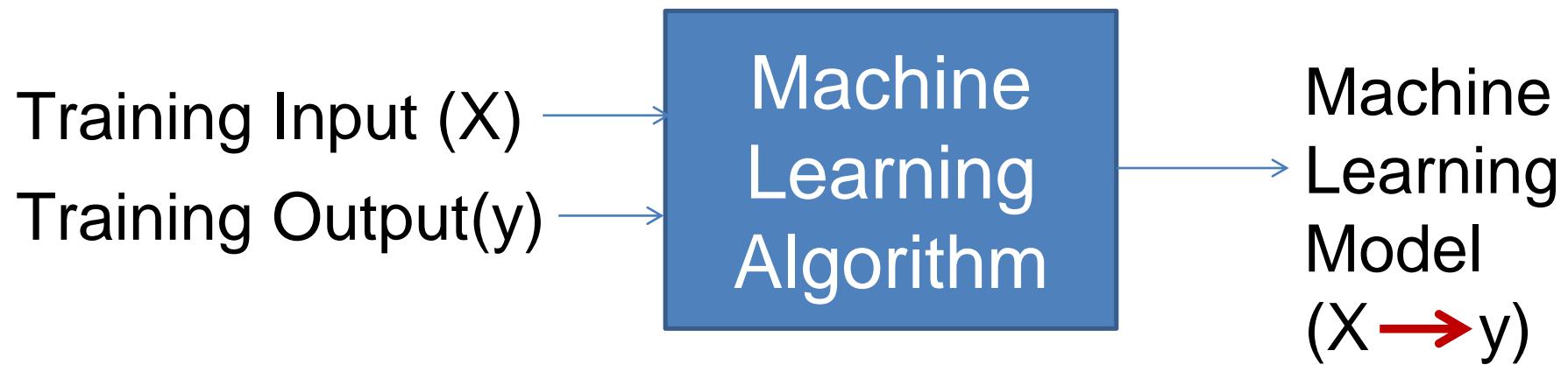
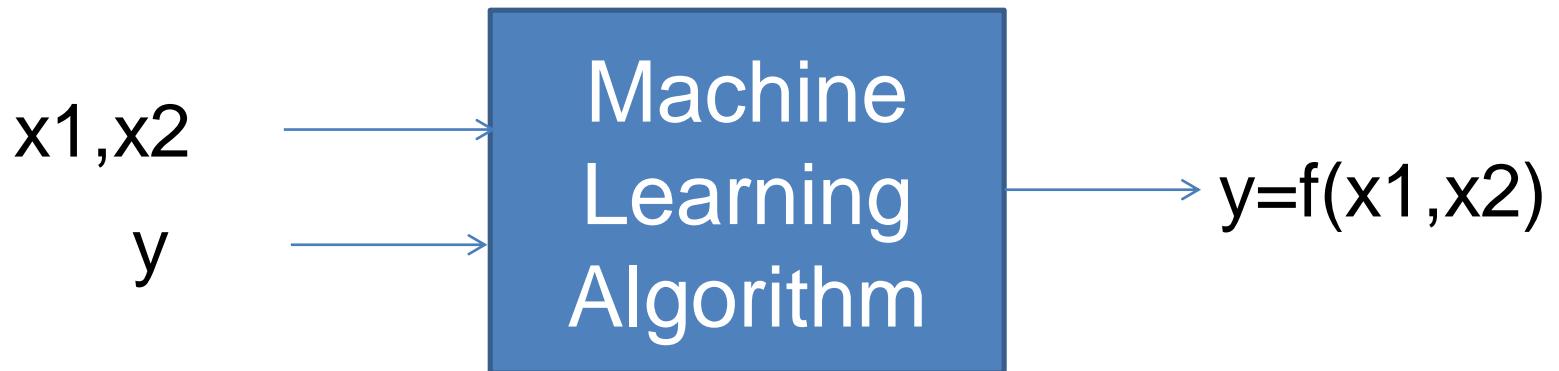
$$y = x_1 + x_2 \text{ (Model)}$$

$$x_1=17, x_2=10 \rightarrow y=27$$

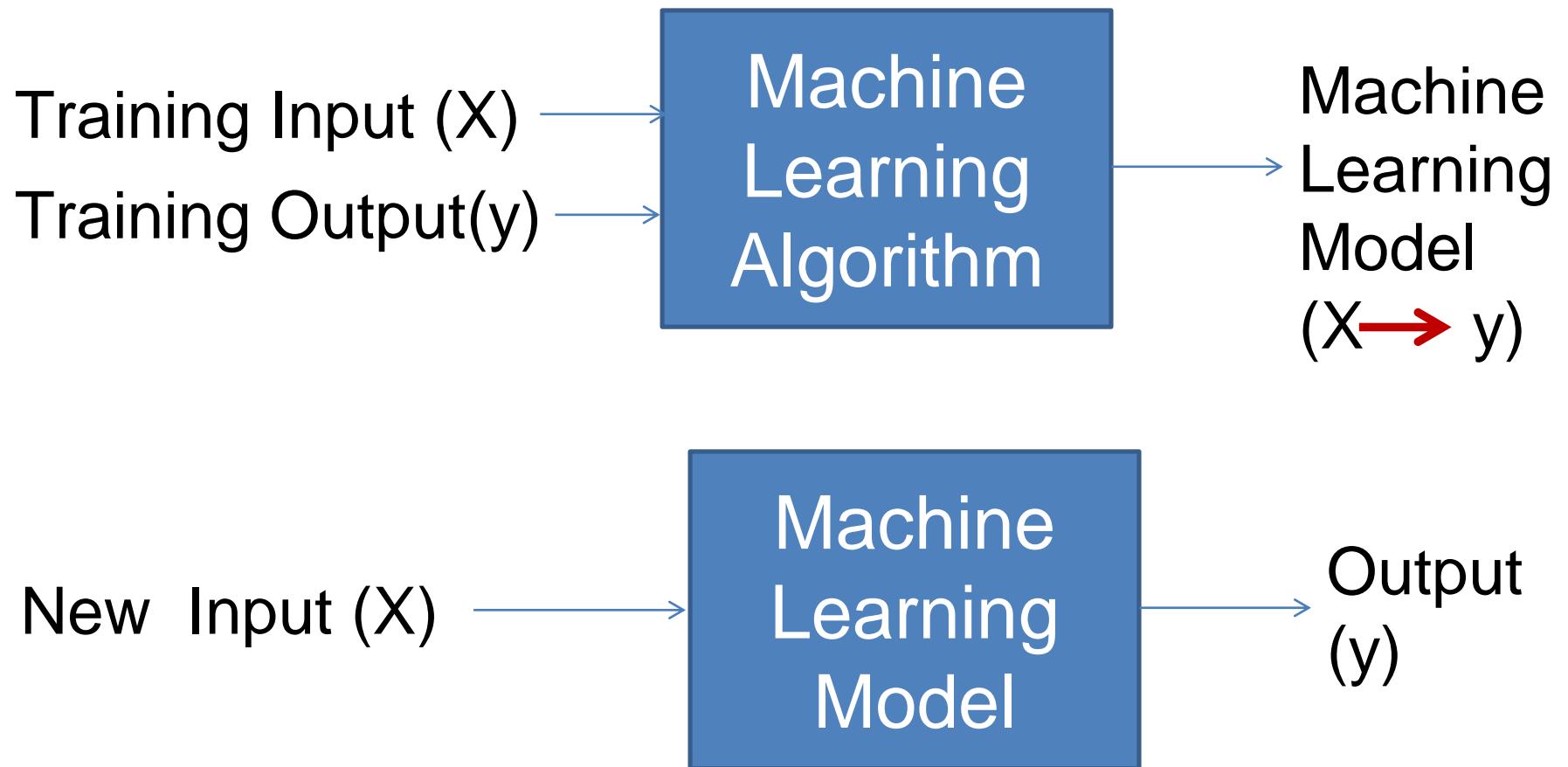
Machine Learning



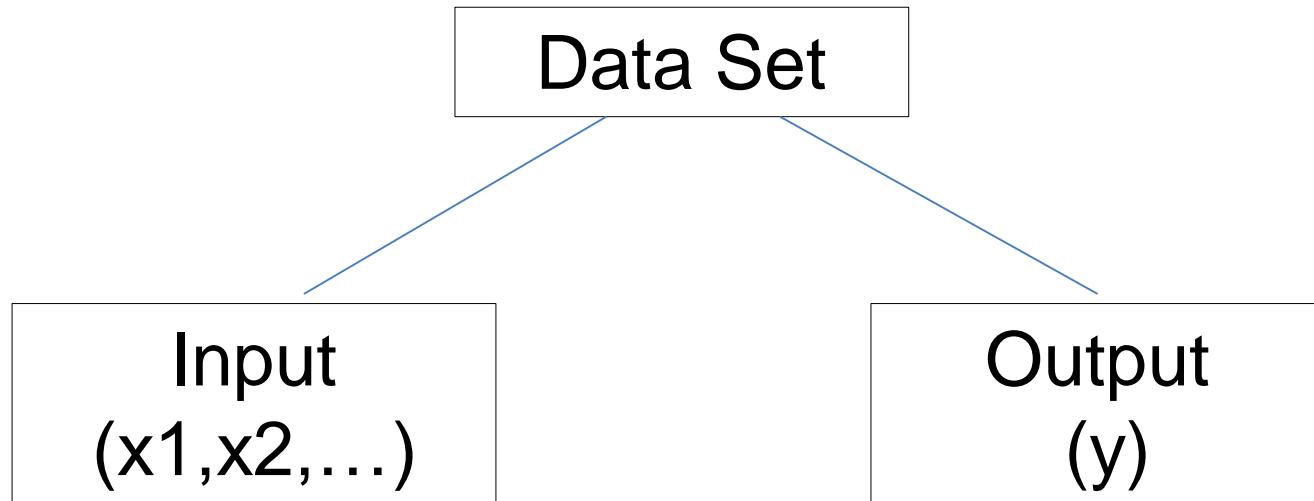
What kind of problem Machine Learning can solve:



What kind of problem Machine Learning can solve:



What is requirement of Machine Learning Problem:



From where to get Data?

UCI Repository

<https://archive.ics.uci.edu>

Kaggle web site:

<https://www.kaggle.com/datasets>

United Nations

<http://data.un.org/>

India

<https://data.gov.in/>

What kind of problem Machine Learning can solve:

S. No.	Area	bedrooms	bathrooms	stories	parking	price
1	585	3	1	2	no	4200000
2	400	2	1	1	no	3850000
3	306	3	1	1	no	4950000
4	665	3	1	2	yes	6050000
5	636	2	1	1	no	6100000
6	416	3	1	1	yes	6600000
7	388	3	2	2	no	6600000
8	416	3	1	3	no	6900000
9	480	3	1	1	yes	8380000
10	550	3	2	4	yes	8850000
11	720	3	2	1	no	9000000

This is called Regression Problem



- [Machine Learning \(Regression\)](#)
- [Kind of Problems Machine Learning \(Regression\)](#)
- [Admission Prediction Problem](#)
- [Dataset](#)
- [Features in the data set:](#)
- [Problem Statement](#)
- [Reading Data in Python](#)
- [Data Pre-processing](#)
- [Data Normalization – Scaling](#)
- [Split Data: Train and Test](#)
- [Applying Machine Learning Algorithm](#)
- [Random Forest Regressor](#)
- [Decision Tree Regressor](#)
- [Deployment of Model](#)
- [Error](#)
- [Absolute Mean Error & Root Mean Square Error](#)
- [References](#)

Machine Learning (Regression)



- Regression analysis is a form of predictive modelling technique which investigates the relationship between a dependent (target) and independent variable (s) (predictor).
- This technique is used for forecasting, time series modelling and finding the causal effect relationship between the variables. For example, relationship between rash driving and number of road accidents by a driver is best studied through regression.
- Calculates approximation of a continuous variable.

Kind of Machine Learning (Regression) Problems



- Estimated household income for loan processing or credit card limit etc.
- Estimated Crop production in an area.
- Estimated traffic at a place.
- Estimated future requirement for saving/investment.
- Admission Prediction Problem
- Sales Prediction which indicates that the growth in sales.
- Salary Prediction of an employee based on his qualification and experience.

Admission Prediction Problem



Serial No.	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research	Chance of Admit
1	337	118	4	4.5	4.5	9.65	1	0.92
2	324	107	4	4	4.5	8.87	1	0.76
3	316	104	3	3	3.5	8	1	0.72
4	322	110	3	3.5	2.5	8.67	1	0.8
5	314	103	2	2	3	8.21	0	0.65
6	330	115	5	4.5	3	9.34	1	0.9
7	321	109	3	3	4	8.2	1	0.75
8	308	101	2	3	4	7.9	0	0.68
9	302	102	1	2	1.5	8	0	0.5
10	323	108	3	3.5	3	8.6	0	0.45

Dataset



- This dataset is created for prediction of graduate admissions and the dataset link is below:

<https://www.kaggle.com/mohansacharya/graduate-admissions>

- The objective is to predict the chance of admission of a student based on their scores such as SOP, CGPA, TOEFL, GRE etc.

Features in the data set:



- GRE Scores (290 to 340)
- TOEFL Scores (92 to 120)
- University Rating (1 to 5)
- Statement of Purpose (1 to 5)
- Letter of Recommendation Strength (1 to 5)
- Undergraduate CGPA (6.8 to 9.92)
- Research Experience (0 or 1)
- Chance of Admit (0.34 to 0.97)

Problem Statement



- 400 applicants have been surveyed as potential students for UCLA. The university weighs certain aspects of a student's education to determine their acceptance.
- The objective is to explore the data and determine the most important factors that contribute to a student's chance of admission.
- Select the most accurate model to predict the probability of admission.

Reading Data in Python



- **Reading the 'Admission_Predict.csv' file in to pandas data frame for processing**

```
import pandas as pd  
  
df = pd.read_csv("Admission_Predict.csv",sep = ",")
```



- df.describe():

Serial No.	GRE Score	TOEFL Score	University Rating	SOP
count	400.000	400.000	400.000	400.0000
mean	200.500000	316.807500	107.410000	3.087500
std	115.614301	11.473646	6.069514	1.143728
min	1.000000	290.000000	92.000000	1.000000
25%	100.750000	308.000000	103.000000	2.000000
50%	200.500000	317.000000	107.000000	3.000000
75%	300.250000	325.000000	112.000000	4.000000
max	400.000000	340.000000	120.000000	5.000000

Data Pre-processing



- Dropping ‘Serial No.’ column from dataframe as it is not useful for Machine Learning.

```
df.drop(["Serial No."],axis=1,inplace = True)
```

- Renaming ‘Chance_of_Admit ‘ to ‘Chance_of_Admit’ as it has space after word ‘Admit ’.

```
df=df.rename(columns = {'Chance of Admit ':'Chance of Admit'})
```

Data Pre-processing



- **Extracting independent variables from the dataset:**

```
x = df.drop(["Chance of Admit"],axis=1)
```

- **Extract dependent variable from the dataset:**

```
y = df["Chance of Admit"]
```

- **Checking Missing values in x:**

```
x.isnull().sum()
```

Data Normalization - Scaling



- **Scaling the numerical field columns between 0 and 1**

```
from sklearn.preprocessing import MinMaxScaler  
  
scalerX = MinMaxScaler(feature_range=(0, 1))  
  
x[x.columns] = scalerX.fit_transform(x [x.columns])
```

Split Data: Train and Test



□ Splitting the data in to train and test set

```
from sklearn.model_selection import train_test_split
```

```
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size = 0.20)
```

Applying Machine Learning Algorithm



□ Applying Linear Regression Algorithm

```
from sklearn.linear_model import LinearRegression  
  
model = LinearRegression()  
  
#Fitting the x_train and y_train on the model  
  
model.fit(x_train,y_train)
```

Applying Machine Learning Algorithm



- **Viewing the Score of the applied model**

```
score=model.score(x_test, y_test)
```

```
print(score)
```

- **Predicting the x_test on the model**

```
y_predict = model.predict(x_test)
```

```
print (y_predict[0:5])
```

Random Forest Regressor



□ Applying Random Forest Regressor

```
from sklearn.ensemble import RandomForestRegressor  
rfr = RandomForestRegressor(n_estimators = 100, random_state = 42)  
rfr.fit(x_train,y_train)
```

```
y_predict_rfr = rfr.predict(x_test)
```

```
score_rfr=rfr.score(x_test, y_test)  
print(score_rfr)
```

Decision Tree Regressor



- Applying DecisionTreeRegressor on x_train and y_train

```
from sklearn.tree import DecisionTreeRegressor  
  
dtr = DecisionTreeRegressor(random_state = 42)  
  
dtr.fit(x_train,y_train)  
  
y_predict_dtr = dtr.predict(x_test)  
  
score_dtr=dtr.score(x_test, y_test)  
  
print(score_dtr)
```



- [Performance Evaluation of Regression Problem](#)
- [Mean Absolute Error](#)
- [Mean Squared Error](#)
- [Root Mean Squared Error \(RMSE\)](#)
- [Deployment of Model](#)

Performance Evaluation of Regression Problem



The performance of regression problem can be measured by:

- Mean Absolute Error (MAE)**
- Mean Squared Error (MSE)**
- Root Mean Squared Error (RMSE)**

Mean Absolute Error



- The standard error of the regression problem is also known as the standard error of the estimate.
- It represents the average distance that the observed values fall from the regression line.
- An error basically is the absolute difference between the actual or true values and the values that are predicted. Absolute difference means that if the result has a negative sign, it is ignored.
- Calculating the mean absolute error between Actual output and Predicted output

```
from sklearn.metrics import mean_absolute_error
#Mean Absolute Error of LinearRegression Model
y_predict = model.predict(X_test)
lre1=mean_absolute_error(y_test,y_predict)
print("Mean Absolute Error:",lre1)
```

Mean Squared Error



- Mean Squared Error (MSE) is most popular method to evaluate regression problem.
- MSE is calculated by taking the average of the square of the difference between the original and predicted values of the data.

$$MSE = \frac{1}{N} \sum_{i=1}^n (\text{actual values} - \text{predicted values})^2$$

Here N is the total number of observations/rows in the dataset. The sigma symbol denotes that the difference between actual and predicted values taken on every i value ranging from **1 to n**.

Mean Squared Error



□ Calculating Mean Squared Error for Linear Regression Model

```
from sklearn.metrics import mean_squared_error  
  
#Mean Squared Error for Linear Regression Model  
  
lrp2=mean_squared_error(y_test,y_predict)  
  
print("mean_squared_error:",lrp2)
```

Root Mean Squared Error (RMSE)



- RMSE is the standard deviation of the errors which occur when a prediction is made on a dataset.
- In RMSE, the errors are squared before they are averaged.
- This basically implies that RMSE assigns a higher weight to larger errors.
- This indicates that RMSE is much more useful when large errors are present and they drastically affect the model's performance.



- Calculating Root Mean Squared Error for LinearRegression Model

```
import numpy as np
```

```
#Root Mean Squared Error for Linear Regression Model
```

```
lrp3=np.sqrt(lrp2)
```

```
print("Square root of mean_squared_error:",lrp3)
```



□ **Calculating Mean Absolute Error and Mean Squared Error for Decision Tree Regression Model**

```
y_predict1 = dtr.predict(X_test)

dtrp1=mean_absolute_error(y_test,y_predict1)

dtrp2=mean_squared_error(y_test,y_predict1)

drtp3=np.sqrt(p2)

print("Mean Absolute Error:",dtrp1)

print("mean_squared_error:",dtrp2)

print("Square root of mean_squared_error:",drtp3)
```



□ **Calculating Mean Absolute Error and Mean Squared Error for Random Forest Regressor Model**

```
y_predict2 = rfr.predict(X_test)  
rfrp1=mean_absolute_error(y_test,y_predict2)  
rfrp2=mean_squared_error(y_test,y_predict2)  
rfrp3=np.sqrt(rfrp2)  
print("Mean Absolute Error:",rfrp1)  
print("mean_squared_error:",rfrp2)  
print("Square root of mean_squared_error:",rfrp3)
```

Deployment of Model



```
gre=int(input("What is your GRE Score (between 290 to 340):"))

toefl=int(input("What is your TOEFL Score (between 90 to 120):"))

univ=int(input("What is your University Rating ( 1 to 5 ):"))

sop=int(input("Rate your Statement of Purpose ( 1 to 5):"))

lor=int(input("What is strength of your Letter of Recommendation ( 1 to 5 ) :"))

cgpa=int(input("What is your CGPA ( 6 to 10):"))

research=int(input("Do You have Research Experience (Enter 0 for No and 1 for Yes:"))

data=[[gre, toefl, univ, sop, lor, cgpa, research]]
```

Deployment of Model



```
newdf=pd.DataFrame(data, columns=['GRE Score', 'TOEFL Score', 'University Rating', 'SOP', 'LOR',  
'CGPA', 'Research'])
```

```
newdf
```

```
#Scaling the dataframe
```

```
newdf[newdf.columns]=mxscaler.transform(newdf[newdf.columns])
```

```
Newdf
```

```
new_predict=lr.predict(newdf)
```

```
print('Your chance of Admit:', new_predict)
```

```
#Regression Problem
```

```
#Admission Prediction problem
```

```
import pandas as pd
```

```
df=pd.read_csv(r'D:\BigData_All_content\csv\Admission_Predict.csv')
```

```
#head(): it will return first 5 records of the dataframe  
df.head()
```

CGPA \ Serial No.	GRE Score	TOEFL Score	University Rating	SOP	LOR
9.65	1	337	118	4	4.5
8.87	2	324	107	4	4.0
8.00	3	316	104	3	3.0
8.67	4	322	110	3	3.5
8.21	5	314	103	2	2.0

	Research	Chance of Admit
0	1	0.92
1	1	0.76
2	1	0.72
3	1	0.80
4	0	0.65

```
df.head(7)
```

CGPA \ Serial No.	GRE Score	TOEFL Score	University Rating	SOP	LOR
9.65	1	337	118	4	4.5
8.87	2	324	107	4	4.0
8.00	3	316	104	3	3.0
8.67	4	322	110	3	3.5
8.21	5	314	103	2	2.0
9.34	6	330	115	5	4.5
8.20	7	321	109	3	3.0

```

Research Chance of Admit
0      1      0.92
1      1      0.76
2      1      0.72
3      1      0.80
4      0      0.65
5      1      0.90
6      1      0.75

df.describe()

      Serial No.    GRE Score    TOEFL Score  University Rating
SOP \
count  400.000000  400.000000  400.000000      400.000000
mean   200.500000  316.807500  107.410000      3.087500
3.400000
std    115.614301  11.473646   6.069514      1.143728
1.006869
min    1.000000   290.000000  92.000000      1.000000
1.000000
25%   100.750000  308.000000  103.000000      2.000000
2.500000
50%   200.500000  317.000000  107.000000      3.000000
3.500000
75%   300.250000  325.000000  112.000000      4.000000
4.000000
max   400.000000  340.000000  120.000000      5.000000
5.000000

      LOR        CGPA    Research Chance of Admit
count  400.000000  400.000000  400.000000  400.000000
mean   3.452500   8.598925   0.547500   0.724350
std    0.898478   0.596317   0.498362   0.142609
min    1.000000   6.800000   0.000000   0.340000
25%   3.000000   8.170000   0.000000   0.640000
50%   3.500000   8.610000   1.000000   0.730000
75%   4.000000   9.062500   1.000000   0.830000
max   5.000000   9.920000   1.000000   0.970000

df.columns

Index(['Serial No.', 'GRE Score', 'TOEFL Score', 'University Rating',
       'SOP',
       'LOR', 'CGPA', 'Research', 'Chance of Admit'],
      dtype='object')

df.dtypes

Serial No.          int64
GRE Score          int64

```

```
T0EFL Score           int64
University Rating      int64
SOP                   float64
LOR                   float64
CGPA                  float64
Research               int64
Chance of Admit       float64
dtype: object
```

#Dropping the Serial No. feature/column from the dataframe

```
del df['Serial No.']
df.head()
```

```
GRE Score  T0EFL Score  University Rating  SOP  LOR  CGPA  Research
\0          337           118                 4    4.5  4.5  9.65     1
1          324           107                 4    4.0  4.5  8.87     1
2          316           104                 3    3.0  3.5  8.00     1
3          322           110                 3    3.5  2.5  8.67     1
4          314           103                 2    2.0  3.0  8.21     0
```

```
Chance of Admit
0            0.92
1            0.76
2            0.72
3            0.80
4            0.65
```

#Prepare independent/input features/variables

```
X=df.drop('Chance of Admit', axis=1)
X.head()
```

```
GRE Score  T0EFL Score  University Rating  SOP  LOR  CGPA  Research
0          337           118                 4    4.5  4.5  9.65     1
1          324           107                 4    4.0  4.5  8.87     1
2          316           104                 3    3.0  3.5  8.00     1
3          322           110                 3    3.5  2.5  8.67     1
4          314           103                 2    2.0  3.0  8.21     0
```

#Preparing outcome/dependent feature/variable

```
y=df['Chance of Admit']
```

```
y.head()  
0    0.92  
1    0.76  
2    0.72  
3    0.80  
4    0.65  
Name: Chance of Admit, dtype: float64
```

#Checking missing data

```
df.isnull().sum()
```

```
GRE Score      0  
TOEFL Score   0  
University Rating 0  
SOP            0  
LOR            0  
CGPA           0  
Research        0  
Chance of Admit 0  
dtype: int64
```

#Feature scaling of numerical features using MinMaxScaler

```
from sklearn.preprocessing import MinMaxScaler
```

```
mx=MinMaxScaler(feature_range=(0,1))
```

#Scaling all features/columns of X

```
X[X.columns]=mx.fit_transform(X[X.columns])  
X.head()
```

	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA
Research						
0	0.94	0.928571		0.75	0.875	0.875
1.0						0.913462
1	0.68	0.535714		0.75	0.750	0.875
1.0						0.663462
2	0.52	0.428571		0.50	0.500	0.625
1.0						0.384615
3	0.64	0.642857		0.50	0.625	0.375
1.0						0.599359
4	0.48	0.392857		0.25	0.250	0.500
0.0						0.451923

#Splitting the data into training set and testing set

```
from sklearn.model_selection import train_test_split
```

```

#test_size represents the size of testing data in percentage
#default value of test_size is 0.25

X_train, X_test, y_train, y_test=train_test_split(X, y,
test_size=0.20)

#In this case, 20% of data will be used for testing and 80% of data
will be used for training

#total data=400

#X_train= 80% data of X= 320 rows
#X_test= 20% data of X= 80 rows

#y_train= 80% data of y= 320 rows
#y_test= 20% data of y= 80 rows

print(X_train.shape)
print(X_test.shape)

print(y_train.shape)
print(y_test.shape)

(320, 7)
(80, 7)
(320,)
(80,)

X_train.head()

      GRE Score  TOEFL Score  University Rating    SOP    LOR     CGPA
\
332      0.36      0.500000        0.50    0.625    0.375   0.451923
143      1.00      1.000000        0.75    0.875    0.750   1.000000
237      0.78      0.785714        1.00    0.875    1.000   0.766026
272      0.08      0.107143        0.00    0.125    0.125   0.269231
100      0.64      0.535714        0.50    0.625    0.625   0.532051

      Research
332      1.0
143      1.0
237      1.0

```

```
272      0.0
100      1.0

y_train.head()

332    0.75
143    0.97
237    0.86
272    0.49
100    0.71
Name: Chance of Admit, dtype: float64

#Now, our data is ready to apply Machine Learning algorithms/model

#We are using LinearRegression algo/model to solve this regression problem

from sklearn.linear_model import LinearRegression

#Creating object of LinearRegression algo/model

lr=LinearRegression()

#Training the LinearRegression algo using X_train and y_train

lr.fit(X_train, y_train)

LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None,
normalize=False)

#Testing the LinearRegression algo/model

#To view the score of LinearRegression model using X_test and y_test

lr.score(X_test, y_test)

0.8114415881955475

#Here, score is ~81%

#It means that the LinearRegression algorithm predicts the value of y_test on X_test with 81% of #accuracy

#In score(), first predict() is called to predict the outcome of y_test on the basis of X_test

#Then it compares the predicted outcome with the actual outcome.
```

```

"""
Actual          Predicted
Outcome        Outcome
y_test         y_predict

0              0
1              0
0              0
1              1

```

We can say that actual outcome and predicted outcome is matched at 3 places.

So accuracy is 75%

```

'\nActual          Predicted\nOutcome        Outcome\ny_test         \t
y_predict\n\n0\t          0\n1\t          0\n0\t          0\n1\t
1\n\nWe can say that actual outcome and predicted outcome is matched
at 3 places.\nSo accuracy is 75%\n'

```

#We can view the predicted outcome on the basis of X_test by the LinearRegression

```
import numpy as np
```

```
y_pred=lr.predict(X_test)

y_pred1=np.around(y_pred, 2)
y_pred1[0:5]
```

```
array([0.79, 0.89, 0.89, 0.96, 0.91])
```

```
y_test[0:5]
```

```
245    0.81
142    0.92
286    0.92
25     0.94
259    0.90
Name: Chance of Admit, dtype: float64
```

#Now comparing the Actual Outcome and Predicted Outcome for X_test

```
df_pred=pd.DataFrame({'Actual Outcome':y_test, 'Predicted
Outcome':y_pred1 })
df_pred.head()
```

	Actual Outcome	Predicted Outcome
245	0.81	0.79
142	0.92	0.89
286	0.92	0.89

```
25          0.94          0.96
259         0.90          0.91
```

#Applying RandomForestRegressor

```
from sklearn.ensemble import RandomForestRegressor
rfr=RandomForestRegressor()
rfr.fit(X_train, y_train)
C:\Users\hp\Anaconda3\lib\site-packages\sklearn\ensemble\
forest.py:245: FutureWarning: The default value of n_estimators will
change from 10 in version 0.20 to 100 in 0.22.
"10 in version 0.20 to 100 in 0.22.", FutureWarning)
RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=None,
                     max_features='auto', max_leaf_nodes=None,
                     min_impurity_decrease=0.0,
                     min_impurity_split=None,
                     min_samples_leaf=1, min_samples_split=2,
                     min_weight_fraction_leaf=0.0, n_estimators=10,
                     n_jobs=None, oob_score=False, random_state=None,
                     verbose=0, warm_start=False)
```

#Applying DecisionTreeRegressor

```
from sklearn.tree import DecisionTreeRegressor
dtr=DecisionTreeRegressor()
dtr.fit(X_train, y_train)
DecisionTreeRegressor(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=False, random_state=None,
                     splitter='best')
```

```
print('Score of LinearRegression:', lr.score(X_test, y_test))
print('Score of RandomForestRegressor:', rfr.score(X_test, y_test))
print('Score of DecisionTreeRegressor:', dtr.score(X_test, y_test))
```

```
Score of LinearRegression: 0.8114415881955475
Score of RandomForestRegressor: 0.8254798196900341
Score of DecisionTreeRegressor: 0.705834514487534
```

#The score of RandomForestRegressor is highest i.e. ~82%

#So, on the basis of score, we can select RandomForestRegressor for implementation

#But, we are using the LinearRegression because in past, it has been found that it is suited best
#for regression problem

#Implementing the LinearRegression algo on new data

X.dtypes

```
GRE Score          float64
TOEFL Score        float64
University Rating   float64
SOP                 float64
LOR                 float64
CGPA                float64
Research             float64
dtype: object
```

df.describe()

	GRE Score	TOEFL Score	University Rating	SOP
LOR \				
count	400.000000	400.000000	400.000000	400.000000
mean	316.807500	107.410000	3.087500	3.400000
std	11.473646	6.069514	1.143728	1.006869
min	290.000000	92.000000	1.000000	1.000000
25%	308.000000	103.000000	2.000000	2.500000
50%	317.000000	107.000000	3.000000	3.500000
75%	325.000000	112.000000	4.000000	4.000000
max	340.000000	120.000000	5.000000	5.000000

	CGPA	Research	Chance of Admit
count	400.000000	400.000000	400.000000
mean	8.598925	0.547500	0.724350
std	0.596317	0.498362	0.142609
min	6.800000	0.000000	0.340000
25%	8.170000	0.000000	0.640000
50%	8.610000	1.000000	0.730000

```
75%      9.062500    1.000000      0.830000
max      9.920000    1.000000      0.970000
```

#Implementation

```
gre=float(input('Enter GRE Score between 290 to 340:'))
toefl=float(input('Enter TOEFL Score between 92 to 120='))
rating=float(input('Enter University Rating between 1 and 5='))
sop=float(input('Enter SOP between 1 and 5='))
lor=float(input('Enter LOR between 1 and 5='))
cgpa=float(input('Enter CGPA between 6.82 and 9.92='))
research=float(input('Enter Research 0-For NO and 1- for YES ='))
```

Enter GRE Score between 290 to 340:318
Enter TOEFL Score between 92 to 120=108
Enter University Rating between 1 and 5=5
Enter SOP between 1 and 5=4
Enter LOR between 1 and 5=4
Enter CGPA between 6.82 and 9.92=7.82
Enter Research 0-For NO and 1- for YES =1

```
type(X_train)
```

```
pandas.core.frame.DataFrame
```

```
X_train.columns
```

```
Index(['GRE Score', 'TOEFL Score', 'University Rating', 'SOP', 'LOR',
       'CGPA',
       'Research'],
      dtype='object')
```

*#Now, Prepare the new data as a dataframe because we have given the training data in fit() as dataframe
#Order and name of the columns in new dataframe must match with the training data*

#Transforming the new data as a dataframe

```
value=[[gre, toefl, rating, sop, lor, cgpa, research]]
```

```
newdf=pd.DataFrame(value, columns=['GRE Score', 'TOEFL Score',
                                    'University Rating', 'SOP', 'LOR',
                                    'CGPA',
                                    'Research'])
```

```
newdf
```

	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research
0	318.0	108.0	5.0	4.0	4.0	7.82	1.0

#So, new data is ready for machine learning

```
#Now, scale the new data using MinMaxScaler because we have scaled the training data
```

```
#Don't fit the new data in MinMaxScaler. Just transform the new data using object of MinMaxScaler
```

```
newdf[newdf.columns]=mx.transform(newdf[newdf.columns])  
newdf
```

	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA
Research						
0	0.56	0.571429		1.0	0.75	0.75
1.0						0.326923

```
#Now, your new data is ready for prediction
```

```
#Predicting the value of 'Chance of Admit' on this new data using LinearRegression algo/model
```

```
new_pr=lr.predict(newdf)
```

```
print('Your chance of admit is:', np.around(new_pr[0],2))
```

```
Your chance of admit is: 0.67
```

```

lr_predict=lr.predict(X_test)
lr_predict[0:5]

array([0.95376269, 0.91385409, 0.6797497 , 0.78426403, 0.90293742])

#Now, creating a dataframe on Actual 'Chance of Admit' and Predicted
'Chance of Admit'

df_pred=pd.DataFrame({'Actual Outcome':y_test, 'Predicted
Outcome':lr_predict})
df_pred.head()

```

	Actual Outcome	Predicted Outcome
384	0.96	0.953763
108	0.93	0.913854
112	0.62	0.679750
133	0.79	0.784264
397	0.91	0.902937

#Error: the difference between actual outcome and predicted outcome

#Calculating the error

```

df_pred['Error']=df_pred['Actual Outcome']-df_pred['Predicted
Outcome']
df_pred.head(10)

```

	Actual Outcome	Predicted Outcome	Error
384	0.96	0.953763	0.006237
108	0.93	0.913854	0.016146
112	0.62	0.679750	-0.059750
133	0.79	0.784264	0.005736
397	0.91	0.902937	0.007063
281	0.80	0.822589	-0.022589
151	0.94	0.904240	0.035760
158	0.61	0.612795	-0.002795
275	0.78	0.784053	-0.004053
80	0.50	0.653263	-0.153263

#Calculating the absolute value of df_pred['Error']

```

df_pred['Absolute Error']=abs(df_pred['Error'])
df_pred.head()

```

	Actual Outcome	Predicted Outcome	Error	Absolute Error
384	0.96	0.953763	0.006237	0.006237
108	0.93	0.913854	0.016146	0.016146
112	0.62	0.679750	-0.059750	0.059750
133	0.79	0.784264	0.005736	0.005736
397	0.91	0.902937	0.007063	0.007063

```

#Now, calculating the mean value of Absolute Error

import numpy as np

print('Mean Absolute Error of LinearRegression:', 
np.mean(df_pred['Absolute Error']))

Mean Absolute Error of LinearRegression: 0.04846612484273518

#Now, calculating the Mean Absolute Error of LinearRegression using
in-built function

from sklearn.metrics import mean_absolute_error

mea_lr=mean_absolute_error(y_test, lr_predict)
mea_lr

0.0484661248427352

#Calculating Mean Squared Error

#First calculate squared error

df_pred['Squared Error']=df_pred['Error']*df_pred['Error']

df_pred.head()

      Actual Outcome  Predicted Outcome    Error  Absolute Error \
384           0.96        0.953763  0.006237       0.006237
108           0.93        0.913854  0.016146       0.016146
112           0.62        0.679750 -0.059750       0.059750
133           0.79        0.784264  0.005736       0.005736
397           0.91        0.902937  0.007063       0.007063

      Squared Error
384      0.000039
108      0.000261
112      0.003570
133      0.000033
397      0.000050

#Then calculate mean value of Squared error

print('Mean Squared Error for LinearRegression:', 
np.mean(df_pred['Squared Error']))

Mean Squared Error for LinearRegression: 0.004161456493236137

#Calculating Mean Squared Error using in-built function

from sklearn.metrics import mean_squared_error

```

```
mse_lr=mean_squared_error(y_test, lr_predict)
mse_lr
0.004161456493236139

#Calculating Root Mean Squared Error(RMSE)

#It can be obtained by taking the root of Mean Squared Error

print('Root Mean Squared Error for LinearRegression:', 
np.sqrt(mse_lr))

Root Mean Squared Error for LinearRegression: 0.064509351982764

#Calculating Mean Squared Error for RandomForestRegressor

#Predicting the y_test on X_test using object of RandomForestRegressor
rfr_predict=rfr.predict(X_test)

mse_rfr=mean_squared_error(y_test, rfr_predict)
mse_rfr
0.005035199999999999

#Calculating Mean Squared Error for DecisionTreeRegressor

#Predicting the y_test on X_test using object of DecisionTreeRegressor
dtr_predict=dtr.predict(X_test)

mse_dtr=mean_squared_error(y_test, dtr_predict)
mse_dtr
0.01043249999999999

#Calculating Root Mean Squared Error

print('Root Mean Squared Error for LinearRegression:', 
np.sqrt(mse_lr))
print('Root Mean Squared Error for RandomForestRegressor:', 
np.sqrt(mse_rfr))
print('Root Mean Squared Error for DecisionTreeRegressor:', 
np.sqrt(mse_dtr))

Root Mean Squared Error for LinearRegression: 0.064509351982764
Root Mean Squared Error for RandomForestRegressor: 0.07095914317408293
Root Mean Squared Error for DecisionTreeRegressor: 0.10213961033800745
```



- [Loan Prediction Problem](#)
- [Data Understanding and Requirements](#)
- [Data Pre-processing](#)
 - Handling Null Values – Categorical Features
 - Handling Null Values – Numerical Features
 - Changing Categorical Values into Numerical Values
 - Train and Test Split
 - Applying Machine Learning Algorithm – Logistic
- [Regression](#)
- [Applying Machine Learning Algorithm – Support Vector Mach...](#)
- [Applying Machine Learning Algorithm – Decision Tree Class...](#)
- [Applying Machine Learning Algorithm – KNeighborsClassifier](#)
- [Applying Machine Learning Algorithm](#)

Loan Prediction Problem



- A Finance company deals in all home loans.
- They have presence across all urban, semi urban and rural areas.
- Customer first apply for home loan after that company validates the customer eligibility for loan.
- Company wants to automate the loan eligibility process (real time) based on customer detail provided while filling online application form.

Loan Prediction Problem



Variable	Description
Loan_ID	Unique Loan ID
Gender	Male/ Female
Married	Applicant married (Y/N)
Dependents	Number of dependents
Education	Applicant Education (Graduate/ Under Graduate)
Self_Employed	Self employed (Y/N)
ApplicantIncome	Applicant income
CoapplicantIncome	Co-applicant income
LoanAmount	Loan amount in thousands
Loan_Amount_Term	Term of loan in months
Credit_History	credit history meets guidelines
Property_Area	Urban/ Semi Urban/ Rural
Loan_Status	Loan approved (Y/N)

Data Understanding and Requirement Understanding



```
import pandas as pd  
df=pd.read_csv('train.csv')
```

```
df.head()
```

```
df.describe()
```

Data Understanding and Requirement Understanding



	Applicant Income	Coapplicant Income	Loan Amount	Loan_Amou nt_Term	Credit_Histo ry
count	500.000000	500.000000	482.000000	486.000000	459.000000
mean	5493.644000	1506.307840	144.020747	342.543210	0.843137
std	6515.668972	2134.432188	82.344919	63.834977	0.364068
min	150.000000	0.000000	17.000000	12.000000	0.000000
25%	2874.500000	0.000000	100.000000	360.000000	1.000000
50%	3854.000000	1125.500000	126.500000	360.000000	1.000000
75%	5764.000000	2253.250000	161.500000	360.000000	1.000000
max	81000.000000	20000.000000	700.000000	480.000000	1.000000

Data Pre-processing



- Preparing X and y

```
X=df.drop(['Loan_Status','Loan_ID'], axis=1)  
y=df['Loan_Status']
```

- Checking Missing Values

```
X.isnull().sum()
```

- Counting frequency in ‘Credit_History’

```
X['Credit_History'].value_counts()
```

```
X['Gender'].value_counts()
```

Handling Null Values – Categorical Features



```
X['Gender'].fillna("Male", inplace=True)
```

```
X.isnull().sum()
```

```
X['Married'].value_counts()
```

```
X['Married'].fillna("Yes", inplace=True)
```

```
X.isnull().sum()
```

```
X['Dependents'].value_counts()
```

```
X['Dependents'].fillna(0,inplace=True)
```

```
X['Self_Employed'].value_counts()
```

```
X['Self_Employed'].fillna('No',inplace=True)
```

Handling Null Values – Numerical Features



```
mean_loan=X['LoanAmount'].mean()  
X['LoanAmount'].fillna(mean_loan,inplace=True)  
X.isnull().sum()
```

```
X['Loan_Amount_Term'].fillna(X['Loan_Amount_Term'].mean(),inplace=True)
```

```
X['Credit_History'].fillna(X['Credit_History'].mean(),inplace=True)
```

```
X.isnull().sum()
```

Changing Categorical Values into Numerical Values



- One-hot Encoding

Gender
Male
Female
Male
Male
Male
Female
Male

Gender
1
0
1
1
1
0
1

Gender_0	Gender_1
0	1
1	0
0	1
0	1
0	1
1	0
0	1

Changing Categorical Values into Numerical Values

```
X=pd.get_dummies(X)
```



Train and Test Split



```
from sklearn.model_selection import train_test_split  
  
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size = 0.30)  
  
X_train.shape  
  
X_test.shape  
  
y_test.shape
```

Applying Machine Learning Algorithm – Logistic Regression



```
from sklearn.linear_model import LogisticRegression
```

```
model = LogisticRegression()
```

```
model.fit(X,y)
```

```
model.score(X,y)
```

Applying Machine Learning Algorithm – Support Vector Machine

```
from sklearn.svm import SVC  
svc = SVC()  
  
svc.fit(X, y)  
  
svc.score(X,y)
```

Applying Machine Learning Algorithm – Decision Tree Classifier

```
from sklearn.tree import DecisionTreeClassifier
```

```
dft = DecisionTreeClassifier()
```

```
dft.fit(X_train, y_train)
```

Applying Machine Learning Algorithm – Gaussian NB



```
from sklearn.naive_bayes import GaussianNB
```

```
n_b = GaussianNB()
```

```
n_b.fit(X_train, y_train)
```

Applying Machine Learning Algorithm - KNeighborsClassifier



```
from sklearn.neighbors import KNeighborsClassifier
```

```
knn = KNeighborsClassifier()
```

```
knn.fit(X_train, y_train)
```

Applying Machine Learning Algorithm



```
print(lr.score(X_test, y_test))  
  
print(dtf.score(X_test, y_test))  
  
print(n_b.score(X_test, y_test))  
  
print(knn.score(X_test, y_test))  
  
print(svc.score(X_test, y_test))
```



- [Best Model Deployment](#)
- [All Model Deployment – Ensemble Learning](#)
- [References](#)

Best Model Deployment



```
gender=input("What is your gender:")  
married=input("Married:")  
dependents=int(input("dependents value:"))  
Education=input("enter your education")  
SelfEmployed=input("Self Employed:")  
Applicantincome=int(input("enter applicant income"))  
coapplicantincome=int(input("enter co applicant income:"))  
loanamount=int(input("enter loan amount:"))  
loanamountterm=int(input("enter loan amount term:"))  
credithistory=int(input("enter credit history:"))  
propertyarea=input("enter property area:")
```



Converting the values into list

```
data = [[gender,married,dependents,Education,SelfEmployed,Applicantincome,  
coapplicantincome,loanamount, loanamountterm, credithistory, propertyarea]]
```

Converting the list into dataframe

```
newdf = pd.DataFrame(data, columns =  
['Gender','Married','Dependents','Education','Self_Employed','ApplicantIncome','CoapplicantIncome','L  
oanAmount','Loan_Amount_Term','Credit_History','Property_Area'])
```



- **Transforming categorical values into numerical one**

```
newdf = pd.get_dummies(newdf)
```

- **Identifying missing value**

```
missing_cols = set( X_train.columns ) - set( newdf.columns )
```

- **Add a missing column in test set with default value equal to 0**

```
for c in missing_cols:  
    newdf[c] = 0
```



- Ensure the order of column in the test set is in the same order than in train set

```
newdf = newdf[X_train.columns]
```

- Predicting the Loan_Status on new values

```
yp=n_b.predict(newdf)  
print(yp)
```

```
if (yp[0]=='Y'):  
    print("Your Loan is approved, Please contact at HDFC Bank Branch for further processing")  
else:  
    print("Sorry ! Your Loan is not approved")
```

All Model Deployment – Ensemble Learning



```
yp1=n_b.predict(newdf)
yp2=lr.predict(newdf)
yp3=dtf.predict(newdf)
yp4=knn.predict(newdf)
yp5=svc.predict(newdf)
```

```
print(yp1)
print(yp2)
print(yp3)
print(yp4)
print(yp5)
```



- [Classification Accuracy](#)
- [Classification Error Rate](#)
- [Problems with Accuracy](#)
- [Confusion Matrix](#)
- [Example](#)
- [Definition of the Terms:](#)
- [Example Confusion Matrix](#)
- [Observation](#)
- [Improving the Confusion Matrix](#)
- [Confusion Matrix Improvement](#)
- [Model Accuracy](#)

Classification Accuracy



- Classification accuracy alone can be misleading if you have an unequal number of observations in each class or if you have more than two classes in your dataset.
- Calculating a confusion matrix can give you a better idea of what your classification model is getting right and what types of errors it is making.

Classification Accuracy



- Classification Accuracy and its Limitations
- Classification accuracy is the ratio of correct predictions to total predictions made.

classification accuracy = correct predictions / total predictions

- It is often presented as a percentage by multiplying the result by 100.

classification accuracy = correct predictions / total predictions *100

Classification Error Rate



Error Rate

Classification accuracy can also easily be turned into a misclassification rate or error rate by inverting the value, such as:

error rate = (1 - (correct predictions / total predictions)) * 100

Error rate=(1-classification accuracy)

Problems with Accuracy



- The main problem with classification accuracy is that it hides the detail you need to better understand the performance of your classification model. There are two examples where you are most likely to encounter this problem:
- **Scenario 1:**

When your data has more than 2 classes. With 3 or more classes you may get a classification accuracy of 80%, but you don't know if that is because all classes are being predicted equally well or whether one or two classes are being neglected by the model.

Problems with Accuracy



□ Scenario 2

When your data does not have an even number of classes. You may achieve accuracy of 90% or more, but this is not a good score if 90 records for every 100 belong to one class and you can achieve this score by always predicting the most common class value.

Confusion Matrix



- A confusion matrix is a summary of prediction results on a classification problem.
- A confusion matrix is a table that is often used to describe the performance of a classification model (or “classifier”) on a set of test data for which the true values are known.
- It allows the visualization of the performance of an algorithm.
- It allows easy identification of confusion between classes e.g. one class is commonly mislabelled as the other.
- Most performance measures are computed from the confusion matrix.
- The confusion matrix shows the ways in which your classification model is confused when it makes predictions.

Confusion Matrix



- A confusion matrix is a summary of prediction results on a classification problem.
- The number of correct and incorrect predictions are summarized with count values and broken down by each class.
- The confusion matrix shows the ways in which your classification model is confused when it makes predictions.
- It gives us insight not only into the errors being made by a classifier but more importantly the types of errors that are being made.



Here,

- Class 1 : Positive
- Class 2 : Negative

	<i>Class 1 Predicted</i>	<i>Class 2 Predicted</i>
<i>Class 1 Actual</i>	TP	FN
<i>Class 2 Actual</i>	FP	TN

Definition of the Terms:



- **Positive (P)** : Observation is positive (for example: is an apple).
- **Negative (N)** : Observation is not positive (for example: is not an apple).
- **True Positive (TP)** : Observation is positive, and is predicted to be positive.
- **False Positive (FP)** : Observation is negative, but is predicted positive.
- **True Negative (TN)**: Observation is negative, and is predicted to be negative.
- **False Negative (FN)** : Observation is positive, but is predicted negative.

Example to interpret confusion matrix



$n = 165$	Predicted: No	Predicted: Yes
Actual: No	50	10
Actual: Yes	5	100

Example Confusion Matrix



- For the simplification of the above confusion matrix, added all the terms like TP, FP, etc and the row and column totals in the following image

		Predicted: No	Predicted: Yes	
		Tn = 50	FP=10	60
Actual: No	Actual: No	Tn = 50	FP=10	60
	Actual: Yes	Fn=5	Tp=100	105
		55	110	

Confusion Matrix using Python



□ Applying Confusion Matrix on Logistic Regression Model

```
from sklearn.metrics import confusion_matrix  
  
y_predict=lr.predict(X_test)  
  
results = confusion_matrix(y_test, y_predict)  
  
print('Confusion Matrix :')  
  
print(results)
```





□ Putting the result in to proper matrix

	Predicted No	Predicted Yes
Actual No	20	25
Actual Yes	3	102



- The above confusion matrix clearly shows that the logistic regression model accurately predicts the 'YES' class well. The model does 102 'YES' out of 105 'YES'. The model only predicts 3 'NO'.
- But look at the 'NO' class. It predicts only 20 'NO' out of 45 'NO'.
- This clearly shows that the Logistic Model precisely classifies the 'YES' class but unable to classify 'NO' class well.

Improving the Confusion Matrix



- The confusion matrix can be improved by changing the probability of predicting a class.
- Generally, a class can be predicted by calculating the probability of occurrence of the class. The default value is $>=0.50$. For example, if the probability of a class is $>=0.50$, then that class will be selected.
- By changing the default probability, we can improve the confusion matrix.

Confusion Matrix Improvement



- **Binarization is used when you want to convert a numerical feature vector into a Boolean vector.**
- **binarize():** transforms the data into 0 and 1.
- For example, binarize(y_predict,0.70) will binarize the y_predict into 0 and 1. All values of y_predict where probability is >0.70 will be allotted 1 and remaining 0.

#Predicting the probabilities of y_test on X_test

```
y_predict=lr.predict_proba(X_test)
```

- **Binarizing the predicted probabilities in to 0 and 1.**

#Where probability is >1

```
y_pred_class=binarize(y_predict,0.70)
```

Confusion Matrix Improvement



- Accessing class '1' from y_pred_class

```
y_pred_class1=y_pred_class[:,1]
```

- Transforming the y_pred as integer

```
y_pred=y_pred_class1.astype(int)
```

- Preparing the confusion matrix to improve the prediction

```
results = confusion_matrix(y_test, y_pred)
```

```
print('Confusion Matrix :')
```

```
print(results)
```

Confusion Matrix Improvement



- We can improve the confusion matrix by changing the different values in binarize().

`y_pred_class=binarize(y_predict, 0.75)`

or

`y_pred_class=binarize(y_predict, 0.78)`



- Model Accuracy specifies how often is the classifier correct? We can calculate the accuracy of any model as:

```
from sklearn import metrics
```

```
# Model Accuracy, how often is the classifier correct?
```

```
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

```
#Classification Problem
```

```
#Loan Prediction Problem
```

```
import pandas as pd
```

```
df=pd.read_csv(r'D:\BigData_All_content\csv\train_loan.csv')  
df.head()
```

```
    Loan_ID Gender Married Dependents Education Self_Employed \
0   LP001002   Male     No        0   Graduate       No
1   LP001003   Male    Yes        1   Graduate       No
2   LP001005   Male    Yes        0   Graduate      Yes
3   LP001006   Male    Yes        0  Not Graduate  No
4   LP001008   Male     No        0   Graduate       No

    ApplicantIncome CoapplicantIncome  LoanAmount  Loan_Amount_Term \
0           5849             0.0        NaN          360.0
1           4583            1508.0      128.0         360.0
2           3000             0.0        66.0         360.0
3           2583            2358.0      120.0         360.0
4           6000             0.0        141.0        360.0

    Credit_History Property_Area Loan_Status
0              1.0      Urban        Y
1              1.0      Rural        N
2              1.0      Urban        Y
3              1.0      Urban        Y
4              1.0      Urban        Y
```

```
df.describe()
```

```
    ApplicantIncome  CoapplicantIncome  LoanAmount  
Loan_Amount_Term \
count       614.000000      614.000000  592.000000
600.000000
mean       5403.459283     1621.245798  146.412162
342.000000
std        6109.041673     2926.248369   85.587325
65.12041
min        150.000000      0.000000    9.000000
12.000000
25%        2877.500000      0.000000   100.000000
360.000000
50%        3812.500000     1188.500000   128.000000
360.000000
75%        5795.000000     2297.250000   168.000000
360.000000
max       81000.000000    41667.000000   700.000000
480.000000
```

```

        Credit_History
count      564.000000
mean       0.842199
std        0.364878
min       0.000000
25%       1.000000
50%       1.000000
75%       1.000000
max       1.000000

df.dtypes

Loan_ID           object
Gender            object
Married           object
Dependents        object
Education          object
Self_Employed     object
ApplicantIncome    int64
CoapplicantIncome   float64
LoanAmount         float64
Loan_Amount_Term   float64
Credit_History     float64
Property_Area      object
Loan_Status         object
dtype: object

df.columns

Index(['Loan_ID', 'Gender', 'Married', 'Dependents', 'Education',
       'Self_Employed', 'ApplicantIncome', 'CoapplicantIncome',
       'LoanAmount',
       'Loan_Amount_Term', 'Credit_History', 'Property_Area',
       'Loan_Status'],
      dtype='object')

#Removing Loan_ID feature

del df['Loan_ID']
df.head()

   Gender Married Dependents      Education Self_Employed
ApplicantIncome \
0   Male      No          0      Graduate           No
5849
1   Male      Yes          1      Graduate           No
4583
2   Male      Yes          0      Graduate          Yes
3000
3   Male      Yes          0  Not Graduate          No
2583

```

```

4   Male      No       0     Graduate      No
6000

          CoapplicantIncome  LoanAmount  Loan_Amount_Term  Credit_History \
0             0.0           NaN        360.0            1.0
1            1508.0         128.0      360.0            1.0
2              0.0           66.0      360.0            1.0
3            2358.0         120.0      360.0            1.0
4              0.0          141.0      360.0            1.0

  Property_Area  Loan_Status
0      Urban        Y
1     Rural         N
2      Urban        Y
3      Urban        Y
4      Urban        Y

```

#Prepare independent/input feature X

```
X=df.drop('Loan_Status', axis=1)
X.head()
```

	Gender	Married	Dependents	Education	Self_Employed
ApplicantIncome \					
0	Male	No	0	Graduate	No
5849					
1	Male	Yes	1	Graduate	No
4583					
2	Male	Yes	0	Graduate	Yes
3000					
3	Male	Yes	0	Not Graduate	No
2583					
4	Male	No	0	Graduate	No
6000					

	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History \
0	0.0	NaN	360.0	1.0
1	1508.0	128.0	360.0	1.0
2	0.0	66.0	360.0	1.0
3	2358.0	120.0	360.0	1.0
4	0.0	141.0	360.0	1.0

	Property_Area
0	Urban
1	Rural
2	Urban
3	Urban
4	Urban

```
#Prepare dependent/outcome feature y

y=df['Loan_Status']
y.head()

0      Y
1      N
2      Y
3      Y
4      Y
Name: Loan_Status, dtype: object

#Label encoding of outcome between 0 and 1

#setting N=0 and Y=1

from sklearn.preprocessing import LabelBinarizer
lb=LabelBinarizer()
y=lb.fit_transform(y)
y[0:5]

array([[1],
       [0],
       [1],
       [1],
       [1]]))

#Checking missing values in X

X.isnull().sum()

Gender          13
Married          3
Dependents      15
Education         0
Self_Employed   32
ApplicantIncome   0
CoapplicantIncome  0
LoanAmount       22
Loan_Amount_Term  14
Credit_History    50
Property_Area     0
dtype: int64

X.dtypes

Gender          object
Married          object
Dependents      object
Education        object
Self_Employed   object
```

```
ApplicantIncome      int64
CoapplicantIncome    float64
LoanAmount          float64
Loan_Amount_Term    float64
Credit_History      float64
Property_Area       object
dtype: object
```

#Filling missing data in Gender

#Counting frequency of each class in 'Gender'

```
X.Gender.value_counts()
```

```
Male      489
Female    112
Name: Gender, dtype: int64
```

```
X.Gender.fillna('Male', inplace=True)
X.isnull().sum()
```

```
Gender          0
Married         3
Dependents     15
Education       0
Self_Employed   32
ApplicantIncome 0
CoapplicantIncome 0
LoanAmount      22
Loan_Amount_Term 14
Credit_History   50
Property_Area    0
dtype: int64
```

#Fill missing data in 'Married'

```
X.Married.value_counts()
```

```
Yes      398
No      213
Name: Married, dtype: int64
```

```
X.Married.fillna('Yes', inplace=True)
X.isnull().sum()
```

```
Gender          0
Married         0
Dependents     15
Education       0
Self_Employed   32
ApplicantIncome 0
CoapplicantIncome 0
```

```
LoanAmount      22  
Loan_Amount_Term    14  
Credit_History     50  
Property_Area       0  
dtype: int64
```

#Filling missing data in 'Dependents'

```
X.Dependents.value_counts()
```

```
0      345  
1      102  
2      101  
3+     51  
Name: Dependents, dtype: int64
```

```
X.Dependents.fillna('0', inplace=True)  
X.isnull().sum()
```

```
Gender      0  
Married     0  
Dependents  0  
Education    0  
Self_Employed 32  
ApplicantIncome 0  
CoapplicantIncome 0  
LoanAmount   22  
Loan_Amount_Term 14  
Credit_History 50  
Property_Area  0  
dtype: int64
```

#Filling missing values in 'Self_Employed'

```
X.Self_Employed.value_counts()
```

```
No      500  
Yes     82  
Name: Self_Employed, dtype: int64
```

```
X.Self_Employed.fillna('No', inplace=True)  
X.isnull().sum()
```

```
Gender      0  
Married     0  
Dependents  0  
Education    0  
Self_Employed 0  
ApplicantIncome 0  
CoapplicantIncome 0  
LoanAmount   22  
Loan_Amount_Term 14
```

```
Credit_History      50  
Property_Area       0  
dtype: int64
```

```
X.dtypes
```

```
Gender            object  
Married           object  
Dependents        object  
Education          object  
Self_Employed     object  
ApplicantIncome    int64  
CoapplicantIncome   float64  
LoanAmount         float64  
Loan_Amount_Term    float64  
Credit_History     float64  
Property_Area      object  
dtype: object
```

#Filling missing data in 'Loan_Amount_Term'

```
import numpy as np
```

```
X.Loan_Amount_Term.fillna(np.mean(X.Loan_Amount_Term), inplace=True)  
X.isnull().sum()
```

```
Gender            0  
Married           0  
Dependents        0  
Education          0  
Self_Employed     0  
ApplicantIncome    0  
CoapplicantIncome   0  
LoanAmount         22  
Loan_Amount_Term    0  
Credit_History     50  
Property_Area      0  
dtype: int64
```

#Filling missing data in LoanAmount

```
X.LoanAmount.fillna(np.mean(X.LoanAmount), inplace=True)  
X.isnull().sum()
```

```
Gender            0  
Married           0  
Dependents        0  
Education          0  
Self_Employed     0  
ApplicantIncome    0  
CoapplicantIncome   0
```

```
LoanAmount          0  
Loan_Amount_Term   0  
Credit_History     50  
Property_Area      0  
dtype: int64
```

#Filling missing values in Credit_History

```
X.Credit_History.value_counts()
```

```
1.0    475  
0.0     89  
Name: Credit_History, dtype: int64
```

```
X.Credit_History.fillna(1.0, inplace=True)  
X.isnull().sum()
```

```
Gender            0  
Married           0  
Dependents        0  
Education          0  
Self_Employed     0  
ApplicantIncome    0  
CoapplicantIncome   0  
LoanAmount         0  
Loan_Amount_Term   0  
Credit_History     0  
Property_Area      0  
dtype: int64
```

#Encoding categorical features/columns of X into numerical type using one-hot encoding

```
X=pd.get_dummies(X)  
X.head()
```

	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	\
0	5849	0.0	146.412162		360.0
1	4583	1508.0	128.000000		360.0
2	3000	0.0	66.000000		360.0
3	2583	2358.0	120.000000		360.0
4	6000	0.0	141.000000		360.0

\	Credit_History	Gender_Female	Gender_Male	Married_No	Married_Yes
0	1.0	0	1	1	0
1	1.0	0	1	0	1
2	1.0	0	1	0	1

```

3           1.0          0           1          0           1
4           1.0          0           1          1           0

Dependents_0  Dependents_1  Dependents_2  Dependents_3+ \
0            1            0            0            0
1            0            1            0            0
2            1            0            0            0
3            1            0            0            0
4            1            0            0            0

Education_Graduate  Education_Not Graduate  Self_Employed_No \
0                  1                      0                      1
1                  1                      0                      1
2                  1                      0                      0
3                  0                      1                      1
4                  1                      0                      1

Self_Employed_Yes  Property_Area_Rural  Property_Area_Semiurban \
0                  0                      0                      0
1                  0                      1                      0
2                  1                      0                      0
3                  0                      0                      0
4                  0                      0                      0

Property_Area_Urban
0                  1
1                  0
2                  1
3                  1
4                  1

```

#Splitting the data in to training set and testing set

```

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test=train_test_split(X,y, test_size=0.2)
#Applying classification ML algorithms

```

#Applying LogisticRegression algo

```
from sklearn.linear_model import LogisticRegression
```

```
LR=LogisticRegression()
LR.fit(X_train, y_train)
```

```
C:\Users\hp\Anaconda3\lib\site-packages\sklearn\linear_model\
logistic.py:432: FutureWarning: Default solver will be changed to
```

```
'lbfgs' in 0.22. Specify a solver to silence this warning.  
FutureWarning)  
C:\Users\hp\Anaconda3\lib\site-packages\sklearn\utils\  
validation.py:724: 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)  
  
LogisticRegression(C=1.0, class_weight=None, dual=False,  
fit_intercept=True,  
intercept_scaling=1, l1_ratio=None, max_iter=100,  
multi_class='warn', n_jobs=None, penalty='l2',  
random_state=None, solver='warn', tol=0.0001,  
verbose=0,  
warm_start=False)
```

#Applying Support Vector Classifier(SVC)

```
from sklearn.svm import SVC  
  
svc=SVC()  
svc.fit(X_train, y_train)  
  
C:\Users\hp\Anaconda3\lib\site-packages\sklearn\utils\  
validation.py:724: 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)  
C:\Users\hp\Anaconda3\lib\site-packages\sklearn\svm\base.py:193:  
FutureWarning: The default value of gamma will change from 'auto' to  
'scale' in version 0.22 to account better for unscaled features. Set  
gamma explicitly to 'auto' or 'scale' to avoid this warning.  
"avoid this warning.", FutureWarning)  
  
SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,  
decision_function_shape='ovr', degree=3, gamma='auto_deprecated',  
kernel='rbf', max_iter=-1, probability=False, random_state=None,  
shrinking=True, tol=0.001, verbose=False)
```

#Applying DecisionTreeClassifier

```
from sklearn.tree import DecisionTreeClassifier  
  
dtc=DecisionTreeClassifier()  
dtc.fit(X_train, y_train)  
  
DecisionTreeClassifier(class_weight=None, criterion='gini',  
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=False,
    random_state=None, splitter='best')
```

#Applying RandomForestClassifier

```
from sklearn.ensemble import RandomForestClassifier

rfc=RandomForestClassifier()
rfc.fit(X_train, y_train)

C:\Users\hp\Anaconda3\lib\site-packages\sklearn\ensemble\
forest.py:245: FutureWarning: The default value of n_estimators will
change from 10 in version 0.20 to 100 in 0.22.
    "10 in version 0.20 to 100 in 0.22.", FutureWarning)
C:\Users\hp\Anaconda3\lib\site-packages\ipykernel_launcher.py:6:
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().
```

```
RandomForestClassifier(bootstrap=True, class_weight=None,
criterion='gini',
                      max_depth=None, max_features='auto',
max_leaf_nodes=None,           min_impurity_decrease=0.0,
min_impurity_split=None,       min_samples_leaf=1, min_samples_split=2,
                                min_weight_fraction_leaf=0.0, n_estimators=10,
                                n_jobs=None, oob_score=False,
random_state=None,             verbose=0, warm_start=False)
```

#Applying GaussianNB

```
from sklearn.naive_bayes import GaussianNB

gnb=GaussianNB()
gnb.fit(X_train, y_train)

C:\Users\hp\Anaconda3\lib\site-packages\sklearn\utils\
validation.py:724: 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)
```

```
GaussianNB(priors=None, var_smoothing=1e-09)
```

#Score

```
print('Score of LogisticRegression:', LR.score(X_test, y_test))
```

```
print('Score of Support Vector Classifier=' , svc.score(X_test,
y_test))
print('Score of DecisionTreeClassifier=' , dtc.score(X_test, y_test))
print('Score of RandomForestClassifier=' , rfc.score(X_test, y_test))
print('Score of GaussianNB=' , gnb.score(X_test, y_test) )

Score of LogisticRegression: 0.7886178861788617
Score of Support Vector Classifier= 0.6910569105691057
Score of DecisionTreeClassifier= 0.6991869918699187
Score of RandomForestClassifier= 0.7317073170731707
Score of GaussianNB= 0.7804878048780488
```

*#Since, the score of LogisticRegression algo/model is highest
#WE can say that LogisticRegression is best algo/model for this problem*

```
#Preparing the confusion matrix on LogisticRegression

#confusion matrix is prepared on testing data

#syntax: confusion_matrix(actual outcome, predicted outcome)

from sklearn.metrics import confusion_matrix

#Predicting the value of 'Loan_Status' on testing data using object of LogisticRegression

lr_pred=LR.predict(X_test)

LR_conf=confusion_matrix(y_test, lr_pred)
LR_conf

array([[15, 24],
       [ 2, 82]], dtype=int64)
```

```
-----
-----
AttributeError                                Traceback (most recent call
last)
<ipython-input-35-2820df361543> in <module>
----> 1 y_test.value_counts()
```

AttributeError: 'numpy.ndarray' object has no attribute 'value_counts'

#Preparing data from for confusion matrix

```
df_conf=pd.DataFrame(LR_conf, columns=[ 'N', 'Y'], index=[ 'N', 'Y'])
```

```
        min_weight_fraction_leaf=0.0, n_estimators=10,
n_jobs=None, oob_score=False,
random_state=None,
verbose=0, warm_start=False)
```

#Applying GaussianNB

```
from sklearn.naive_bayes import GaussianNB
```

```
gnb=GaussianNB()
gnb.fit(X_train, y_train)
```

```
GaussianNB(priors=None, var_smoothing=1e-09)
```

#Score

```
print('Score of LogisticRegression:', LR.score(X_test, y_test))
print('Score of Support Vector Classifier=', svc.score(X_test,
y_test))
print('Score of DecisionTreeClassifier=', dtc.score(X_test, y_test))
print('Score of RandomForestClassifier=', rfc.score(X_test, y_test))
print('Score of GaussianNB=', gnb.score(X_test, y_test) )
```

Score of LogisticRegression: 0.8373983739837398

Score of Support Vector Classifier= 0.7235772357723578

Score of DecisionTreeClassifier= 0.7154471544715447

Score of RandomForestClassifier= 0.7479674796747967

Score of GaussianNB= 0.7886178861788617

#Since, the score of LogisticRegression algo/model is highest
#WE can say that LogisticRegression is best algo/model for this problem

#Predicting the outcome of testing data using LogisticRegression

```
LR_pred=LR.predict(X_test)
```

```
df_pred=pd.DataFrame({'Actual Outcome':y_test, 'Predicted
Outcome':LR_pred})
df_pred.head(10)
```

	Actual Outcome	Predicted Outcome
192	N	Y
461	Y	Y
111	Y	Y
546	N	Y
184	Y	Y
537	Y	Y
481	Y	Y
407	Y	Y

```
290          Y          Y  
477          N          N
```

```
#Implementing the LogisticRegression model on new data
```

```
df.dtypes
```

```
Gender          object  
Married         object  
Dependents     object  
Education       object  
Self_Employed  object  
ApplicantIncome int64  
CoapplicantIncome float64  
LoanAmount      float64  
Loan_Amount_Term float64  
Credit_History  float64  
Property_Area   object  
Loan_Status     object  
dtype: object
```

```
df.describe()
```

	ApplicantIncome	CoapplicantIncome	LoanAmount
Loan_Amount_Term \ count	614.000000	614.000000	592.000000
600.000000			
mean	5403.459283	1621.245798	146.412162
342.000000			
std	6109.041673	2926.248369	85.587325
65.12041			
min	150.000000	0.000000	9.000000
12.000000			
25%	2877.500000	0.000000	100.000000
360.000000			
50%	3812.500000	1188.500000	128.000000
360.000000			
75%	5795.000000	2297.250000	168.000000
360.000000			
max	81000.000000	41667.000000	700.000000
480.000000			

	Credit_History
count	564.000000
mean	0.842199
std	0.364878
min	0.000000
25%	1.000000
50%	1.000000
75%	1.000000
max	1.000000

```
df.dtypes
```

```
Gender          object
Married         object
Dependents     object
Education       object
Self_Employed   object
ApplicantIncome int64
CoapplicantIncome float64
LoanAmount      float64
Loan_Amount_Term float64
Credit_History  float64
Property_Area   object
Loan_Status     object
dtype: object
```

```
#input new data
```

```
gender=input('Enter Your Gender (Male or Female)=')
married=input('Enter Marital Status (No or Yes)=')
depen=input('Enter Dependents(0/1/2/3+)=')
edu=input('Enter Your Education (Graduate/Not Graduate)=')
self=input('Enter Self employed status (No or Yes)=')
app=int(input('Enter Applicant Income (150 to 81000)='))
coapp=float(input('Enter Coapplicant Income (0 to 41667)='))
loan=float(input('Enter Loan Amount (9 to 700)='))
loanterm=float(input('Enter Loan Amount Term (12 to 480)='))
credit=float(input('Enter Credit History (0 or 1)='))
prop=input('Enter Property Area (Rural/Urban/Semiurban)=')
```

```
Enter Your Gender (Male or Female)=Male
Enter Marital Status (No or Yes)=Yes
Enter Dependents(0/1/2/3+)=0
Enter Your Education (Graduate/Not Graduate)=Graduate
Enter Self employed status (No or Yes)=No
Enter Applicant Income (150 to 81000)=2500
Enter Coapplicant Income (0 to 41667)=100
Enter Loan Amount (9 to 700)=400
Enter Loan Amount Term (12 to 480)=360
Enter Credit History (0 or 1)=1
Enter Property Area (Rural/Urban/Semiurban)=Urban
```

```
df.columns
```

```
Index(['Gender', 'Married', 'Dependents', 'Education',
'Self_Employed',
       'ApplicantIncome', 'CoapplicantIncome', 'LoanAmount',
       'Loan_Amount_Term', 'Credit_History', 'Property_Area',
'Loan_Status'],
      dtype='object')
```

```

#First arange these values in a list in same order as in training data

value=[[gender, married, depen, edu, self, app, coapp, loan, loanterm,
credit, prop]]

print(value)
[['Male', 'Yes', '0', 'Graduate', 'No', 2500, 100.0, 400.0, 360.0,
1.0, 'Urban']]

#convert the list into dataframe

newdf=pd.DataFrame(value, columns=['Gender', 'Married', 'Dependents',
'Education', 'Self_Employed',
'ApplicantIncome', 'CoapplicantIncome', 'LoanAmount',
'Loan_Amount_Term', 'Credit_History', 'Property_Area'])
newdf

   Gender Married Dependents Education Self_Employed
ApplicantIncome \
0   Male     Yes           0   Graduate           No        2500

   CoapplicantIncome  LoanAmount  Loan_Amount_Term Credit_History \
0            100.0       400.0          360.0             1.0

   Property_Area
0            Urban

#Encode categorical data into numerical one by using one-hot encoding

newdf=pd.get_dummies(newdf)
newdf

   ApplicantIncome  CoapplicantIncome  LoanAmount  Loan_Amount_Term \
0            2500              100.0       400.0          360.0

   Credit_History  Gender_Male  Married_Yes  Dependents_0
Education_Graduate \
0                 1.0           1           1           1
1

   Self_Employed_No  Property_Area_Urban
0                  1                      1

X_train.columns

Index(['ApplicantIncome', 'CoapplicantIncome', 'LoanAmount',
       'Loan_Amount_Term', 'Credit_History', 'Gender_Female',
       'Gender_Male',
       'Married_No', 'Married_Yes', 'Dependents_0', 'Dependents_1'],
      dtype='object')

```

```

'Dependents_2', 'Dependents_3+', 'Education_Graduate',
'Education_Not Graduate', 'Self_Employed_No',
'Self_Employed_Yes',
'Property_Area_Rural', 'Property_Area_Semiurban',
'Property_Area_Urban'],
dtype='object')

newdf.columns

Index(['ApplicantIncome', 'CoapplicantIncome', 'LoanAmount',
       'Loan_Amount_Term', 'Credit_History', 'Gender_Male',
'Married_Yes',
       'Dependents_0', 'Education_Graduate', 'Self_Employed_No',
       'Property_Area_Urban'],
      dtype='object')

print('NO. of features/columns in training data=',
len(X_train.columns))
print('NO. of features/columns in new data= ', len(newdf.columns))

NO. of features/columns in training data= 20
NO. of features/columns in new data= 11

#If we apply/implement any ML algo on new data

#No. of features/columns in new data must match with the no. of columns in training data

#Identify those columns/features which are available in X_train but in newdf

missing_cols=set(X_train.columns)-set(newdf.columns)

print('Missing columns/features in newdf')
missing_cols

Missing columns/features in newdf

{'Dependents_1',
'Dependents_2',
'Dependents_3+',
'Education_Not Graduate',
'Gender_Female',
'Married_No',
'Property_Area_Rural',
'Property_Area_Semiurban',
'Self_Employed_Yes'}

#Adding these missing columns in newdf with 0

```

```

for i in missing_cols:
    newdf[i]=0

print('NO. of features/columns in training data=',
len(X_train.columns))
print('NO. of features/columns in new data=' , len(newdf.columns))

NO. of features/columns in training data= 20
NO. of features/columns in new data= 20

#The order of the columns in newdf must match with the order of the
columns in X_train

#Arrange the columns of newdf as per X_train

newdf=newdf[X_train.columns]
newdf

   ApplicantIncome  CoapplicantIncome  LoanAmount  Loan_Amount_Term \
0           2500            100.0       400.0        360.0

   Credit_History  Gender_Female  Gender_Male  Married_No  Married_Yes \
\0             1.0                 0             1             0             1

   Dependents_0  Dependents_1  Dependents_2  Dependents_3+ \
0             1                 0                 0                 0

   Education_Graduate  Education_Not Graduate  Self_Employed_No \
0                  1                      0                     1

   Self_Employed_Yes  Property_Area_Rural  Property_Area_Semiurban \
0                   0                         0                         0

   Property_Area_Urban
0                   1

#Now, our new data is ready for Machine Learning

#Predicting the value of 'Loan_Status' on this new data using object
of LogisticRegression

new_pred=LR.predict(newdf)
new_pred

array(['Y'], dtype=object)

```

```
if(new_pred[0]=='Y'):
    print('Congrats! Your loan may be approved. Please contact
branch')
else:
    print('Sorry! Your loan is not approved')
```

Congrats! Your loan may be approved. Please contact branch

	N	Y
N	15	24
Y	2	82

#On the basis of confusion matrix, we can conclude

#LogisticRegression model is able to identify 'Y' class perfectly with 99% accuracy.

#But, it has some difficulty in identifyingn 'N' class with 57% accuracy

#Preparing the confusion matrix on Support vector classifier

```
svc_pred=svc.predict(X_test)
```

```
svc_conf=confusion_matrix(y_test, svc_pred)
```

```
df_conf1=pd.DataFrame(svc_conf, columns=['N', 'Y'], index=['N', 'Y'])
```

	N	Y
N	1	38
Y	0	84

#Preparing the confusion matrix on Decision tree classifier

```
dtc_pred=dtc.predict(X_test)
```

```
dtc_conf=confusion_matrix(y_test, dtc_pred)
```

```
df_conf12=pd.DataFrame(dtc_conf, columns=['N', 'Y'], index=['N', 'Y'])
```

	N	Y
N	21	18
Y	19	65

#Preparing the confusion matrix on Random Forest classifier

```
rfc_pred=rfc.predict(X_test)
```

```
rfc_conf=confusion_matrix(y_test, rfc_pred)
```

```
df_conf13=pd.DataFrame(rfc_conf, columns=['N', 'Y'], index=['N', 'Y'])
```

	N	Y
N	16	23
Y	10	74

```
#Preparing the confusion matrix on GaussianNB
gnb_pred=gnb.predict(X_test)

gnb_conf=confusion_matrix(y_test, gnb_pred)
df_conf14=pd.DataFrame(gnb_conf, columns=['N', 'Y'], index=['N', 'Y'])
df_conf14

      N    Y
N    16   23
Y     4   80

#the objective of this problem is to accept the loan application of all eligible applicants.
#Try not to reject the loan application of eligible candidates

#The score of LogisticRegression is 83% and no. of incorrect prediction in 'No' class is very low
#1

#so, on the basis of score and confusion matrix, LogisticRegression is the best model for this problem

#Improvement of confusion matrix

#For predicting the output of any data, classification algo/model first calculates the probability #of each class
#Then, it selects the class as Final output whose probability is > 0.50
```

	<i>Prob of N</i>	<i>Prob of Y</i>	<i>Final Outcome</i>
D1	0.3	0.7	Y
D2	0.75	0.25	N
D3	0.45	0.55	Y

```
'\n\n      Prob of N\t\tProb of Y\t\tFinal Outcome\n\nD1      \
t0.3\t\t    0.7\t\t      Y\n\nD2\t\t    0.75\t\t 0.25\t\t
N\n\nD3\t\t    0.45\t\t 0.55\t\t      Y\n'
```

#Now, printing the probability of each class on testing data using LogisticRegression

```
LR_prob=LR.predict_proba(X_test)
LR_prob_df=pd.DataFrame(LR_prob, columns=[ 'N' , 'Y'])
LR_prob_df.head()
```

```

N          Y
0  0.121314  0.878686
1  0.056206  0.943794
2  0.149264  0.850736
3  0.261450  0.738550
4  0.328526  0.671474

LR_pred=LR.predict(X_test)
LR_pred[0:5]

array([1, 1, 1, 1, 1])

LR_prob_df['Final Output']=LR_pred
LR_prob_df.head(10)

      N          Y  Final Output
0  0.121314  0.878686          1
1  0.056206  0.943794          1
2  0.149264  0.850736          1
3  0.261450  0.738550          1
4  0.328526  0.671474          1
5  0.304481  0.695519          1
6  0.327381  0.672619          1
7  0.126139  0.873861          1
8  0.240794  0.759206          1
9  0.683340  0.316660          0

```

#Improving the confusion matrix

#Confusion matrix can be improved by changing the default threshold value for probability

#Ex: setting the threshold value to 0.6

```
from sklearn.preprocessing import binarize

LR_class=binarize(LR_prob, 0.6)

LR_class1=LR_class[:, 1]
LR_class1_int=LR_class1.astype(int)

LR_class1_int

array([1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0,
       1,
       1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0,
       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,
```

```

    0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1,
0,
    1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1])

#Printing the confusion matrix of LR before improvement

df_conf

    N   Y
N  15  24
Y   2  82

#Printing the confusion matrix of LR after improvement

LR_conf60=confusion_matrix(y_test, LR_class1_int)
df_LR_60=pd.DataFrame(LR_conf60, columns=['N', 'Y'], index=['N', 'Y'])
df_LR_60

    N   Y
N  16  23
Y   3  81

LR_class=binarize(LR_prob, 0.7)

LR_class70=LR_class[:, 1]
LR_class70_int=LR_class70.astype(int)

#Printing the confusion matrix of LR after improvement

LR_conf70=confusion_matrix(y_test, LR_class70_int)
df_LR_70=pd.DataFrame(LR_conf70, columns=['N', 'Y'], index=['N', 'Y'])
df_LR_70

    N   Y
N  20  19
Y  23  61

```



- [Image Processing](#)
- [Transforming the Dataframe into Image](#)
- [Image Preprocessing](#)
- [View the shape of the image](#)
- [Image rescalling](#)
- [Image Reshaping](#)
- [References](#)



- Image, voice and video are called unstructured data. Images, voice and video are stored in the computer as data. Images are made up of **individual pixels**. The color of each pixel is represented as a binary number so the whole image is therefore stored as a series of binary numbers.

For Example:

- The sample image is given in Day17_1.csv file.
- In this file, image is represented in the form of matrix i.e. rows and columns.
- No. of rows and columns are dependent upon the resolution of the image.
- Single cell represents the colour value of an individual pixel.



□ Reading Sample Image

```
import pandas as pd  
  
df=pd.read_csv("Day17_1.csv")  
  
df.head()  
  
df.shape
```

Transforming the Dataframe into Image



- A image from Day17_1.csv file can be displayed using skimage.io and matplotlib package.
- **skimage.io:** Utilities to read and write images in various formats. Here, imshow() is used to show the image.
- **imshow():** Used to Display an image.
- **Syntax:**

`skimage.io.imshow(arr, plugin=None, **plugin_args)`

Transforming the Dataframe into Image



- showing the csv file as image

```
from skimage.io import imshow  
  
import matplotlib.pyplot as plt  
  
plt.imshow(df)  
  
plt.show()
```



Image Preprocessing



- skimage pkg is used for image processing in python
- imread: used to read images
- imshow: used to show images
- Reading 'MY.jpg' image

```
from skimage.io import imshow, imread  
  
df=imread("d:\\MY.jpg")  
  
imshow(df)
```

Image Preprocessing



□ viewing the shape of image

`df.shape`

#here 1006 is the height of the image

#768 is the width of the image

3 is colour value

`print(1006*768*3)`

□ viewing the colour value at a specific pixel

`print(df[600][600])`

#R=151

#G=67

#B=83

#RGB is between 0 to 255

Image Preprocessing



- Image classification can not be done on the basis of color
- Reading the image gray i.e. excluding the color information from the image

```
df_gray=imread("d:\\MY.jpg", as_gray=True)  
imshow(df_gray)
```

View the shape of the image



```
df_gray.shape  
  
#here the shape of the image is 1006*768  
  
#here colour information is not included  
  
print('Resolution of image:', 1006*768)  
  
print('Height of the image:', df_gray.shape[0], 'pixel')  
  
print('Width of the image:', df_gray.shape[1], 'pixel')  
  
print('Resolution of image:', df_gray.shape[0]*df_gray.shape[1])
```



- Image rescaling: making image smaller
- rescale(): used to rescale a image
- Syntax: rescale(image_name, rescaling_factor)

```
from skimage.transform import rescale  
  
df_rescale=rescale(df_gray, 0.25)  
  
imshow(df_rescale)
```



- Saving the image as csv

```
import pandas as pd  
  
df_res=pd.DataFrame(df_rescale)  
  
df_res.to_csv('d:\\image_rescale.csv')
```



- Generally, an image is stored in the form of rows and columns. Image reshaping is used to reshape the image data into single row or column. For storage purpose, an image can be stored in single row or column.
- Reshaping the image to (48384, 1)

```
#making the shape of df_rescale to (1,48384)
```

```
import pandas as pd
```

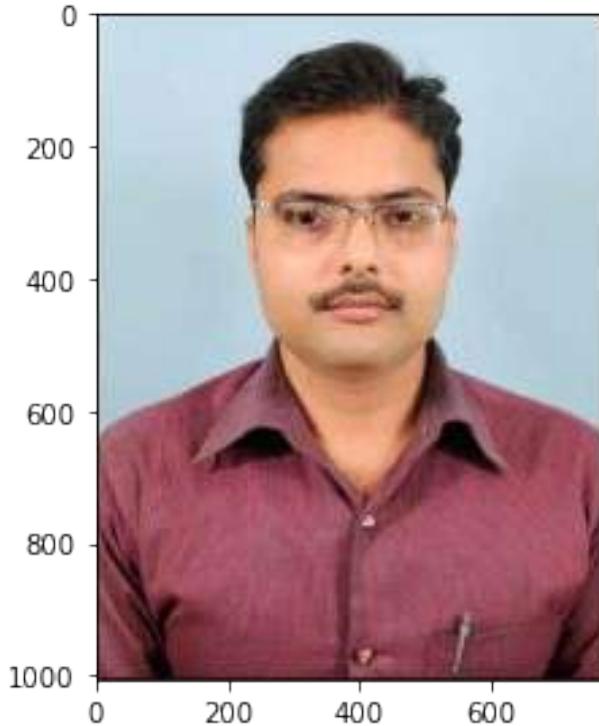
```
import numpy as np
```

```
#df_reshape=np.reshape(df_rescale, 252*192)
```

```
df_reshape=np.reshape(df_rescale, df_rescale.shape[0]*df_rescale.shape[1])
```

```
df_reshape.shape
```

```
#Image Classification  
  
#Image preprocessing in Python  
  
#skimage: pkg is used for image preprocessing  
  
#imread(): used to read the image in python  
#imshow(): used to display the image in python  
  
from skimage.io import imread, imshow  
  
#Reading the sample image using imread()  
  
#syntax: var_name=imread('location/path of the image')  
img=imread('d:\\MY.jpg')  
  
#Showing the image using imshow()  
  
imshow(img)  
<matplotlib.image.AxesImage at 0x148e129e3c8>
```



```
#To view the shape of the image  
  
img.shape
```

```
(1006, 768, 3)

img[0]

Array([[ 0,  3,  6],
       [10, 18, 21],
       [21, 29, 32],
       ...,
       [10, 18, 21],
       [ 2, 10, 13],
       [ 0,  6,  9]], dtype=uint8)

#1006 is the height of the image in pixels
#768 is the width of the image in pixels
#3 is the color value of each pixel

print("Height of the Image:", img.shape[0], 'pixels')
print('Width of the Image:', img.shape[1], 'pixels')
print('Color value of each pixel:', img.shape[2])

print('Resolution of the Image:',
      img.shape[0]*img.shape[1]*img.shape[2])

Height of the Image: 1006 pixels
Width of the Image: 768 pixels
Color value of each pixel: 3
Resolution of the Image: 2317824

#To view the color information of any specific pixel

img[600][600]

Array([151,  67,  83], dtype=uint8)

#R=151
#G=67
#B=83

img[200][150]

Array([197, 216, 230], dtype=uint8)

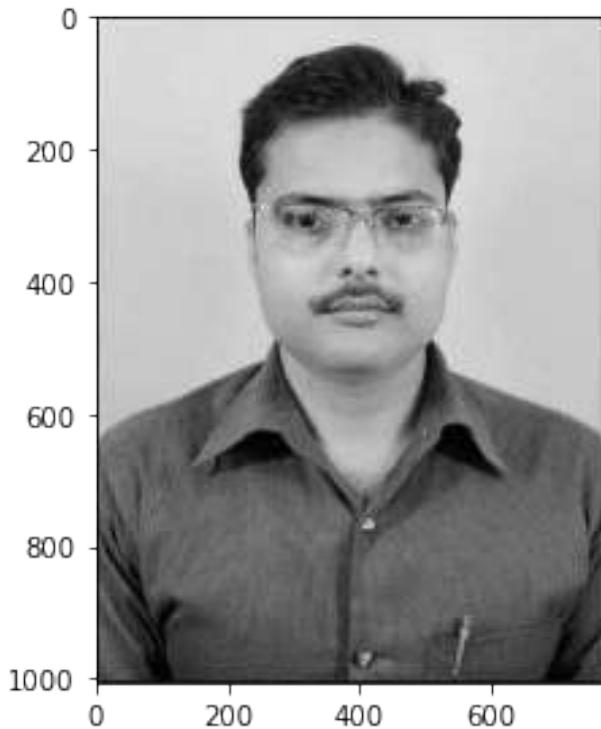
#Image classification can not be done on the basis of color of the image

#Reading the image as gray i.e. excluding the color information

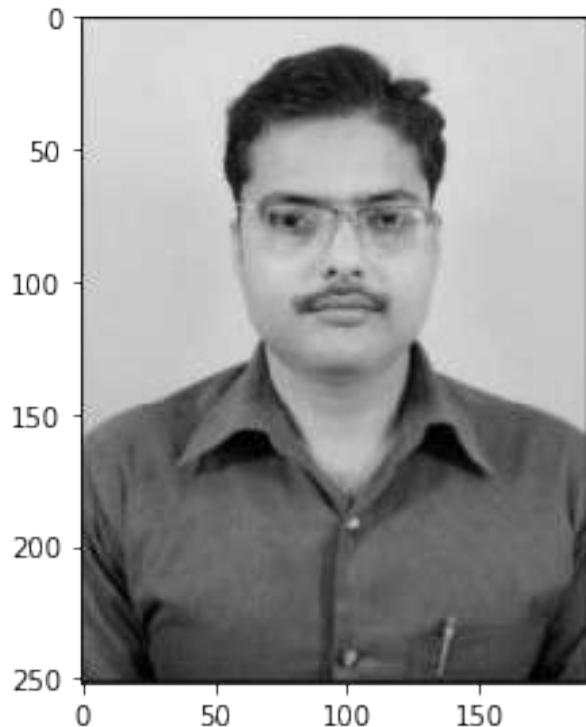
img_gray.imread('D:\\MY.jpg', as_gray=True)

imshow(img_gray)

<matplotlib.image.AxesImage at 0x148e132a470>
```



```
img_gray.shape  
(1006, 768)  
  
print('Height of the image:', img_gray.shape[0])  
print('Width of the image:', img_gray.shape[1])  
print('Resolution of the image:', img_gray.shape[0]*img_gray.shape[1])  
  
Height of the image: 1006  
Width of the image: 768  
Resolution of the image: 772608  
  
#Image rescaling: Making image smaller  
  
from skimage.transform import rescale  
  
#rescale(): used to rescale an image  
#Syntax: rescale(image, scaling_factor)  
  
img_rescale=rescale(img_gray, 0.25)  
imshow(img_rescale)  
  
<matplotlib.image.AxesImage at 0x148e14889b0>
```



```
img_rescale.shape
(252, 192)
print('Image resolution:', 252*192)
print('Image resolution:', img_rescale.shape[0]*img_rescale.shape[1])
Image resolution: 48384
Image resolution: 48384

import pandas as pd

#Transforming the rescaled image as dataframe
df=pd.DataFrame(img_rescale)

#Transferring the contents of df as csv file
df.to_csv('e:\\Image_rescale.csv')

#Image Reshaping

#reshaping the image to (1, 48384)

import numpy as np

#image_reshape=np.reshape(img_rescale, (1,48384))
#image_reshape=np.reshape(img_rescale, (1,252*192))
```

```
image_reshape=np.reshape(img_rescale,
(1,img_rescale.shape[0]*img_rescale.shape[1]))  
  
df_re=pd.DataFrame(image_reshape)  
df_re.to_csv('e:\\Image_reshape.csv')
```



- [Image Classification](#)
- [Data Pre-processing for Image Classification](#)
- [Reading Training Data](#)
- [Data Preparation](#)
- [Preparing Training Data](#)
- [Preparing X_train and y_train](#)
- [Preparing testing data](#)
- [Preparing X_test & y_test](#)
- [Applying GaussianNB](#)
- [Confusion matrix of GaussianNB](#)
- [Applying KNeighborsClassifier](#)
- [Confusion matrix of KNeighborsClassifier](#)
- [Applying SVC\(\) algorithm](#)
- [Deployment](#)
- [References](#)

Image Classification



- Image classification refers to the task of extracting information classes from an image. The resulting image classification can be used to identify a particular class. In given Example:
- Take few photos of two persons that will work as historical data.
- We will create a ML Model, which will take new image of any of the two people and identify him.
- Number of classes is 2.
- You can extend this problem for more than 2 images easily.

Data Pre-processing for Image Classification



- For image classification, the data should be prepared before applying Machine Learning algorithm.
- **Dataset:-** We have two folder of images one is training and other is test.
- For this purpose, we have clicked some random photos of two people and place them in a folder.
- In images folder we have total 20 images where 09 images belongs to person Icy and 11 images belongs to person Saumya.
- All these images are captured by mobile.
- We have prepared CSV file using Excel with two columns one is **name of images** and column 2 is **label of images**. The file is saved as image_train.csv
- Now our data is in two parts first part is image_train.csv file and other part is training folder that contains actual images.

Reading Training Data



```
import pandas as pd  
  
df_img=pd.read_csv('i:\\image_data\\train_image.csv')
```

```
df_img.head()
```

- #Preparing X_train

```
from skimage.io import imread, imshow  
from skimage.transform import rescale  
import numpy as np
```

6 Data Preparation



- **Specify the physical path of your training images**

```
path='I:\\image_data\\training\\'
```

- **Specify name of the image file**

```
file=df_img.iloc[0].Image
```

```
#file='1.jpg'
```

- **Reading first image from training data**

```
image=imread(path+file, as_gray=True)
```

```
imshow(image)
```

Preparing Training Data



```
#create an empty list
data=[]
for i in range(33):
    #specify the physical path of your training images
    path='I:\\image_data\\training\\'
    #specify name of the image file
    file=df_img.iloc[i].Image

    #reading first image from training data
    image=imread(path+file, as_gray=True)
    img_rescale=rescale(image, 0.25)
    image1=np.reshape(img_rescale, img_rescale.shape[0]*img_rescale.shape[1])

    #appending the reshaped data to list named data.
    data.append(image1)
```

Preparing X_train and y_train



□ Preparing X_train

```
X_train=pd.DataFrame(data)
```

□ Preparing y_train

```
y_train=df_img.Name
```

Preparing testing data



```
#reading testing file
df_test=pd.read_csv('i:\\image_data\\test_image.csv')
data=[]
for i in range(6):
    #specify the physical path of your testing images
    path='I:\\image_data\\test\\'
    
    #specify name of the image file
    file=df_test.iloc[i].Image

#reading first image from training data
image=imread(path+file, as_gray=True)
img_rescale=rescale(image, 0.25)
image1=np.reshape(img_rescale, img_rescale.shape[0]*img_rescale.shape[1])
#appending the reshaped data to list named data.
data.append(image1)
```

Preparing X_test & y_test



- #preparing X_test

```
X_test=pd.DataFrame(data)
```

- Preparing y_test

```
y_test=df_test.Name
```

Applying GaussianNB



```
from sklearn.naive_bayes import GaussianNB  
  
nb=GaussianNB()  
  
nb.fit(X_train, y_train)  
  
nb.score(X_test, y_test)
```

Confusion matrix of GaussianNB



```
from sklearn.metrics import confusion_matrix  
  
nb_predict=nb.predict(X_test)  
  
conf_nb=confusion_matrix(y_test, nb_predict)  
  
conf_nb
```

Applying KNeighborsClassifier



```
from sklearn.neighbors import KNeighborsClassifier  
knn=KNeighborsClassifier()  
knn.fit(X_train, y_train)  
knn.score(X_test, y_test)
```

Confusion matrix of KNeighborsClassifier



```
knn_predict=knn.predict(X_test)  
conf_knn=confusion_matrix(y_test, knn_predict)  
conf_knn
```

Applying SVC() algorithm



```
from sklearn.svm import SVC  
  
svc=SVC()  
  
svc.fit(X_train,y_train)  
  
print(svc.score(X_test,y_test))
```

Deployment



- GUI application for this ML Model deployment

```
from tkinter import Tk,Label,filedialog,Canvas,Button
```

```
win=Tk()
```

```
#Create a Canvas that can fit our input image
```

```
canvas=Canvas(win, width=900, height=600)
```

```
#Create a Label for predicting (displaying) outcome to the user
```

```
l1=Label(win)
```



- #Define OpenImage function here before button is created

```
def OpenImage():
    path=filedialog.askopenfilename(initialdir="I:\\image_data\\",title="Select an Image")
    print(path)
    rows1=[]
    image_black=imread(path,as_gray=True)
    small_bi=rescale(image_black,0.25)
    X1=np.reshape(small_bi,small_bi.shape[0]*small_bi.shape[1])
    rows1.append(X1)
    new_image=pd.DataFrame(rows1)
    y_predict=nb.predict(new_image)
    print(y_predict)
```



```
b1=Button(win, text="Open Image", command=OpenImage)
```

```
#Organizing widget in our window
```

```
canvas.grid(row=0)
```

```
l1.grid(row=1,column=0)
```

```
b1.grid(row=2)
```

```
win.mainloop()
```

```
import PIL
import os
import os.path
from PIL import Image

#f=path of your image folder
#My sample images are stored in Image folder of e:

path='e:\\Image\\'

for file in os.listdir(path):
    f_img = path+"/"+file
    img = Image.open(f_img)
    #in resize, first specify desired width of the image and then
    #specify desired height in pixels.
    #making the resolution of each image to 568*824

    img = img.resize((568,824))
    img.save(f_img)
```

```
#Image classification

import pandas as pd

df_train=pd.read_csv(r'D:\Image_data\train_image.csv')
df_train.head()

   Image Name
0  1.jpg    GH
1  2.jpg    GH
2  3.jpg    GH
3  4.jpg    GH
4  5.jpg    GH

from skimage.io import imread, imshow
from skimage.transform import rescale
import numpy as np

#Setting the physical path of your training images

path='D:\\Image_data\\training\\'

#Specify name of the first training image file

file=df_train.Image[0]

print('Physical path of training image:', path)
print('Name of the first training image:', file)

Physical path of training image: D:\\Image_data\\training\
Name of the first training image: 1.jpg

#Reading first training image

img=imread(path+file, as_gray=True)
imshow(img)

<matplotlib.image.AxesImage at 0x1650871af0>

#Preparing Training Data

data=[]
path='D:\\Image_data\\training\\'

for i in range(0,33):
    file=df_train.Image[i]
```

```
image=imread(path+file, as_gray=True)
img_rescale=rescale(image, 0.25)
img_reshape=np.reshape(img_rescale,
img_rescale.shape[0]*img_rescale.shape[1])

data.append(img_reshape)
print(file)

C:\Users\hp\Anaconda3\lib\site-packages\skimage\transform\
_warps.py:23: UserWarning: The default multichannel argument (None) is
deprecated. Please specify either True or False explicitly.
multichannel will default to False starting with release 0.16.
  warn('The default multichannel argument (None) is deprecated.
Please '

1.jpg
2.jpg
3.jpg
4.jpg
5.jpg
6.jpg
7.jpg
8.jpg
9.jpg
10.jpg
11.jpg
12.jpg
13.jpg
14.jpg
15.jpg
16.jpg
17.jpg
18.jpg
19.jpg
20.jpg
21.jpg
22.jpg
23.jpg
24.jpg
25.jpg
26.jpg
27.jpg
28.jpg
29.jpg
30.jpg
31.jpg
32.jpg
33.jpg
```

```
print(img_rescale.shape[0])
print(img_rescale.shape[1])

print(img_rescale.shape[0]*img_rescale.shape[1])

260
146
37960

data[0]

array([0.43822131, 0.43798112, 0.44063865, ..., 0.12551414,
0.12622018,
       0.12596943])

len(data)

33

#Preparing X_train

X_train=pd.DataFrame(data)
X_train.to_csv('e:\\Training_Image.csv')

#Preparing y_train

y_train=df_train.Name
y_train.head()

0      GH
1      GH
2      GH
3      GH
4      GH
Name: Name, dtype: object

#Reading Testing data

df_test=pd.read_csv(r'D:\\Image_data\\test_image.csv')
df_test.head()

   Image  Name
0  19.jpg    GH
1  20.jpg    GH
2  21.jpg    GH
3  37.jpg   RR
4  38.jpg   RR

#Preparing Testing data

path='D:\\Image_data\\test\\'
```

```

data=[]
for i in range(0,6):
    file=df_test.Image[i]

    image=imread(path+file, as_gray=True)
    img_rescale=rescale(image, 0.25)
    img_reshape=np.reshape(img_rescale,
img_rescale.shape[0]*img_rescale.shape[1]))


    data.append(img_reshape)
    print(file)

```

```

19.jpg
20.jpg
21.jpg
37.jpg
38.jpg
39.jpg

```

```

C:\Users\hp\Anaconda3\lib\site-packages\skimage\transform\
_warps.py:23: UserWarning: The default multichannel argument (None) is
deprecated. Please specify either True or False explicitly.
multichannel will default to False starting with release 0.16.
  warn('The default multichannel argument (None) is deprecated.
Please '

```

#Preparing X_test

```

X_test=pd.DataFrame(data)
X_test.shape

(6, 37960)

```

#Preparing y_test

```

y_test=df_test.Name

```

#Now, we can train classification algo using X_train and y_train

```

from sklearn.tree import DecisionTreeClassifier

```

```

dtc=DecisionTreeClassifier()
dtc.fit(X_train, y_train)

```

```

DecisionTreeClassifier(class_weight=None, criterion='gini',
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=False,
        random_state=None, splitter='best')

#Applying RandomForestClassifier

from sklearn.ensemble import RandomForestClassifier

rfc=RandomForestClassifier()
rfc.fit(X_train, y_train)

C:\Users\hp\Anaconda3\lib\site-packages\sklearn\ensemble\
forest.py:245: FutureWarning: The default value of n_estimators will
change from 10 in version 0.20 to 100 in 0.22.
    "10 in version 0.20 to 100 in 0.22.", FutureWarning)

RandomForestClassifier(bootstrap=True, class_weight=None,
criterion='gini',
                      max_depth=None, max_features='auto',
max_leaf_nodes=None,
                      min_impurity_decrease=0.0,
min_impurity_split=None,
                      min_samples_leaf=1, min_samples_split=2,
                      min_weight_fraction_leaf=0.0, n_estimators=10,
n_jobs=None, oob_score=False,
random_state=None,
                      verbose=0, warm_start=False)

```

#Applying GaussianNB

```

from sklearn.naive_bayes import GaussianNB

gnb=GaussianNB()

gnb.fit(X_train, y_train)

GaussianNB(priors=None, var_smoothing=1e-09)

print('Score of DecisionTree Classifier:', dtc.score(X_test, y_test))
print('Score of RandomForest Classifier:', rfc.score(X_test, y_test))
print('Score of GaussianNB:', gnb.score(X_test, y_test))

```

```

Score of DecisionTree Classifier: 1.0
Score of RandomForest Classifier: 1.0
Score of GaussianNB: 1.0

```

#Confusion Matrix for DecisionTree Classifier

```
from sklearn.metrics import confusion_matrix
```

```

dtc_pred=dtc.predict(X_test)

conf_dtc=confusion_matrix(y_test, dtc_pred)

dtc_df=pd.DataFrame(conf_dtc, columns=['GH', 'RR'], index=['GH',
'RR'])
dtc_df

      GH  RR
GH    3   0
RR    0   3

#Confusion Matrix for RandomForest Classifier

rfc_pred=rfc.predict(X_test)

conf_rfc=confusion_matrix(y_test, rfc_pred)

rfc_df=pd.DataFrame(conf_rfc, columns=['GH', 'RR'], index=['GH',
'RR'])
rfc_df

      GH  RR
GH    3   0
RR    0   3

#Confusion Matrix for GaussianNB

gnb_pred=gnb.predict(X_test)

conf_gnb=confusion_matrix(y_test, gnb_pred)

gnb_df=pd.DataFrame(conf_gnb, columns=['GH', 'RR'], index=['GH',
'RR'])
gnb_df

      GH  RR
GH    3   0
RR    0   3

#Implementation

from PIL import Image
from PIL import ImageTk
from tkinter import Tk,Label,Button,Canvas,StringVar
from tkinter import filedialog
import tkinter

win = Tk() # Create instance

```

```

# Create a canvas that can fit the above image
canvas = Canvas(win, width = 800, height = 900)
canvas.grid(row=0)

labelText = StringVar()
tkinter.Label(win,fg='Red',font=("Helvetica", 40),textvariable=labelText).grid(row=1,column=0)

# Open an image file
def openImage():
    global photo
    rows2=[]
    fullfilename = filedialog.askopenfilename(initialdir="/",
    title="Select a file", filetypes=[("Image files", "*.jpg; *.png")]) # select png or jpg image file from the hard drive
    image =imread(fullfilename,as_gray=True)

    image1_rescaled = rescale(image, 0.25)
    x2=np.reshape(image1_rescaled,
    image1_rescaled.shape[0]*image1_rescaled.shape[1])
    rows2.append(x2)
    X_new = pd.DataFrame(rows2)
    y_new=gnb.predict(X_new)
    print(y_new)

# Use PIL (Pillow) to convert the NumPy ndarray to a PhotoImage
photo = Image.open(fullfilename)
#photo = photo.resize((800, 710),Image.ANTIALIAS)
#photo = photo.rotate(270)
photo = ImageTk.PhotoImage(photo)
canvas.create_image(20,40,image=photo)

message="This is looking like " + y_new[0]
labelText.set(message)

# img = PhotoImage(file=fullfilename)

action_pic = Button(win, text="Open Image", command=openImage, padx=2)
action_pic.grid(row=2)

win.mainloop()

['GH']

C:\Users\hp\Anaconda3\lib\site-packages\skimage\transform\_warps.py:23: UserWarning: The default multichannel argument (None) is

```

deprecated. Please specify either True or False explicitly.
multichannel will default to False starting with release 0.16.
warn('The default multichannel argument (None) is deprecated.
Please '

['RR']



- [Text Classification](#)
- [Applications of NLP](#)
- [Text transformation](#)
- [Step 1: Collect Data](#)
- [Step 2: Design the Vocabulary](#)
- [Step 3: Sparse Matrix](#)
- [Text Processing using Python](#)
- [References](#)

Text Classification



- Text classification also known as text tagging or text categorization is the process of categorizing text into organized groups. By using Natural Language Processing (NLP), text classifiers can automatically analyze text and then assign a set of pre-defined tags or categories based on its content.
- **Natural Language Processing**
- NLP stands for Natural Language Processing, which is a part of Computer Science, Human language, and Artificial Intelligence. It is the technology that is used by machines to understand, analyses, 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.

Applications of NLP



- **Question Answering**
- **Spam Detection**
- **Sentiment Analysis**
- **Machine Translation**
- **Speech Recognition**
- **Chatbot**

Text transformation



- A text transformation is a technique that is used to control the capitalization of the text.
- **Bag-of-Words Model**
- The bag-of-words model is used to represent text data for machine learning algorithms. The bag-of-words model is simple to understand and implement and has seen great success in language modeling and text classification.
- A bag-of-words is a representation of text that describes the occurrence of words within a document. It involves two things:
 - A vocabulary of known words.
 - A measure of the presence of known words.

Step 1: Collect Data



For example, An English man speaks some text to villagers:

- Good Job Done
- Very Good Job Done
- Bad Job Done
- Very Bad Job Done
- Bad Job

Step 2: Design the Vocabulary



- Now we can make a list of all of the words in our model vocabulary. The unique words here (ignoring case and punctuation) are:

- Good
- Job
- Done
- Very
- Bad

Step 3: Sparse Matrix



- Matrices that contain mostly zero values are called sparse matrix. The sparse matrix of above example can be constructed as:

Dictionary	Good	Job	Done	Very	Bad
Good Job Done	1	1	1	0	0
Very Good Job Done	1	1	1	1	0
Bad Job Done	0	1	1	0	1
Very Bad Job Done	0	1	1	1	1
Bad Job	0	1	0	0	1



- The above matrix can be represented as sparse matrix:

Row/Column index	0	1	2	3	4
0	1	1	1	0	0
1	1	1	1	1	0
2	0	1	1	0	1
3	0	1	1	1	1
4	0	1	0	0	1



□ **CountVectorizer():**

The CountVectorizer() function provides a simple way to both tokenize a collection of text documents and build a vocabulary of known words. It also used to encode new documents using that vocabulary.

Sample Text

```
simple_text=["Good Job Done", "Very Good Job Done", "Bad Job Done", "Very Bad Job Done",  
"Bad Job"]
```



□ **Building the vocabulary using CountVectorizer()**

```
from sklearn.feature_extraction.text import CountVectorizer
```

```
vect=CountVectorizer()
```

#Fit : Learn the "Dictionary" of the data provided.

```
vect.fit(simple_text)
```

□

□ **To see the dictionary made from data**

□ **get_feature_names() - Returns a list of feature names, ordered by their indices.**

```
vect.get_feature_names()
```



□ **To prepare data matrix i.e. index having the value 1**

```
data_matrix=vect.transform(simple_text)
```

```
print(data_matrix)
```

□ **To prepare dense matrix**

```
dense_matrix=data_matrix.toarray()
```

```
print(dense_matrix)
```



- **Converting the transformed data into dataframe**

```
import pandas as pd

df=pd.DataFrame(data_matrix.toarray(),columns=vect.get_feature_names())

print(df)
```

- **Note: These operations must be performed on the text data on which you want to perform Machine Learning. Now your training data is ready.**

```
#Text Processing
```

```
sample_text=['Good Job Done', 'Very Good Job Done', 'Bad Job Done',
'Very Bad Job Done', 'Bad Job']
sample_text

['Good Job Done',
 'Very Good Job Done',
 'Bad Job Done',
 'Very Bad Job Done',
 'Bad Job']

#CountVectorizer(): tokenize a collection of text document/sentences
and builds a vocabulary of known
#words

from sklearn.feature_extraction.text import CountVectorizer

vect=CountVectorizer()

#fitting the sample text on CountVectorizer()

#fit() will identify the unique words which are available in
sample_text and arange them in
#ascending order

vect.fit(sample_text)

CountVectorizer(analyzer='word', binary=False, decode_error='strict',
               dtype=<class 'numpy.int64'>, encoding='utf-8',
               input='content',
               lowercase=True, max_df=1.0, max_features=None,
               min_df=1,
               ngram_range=(1, 1), preprocessor=None,
               stop_words=None,
               strip_accents=None, token_pattern='(\\w+\\b)\\w+\\b',
               tokenizer=None, vocabulary=None)

#To see the dictionary of known words i.e. unique words in sample_text
vect.get_feature_names()

['bad', 'done', 'good', 'job', 'very']

#Preparing sparse matrix.
#transform() will make sparse matrix using unique words of sample_text

sparse=vect.transform(sample_text)
print(sparse)

(0, 1)    1
(0, 2)    1
```

```
(0, 3)    1
(1, 1)    1
(1, 2)    1
(1, 3)    1
(1, 4)    1
(2, 0)    1
(2, 1)    1
(2, 3)    1
(3, 0)    1
(3, 1)    1
(3, 3)    1
(3, 4)    1
(4, 0)    1
(4, 3)    1
```

#Preparing dense matrix using sparse matrix

```
dense=sparse.toarray()
dense

array([[0, 1, 1, 1, 0],
       [0, 1, 1, 1, 1],
       [1, 1, 0, 1, 0],
       [1, 1, 0, 1, 1],
       [1, 0, 0, 1, 0]], dtype=int64)
```

#Transforming the dense matrix into data frame

```
import pandas as pd
```

```
df_text=pd.DataFrame(dense, columns=vect.get_feature_names(),
index=sample_text)
df_text
```

	bad	done	good	job	very
Good Job Done	0	1	1	1	0
Very Good Job Done	0	1	1	1	1
Bad Job Done	1	1	0	1	0
Very Bad Job Done	1	1	0	1	1
Bad Job	1	0	0	1	0



- [Text Classification Problem: Spam Detection \(Mini Project 3\)](#)
- [Data Preparation](#)
- [Making dictionary of message for processing](#)
- [Prepare X_train, y_train and X_test, y_test](#)
- [Applying MultinomialNB Model](#)
- [Applying Logistic Regression](#)
- [Confusion matrix of MultinomialNB](#)
- [References](#)

Text Classification Problem: Spam Detection (Mini Project 3)



- **Problem Statement: Classify the incoming email as ham (good) or spam (bad)**
- **Reading sms.txt file**

```
import pandas as pd  
  
sms=pd.read_csv("sms.txt",header=None,names=['label','message'],sep='\t')
```

- **To view data**

sms.head()
- **To view entries in label column**

sms.label.value_counts()

Data Preparation



□ Preparing X and y

X=sms.message

y=sms.label

Making dictionary of message for processing



```
from sklearn.feature_extraction.text import CountVectorizer  
vect=CountVectorizer()  
vect.fit(X)      #making dictionary  
X_matrix=vect.transform(X)
```

Prepare X_train, y_train and X_test, y_test



```
from sklearn.model_selection import train_test_split  
  
X_train, X_test, y_train, y_test=train_test_split(X_matrix, y, test_size=0.3)
```

Applying MultinomialNB Model on X_train_matrix and y_train



```
from sklearn.naive_bayes import MultinomialNB  
  
mb=MultinomialNB()  
  
mb.fit(X_train, y_train)  
  
mb.score(X_test, y_test)
```

Applying Logistic Regression



```
from sklearn.linear_model import LogisticRegression  
  
lr=LogisticRegression()  
  
lr.fit(X_train_matrix,y_train)  
  
print(lr.score(X_test_matrix,y_test))
```

Confusion matrix of MultinomialNB



```
from sklearn.metrics import confusion_matrix  
  
mb_predict=mb.predict(X_test)  
  
conf_mb=confusion_matrix(y_test, mb_predict)  
  
conf_df=pd.DataFrame(conf_mb, columns=['Ham', 'Spam'])  
  
conf_df
```

#Text classification

#Spam Detection Problem

```
import pandas as pd
```

```
sms=pd.read_csv('sms.txt', names=['Label', 'Message'], sep='\t', header=None)
sms.head()
```

Label	Message
0 ham	Go until jurong point, crazy.. Available only ...
1 ham	Ok lar... Joking wif u oni...
2 spam	Free entry in 2 a wkly comp to win FA Cup fina...
3 ham	U dun say so early hor... U c already then say...
4 ham	Nah I don't think he goes to usf, he lives aro...

#Prepare X and y

X=sms.Message
y=sms.Label

#Preparing X for Machine Learning

```
from sklearn.feature_extraction.text import CountVectorizer
```

```
v=CountVectorizer()
```

v.fit(X)

```
X sparse=v.transform(X)
```

X sparse.shape

(5572, 8713)

#Converting X sparse to dataframe

```
df_text=pd.DataFrame(X_sparse.toarray(),  
columns=v.get_feature_names())  
df_text.head()
```

[5 rows x 8713 columns]

*#Now, our text data is ready for Machine Learning
#We can apply any classification algorithm on this data*

#Splitting the data in training and testing set

```
from sklearn.model_selection import train_test_split
```

```
X_train, X_test, y_train, y_test=train_test_split(df_text, y, test_size=0.2)
```

#Applying DecisionTreeClassifier

```
from sklearn.tree import DecisionTreeClassifier
```

```
dtc=DecisionTreeClassifier()  
dtc.fit(X_train, y_train)
```

```
DecisionTreeClassifier(class_weight=None, criterion='gini',  
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=False,
        random_state=None, splitter='best')

#Applying RandomForestClassifier

from sklearn.ensemble import RandomForestClassifier
rfc=RandomForestClassifier()

rfc.fit(X_train, y_train)

C:\Users\hp\Anaconda3\lib\site-packages\sklearn\ensemble\
forest.py:245: FutureWarning: The default value of n_estimators will
change from 10 in version 0.20 to 100 in 0.22.
    "10 in version 0.20 to 100 in 0.22.", FutureWarning)

RandomForestClassifier(bootstrap=True, class_weight=None,
criterion='gini',
                      max_depth=None, max_features='auto',
max_leaf_nodes=None,
                      min_impurity_decrease=0.0,
min_impurity_split=None,
                      min_samples_leaf=1, min_samples_split=2,
                      min_weight_fraction_leaf=0.0, n_estimators=10,
n_jobs=None, oob_score=False,
random_state=None,
                      verbose=0, warm_start=False)

```

#Applying GaussianNB

```

from sklearn.naive_bayes import GaussianNB

gnb=GaussianNB()
gnb.fit(X_train, y_train)

GaussianNB(priors=None, var_smoothing=1e-09)

```

#Applying MultinomialNB

```

from sklearn.naive_bayes import MultinomialNB

mb=MultinomialNB()
mb.fit(X_train, y_train)

MultinomialNB(alpha=1.0, class_prior=None, fit_prior=True)

print('Score of DecisionTreeClassifier=', dtc.score(X_test, y_test))
print('Score of RandomForestClassifier=', rfc.score(X_test, y_test))
print('Score of GaussianNB=', gnb.score(X_test, y_test))
print('Score of MultinomialNB=', mb.score(X_test, y_test))

```

```

Score of DecisionTreeClassifier= 0.968609865470852
Score of RandomForestClassifier= 0.97847533632287

```

```

Score of GaussianNB= 0.8869955156950673
Score of MultinomialNB= 0.9856502242152466

y.value_counts()

ham      4825
spam     747
Name: Label, dtype: int64

y_test.value_counts()

ham      971
spam    144
Name: Label, dtype: int64

#Preparing confusion matrix on MultinomialNB

from sklearn.metrics import confusion_matrix

mb_pred=mb.predict(X_test)

conf_mb=confusion_matrix(y_test, mb_pred)
conf_mb_df=pd.DataFrame(conf_mb, columns=['Ham', 'Spam'],
index=['Ham', 'Spam'])
conf_mb_df

      Ham   Spam
Ham    958    13
Spam     3   141

import numpy as np
print('% of correct prediction in ham class:',
np.around((958/971)*100, 2), '%')

print('% of incorrect prediction in ham class:',
np.around((13/971)*100, 2), '%')

% of correct prediction in ham class: 98.66 %
% of incorrect prediction in ham class: 1.34 %

print('% of correct prediction in spam class:',
np.around((141/144)*100, 2), '%')
print('% of incorrect prediction in spam class:',
np.around((3/144)*100, 2), '%')

% of correct prediction in spam class: 97.92 %
% of incorrect prediction in spam class: 2.08 %

#Preparing confusion matrix of RandomForestClassifier

rfc_pred=rfc.predict(X_test)

```

```

conf_rfc=confusion_matrix(y_test, rfc_pred)
conf_rfc_df=pd.DataFrame(conf_rfc, columns=['Ham', 'Spam'],
index=['Ham', 'Spam'])
conf_rfc_df

      Ham  Spam
Ham    970     1
Spam    23    121

print('% of correct prediction in ham class:',
np.around((970/971)*100, 2), '%')
print('% of incorrect prediction in ham class:',
np.around((1/971)*100, 2), '%')

% of correct prediction in ham class: 99.9 %
% of incorrect prediction in ham class: 0.1 %

print('% of correct prediction in spam class:',
np.around((121/144)*100, 2), '%')
print('% of incorrect prediction in spam class:',
np.around((23/144)*100, 2), '%')

% of correct prediction in spam class: 84.03 %
% of incorrect prediction in spam class: 15.97 %

#Since the accuracy of MultinomialNB is best for 'Ham' class and
'Spam' class is best and overall
accuracy is also highest. So, we can implement this problem using
MultinomialNB model

#Implementing the MultinomialNB model on new message

text=[input('Enter Your Message:')]
text1=v.transform(text)

mb_pr=mb.predict(text1)
print('Message is:', mb_pr[0])

Enter Your Message:Click on following link to win the 50000000 cash.
Message is: spam

```



- [Text classification Improvement](#)
- [Stop Words](#)
- [ngram_range](#)
- [Remove rare words from the text](#)
- [Remove frequently used words](#)
- [Improving text classification problem](#)
- [Prepare X and y](#)
- [Process X for Machine Learning with improvement](#)
- [Prepare X_train, y_train and X_test, y_test](#)
- [Applying MultinomialNB](#)
- [Confusion Matrix](#)
- [Implementation](#)
- [References](#)

Text classification Improvement



```
sample_text=['Good Job Done By You', 'Very Good Job Done By You',  
           'Bad Job Done', 'Very Bad Job Done', 'Bad Job']
```

```
sample_text
```

Stop Words



- These are the words which may not be playing any role with respect to classification as ham or spam. Examples- the, is, have, has, would, in, an etc.
- Therefore stop words should be removed from the features.

```
import pandas as pd

from sklearn.feature_extraction.text import CountVectorizer

vect=CountVectorizer(stop_words='english')

vect.fit(sample_text)

data_matrix=vect.transform(sample_text)

df_dense=pd.DataFrame(data_matrix.toarray(), columns=vect.get_feature_names())

df_dense
```

ngram_range



- **Specifying the ngram_range in CountVectorizer():** To use combination of two words in sparse matrix.
- ngram_range: The lower and upper boundary of the range of n-values for different word n-grams or char n-grams to be extracted.
- It means including combination of two, three or four words in dense/sparse matrix
- How to differentiate between, 'happy', 'not happy' and 'very happy'
- Adjective: not and very
- Include 1-gram (only one word)
- Include 2-gram (combination of two adjacent words)
- Include 3-gram (combination of three adjacent words)



```
vect=CountVectorizer(ngram_range=(1,2))
```

```
vect.fit(sample_text)
```

```
data_matrix12=vect.transform(sample_text)
```

```
df_dense12=pd.DataFrame(data_matrix12.toarray(), columns=vect.get_feature_names())
```

```
df_dense12
```

Remove rare words from the text



- Remove rare words used in the messages/text
- Only keep those words which appear atleast in 2 messages/sentences/documents

```
vect=CountVectorizer(min_df=2)
```

```
vect.fit(sample_text)
```

```
data_matrix12=vect.transform(sample_text)
```

```
df_dense12=pd.DataFrame(data_matrix12.toarray(), columns=vect.get_feature_names())
```

```
df_dense12
```

Remove frequently used words



- Removing words that appears in more than 50% of the sentences/messages.
- It helps in removing address, name kind of things
- Removing most frequent words

```
vect=CountVectorizer(max_df=0.5)
```

```
vect.fit(sample_text)
```

```
data_matrix12=vect.transform(sample_text)
```

```
df_dense12=pd.DataFrame(data_matrix12.toarray(), columns=vect.get_feature_names())
```

```
df_dense12
```

Improving text classification problem



- #text classification
- #reading sms.txt file

```
import pandas as pd
```

```
sms=pd.read_csv('sms.txt', header=None, names=['Label', 'Message'], sep='\t')
```

```
sms.head()
```

Prepare X and y



X=sms.Message

y=sms.Label

Process X for Machine Learning with improvement



```
from sklearn.feature_extraction.text import CountVectorizer  
  
vect1=CountVectorizer(stop_words='english', ngram_range=(1,2), min_df=2, max_df=0.5)  
  
vect1.fit(X)  
  
X_matrix=vect1.transform(X)
```

Prepare X_train, y_train and X_test, y_test



```
from sklearn.model_selection import train_test_split  
  
X_train, X_test, y_train, y_test=train_test_split(X_matrix, y, test_size=0.3)
```

Applying MultinomialNB



```
from sklearn.naive_bayes import MultinomialNB  
  
mb=MultinomialNB()  
  
mb.fit(X_train, y_train)  
  
mb.score(X_test, y_test)
```

Confusion Matrix



```
#confusion matrix
```

```
from sklearn.metrics import confusion_matrix
```

```
mb_predict=mb.predict(X_test)
```

```
conf_mb=confusion_matrix(y_test, mb_predict)
```

```
conf_df=pd.DataFrame(conf_mb, columns=['Ham', 'Spam'])
```

```
conf_df
```



```
text=[input('Enter Message:')]  
  
text1=vect1.transform(text)  
  
predict=mb.predict(text1)  
  
print('Message is:', predict[0])
```

#Text Classification Improvement

```
sample=['Good Job Done at office', 'Very Good Job Done by you', 'Bad Job Done by You', 'Very Bad Job Done', 'Have Bad Job']
sample
['Good Job Done at office',
 'Very Good Job Done by you',
 'Bad Job Done by You',
 'Very Bad Job Done',
 'Have Bad Job']
```

#Remove stop words of english language from the sentences/text document

#Stop words: the, is , at, would, have, are etc.

#These are words which may not play any role in text classification

```
from sklearn.feature_extraction.text import CountVectorizer
```

```
v1=CountVectorizer(stop_words='english')
```

```
v1.fit(sample)
```

```
sparse=v1.transform(sample)
```

```
v1.get_feature_names()
```

```
['bad', 'good', 'job', 'office']
```

```
import pandas as pd
```

```
df=pd.DataFrame(sparse.toarray(), columns=v1.get_feature_names())
df
```

	bad	good	job	office
0	0	1	1	1
1	0	1	1	0
2	1	0	1	0
3	1	0	1	0
4	1	0	1	0

#Include N-gram words

#How to differentiate between 'Happy', 'Not Happy' and 'Very Happy'

#ngram_range=(1,2), will include 1-word and 2-word combination in dense/sparse matrix

#ngram_range=(1,2,3), will include 1-word , 2-word and 3-word

combination in dense/sparse matrix

```
v2=CountVectorizer(ngram_range=(1,2))

v2.fit(sample)

sp1=v2.transform(sample)

df1=pd.DataFrame(sp1.toarray(), columns=v2.get_feature_names())
df1

      at   at office   bad   bad job   by   by you   done   done at   done by
good \
0      1           1     0           0     0           0     1           1     0
1
1      0           0     0           0     1           1     1           0     1
1
2      0           0     1           1     1           1     1           0     1
0
3      0           0     1           1     0           0     1           0     0
0
4      0           0     1           1     0           0     0           0     0
0

      good job   have   have bad   job   job done   office   very   very bad
very good \
0      1     0           0     1           1     1           1     0           0
0
1      1     0           0     1           1     0           0     1           0
1
2      0     0           0     1           1     0           0     0           0
0
3      0     0           0     1           1     0           0     1           1
0
4      0     1           1     1           0     0           0     0           0
0

      you
0      0
1      1
2      1
3      0
4      0
```

#Remove those words which are used rarely in your text document/sentences

#Ex: Only keep those words which appear in atleast 2 text document/sentences

```

v3=CountVectorizer(min_df=2)

v3.fit(sample)

sp2=v3.transform(sample)

df3=pd.DataFrame(sp2.toarray(), columns=v3.get_feature_names())
df3

   bad  by  done  good  job  very  you
0    0    0     1     1     1      0     0
1    0    1     1     1     1      1     1
2    1    1     1     0     1      0     1
3    1    0     1     0     1      1     0
4    1    0     0     0     1      0     0

#Remove most frequent/common words from your text document/sentences

#Ex: removing words which are appearing in more than 50% of the text document/sentences/messages
#It helps in removing name, address kind of things

v4=CountVectorizer(max_df=0.5)
v4.fit(sample)

sm3=v4.transform(sample)
df4=pd.DataFrame(sm3.toarray(), columns=v4.get_feature_names())
df4

   at  by  good  have  office  very  you
0   1    0     1     0      1      0     0
1   0    1     1     0      0      1     1
2   0    1     0     0      0      0     1
3   0    0     0     0      0      1     0
4   0    0     0     1      0      0     0

#Now, applying these modifications to spam detection problem

#Text classification

#Spam Detection Problem

import pandas as pd

sms=pd.read_csv('sms.txt', names=['Label', 'Message'], sep='\t',
header=None)
sms.head()

  Label                               Message
0  ham  Go until jurong point, crazy.. Available only ...

```

```
1 ham Ok lar... Joking wif u oni...
2 spam Free entry in 2 a wkly comp to win FA Cup fina...
3 ham U dun say so early hor... U c already then say...
4 ham Nah I don't think he goes to usf, he lives aro...
```

#Prepare X and y

```
X=sms.Message  
y=sms.Label
```

#Preparing X for Machine Learning

```
from sklearn.feature_extraction.text import CountVectorizer  
  
v=CountVectorizer(stop_words='english', ngram_range=(1,2), min_df=2,  
max_df=0.5)  
  
v.fit(X)  
X_sparse=v.transform(X)  
  
X_sparse.shape  
(5572, 9207)
```

#Converting X_sparse to dataframe

```
df_text=pd.DataFrame(X_sparse.toarray(),  
columns=v.get_feature_names())  
df_text.head()
```

```
    00 00 sub 00 subs 000 000 bonus 000 cash 000 homeowners 000  
pounds \
0 0 0 0 0 0 0 0 0 0 0 0
0
1 0 0 0 0 0 0 0 0 0 0 0
0
2 0 0 0 0 0 0 0 0 0 0 0
0
3 0 0 0 0 0 0 0 0 0 0 0
0
4 0 0 0 0 0 0 0 0 0 0 0
0
0
000 prize 000 xmas ... yuo raed yup yup having yup lor yup
ok \
0 0 0 ... 0 0 0 0 0 0 0
0
1 0 0 ... 0 0 0 0 0 0 0
0
2 0 0 ... 0 0 0 0 0 0 0
0
```

```

3      0      0  ...
0      0      0  ...
4      0      0  ...
0

    yup thk  zed  zed 08701417012  zed 08701417012150p  zoe
0      0      0      0      0      0      0      0
1      0      0      0      0      0      0      0
2      0      0      0      0      0      0      0
3      0      0      0      0      0      0      0
4      0      0      0      0      0      0      0

[5 rows x 9207 columns]

#Splitting the data in training and testing set

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test=train_test_split(df_text, y,
test_size=0.2)

#Applying DecisionTreeClassifier

from sklearn.tree import DecisionTreeClassifier

dtc=DecisionTreeClassifier()
dtc.fit(X_train, y_train)

DecisionTreeClassifier(class_weight=None, criterion='gini',
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=False,
           random_state=None, splitter='best')

#Applying RandomForestClassifier

from sklearn.ensemble import RandomForestClassifier
rfc=RandomForestClassifier()

rfc.fit(X_train, y_train)

C:\Users\hp\Anaconda3\lib\site-packages\sklearn\ensemble\
forest.py:245: FutureWarning: The default value of n_estimators will
change from 10 in version 0.20 to 100 in 0.22.
  "10 in version 0.20 to 100 in 0.22.", FutureWarning)

RandomForestClassifier(bootstrap=True, class_weight=None,
criterion='gini',

```

```

        max_depth=None, max_features='auto',
max_leaf_nodes=None, min_impurity_decrease=0.0,
min_impurity_split=None, min_samples_leaf=1, min_samples_split=2,
min_weight_fraction_leaf=0.0, n_estimators=10,
n_jobs=None, oob_score=False,
random_state=None, verbose=0, warm_start=False)

```

#Applying MultinomialNB

```

from sklearn.naive_bayes import MultinomialNB

mb=MultinomialNB()
mb.fit(X_train, y_train)

MultinomialNB(alpha=1.0, class_prior=None, fit_prior=True)

print('Score of DecisionTreeClassifier=', dtc.score(X_test, y_test))
print('Score of RandomForestClassifier=', rfc.score(X_test, y_test))
print('Score of MultinomialNB=', mb.score(X_test, y_test))

Score of DecisionTreeClassifier= 0.9587443946188341
Score of RandomForestClassifier= 0.9695067264573991
Score of MultinomialNB= 0.9811659192825112

```

#Preparing confusion matrix on MultinomialNB

```

from sklearn.metrics import confusion_matrix

mb_pred=mb.predict(X_test)

conf_mb=confusion_matrix(y_test, mb_pred)
conf_mb_df=pd.DataFrame(conf_mb, columns=[ 'Ham', 'Spam'],
index=[ 'Ham', 'Spam'])
conf_mb_df

      Ham  Spam
Ham    973     6
Spam    15   121

```

```

#Twitter Data Analyisis

import pandas as pd

df=pd.read_csv(r'D:\BigData_All_content\AI Machine Learning Content\Content\Day 22_Twitter Data\train_twitter_less.csv')
df.head()

      label                      tweet
0      0  @userAbc123 when a father is dysfunctional an...
1      0  @user @user thanks for #lyft credit i can't us...
2      0                           bihday your majesty
3      0  #model    i love u take with u all the time in ...
4      0           factsguide: society now    #motivation

#Import required libraries

# import re (regular expression)
import re

#Sample text
t= "@userAbc123 When a Father is DYSfunctional and is so selfish he
drags his kids into his 989835 &$% dysfunction.#run"
t

'@userAbc123 When a Father is DYSfunctional and is so selfish he drags
his kids into his 989835 &$% dysfunction.#run'

#Processing the tweets for sentiment analysis/classification

#1.Remove user name from the tweet. user name is given using @ in your
tweet

#replace all the characters (a-z, A-Z, 0-9, or any other symbol) after
@ symbol with space

t1=re.sub('@[\w]*', ' ', t)
t1

' When a Father is DYSfunctional and is so selfish he drags his kids
into his 989835 &$% dysfunction.#run'

#2. Replace all the characters other than (a-z, A-Z and #)
t1=re.sub('[^a-zA-Z#]', ' ', t1)
t1

' When a Father is DYSfunctional and is so selfish he drags his kids
into his         dysfunction #run'

#Convert the tweet in lower case
t1=t1.lower()
t1

```

```

' when a father is dysfunctional and is so selfish he drags his kids
into his          dysfunction #run'

df.shape
(9998, 2)

processed_tweet=[]

for i in range(9998):
    #1.Replace all the characters (a-z, A-Z, 0-9, or any other symbol)
after @ symbol with space

    t1=re.sub('@[\w]*', ' ', df.tweet[i])

    #2. Replace all the characters other than (a-z, A-Z and #)
    t1=re.sub('[^a-zA-Z#]', ' ', t1)

    #3.Convert the tweet in lower case
    t1=t1.lower()
    processed_tweet.append(t1)

processed_tweet[0:2]

[' when a father is dysfunctional and is so selfish he drags his kids
into his dysfunction      #run',
 ' thanks for #lyft credit i can t use cause they don t offer
wheelchair vans in pdx      #disapointed #getthanked']

#Preparing X

X=pd.DataFrame(processed_tweet, columns=['Processed_Tweet'])
X.head()

          Processed_Tweet
0  when a father is dysfunctional and is so sel...
1  thanks for #lyft credit i can t use cause th...
2                      bihday your majesty
3  #model   i love u take with u all the time in ...
4            factsguide society now      #motivation

#Prapare y
y=df.label

y.head()

0    0
1    0
2    0
3    0
4    0
Name: label, dtype: int64

```

```
#Creating dictionary of X
```

```
from sklearn.feature_extraction.text import CountVectorizer  
cv=CountVectorizer(stop_words='english', ngram_range=(1,2), min_df=2,  
max_df=0.5)
```

```
X_matrix=cv.fit_transform(X.Processed_Tweet)
```

```
#Now, convert the matrix into dataframe
```

```
X1=pd.DataFrame(X_matrix.toarray(), columns=cv.get_feature_names())  
X1.head()
```

```
aa aap aap claim aap spokesperson aarhus abandoned abba abc  
able \
```

```
0 0 0 0 0 0 0 0 0 0 0  
0  
1 0 0 0 0 0 0 0 0 0 0  
0  
2 0 0 0 0 0 0 0 0 0 0  
0  
3 0 0 0 0 0 0 0 0 0 0  
0  
4 0 0 0 0 0 0 0 0 0 0  
0
```

```
able learn ... zealand zedd zelda zen zero zimbabwe zone \  
0 0 ... 0 0 0 0 0 0 0 0  
1 0 ... 0 0 0 0 0 0 0 0  
2 0 ... 0 0 0 0 0 0 0 0  
3 0 ... 0 0 0 0 0 0 0 0  
4 0 ... 0 0 0 0 0 0 0 0
```

```
zone canada zoo zucchini  
0 0 0 0  
1 0 0 0  
2 0 0 0  
3 0 0 0  
4 0 0 0
```

```
[5 rows x 10262 columns]
```

```
#Splitting the data into training set and testing set
```

```
from sklearn.model_selection import train_test_split
```

```
X_train, X_test, y_train, y_test=train_test_split(X1, y,  
test_size=0.2)
```

#Applying MultinomialNB

```
from sklearn.naive_bayes import MultinomialNB  
mb=MultinomialNB()  
mb.fit(X_train, y_train)  
  
MultinomialNB(alpha=1.0, class_prior=None, fit_prior=True)
```

#Applying DecisionTreeClassifier

```
from sklearn.tree import DecisionTreeClassifier  
  
dtc=DecisionTreeClassifier()  
dtc.fit(X_train, y_train)  
  
DecisionTreeClassifier(class_weight=None, criterion='gini',  
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=False,  
          random_state=None, splitter='best')
```

#Applying RandomForestClassifier

```
from sklearn.ensemble import RandomForestClassifier  
  
rfc=RandomForestClassifier()  
rfc.fit(X_train, y_train)  
  
C:\Users\hp\Anaconda3\lib\site-packages\sklearn\ensemble\  
forest.py:245: FutureWarning: The default value of n_estimators will  
change from 10 in version 0.20 to 100 in 0.22.  
    "10 in version 0.20 to 100 in 0.22.", FutureWarning)  
  
RandomForestClassifier(bootstrap=True, class_weight=None,  
criterion='gini',  
          max_depth=None, max_features='auto',  
max_leaf_nodes=None,  
          min_impurity_decrease=0.0,  
min_impurity_split=None,  
          min_samples_leaf=1, min_samples_split=2,  
          min_weight_fraction_leaf=0.0, n_estimators=10,  
          n_jobs=None, oob_score=False,  
random_state=None,  
          verbose=0, warm_start=False)  
  
print('Score of MultinomialNB:', mb.score(X_test, y_test))  
print('Score of DecisionTreeClassifier:', dtc.score(X_test, y_test))  
print('Score of RandomForestClassifier:', rfc.score(X_test, y_test))
```

```
Score of MultinomialNB: 0.933
Score of DecisionTreeClassifier: 0.929
Score of RandomForestClassifier: 0.948
```

#Confusion matrix

```
from sklearn.metrics import confusion_matrix

rfc_conf=confusion_matrix(y_test, rfc.predict(X_test))
rfc_df=pd.DataFrame(rfc_conf, columns=['0', '1'], index=['0', '1'])
rfc_df

          0   1
0  1839  27
1    77  57

mb_conf=confusion_matrix(y_test, mb.predict(X_test))
mb_df=pd.DataFrame(mb_conf, columns=['0', '1'], index=['0', '1'])
mb_df

          0   1
0  1790  76
1    58  76
```

#Applying KNeighborsClassifier

```
from sklearn.neighbors import KNeighborsClassifier

knc=KNeighborsClassifier()
knc.fit(X_train, y_train)
knc.score(X_test, y_test)

0.942

knc_conf=confusion_matrix(y_test, knc.predict(X_test))
knc_df=pd.DataFrame(knc_conf, columns=['0', '1'], index=['0', '1'])
knc_df

          0   1
0  1864  2
1   114  20
```



- [Mathematics behind Regression Problem](#)
- [Graphical Visualisation](#)
- [Observation](#)
- [Coefficient and Intercept](#)
- [To view the Actual outcome and Predicted outcome](#)
- [Predicting the outcome](#)
- [Applying Admission Prediction Problem](#)
- [References](#)

Mathematics behind Regression Problem



- Suppose there is only one column in the dataset i.e. GRE Score
- On which outcome 'Chance of Admit' depends

`X1=df['GRE Score']`

`y1=df['Chance of Admit']`

- Now, make a ML model on these two data only



- If you work on only one column, then it is treated as series
- ML algorithm takes features (X) as dataframe and outcome (y) as series
- We have to convert the series into DataFrame

```
X1=pd.DataFrame(X1, columns=['GRE Score'])
```

```
print(type(X1))
```

- Now X1 and y1 is ready for ML



- splitting the X1 and y1

```
X1_train, X1_test, y1_train, y1_test=train_test_split(X1,y1, test_size=0.25)
```

- Applying LinearRegression algorithm

```
lr1=LinearRegression()
```

```
lr1.fit(X1_train, y1_train)
```

```
lr1.score(X1_train, y1_train)
```

Graphical Visualisation



- ❑ #creating scatter plot to compare the value of 'GRE Score' and 'Chance of Admit'

```
import matplotlib.pyplot as plt  
  
plt.scatter(df['GRE Score'], df['Chance of Admit'])  
  
plt.xlabel("GRE Score")  
  
plt.ylabel("Chance of Admit")  
  
plt.title("GRE Score Vs Chance of Admit")  
  
plt.show()
```

Observation



- Line equation: $y=mx+c$
- where m=slope, c=intercept
- Now, if you know the values of m (slope) and intercept (c), then you can know the value of y for specific x
- Similarly, We can write in ML: $y=\text{beta1}*\text{x1}+\text{alpha}$

- $\text{beta1}=\text{coefficient}$
- $\text{alpha}=\text{intercept}$

Coefficient and Intercept



- ML model will try to find the optimal value of coefficient and intercept to maintain minimum error on the basis of training data
- If we know the value of coefficient (β_1) and intercept (α), then you can find the value of y for any specific X_1
- basically, LinearRegression algorithm will try to find out the value of coefficient and intercept, so that the error is minimum on the basis training data.



- For this problem, X1=GRE Score, y1=Chance of Admit, LinearRegression will write the equation as
- $y1=\text{beta1} \cdot X1 + \alpha$
- Chance of Admit= $\text{beta1} \cdot \text{GRE Score} + \alpha$

- Now, If we know the value of beta1 and alpha
- Then, we can find the value of Chance of Admit for a specific GRE Score.

Coefficient and Intercept



- During fit() operation, LinearRegression algorithm tries to find out the optimal values of coefficient and intercept.
- Coefficient is obtained/calculated for every feature provided in training data.
- Intercept is calculated only once.
- we can access the value of coefficient and intercept for LinearRegression algo

```
print('Coefficient of GRE Score:', lr1.coef_)
```

```
print('Intercept:', lr1.intercept_)
```

Here, lr1 is the object of LinearRegression algorithm

To view the Actual outcome and Predicted outcome



```
Data=pd.DataFrame({'GRE Score':X1_test['GRE Score'], 'Actual Y1':y1_test,  
'Predicted Y1':predict_lr1})
```

```
Data.head()
```

Predicting the outcome



- Coefficient of GRE Score=0.00987647
- Intercept=-2.4001138760209253
- Predict the value of Chance of Admit, When GRE Score=312

$\text{lr1_pr}=0.00987647*312+(-2.4001138760209253)$

lr1_pr

- Predict the value of Chance of Admit, When GRE Score=314

$\text{lr1_pr}=0.00987647*314+(-2.4001138760209253)$

lr1_pr

Applying Admission Prediction Problem



- Fitting the LinearRegression on X_train and y_train

#X_train will have all the features from the dataset

```
lr.fit(X_train, y_train)
```

- In lr object, we have passed all the columns of X_train
- Thats why, lr will produce coefficient for each feature (column) of X_train

```
print(lr.coef_)
```

```
print(lr.intercept_)
```



- [Mathematics Behind Classification Problem](#)
- [Decision Tree](#)
- [Decision Tree Classifier](#)
- [Tree split](#)
- [Decision Tree Algorithm](#)
- [Class Imbalance](#)
- [Gini value](#)
- [Entropy value](#)
- [Decision Tree: Algorithm overview](#)
- [Decision Tree Prediction](#)
- [Performance Metric](#)
- [Parameters & Hyperparameters](#)
- [References](#)

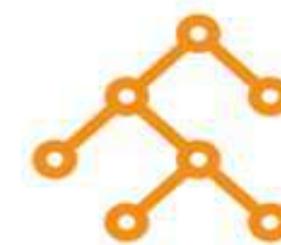


Decision Tree: Overview

Solve both regression and classification problems



Decision Tree works is based on a branch of computer science known as **Information Theory**



The classic use case of decision trees is analysis of segments in business data





Decision Tree



Existing Data of a Bank

Customer	Age	Gender	Marital Status	# cr. Cards	Profitability
1	36	M	M	1	P
2	32	M	S	3	U
3	38	M	M	2	P
4	40	M	S	1	U
5	44	M	M	0	P
6	56	F	M	0	P
7	58	F	S	1	U
8	30	F	S	2	P
9	28	F	M	1	U
10	26	F	M	0	U

Profitable

Unprofitable

To build a predictive model classifying customers logistic, Regression Classifier can be used

Decision Tree



Decision Tree



Existing Data of a Bank

Customer	Age	Gender	Marital Status	# cr. Cards	Profitability
1	36	M	M	1	P
2	32	M	S	3	U
3	38	M	M	2	P
4	40	M	S	1	U
5	44	M	M	0	P
6	56	F	M	0	P
7	58	F	S	1	U
8	30	F	S	2	P
9	28	F	M	1	U
10	26	F	M	0	U

Total Population = 10
 Profitable = 5
 Unprofitable = 5
Profitability rate = 50%

> 35
Age
<= 35

Total Population = 6
 Profitable = 4
 Unprofitable = 2
Profitability rate = 66%

Total Population = 4
 Profitable = 1
 Unprofitable = 3
Profitability rate = 25%



Decision Tree

Total Population = 10

Profitable = 5

Unprofitable = 5

Profitability rate = 50%

> 35

Age

<= 35

Total Population = 6

Profitable = 4

Unprofitable = 2

Profitability rate = 66%

Total Population = 4

Profitable = 1

Unprofitable = 3

Profitability rate = 25%



The segment of data which is >35 has a higher chance of seeing a profitable customer



Decision Tree

Total Population = 10

Profitable = 5

Unprofitable = 5

Profitability rate = 50%

> 35

Age

<= 35

Profitable

Total Population = 6

Profitable = 4

Unprofitable = 2

Profitability rate = 66%

Total Population = 4

Profitable = 1

Unprofitable = 3

Profitability rate = 25%

Married

Marital Status

Single

Total Population = 4

Profitable = 4

Unprofitable = 2

Profitability rate = 100%

Total Population = 2

Profitable = 0

Unprofitable = 2

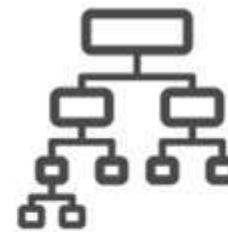
Profitability rate = 0%



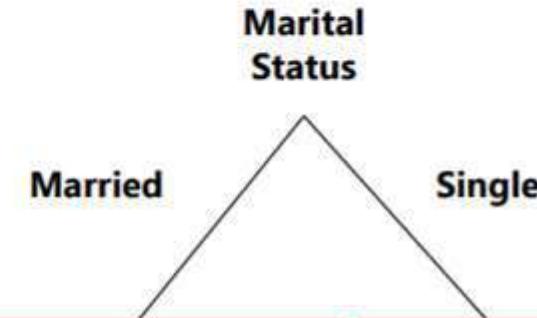


Decision Tree

Decision tree classifier - Recursively sub-setting data can reveal interesting patterns



Data needs to be split in such a way so that the subsets of data end up being dominated by one class of the target variable



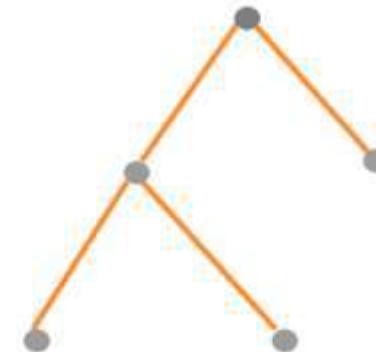
Total Population = 4
Profitable = 4
Unprofitable = 2
Profitability rate = 100%

Total Population = 2
Profitable = 0
Unprofitable = 2
Profitability rate = 0%



Decision Tree

Decision Tree splits into 2 parts at each node



Most implementations of a decision trees
produce binary splits

Binary Tree



Decision Tree: Algorithm



How to decide which variable should be used to create splits?



Understand the intuition behind creating splits

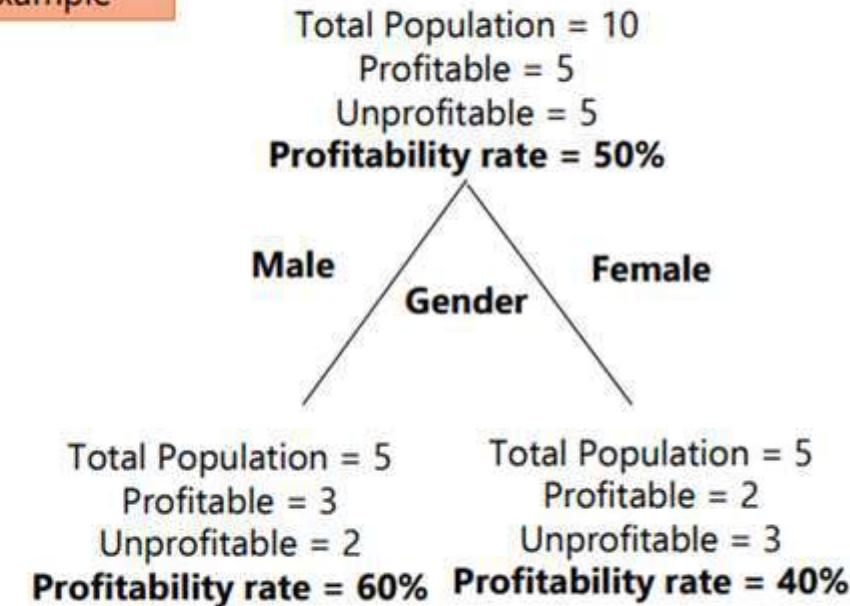
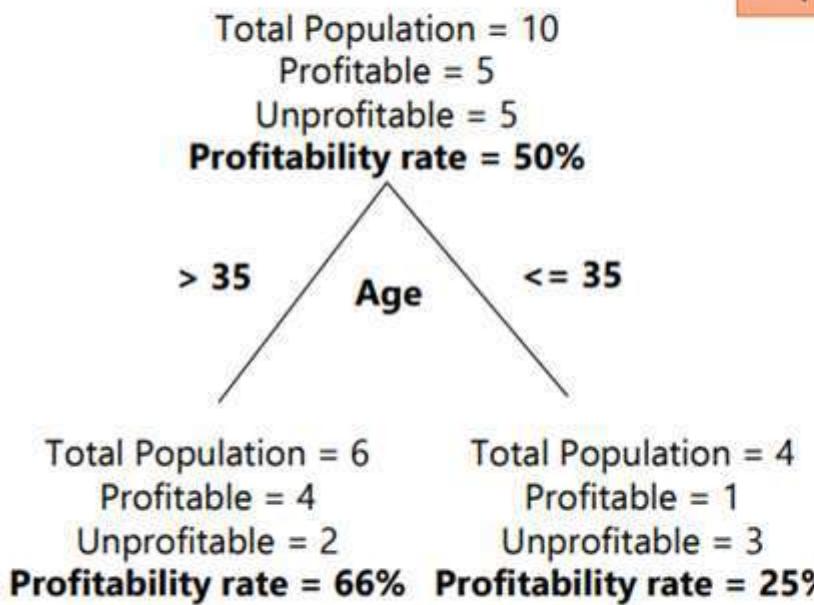
The intuition will be formalized by introducing purity metrics





Decision Tree: Algorithm

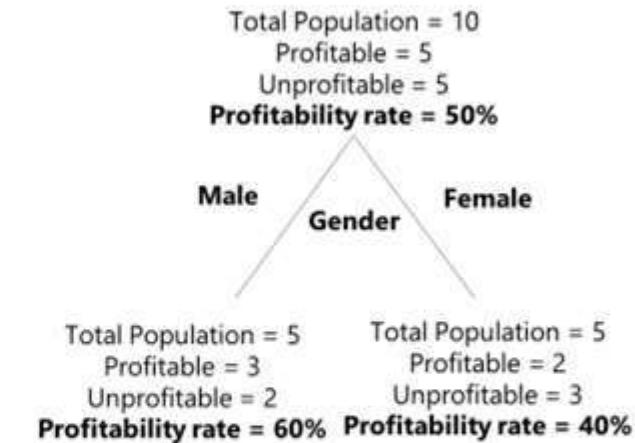
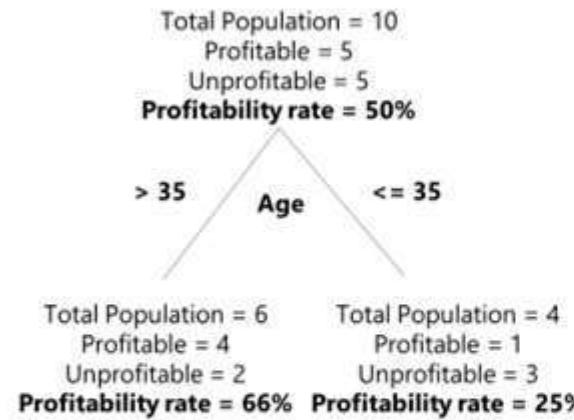
Previous Example



Both splits can be compared to understand which split is better



Decision Tree: Algorithm



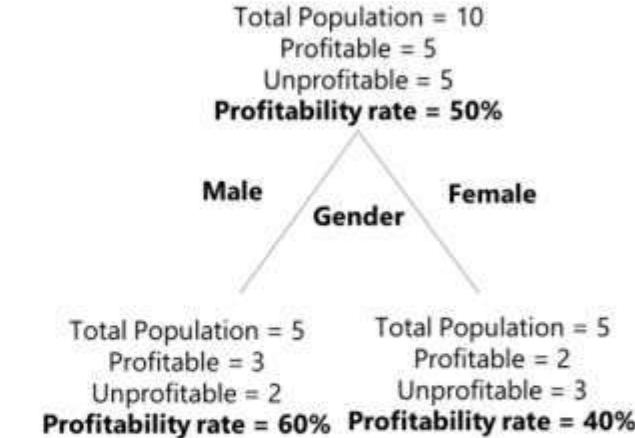
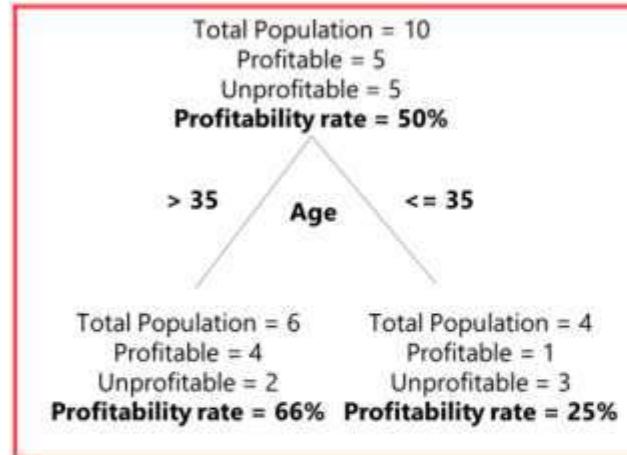
Which variable produces better splits?



Age or Gender?



Decision Tree: Algorithm



Good split in context of classification problem

Split produced by variable age are better than the splits produced by variable gender

Greater the **class imbalance**, better the split



Decision Tree: Algorithm

Class imbalance can be measured by computing Gini or Entropy

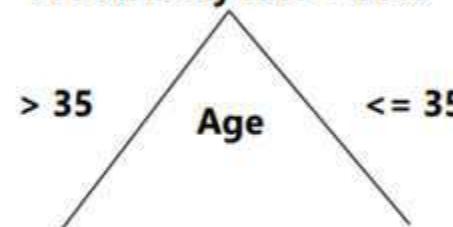
$$Gini = 1 - \sum p_i^2$$

$$Entropy = -\sum p_i \log_2 p_i$$



Decision Tree: Algorithm

Total Population = 10
Profitable = 5
Unprofitable = 5
Profitability rate = 50%



Total Population = 6
Profitable = 4
Unprofitable = 2
Profitability rate = 66%

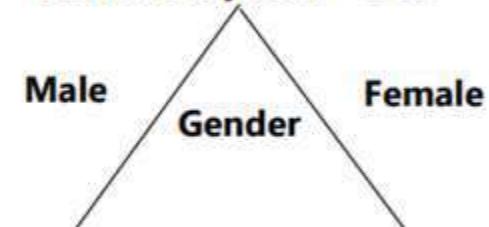
Total Population = 4
Profitable = 1
Unprofitable = 3
Profitability rate = 25%

$$1 - [\left(\frac{4}{6} \right)^2 + \left(\frac{2}{6} \right)^2]$$

0.44

$$Gini = 1 - \sum p_i^2$$

Total Population = 10
Profitable = 5
Unprofitable = 5
Profitability rate = 50%



Total Population = 5
Profitable = 3
Unprofitable = 2
Profitability rate = 60%

Total Population = 5
Profitable = 2
Unprofitable = 3
Profitability rate = 40%

$$1 - [\left(\frac{3}{5} \right)^2 + \left(\frac{2}{5} \right)^2]$$

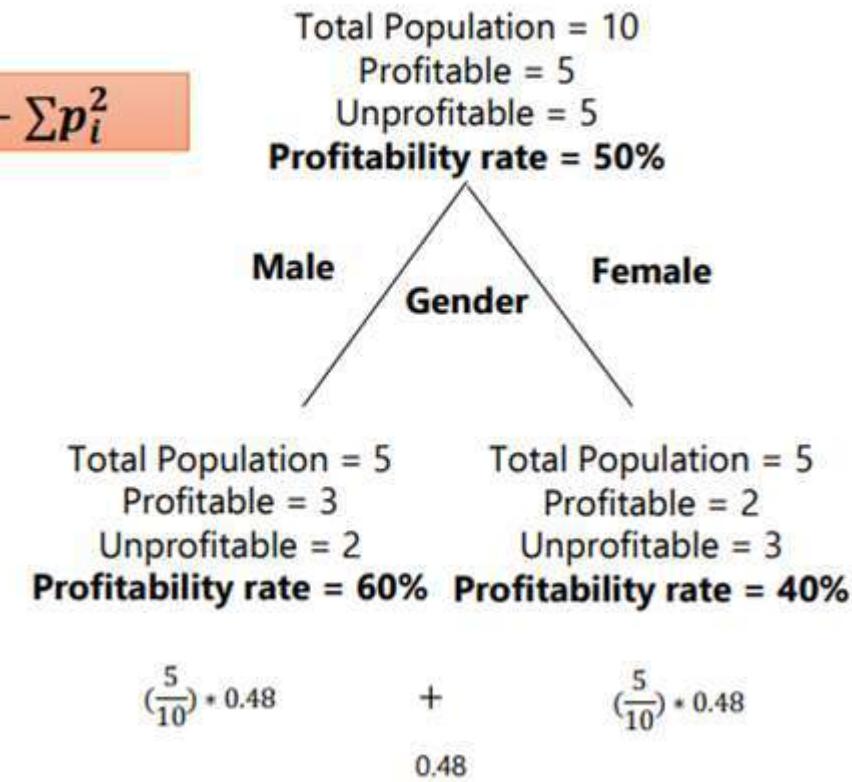
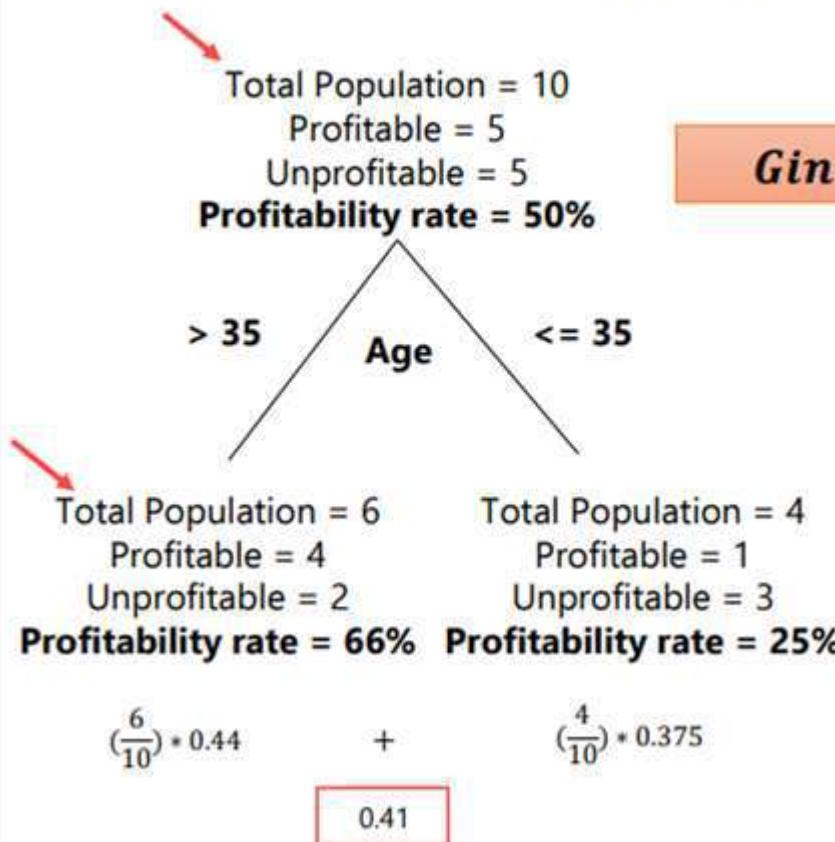
0.48

$$1 - [\left(\frac{2}{5} \right)^2 + \left(\frac{3}{5} \right)^2]$$

0.48



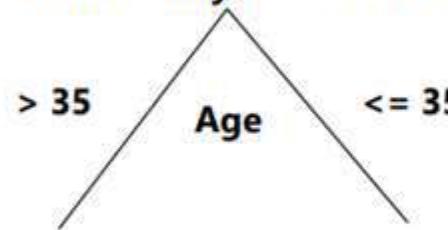
Decision Tree: Algorithm





Decision Tree: Algorithm

Total Population = 10
Profitable = 5
Unprofitable = 5
Profitability rate = 50%



Total Population = 6
Profitable = 4
Unprofitable = 2

Profitability rate = 66%

$$-\left[\left(\frac{4}{6}\right) * \log_2\left(\frac{4}{6}\right) + \left(\frac{2}{6}\right) * \log_2\left(\frac{2}{6}\right)\right]$$

0.91

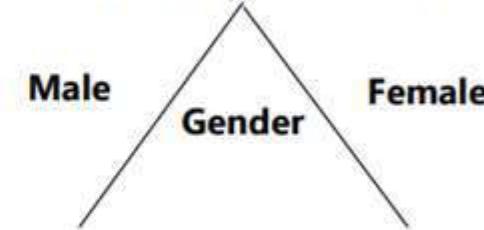
Total Population = 4
Profitable = 1
Unprofitable = 3

Profitability rate = 25%

$$-\left[\left(\frac{1}{4}\right) * \log_2\left(\frac{1}{4}\right) + \left(\frac{3}{4}\right) * \log_2\left(\frac{3}{4}\right)\right]$$

0.81

Total Population = 10
Profitable = 5
Unprofitable = 5
Profitability rate = 50%



Total Population = 5
Profitable = 3
Unprofitable = 2

Profitability rate = 60% **Profitability rate = 40%**

$$-\left[\left(\frac{3}{5}\right) * \log_2\left(\frac{3}{5}\right) + \left(\frac{2}{5}\right) * \log_2\left(\frac{2}{5}\right)\right]$$

0.97

Total Population = 5
Profitable = 2
Unprofitable = 3

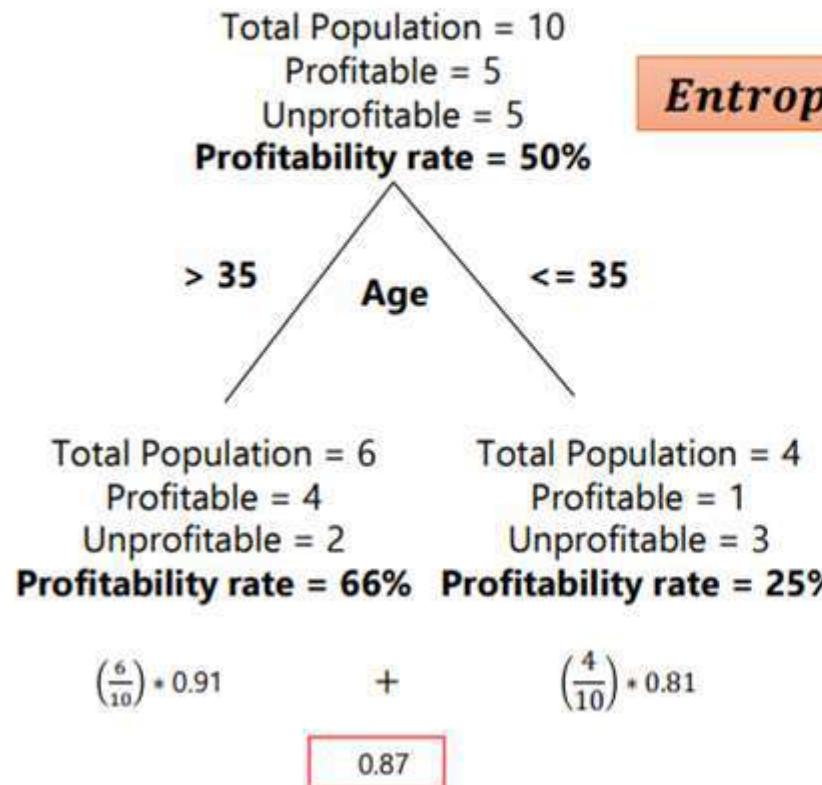
Profitability rate = 40%

$$-\left[\left(\frac{2}{5}\right) * \log_2\left(\frac{2}{5}\right) + \left(\frac{3}{5}\right) * \log_2\left(\frac{3}{5}\right)\right]$$

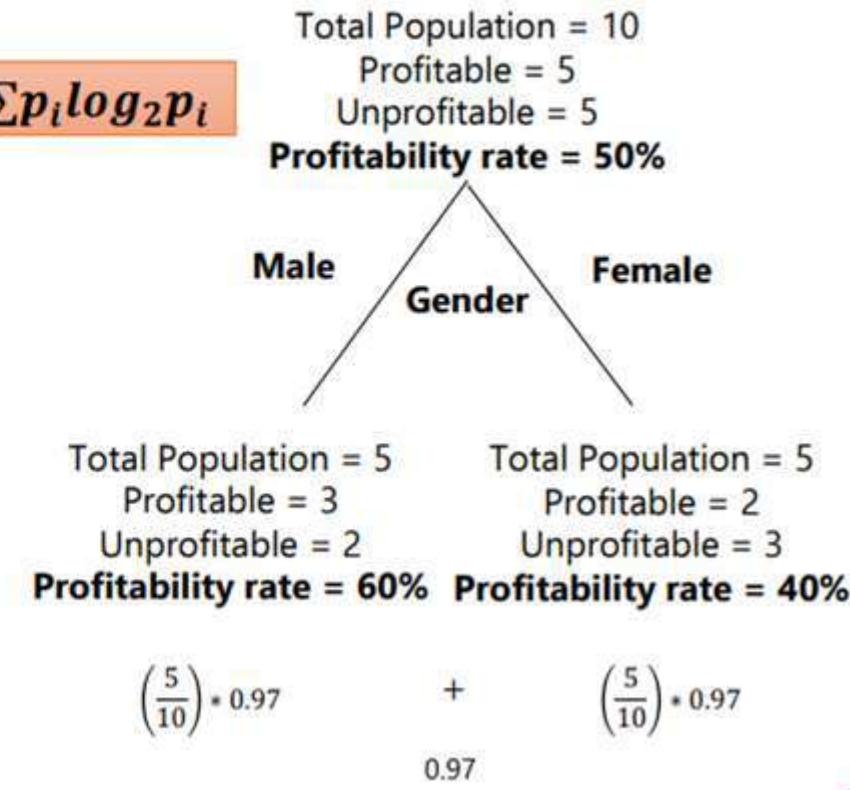
0.97



Decision Tree: Algorithm

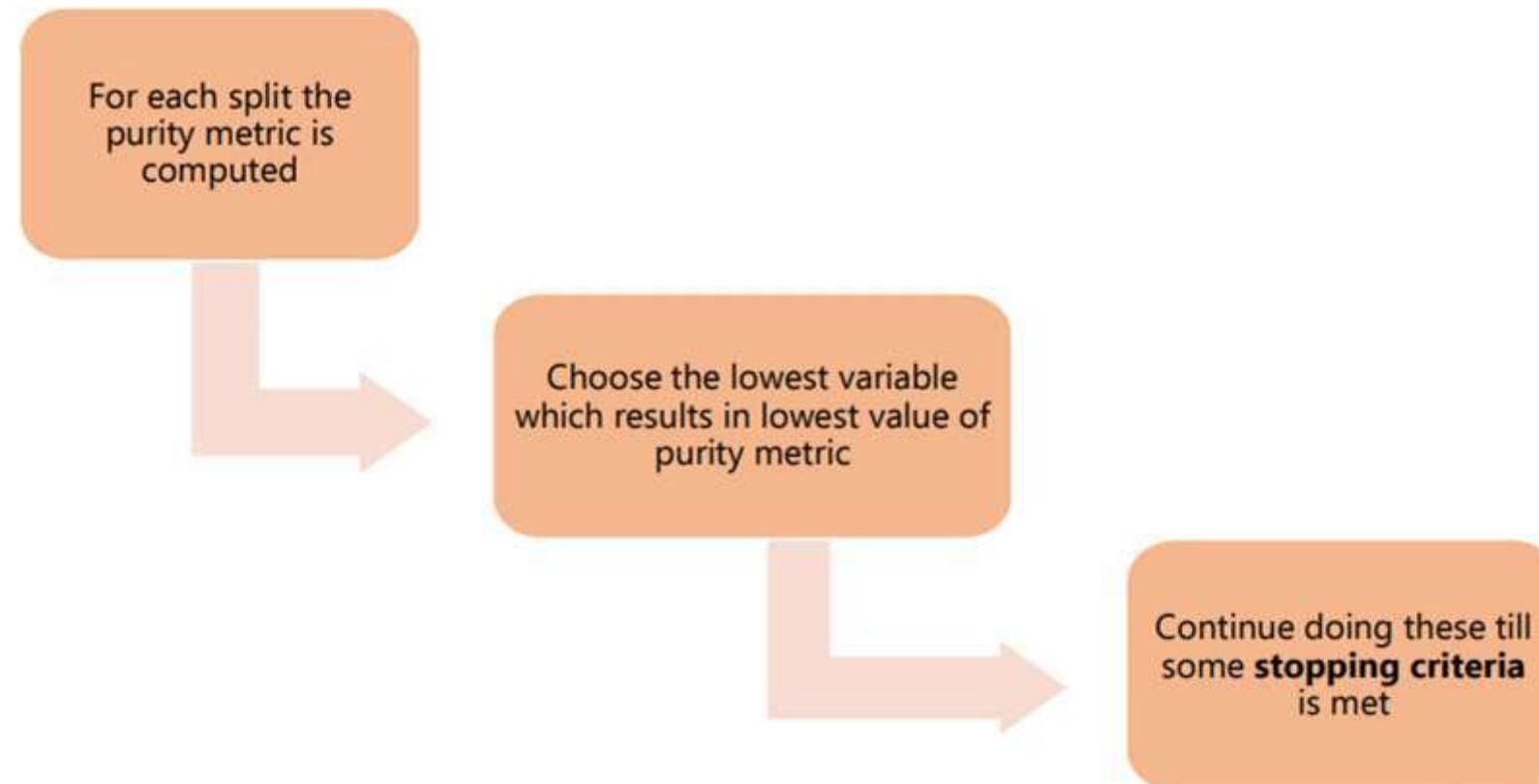


$$\text{Entropy} = -\sum p_i \log_2 p_i$$





Decision Tree: Algorithm Overview





Decision Tree: Algorithm Overview

Stopping Criteria

Depth of tree

Specifying the levels of the tree

Improvement in purity metric

Specifying the minimum change in purity metric from one split to another

Value in terminal node

Specifying the number of value in the terminal node



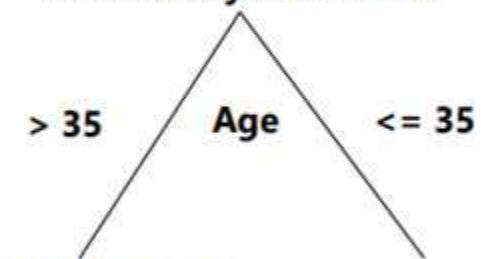
Decision Tree: Prediction

Use decision tree classifier as prediction

Available data – 20 year old person

Prediction – 25% Chance of him being profitable

Total Profitable = 5
population = 10
Unprofitable = 5
Profitability rate = 50%



Total Population = 6
Profitable = 4
Unprofitable = 2
Profitability rate = 66%

Total Population = 4
Profitable = 1
Unprofitable = 3
Profitability rate = 25%



Decision Tree: Performance Metrics

Decision tree classifier output probabilities

ROC curves

Confusion
metrics

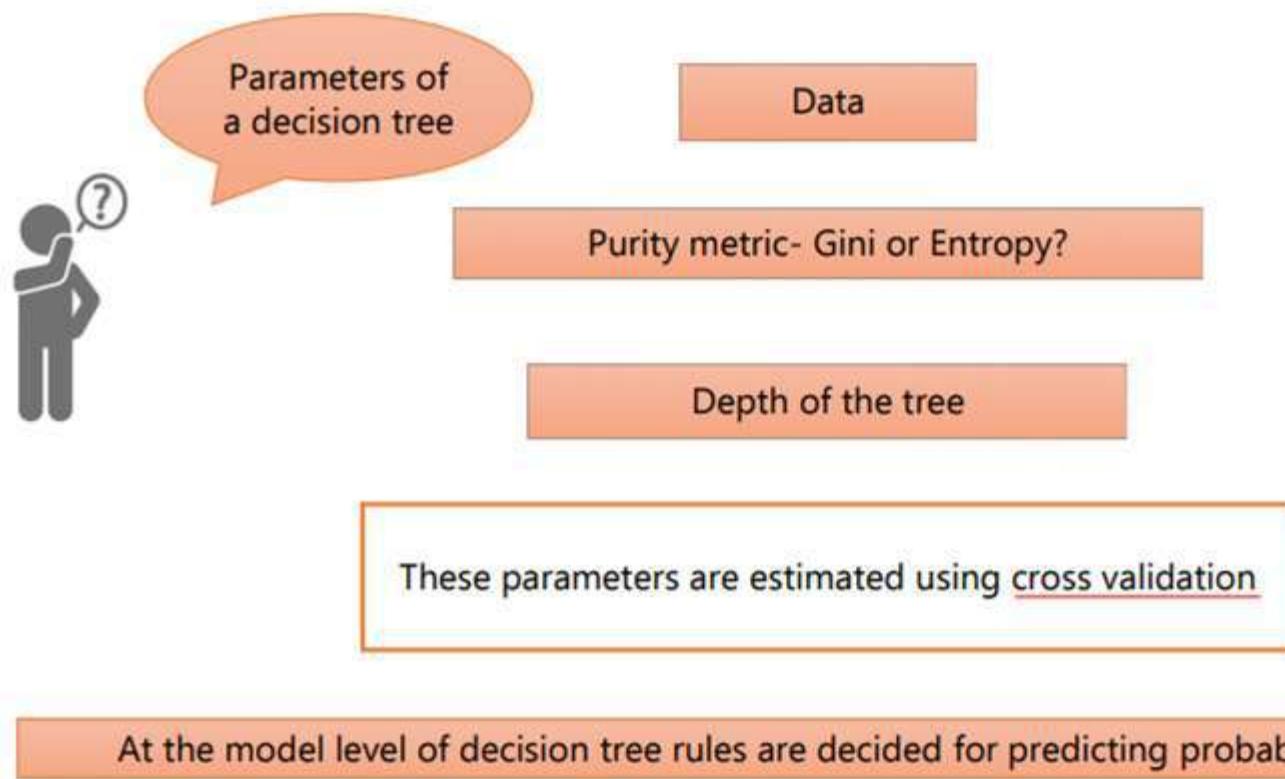
Area under ROC
curves

Performance of
the decision tree
classifier

For multiclass problems, accuracy is used as a performance measure



Decision Tree: Parameters and Hyperparameters





- [Python – Matplotlib](#)
- [Installing Matplotlib](#)
- [Python Matplotlib : Types of Plots](#)
- [Matplotlib Scatter Plot](#)
- [Scatter Plot Example](#)
- [Histogram](#)
- [Bar Chart](#)
- [Pie chart](#)
- [Box Plot](#)
- [Seaborn package](#)
- [References](#)



- Matplotlib is a python library used to create 2D graphs and plots by using python scripts.
- It has a module named pyplot which makes things easy for plotting by providing feature to control line styles, font properties, formatting axes etc.
- It supports a very wide variety of graphs and plots namely - histogram, bar charts, scatter plot, boxplot etc.
- It is used along with NumPy to provide an environment that is an effective open source alternative for MatLab.

Installing Matplotlib



- If matplotlib is not available on your machine, then you can install it by open the command prompt and type the following :

pip install matplotlib

- Matplotlib is preinstalled on Anaconda Distribution
- You can use the matplotlib package in your code by importing the matplotlib package:

from matplotlib import pyplot as plt

Python Matplotlib : Types of Plots



- There are various plots which can be created using python matplotlib. Some of them are listed below:



Bar Graph



Histogram



Scatter Plot



Area Plot



Pie Plot

Matplotlib Line Charts



- A Line chart is a graph that represents information as a series of data points connected by a straight line. In line charts, each data point or marker is plotted and connected with a line or curve.
- Ex: Let's consider the apple yield (tons per hectare) in Shimla. Let's plot a line graph using this data and see how the yield of apples changes over time.

```
#reading csv file
import pandas as pd
#importing matplot
import matplotlib.pyplot as plt
df=pd.read_csv('e:\\fruit_data.csv')
plt.plot(df.Year, df.Apple)
plt.xlabel('Year')
plt.ylabel('Yield (tons per hectare)')
```

Matplotlib Line Chart



- We can add a legend which tells us what each line in our graph means.
- To understand what we are plotting, we can add a title to our graph.

```
plt.plot(df.Year, df.Apple)  
plt.plot(df.Year, df.Orange)  
plt.xlabel('Year')  
plt.ylabel('Yield (tons per hectare)')  
plt.title('Crop Yields in Shimla')  
plt.legend(['Apple', 'Orange'])
```

Matplotlib Line Chart



- To show each data point on our graph, we can highlight them with markers using the marker argument. Many different marker shapes like a circle, cross, square, diamond, etc. are provided by Matplotlib.

```
plt.plot(df.Year, df.Apple, marker='o')
plt.plot(df.Year, df.Orange, marker='x')
plt.xlabel('Year')
plt.ylabel('Yield (tons per hectare)')
plt.title('Crop Yields in Shimla')
plt.legend(['Apple', 'Orange'])
```

Matplotlib Scatter Plot



- It is useful to study the relationship between two variables.
- Each data point has the value from the x-axis values and the value from the y-axis values.
- This type of plot can be used to display trends or correlations.
- To make a scatter plot with Matplotlib, we can use the plt.scatter() function.
- The first argument in plt.scatter() is used for the data on the x-axis, and the second - for the data on the y-axis.

Scatter Plot Example



```
import pandas as pd

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

#Scatter Plot between CGPA and GRE Score

import matplotlib.pyplot as plt

plt.scatter(df['GRE Score'],df['CGPA'])

plt.xlabel("GRE Score")

plt.ylabel("CGPA")

plt.title("CGPA vs GRE Score")
```

Observation of Scatter Plot



- Scatter plot helps in finding relationship
- It helps in finding outliers: outliers are those sample of data who does not show similar relationship as other data show.
- Conclusion: From above graph it is clear that candidates with High GRE Scores usually have high CGPA Score

Scatter Plot Example 2



- We are only considering those students who's CGPA is more than 8.5

```
df[df['CGPA']>=8.5].plot(kind="scatter",x='GRE Score',y='TOEFL Score',color="red")  
plt.xlabel("GRE Score")  
plt.ylabel("TOEFL Score")  
plt.title("GRE Score vs TOEFL Score for CGPA>=8.5")  
plt.grid(True)  
plt.show()
```



- A histogram is used to summarize discrete or continuous data by showing the number of data points that fall within a specified range of values (called “bins”).

```
df['GRE Score'].plot(kind='hist',bins=100,figsize=(6,6))

plt.xlabel("GRE Score")

plt.ylabel("Frequency")

plt.title("GRE Score")

plt.show()
```

Histogram



- It shows the frequency of data (for ex: GRE Score)
- Conclusion: There is a density between 310 and 327
- Being above the range would be a good feature for a candidate to get admitted



- Bar chart represents categorical data with rectangular bars.
- Each bar has a height corresponds to the value it represents.
- To make a bar chart with Matplotlib, we'll need the plt.bar() function.

Bar Chart example



- Bar chart on 'Research' column

```
t1=df['Research'].value_counts()  
  
plt.title("Research Group")  
  
plt.xlabel("Research")  
  
plt.ylabel("Number")  
  
t1.plot(kind='bar')
```

Bar Chart Example



□ #Bar Chart

```
import numpy as np  
  
y=np.array([df["TOEFLScore"].min(), df["TOEFL Score"].mean(), df["TOEFL Score"].max()])  
  
x=["Worst","Average","Best"]  
  
plt.bar(x,y)  
  
plt.xlabel("Level")  
  
plt.ylabel("TOEFL Score")  
  
plt.title("TOEFL Score Comparison")  
  
plt.show()
```

Plotting with Categorical Variable



```
import numpy as np

subjects= ('Python', 'C++', 'Java', 'Perl', 'Scala', 'LISP')

performance = [10,8,6,4,2,1]

plt.bar(subjects, performance, align='center', alpha=0.5)

plt.ylabel('Usage')

plt.title('Programming language usage')

plt.show()
```



- Pie chart: a circular plot, divided into slices to show numerical proportion. They are widely used in the business world.
- Example:

```
sizes = [25, 20, 45, 10]
```

```
labels = ["R", "", "Tigers", "Goats"]
```

```
plt.pie(sizes, labels = labels, autopct = "%.2f")
```

```
plt.axes().set_aspect("equal")
```

```
plt.show()
```



- Boxplots are a measure of how well distributed the data in a data set is. It divides the data set into three quartiles. This graph represents the minimum, maximum, median, first quartile and third quartile in the data set.
- Box plot is another method to view the distribution of data in a numerical column

```
df=pd.read_csv('train_loan.csv')  
  
df.boxplot(column='ApplicantIncome')
```

```
#Data visualization using Matplotlib

import pandas as pd

df=pd.read_csv(r'D:\BigData_All_content\csv\Fruit_data.csv')
df.head()

   Year  Apple  Orange
0  2010  0.895  0.962
1  2011  0.910  0.941
2  2012  0.919  0.930
3  2013  0.926  0.923
4  2014  0.929  0.918
```

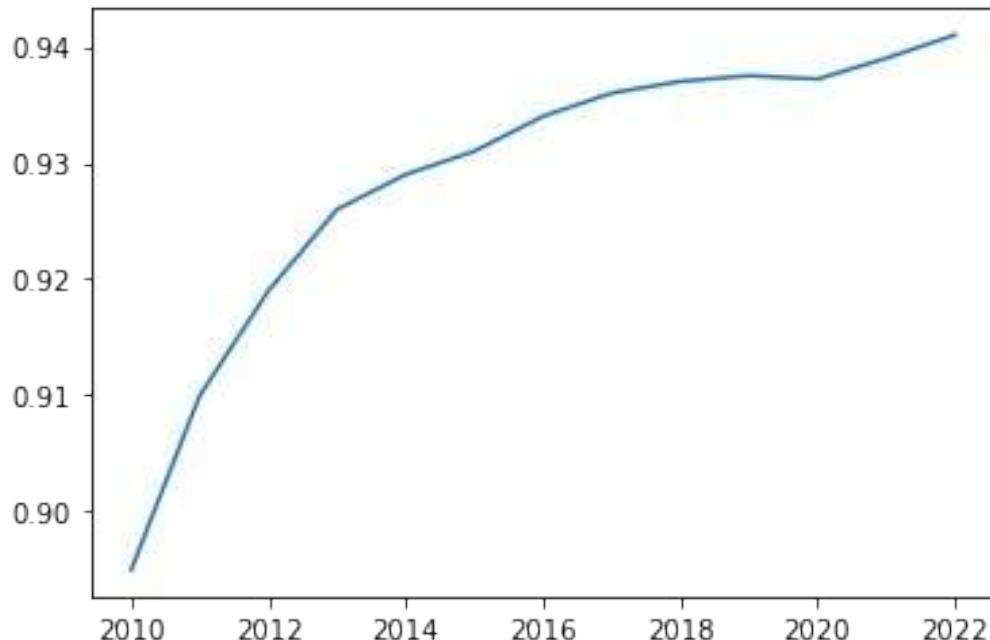
```
#importing matplotlib pkg
```

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

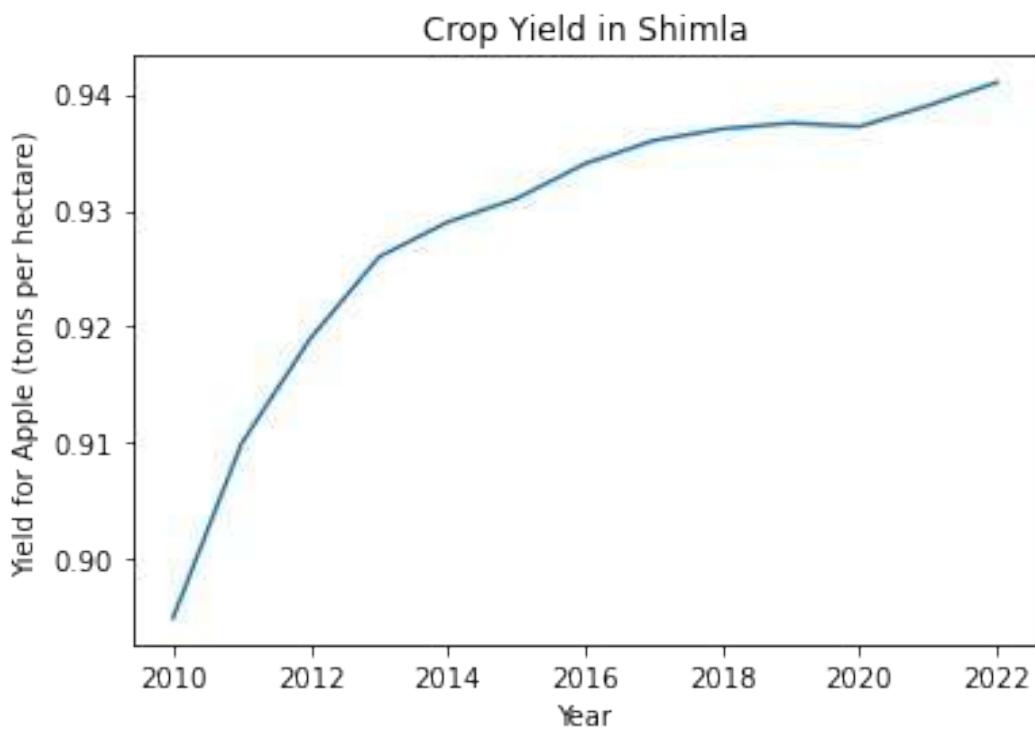
```
#Creating line chart for the yield of 'Apple'
```

```
plt.plot(df.Year, df.Apple)
```

```
[<matplotlib.lines.Line2D at 0x220778500b8>]
```

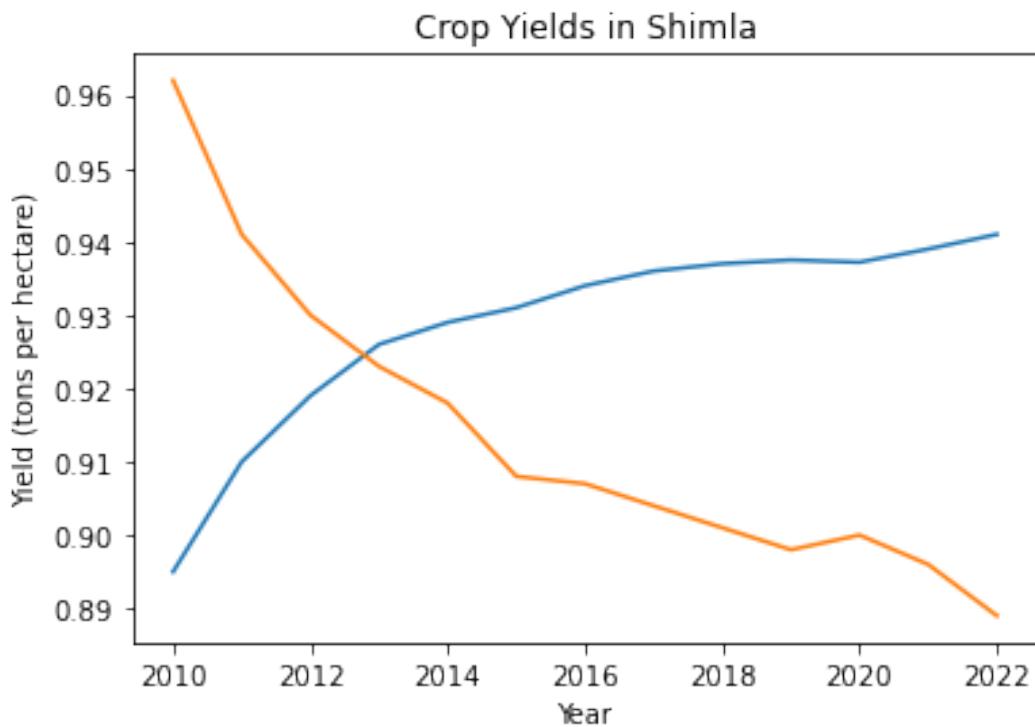


```
plt.plot(df.Year, df.Apple)
plt.title("Crop Yield in Shimla")
plt.xlabel('Year')
plt.ylabel('Yield for Apple (tons per hectare)')
Text(0, 0.5, 'Yield for Apple (tons per hectare)')
```



#Creating line chart for 2 lines

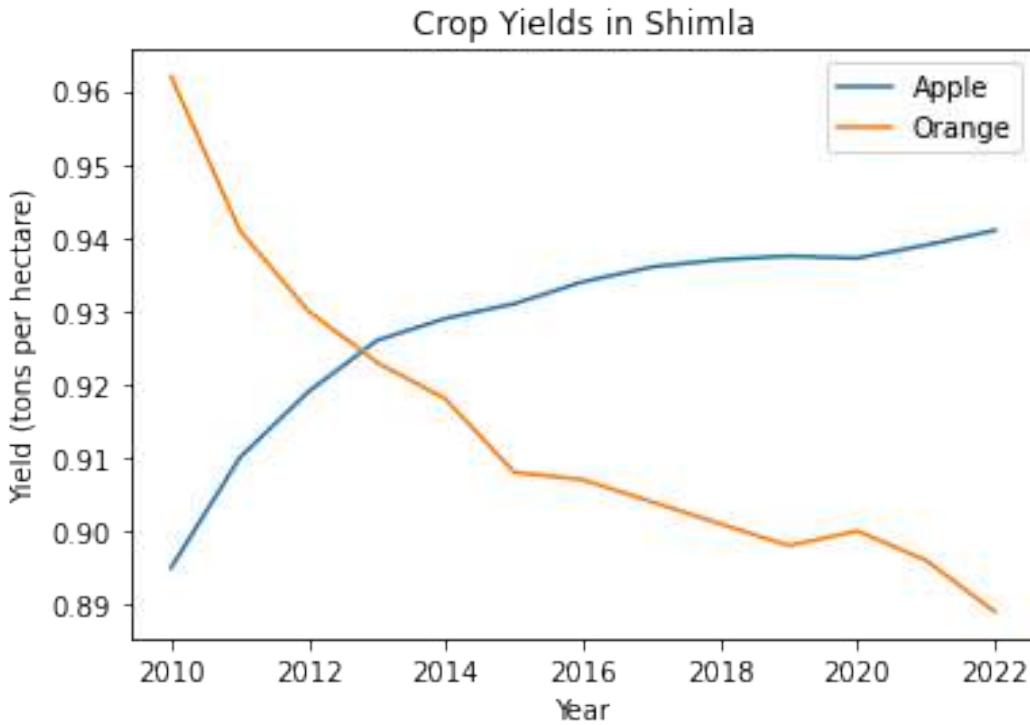
```
plt.plot(df.Year, df.Apple)
plt.plot(df.Year, df.Orange)
plt.title('Crop Yields in Shimla')
plt.xlabel('Year')
plt.ylabel('Yield (tons per hectare)')
Text(0, 0.5, 'Yield (tons per hectare)')
```



#We can add a legend which tells us what each line in our graph means

```
plt.plot(df.Year, df.Apple)
plt.plot(df.Year, df.Orange)
plt.title('Crop Yields in Shimla')
plt.xlabel('Year')
plt.ylabel('Yield (tons per hectare)')
plt.legend(['Apple', 'Orange'])

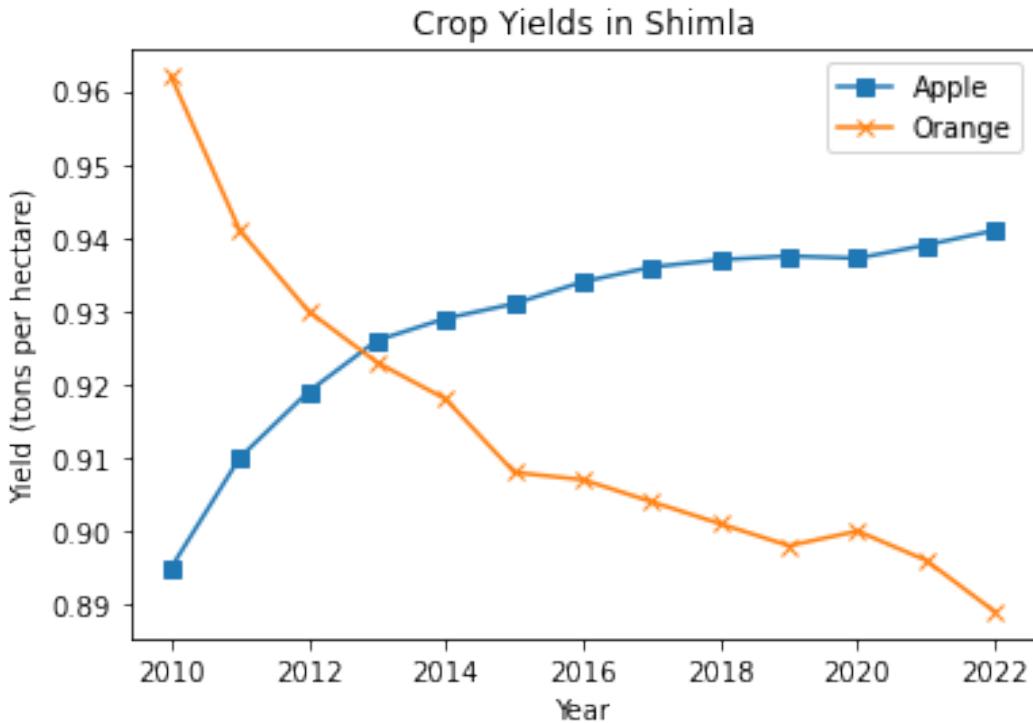
<matplotlib.legend.Legend at 0x220779e4048>
```



```
#We can add marker info. in line chart. We may use different marker symbols:  
#circle (o), cross (x), square (s), diamond (d)
```

```
plt.plot(df.Year, df.Apple, marker='s')  
plt.plot(df.Year, df.Orange, marker='x')  
plt.title('Crop Yields in Shimla')  
plt.xlabel('Year')  
plt.ylabel('Yield (tons per hectare)')  
plt.legend(['Apple', 'Orange'])
```

```
#You can save any graph using the savefig()  
plt.savefig('e:\\fruit_data.jpg', dpi=600)
```



```
df.describe()
```

	Year	Apple	Orange
count	13.00000	13.00000	13.00000
mean	2016.00000	0.928592	0.913615
std	3.89444	0.013383	0.020694
min	2010.00000	0.895000	0.889000
25%	2013.00000	0.926000	0.900000
50%	2016.00000	0.934000	0.907000
75%	2019.00000	0.937200	0.923000
max	2022.00000	0.941000	0.962000

```
df2=pd.read_csv(r'D:\BigData_All_content\csv\Admission_Predict.csv')
df2.head()
```

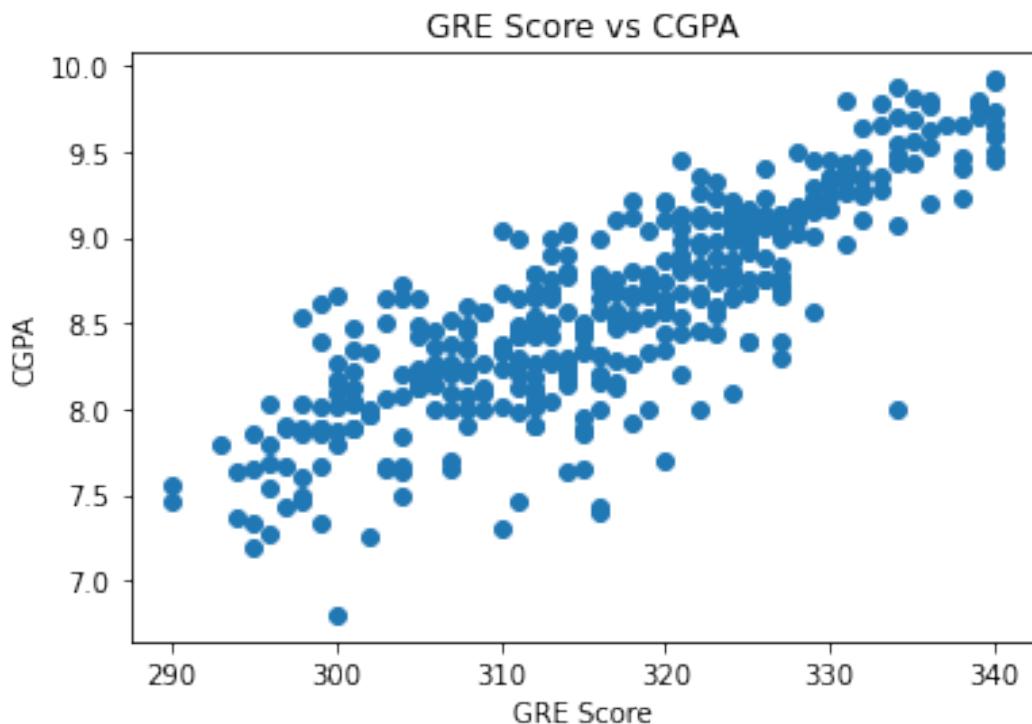
CGPA \ Serial No.	GRE Score	TOEFL Score	University Rating	SOP	LOR
9.65	1	337	118	4	4.5
8.87	2	324	107	4	4.0
8.00	3	316	104	3	3.0
8.67	4	322	110	3	3.5
8.21	5	314	103	2	2.0

	Research	Chance of Admit
0	1	0.92
1	1	0.76
2	1	0.72
3	1	0.80
4	0	0.65

#Creating scatter plot on 'GRE Score' and 'CGPA'

```
plt.scatter(df2['GRE Score'], df2['CGPA'])
plt.xlabel('GRE Score')
plt.ylabel('CGPA')
plt.title('GRE Score vs CGPA')

Text(0.5, 1.0, 'GRE Score vs CGPA')
```



#Observations:

#from above graph, it is clear that candidate with high GRE Score usually have
#higher CGPA

#There is one student having higher GRE Score but having lower CGPA

#Now, observing the relationship b/w GRE Score and CGPA of those students whose #CGPA>8.0

```

x=df2[df2['CGPA']>=8.0]
x

      Serial No.  GRE Score  TOEFL Score  University Rating  SOP  LOR
CGPA \
0           1       337          118                  4  4.5  4.5
9.65
1           2       324          107                  4  4.0  4.5
8.87
2           3       316          104
8.00
3           4       322          110                  3  3.5  2.5
8.67
4           5       314          103                  2  2.0  3.0
8.21
...
...
395         396       324          110                  3  3.5  3.5
9.04
396         397       325          107                  3  3.0  3.5
9.11
397         398       330          116                  4  5.0  4.5
9.45
398         399       312          103                  3  3.5  4.0
8.78
399         400       333          117                  4  5.0  4.0
9.66

      Research  Chance of Admit
0           1          0.92
1           1          0.76
2           1          0.72
3           1          0.80
4           0          0.65
...
395         1          0.82
396         1          0.84
397         1          0.91
398         0          0.67
399         1          0.95

[342 rows x 9 columns]

type(x)
pandas.core.frame.DataFrame

x.columns
Index(['Serial No.', 'GRE Score', 'TOEFL Score', 'University Rating',
       'SOP',

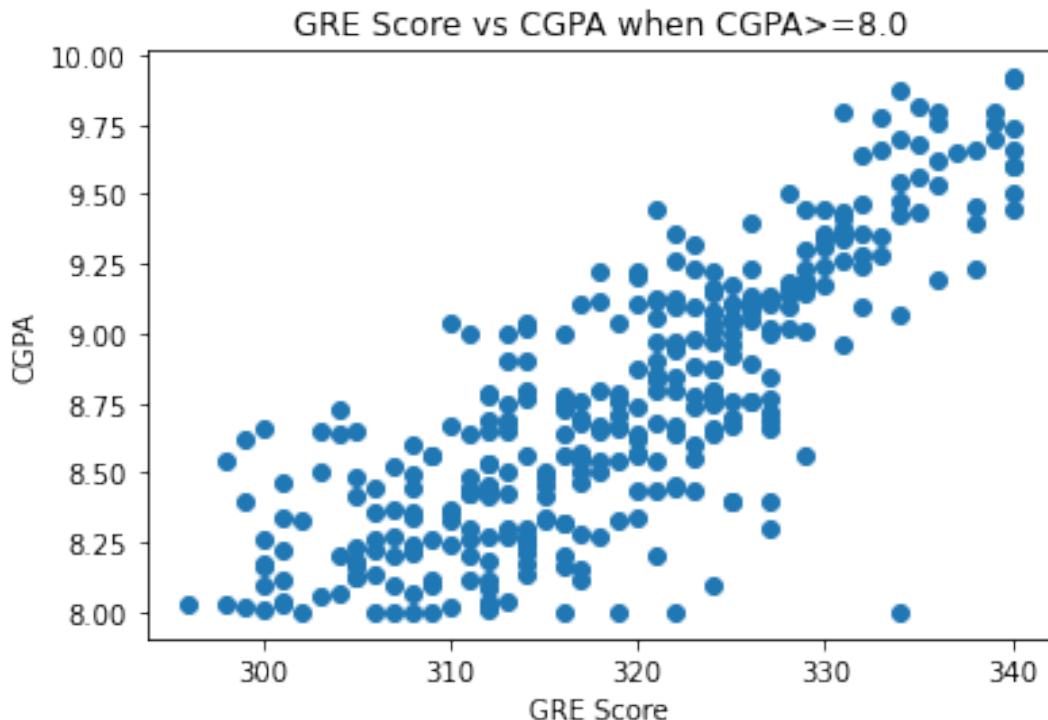
```

```

'LOR', 'CGPA', 'Research', 'Chance of Admit'],
dtype='object')

#x.plot(kind='scatter', x='GRE Score', y='CGPA', color='red')
plt.scatter(x['GRE Score'], x['CGPA'])
plt.xlabel('GRE Score')
plt.ylabel('CGPA')
plt.title('GRE Score vs CGPA when CGPA>=8.0')
plt.show()

```

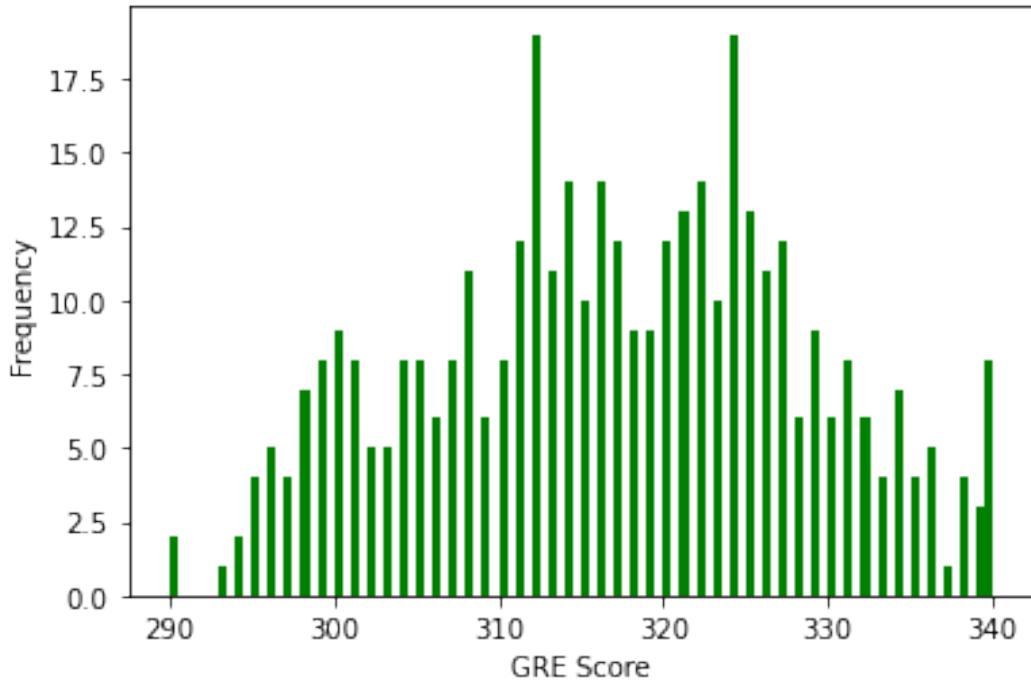


```

#Creating histogram on 'GRE Score'

df2['GRE Score'].plot(kind='hist', bins=100, color='green')
plt.xlabel('GRE Score')
plt.ylabel('Frequency')
Text(0, 0.5, 'Frequency')

```



#Conclusion: there is density between 310 and 330 for GRE Score
#GRE Score>330, would be a good score for a candidate to get admission

#Bar Chart

#Ex: To show the frequency of each category in 'Research' using bar chart

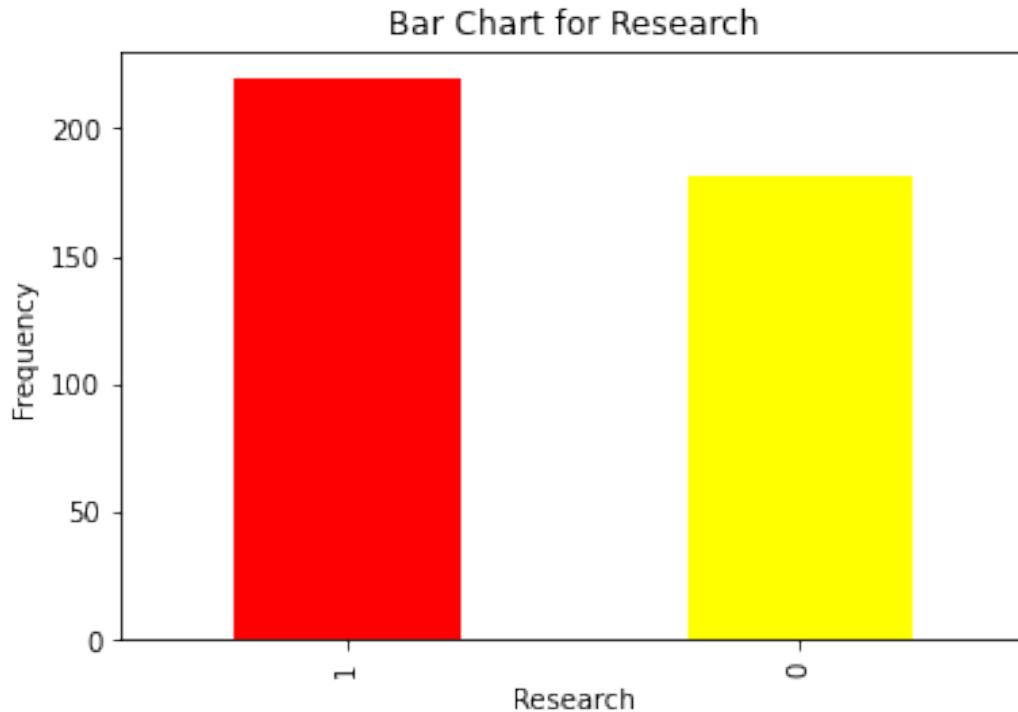
#Count the frequency of each category

```
t=df2.Research.value_counts()
```

#Draw the bar chart

```
t.plot(kind='bar', color=['red', 'yellow'])
plt.xlabel('Research')
plt.ylabel('Frequency')
plt.title('Bar Chart for Research')
```

```
Text(0.5, 1.0, 'Bar Chart for Research')
```



#Usually bar chart is created for categorical data

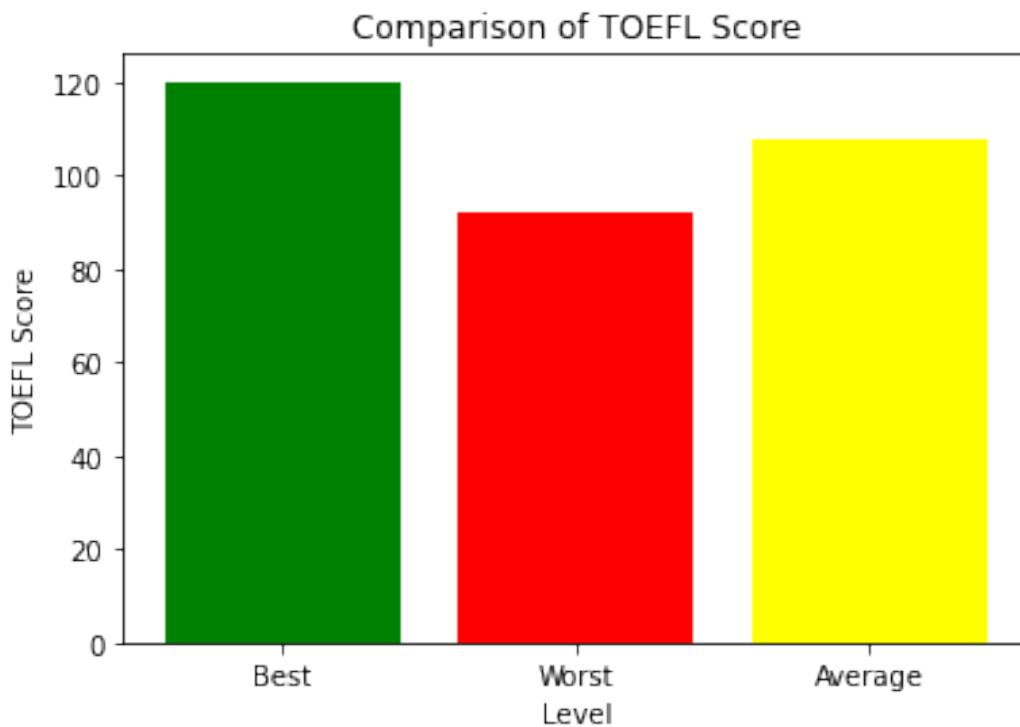
```
#Comparing TOEFL Score graphically
best=df2['TOEFL Score'].max()
worst=df2['TOEFL Score'].min()
avg=df2['TOEFL Score'].mean()

import numpy as np

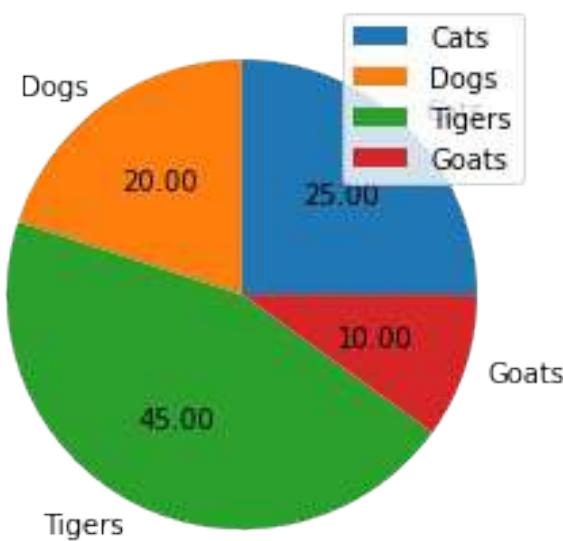
y1=np.array([best, worst, avg])
x1=['Best', 'Worst', 'Average']

plt.bar(x1, y1, color=['green', 'red', 'yellow'])
plt.xlabel('Level')
plt.ylabel('TOEFL Score')
plt.title('Comparison of TOEFL Score')
plt.show

<function matplotlib.pyplot.show(*args, **kw)>
```

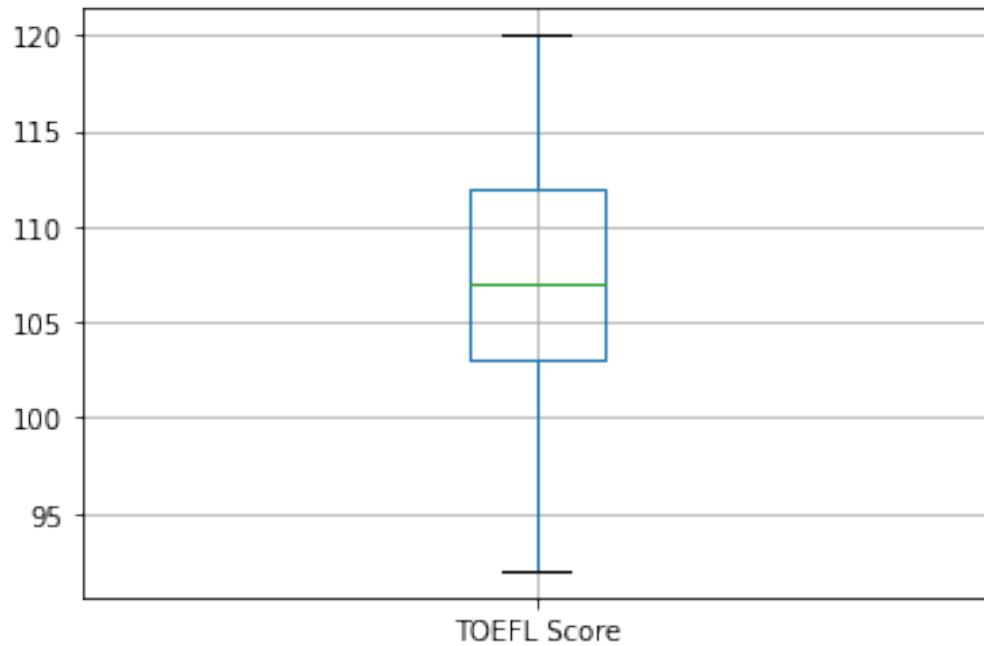


```
#Pie chart
sizes = [25, 20, 45, 10]
labels = ["Cats", "Dogs", "Tigers", "Goats"]
plt.pie(sizes, labels = labels, autopct = "%.2f")
plt.legend()
plt.show()
```



```
#Box plot
```

```
df2.boxplot(column='TOEFL Score')
plt.savefig('e:\\boxplot.jpg', dpi=500)
```



```
df2.describe()
```

	Serial No.	GRE Score	TOEFL Score	University Rating
SOP \ count	400.000000	400.000000	400.000000	400.000000
mean	200.500000	316.807500	107.410000	3.087500
std	115.614301	11.473646	6.069514	1.143728
min	1.000000	290.000000	92.000000	1.000000
25%	100.750000	308.000000	103.000000	2.000000
50%	200.500000	317.000000	107.000000	3.000000
75%	300.250000	325.000000	112.000000	4.000000
max	400.000000	340.000000	120.000000	5.000000

	LOR	CGPA	Research	Chance of Admit
count	400.000000	400.000000	400.000000	400.000000
mean	3.452500	8.598925	0.547500	0.724350
std	0.898478	0.596317	0.498362	0.142609
min	1.000000	6.800000	0.000000	0.340000
25%	3.000000	8.170000	0.000000	0.640000

50%	3.500000	8.610000	1.000000	0.730000
75%	4.000000	9.062500	1.000000	0.830000
max	5.000000	9.920000	1.000000	0.970000



- [Clustering](#)
- [Clustering Example](#)
- [Why do we use clustering in ML?](#)
- [Applications of clustering](#)
- [Clustering v/s Classification](#)
- [Types of Clustering](#)
- [Partitioning/ Centroid-based Clustering](#)
- [Hierarchical Clustering](#)
- [Density-Based Models](#)
- [Model-Based Clustering](#)
- [Fuzzy Clustering](#)
- [K-Means Program Customer Segments](#)
- [Clustering Problem](#)
- [Applying K-means on X](#)
- [Evaluating clustering algorithm](#)
- [Elbow Method](#)
- [References](#)

Clustering



- Have you come across a situation when a Chief Marketing Officer of a company tells you – “Help me understand our customers better so that we can market our products to them in a better manner!”

Or

- Election department has data of election and wants to analyse its data.

Or

- ICC has data of many matches and wants to analyse its data.

Clustering

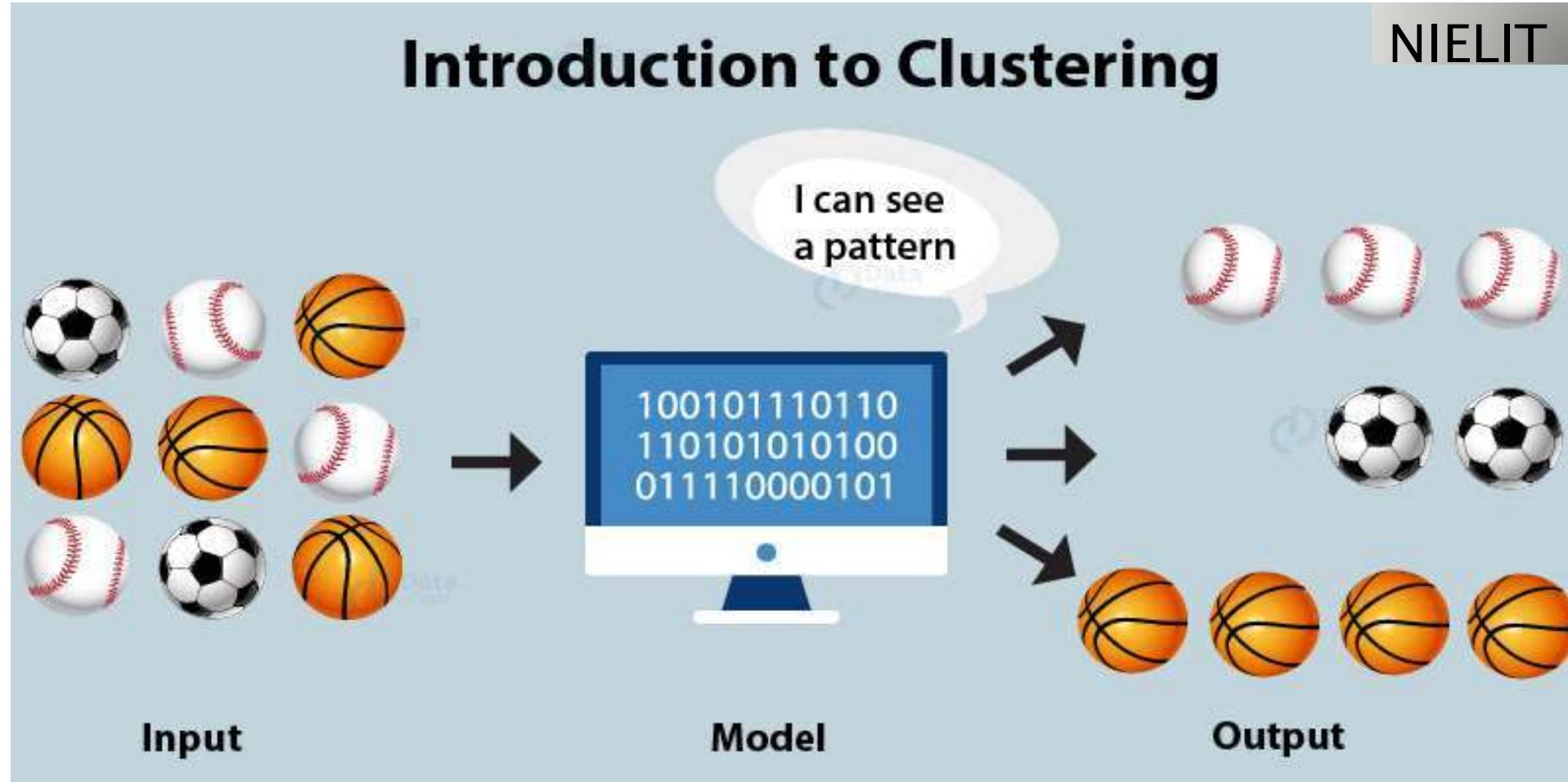


- In all these case analysts was completely clueless what to do!
- As we have studied supervised machine learning and we know that what to find out.
- But I had no clue what to do in this case or we don't know what is the target.
- This is usually the first reaction when you come across an unsupervised learning problem for the first time!
- You are not looking for specific insights for a phenomena, but what you are looking for are structures with in data without them being tied down to a specific outcome.

Clustering



- The method of identifying similar groups of data in a dataset is called clustering. It is one of the most popular techniques in data science.
- Entities in each group are comparatively more similar to entities of that group than those of the other groups.
- Clustering can be easily understand by the given picture-



Clustering



- The basic principle behind cluster is the assignment of a given set of observations into subgroups or clusters such that observations present in the same cluster possess a degree of similarity.
- It is the implementation of the human cognitive ability to discern objects based on their nature.

Example1-

- when you go out for grocery shopping, you easily distinguish between apples and oranges in a given set containing both of them.
- You distinguish these two objects based on their color, texture and other sensory information that is processed by your brain.
- Clustering is an emulation of this process so that machines are able to distinguish between different objects.

Clustering Example



Example2-

- Let's understand with another example. Suppose, you are the head of a rental store and wish to understand preferences of your costumers to scale up your business.
- Is it possible for you to look at details of each costumer and devise a unique business strategy for each one of them?
- Definitely not. But, what you can do is to cluster all of your costumers into say 5 groups based on their purchasing habits and use a separate strategy for costumers in each of these 5 groups. And this is what we call clustering.

Why do we use clustering in ML?



- In basic terms, the objective of clustering is to find different groups within the elements in the data. To do so, clustering algorithms find the structure in the data so that elements of the same cluster (or group) are more similar to each other than to those from different clusters
- Clustering is an important technique as it performs the determination of the intrinsic grouping among the unlabeled dataset.
- In clustering, there are no standard criteria. All of it depends on the user and the suitable criteria that satisfy their needs and requirements.
- For example, to find the homogeneous groups, one can find the representatives through data reduction and describe their suitable properties.
- One can also find unusual data objects for outlier detection. The algorithm then makes the assumption that constitutes what similarity of points makes valid assumptions.

Applications of clustering



- Clustering has a large no. of applications spread across various domains. Some of the most popular applications of clustering are:
- **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.
- **City Planning** : It is used to make groups of houses and to study their values based on their geographical locations and other factors present.
- **Earthquake studies** : By learning the earthquake affected areas we can determine the dangerous zones.

Applications of clustering



Other applications-

- Recommendation engines
- Social network analysis
- Search result grouping
- Medical imaging
- Image segmentation
- Anomaly detection

Clustering v/s Classification



- Classification is the process of classifying the data with the help of class labels whereas, in clustering, there are no predefined class labels.
- Classification is supervised learning, while clustering is unsupervised learning.
- In Classification, algorithms like Decision trees, Bayesian classifiers are used whereas, in Clustering, algorithms like K-means, Expectation Maximization is used.
- Classification has prior knowledge of classes but the cluster doesn't have any prior knowledge of classes.

Clustering v/s Classification



- Mostly, clustering deals with unsupervised data; thus, unlabeled whereas classification works with supervised data; thus, labeled. This is one of the major reasons why clustering does not need training sets while classification does.
- Clustering seeks to verify how data are similar or dissimilar among each other while classification focuses on determining data's "classes" or groups. This makes the clustering process more focused on boundary conditions and the classification analysis more complicated in the sense that it involves more stages.
- **Example of classification:** classification between gender, images.
- **Example of a cluster:** discovery of patterns, grouping.

Types of Clustering



- Broadly speaking, clustering can be divided into two subgroups :
- **Hard Clustering:** In hard clustering, each data point either belongs to a cluster completely or not. For example, each customer is put into one group out of the 10 groups.
- **Soft Clustering:** In soft clustering, instead of putting each data point into a separate cluster, a probability or likelihood of that data point to be in those clusters is assigned. For example, from the above scenario each costumer is assigned a probability to be in either of 10 clusters of the retail store.

Types of Clustering Algorithms



In total, there are five distinct types of clustering algorithms. They are as follows –

- Partitioning Based Clustering
- Hierarchical Clustering
- Model-Based Clustering
- Density-Based Clustering
- Fuzzy Clustering

Partitioning/ Centroid-based Clustering



- In this type of clustering, the algorithm subdivides the data into a subset of k groups.
- These k groups or clusters are to be pre-defined. It divides the data into clusters by satisfying these two requirements –
 - Firstly, Each group should consist of at least one point.
 - Secondly, each point must belong to exactly one group. K-Means Clustering is the most popular type of partitioning clustering method.

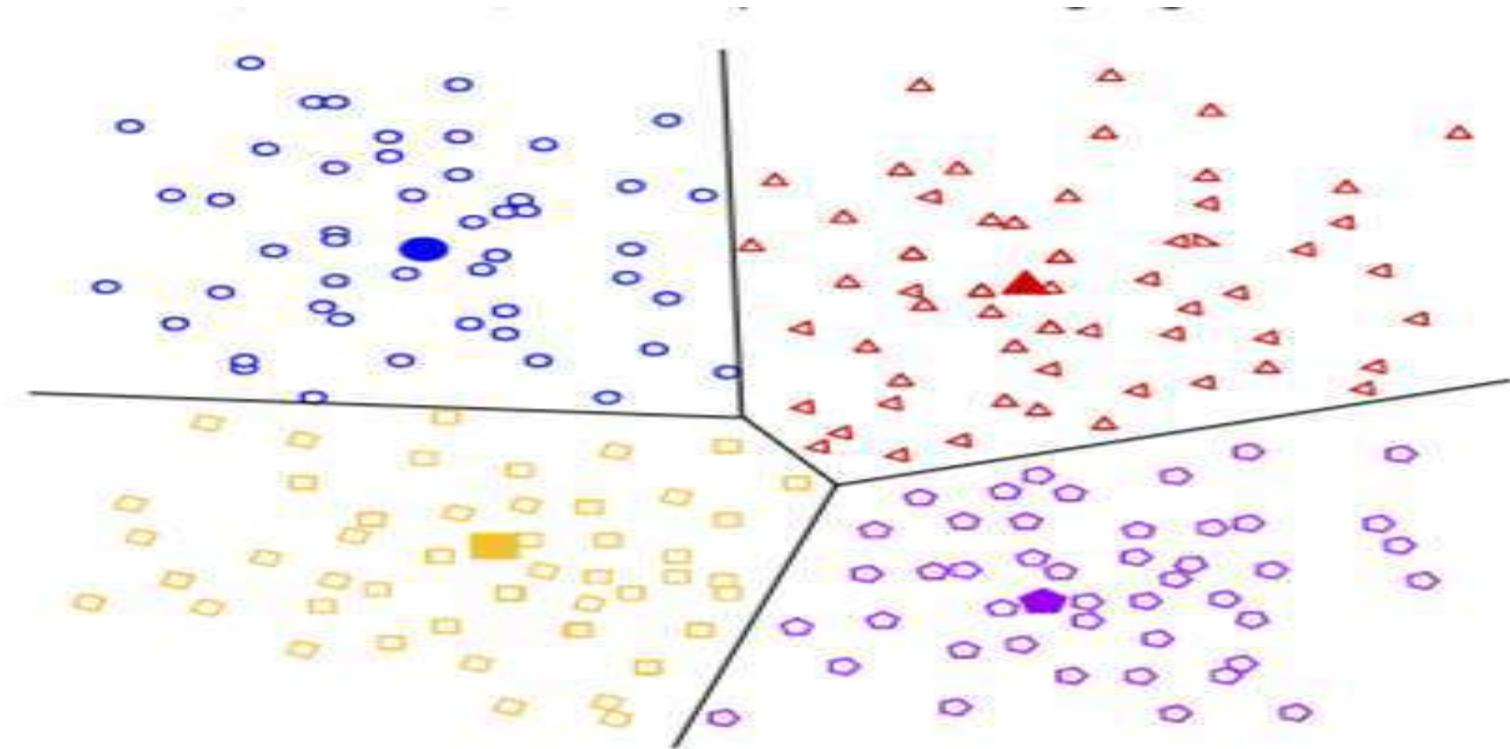
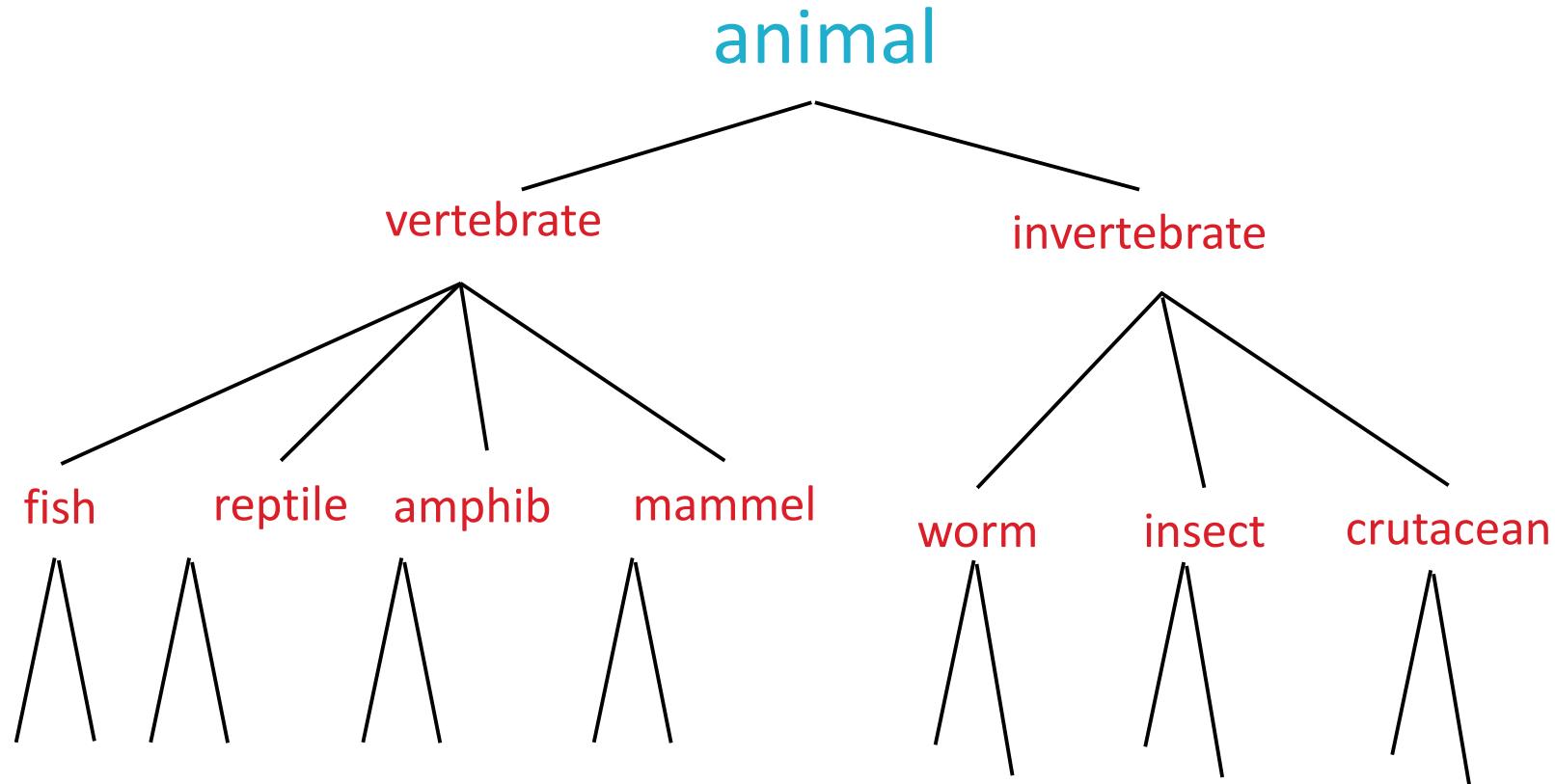


Figure 1: Example of centroid-based clustering.

Hierarchical Clustering



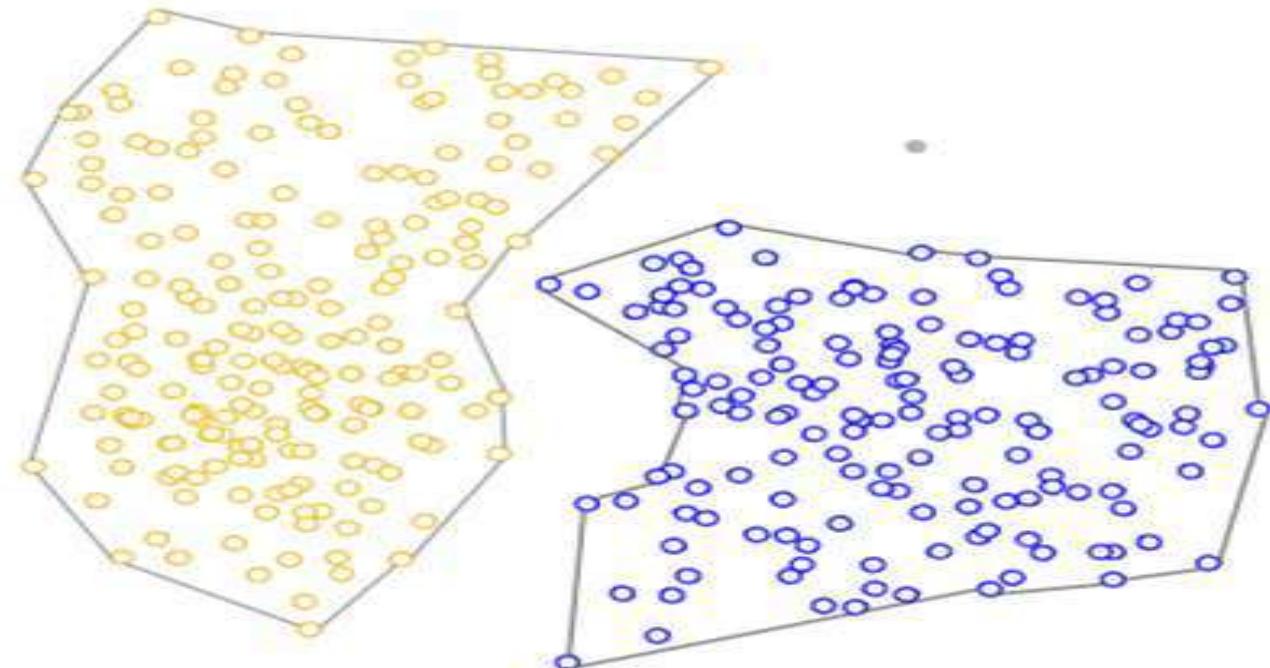
- The basic notion behind this type of clustering is to create a hierarchy of clusters.
- As opposed to Partitioning Clustering, it does not require pre-definition of clusters upon which the model is to be built.
- There are two ways to perform Hierarchical Clustering.
- The first approach is a bottom-up approach, also known as Agglomerative Approach and
- The second approach is the Divisive Approach which moves hierarchy of clusters in a top-down approach. As a result of this type of clustering, we obtain a tree-like representation known as a dendrogram.



Density-Based Models



- In these type of clusters, there are dense areas present in the data space that are separated from each other by sparser areas.
- These type of clustering algorithms play a crucial role in evaluating and finding non-linear shape structures based on density.
- The most popular density-based algorithm is DBSCAN which allows spatial clustering of data with noise.
- It makes use of two concepts – Data Reachability and Data Connectivity.
- These algorithms have difficulty with data of varying densities and high dimensions. Further, by design, these algorithms do not assign outliers to clusters.



Model-Based Clustering



- In this type of clustering technique, the data observed arises from a distribution consisting of a mixture of two or more cluster components.

- Furthermore, each component cluster has a density function having an associated probability or weight in this mixture.



- In this type of clustering, the data points can belong to more than one cluster.

- Each component present in the cluster has a membership coefficient that corresponds to a degree of being present in that cluster.

- Fuzzy Clustering method is also known as a soft method of clustering.

What Is A Good Clustering?



- A good clustering will produce high quality clusters in which:
- The intra-class (that is, intra-cluster) similarity is high
- The inter-class similarity is low
- The measured quality of a clustering depends on both the document representation and the similarity measure used



- As the name itself suggests, this algorithm will regroup n data points into K number of clusters.
 - So given a large amount of data, we need to cluster this data into K clusters.
 - Our goal is to categorise the customers to prepare better strategy.
 - We don't know how many types of customers are present in the dataset (like rich, poor, who loves shopping, who doesn't love shopping etc).



- We don't know exactly how many number of clusters that need to chosen
 - Let's assume for a moment that we are going to segment our data into 3 clusters (or may be 5 cluster).
 - Some of the mathematical terms involved in K-means clustering are centroids, euclidian distance.
 - On a quick note centroid of a data is the average or mean of the data.



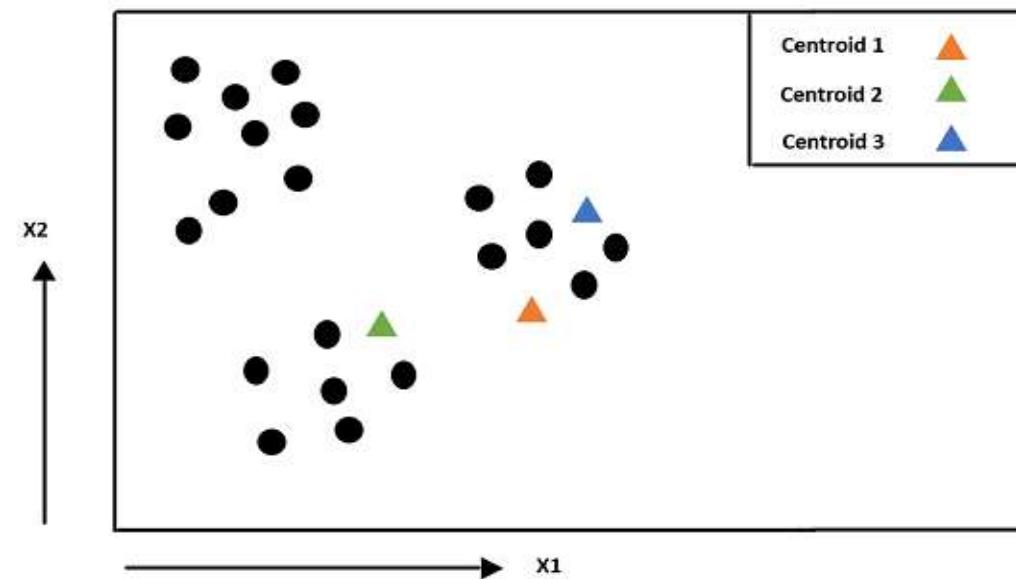
- Euclidian distance is the distance between two points in the coordinate plane. Given two points A(x_1, y_1) and B(x_2, y_2), the euclidian distance between these two points is :

$$\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

- We can use other method also to measure the distance.



- Randomly initialize the cluster centers of each cluster from the data points.
- Let's assume K=3, so we choose randomly 3 data points and assume them as centroids.
- Here three cluster centers or centroids with the green, orange, and blue triangle markers are chosen randomly.



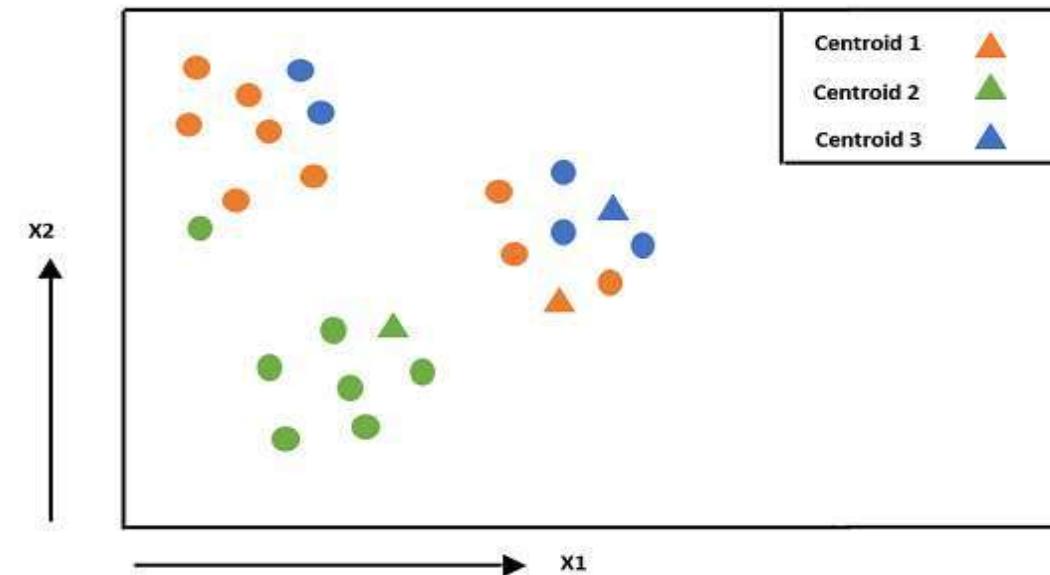


2a.

- For each data point, compute the euclidian distance from all the centroids (3 in this case) and assign the cluster based on the minimal distance to all the centroids.
- In our example, we need to take each black dot, compute its euclidian distance from all the centroids (green, orange and blue), and finally color the black dot to the color of the closest centroid.



- 2.a Assign the centroids to each data point based on computed euclidian distances





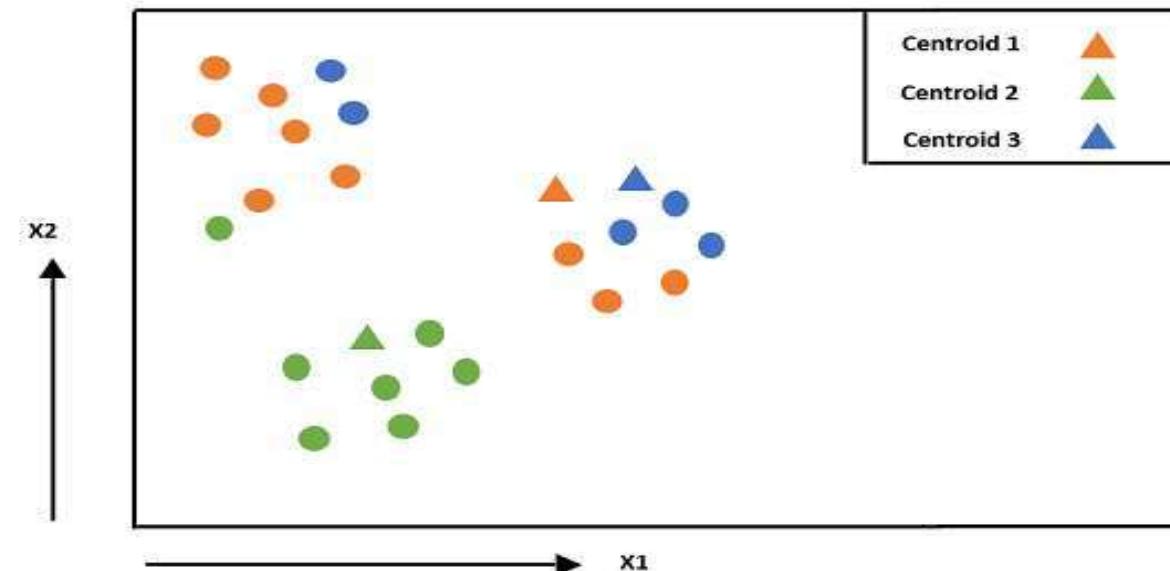
- Adjust the centroid of each cluster by taking the average of all the data points which belong to that cluster on the basis of the computations performed in step 2a.

- In our example, as we have assigned all the data points to one of the clusters, we need to calculate the mean of all the individual clusters and move the centroid to calculated mean.

- Repeat this process till clusters are well separated or convergence is achieved.

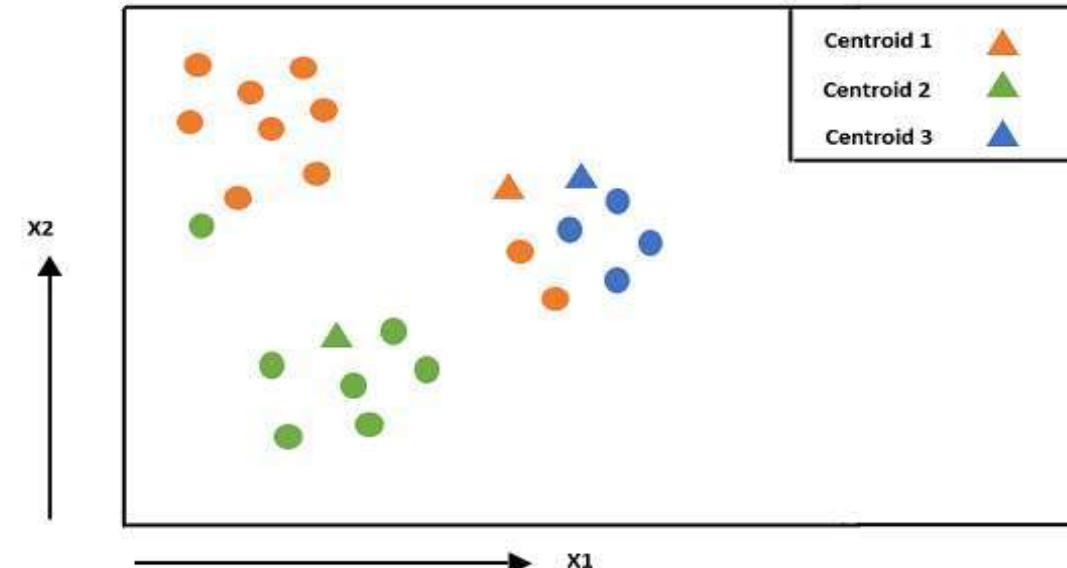


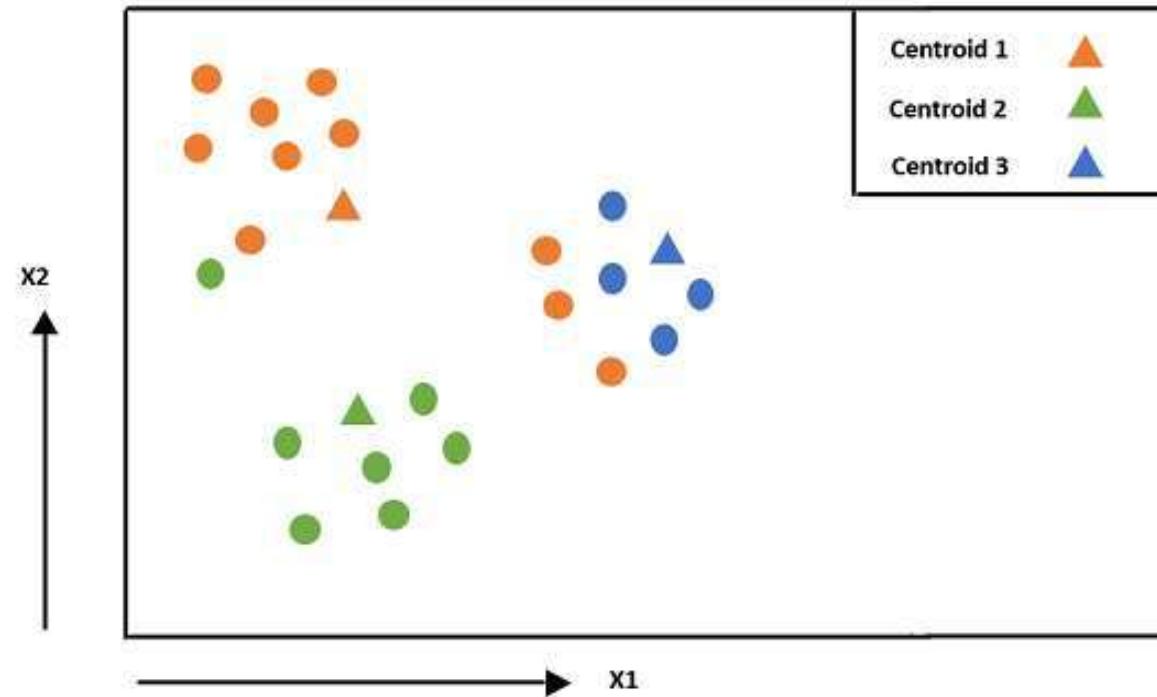
- 2b. Adjust the centroids by taking the average of all data points which belong to that cluster





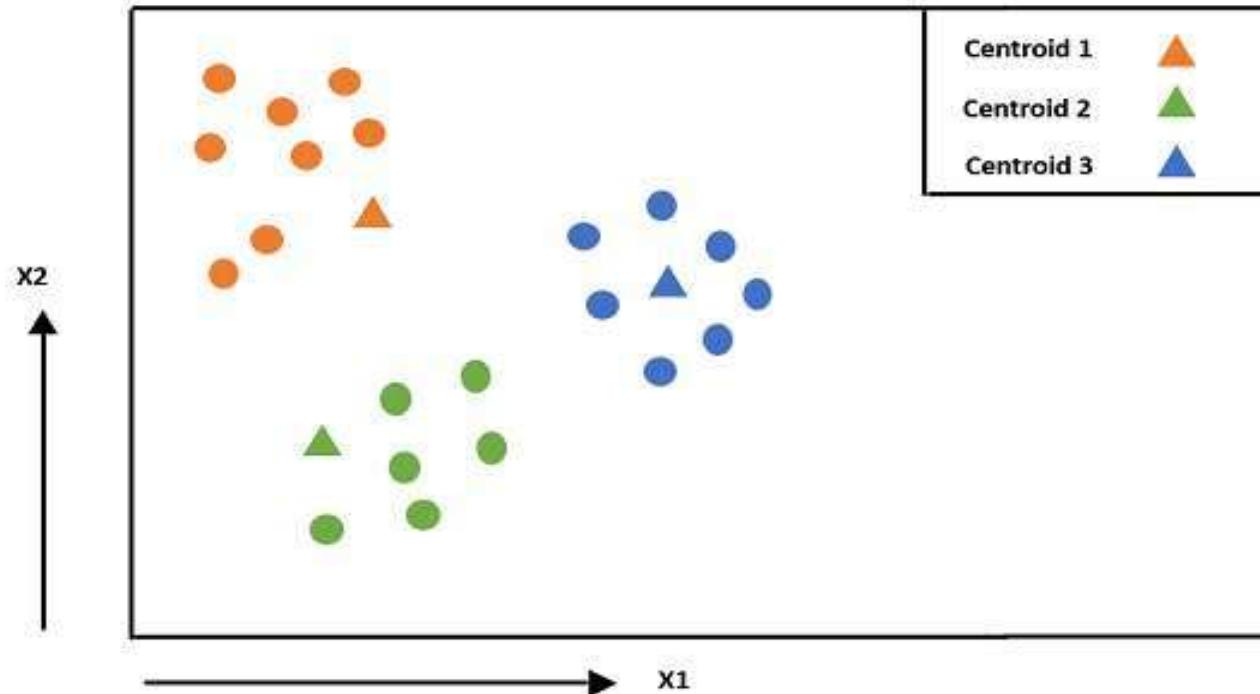
- Repeat these steps until convergence is achieved:





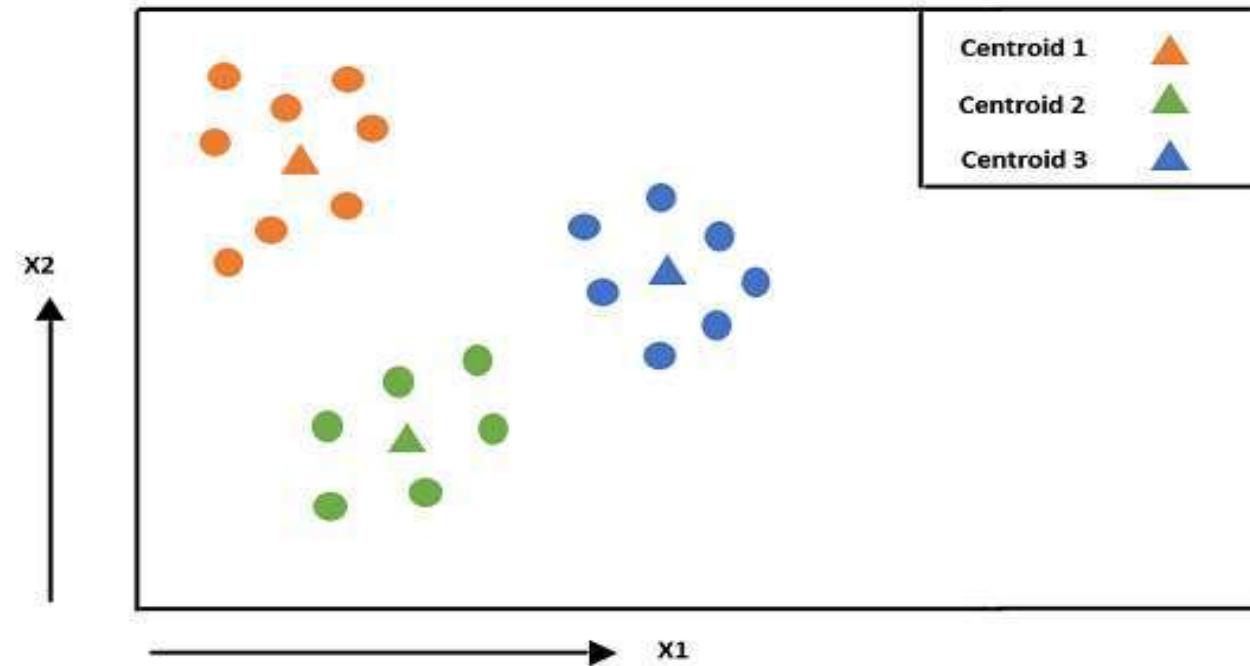


□ Step 2 (A)





□ Step 2 (B)



Clustering Problem



- ❑ #reading the 'Mall_customers.csv'

```
import pandas as pd
```

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

```
df.head()
```

- ❑ **Checking missing values**

```
df.isnull().sum()
```



- ❑ #Ex: we want to segment the data on the basis of 'Annual Income' & 'Spending Score'

```
X=df.iloc[0:100, [3,4]].values
```

```
X.shape
```

Applying K-means on X



- Here we want to segment the data into 5 clusters

```
from sklearn.cluster import KMeans
```

```
#n_clusters specifies the number of clusters to be made on the data
```

```
km=KMeans(n_clusters=5)
```

```
km.fit(X)
```

Predicting the clusters



- #Now using predict function, we can segment the data on these clusters

```
km_predict=km.predict(X)
```

```
km_predict
```

- Viewing the clusters graphically

```
import matplotlib.pyplot as plt  
plt.scatter(X[:,0], X[:,1])
```



```
plt.scatter(X[:,0], X[:,1], c=km_predict, cmap='rainbow')
```

```
plt.xlabel('Annual Income')
```

```
plt.ylabel('Spending Score')
```

Creating 4 clusters



```
km1=KMeans(n_clusters=4)
```

```
km1.fit(X)
```

```
km_predict1=km1.predict(X)
```

```
km_predict1
```

```
plt.scatter(X[:, 0], X[:, 1], c=km_predict1, cmap='rainbow')
```

Creating 3 clusters



```
km3=KMeans(n_clusters=3)
```

```
km3.fit(X)
```

```
km_predict3=km3.predict(X)
```

```
km_predict3
```

```
plt.scatter(X[:,0], X[:,1], c=km_predict3, cmap='rainbow')
```

Creating 2 clusters



```
km2=KMeans(n_clusters=2)
```

```
km2.fit(X)
```

```
km_predict2=km2.predict(X)
```

```
km_predict2
```

```
plt.scatter(X[:,0], X[:,1], c=km_predict2, cmap='rainbow')
```

Evaluating clustering algorithm



- Inertia: metric to evaluate clustering algorithm
- It is the within cluster sum of square of a distance to the cluster centroid
- #Algorithm aims to choose centroid that minimizes the inertia

```
#printing inertia
```

```
print('Intertia:', km2.inertia_)
```



- How many number of clusters dataset should be divided?
- What is the most suitable number of clusters?
- It can be decided by the looking at the data & the problem
- Elbow method: used to find the optimal no. of clusters for a given problem



E=[]

#creating 2 to 10 clusters on the data

for i in range(2,11):

 km_i=KMeans(n_clusters=i)

 km_i.fit(X)

#appending the inertia of each cluster at the end of the list E

 E.append(km_i.inertia_)

for i in E:

 print('Intertia:', i)

Draw Elbow Method



- plt.plot(range(2,11), E)
- plt.title('Elbow Method')
- plt.xlabel('Number of Clusters')
- plt.ylabel('Inertia at each Clusters')
- plt.show()

```
#Clustering problem
```

```
import pandas as pd
```

```
df=pd.read_csv(r'D:\BigData_All_content\csv\Mall_Customers.csv')  
df.head()
```

	CustomerID	Gender	Age	Annual Income	Spending Score (1-100)
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

```
df.describe()
```

	CustomerID	Age	Annual Income	Spending Score (1-100)
count	200.000000	200.000000	200.000000	200.000000
mean	100.500000	38.850000	60.560000	50.200000
std	57.879185	13.969007	26.264721	25.823522
min	1.000000	18.000000	15.000000	1.000000
25%	50.750000	28.750000	41.500000	34.750000
50%	100.500000	36.000000	61.500000	50.000000
75%	150.250000	49.000000	78.000000	73.000000
max	200.000000	70.000000	137.000000	99.000000

```
#checking missing data
```

```
df.isnull().sum()
```

CustomerID	0
Gender	0
Age	0
Annual Income	0
Spending Score (1-100)	0
dtype: int64	

```
#Prepare X
```

```
#Ex: we want to segment the customers on the basis of 'Annual income'  
and 'Spending Score'
```

```
X=df.iloc[:, [3,4]].values
```

```
X.shape
```

```
(200, 2)
```

```
#Applying K-means clustering algorithm on X
```

```
from sklearn.cluster import KMeans
```

```
#Here, we want to segment the data into 5 clusters
```

```
#n_clusters specifies the number of clusters to be made on the given
```

```

data
km=KMeans(n_clusters=5)
km.fit(X)

KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
       n_clusters=5, n_init=10, n_jobs=None,
       precompute_distances='auto',
       random_state=None, tol=0.0001, verbose=0)

#Now, X is fitted on KMeans algo

#It means, KMeans has understood the properties/features of given data
X

#Use predict() to segment the data on these clusters

km_predict=km.predict(X)
km_predict

array([2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2,
       0,
       2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2,
       0,
       2, 0, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4,
       4,
       4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4,
       4,
       4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4,
       4,
       4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 3, 1, 3, 4, 3, 1, 3,
       1,
       4, 3, 1, 3, 1, 3, 1, 3, 4, 3, 1, 3, 1, 3, 1, 3, 1, 3, 1, 3, 1,
       3,
       1, 3, 1, 3, 1, 3, 1, 3, 1, 3, 1, 3, 1, 3, 1, 3, 1, 3, 1, 3, 1,
       3,
       1, 3, 1, 3, 1, 3, 1, 3, 1, 3, 1, 3, 1, 3, 1, 3, 1, 3, 1, 3, 1,
       3,
       1, 3])

```

```

df['Cluster']=km_predict
df.head(10)

```

	CustomerID	Gender	Age	Annual Income	Spending Score (1-100)
Cluster					
0	1	Male	19	15	39
2	2	Male	21	15	81
0	3	Female	20	16	6
2	4	Female	23	16	77

```

0      5   Female    31        17        40
4      6   Female    22        17        76
2      7   Female    35        18         6
5      8   Female    23        18        94
0      9     Male     64        19         3
2      10  Female    30        19        72
9      0

df.to_csv('e:\\Mall_customers_Kmeans.csv')
type(X)
numpy.ndarray

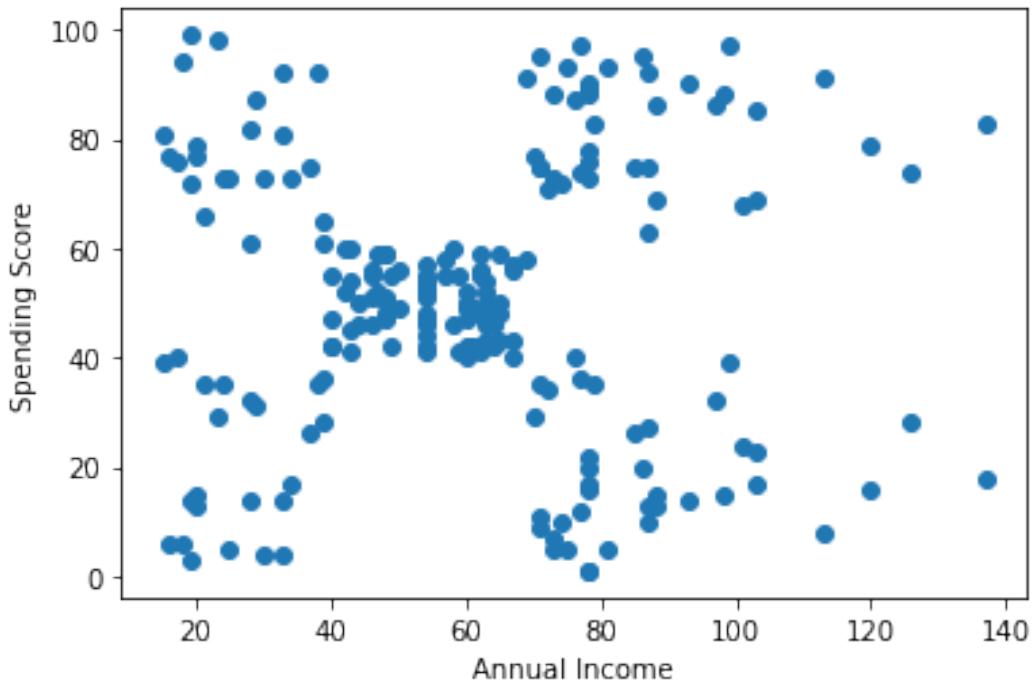
#To view the clusters graphically

import matplotlib.pyplot as plt

plt.scatter(X[:,0], X[:,1])
plt.xlabel('Annual Income')
plt.ylabel('Spending Score')

Text(0, 0.5, 'Spending Score')

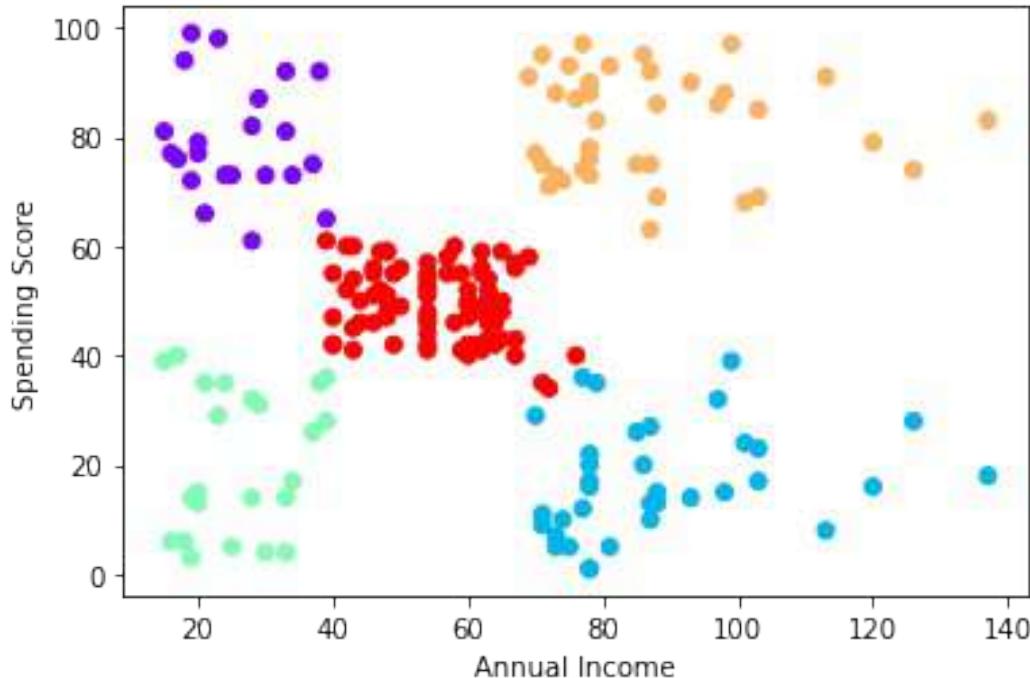
```



```

plt.scatter(X[:,0], X[:,1], c=km_predict, cmap='rainbow')
plt.xlabel('Annual Income')
plt.ylabel('Spending Score')
Text(0, 0.5, 'Spending Score')

```



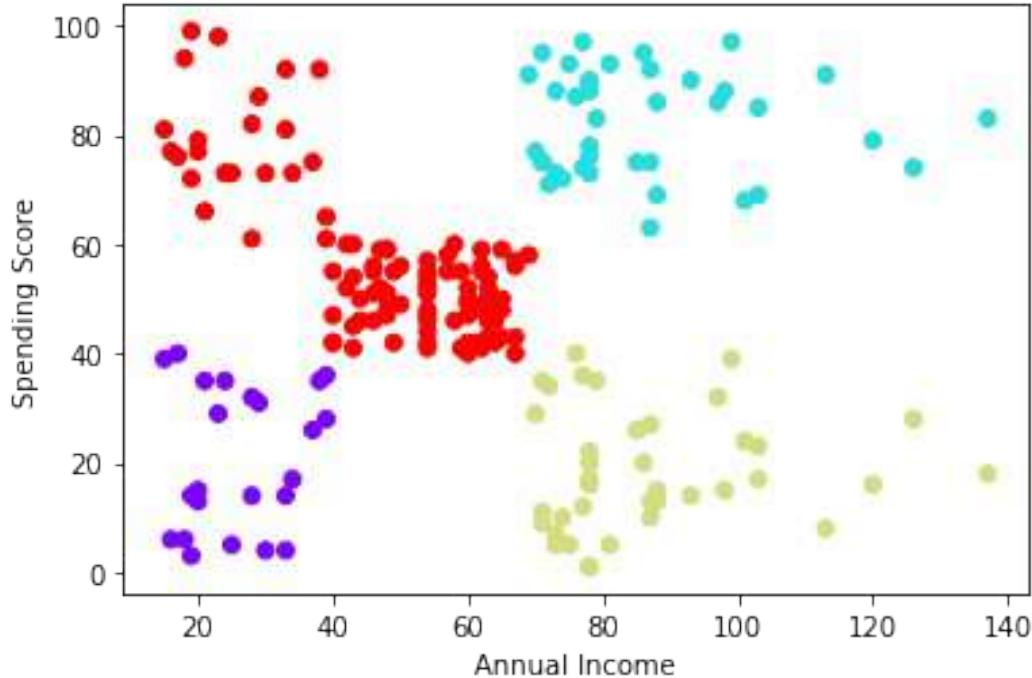
#Creating 4 clusters

```

km4=KMeans(n_clusters=4)
km4.fit(X)
km4_pred=km4.predict(X)

plt.scatter(X[:,0], X[:,1], c=km4_pred, cmap='rainbow')
plt.xlabel('Annual Income')
plt.ylabel('Spending Score')
Text(0, 0.5, 'Spending Score')

```

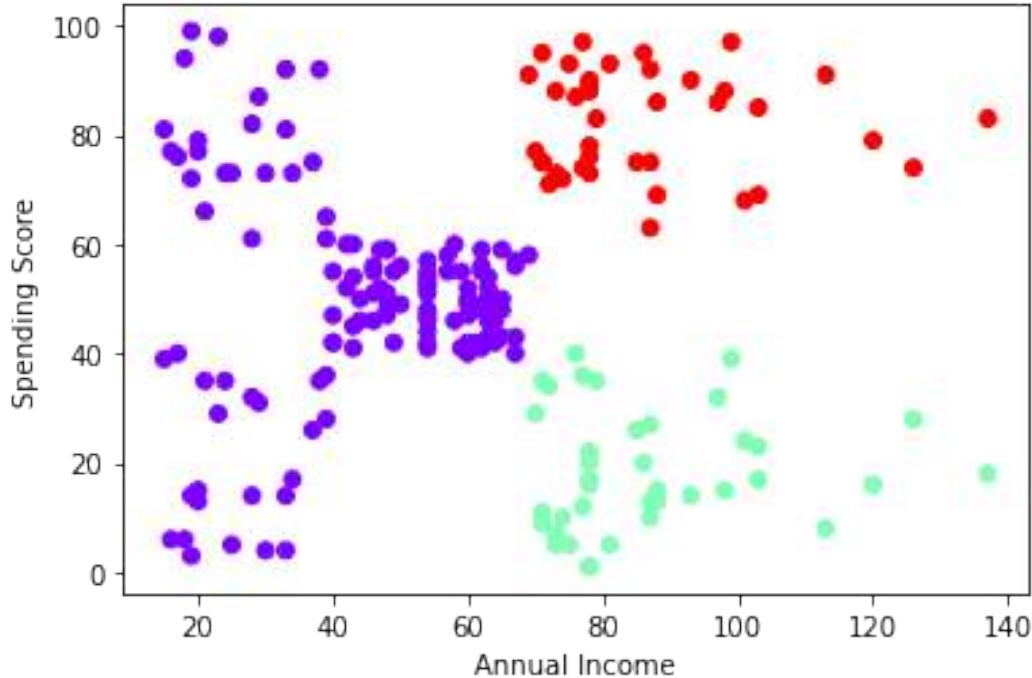


#Creating 3 clusters

```
km3=KMeans(n_clusters=3)
km3.fit(X)
km3_pred=km3.predict(X)

plt.scatter(X[:,0], X[:,1], c=km3_pred, cmap='rainbow')
plt.xlabel('Annual Income')
plt.ylabel('Spending Score')

Text(0, 0.5, 'Spending Score')
```

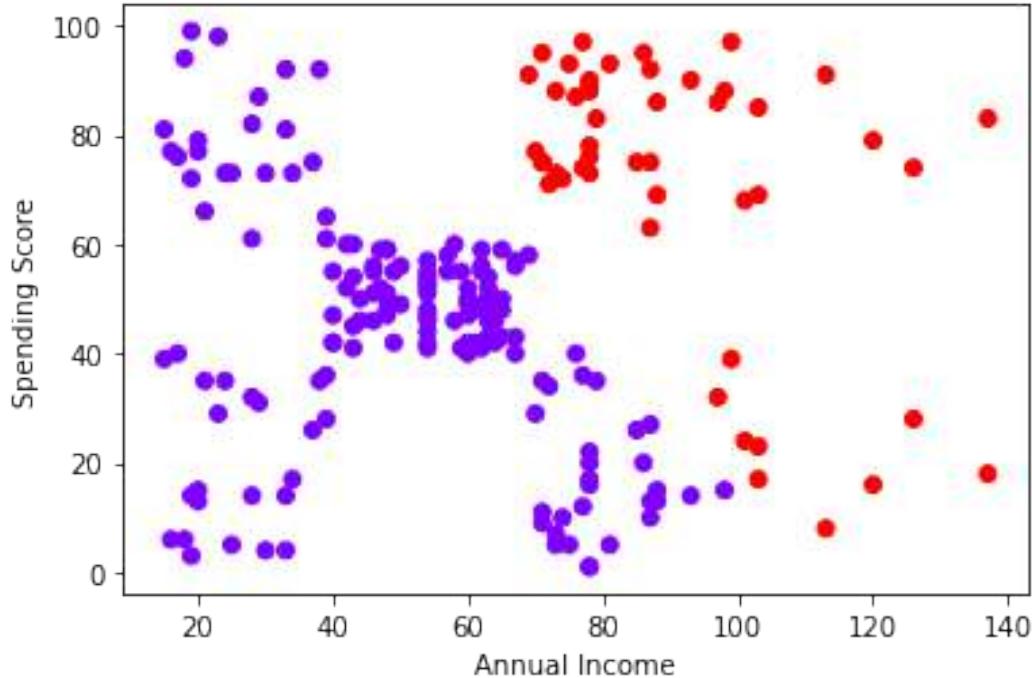


#Creating 2 clusters

```
km2=KMeans(n_clusters=2)
km2.fit(X)
km2_pred=km2.predict(X)

plt.scatter(X[:,0], X[:,1], c=km2_pred, cmap='rainbow')
plt.xlabel('Annual Income')
plt.ylabel('Spending Score')

Text(0, 0.5, 'Spending Score')
```



#Evaluating KMeans clustering algorithm

*#inertia: metric to evaluate clustering algo on the basis of no. of clusters
#It is the within cluster sum of square of a distance to the cluster centroid*

#Algo aims to choose the centroid that minimizes the inertia

```
print('Inertia when cluster=2:', km2.inertia_)
print('Inertia when cluster=3:', km3.inertia_)
print('Inertia when cluster=4:', km4.inertia_)
print('Inertia when cluster=5:', km.inertia_)
```

```
Inertia when cluster=2: 183653.32894736843
Inertia when cluster=3: 106348.37306211118
Inertia when cluster=4: 73679.78903948834
Inertia when cluster=5: 44448.45544793371
```

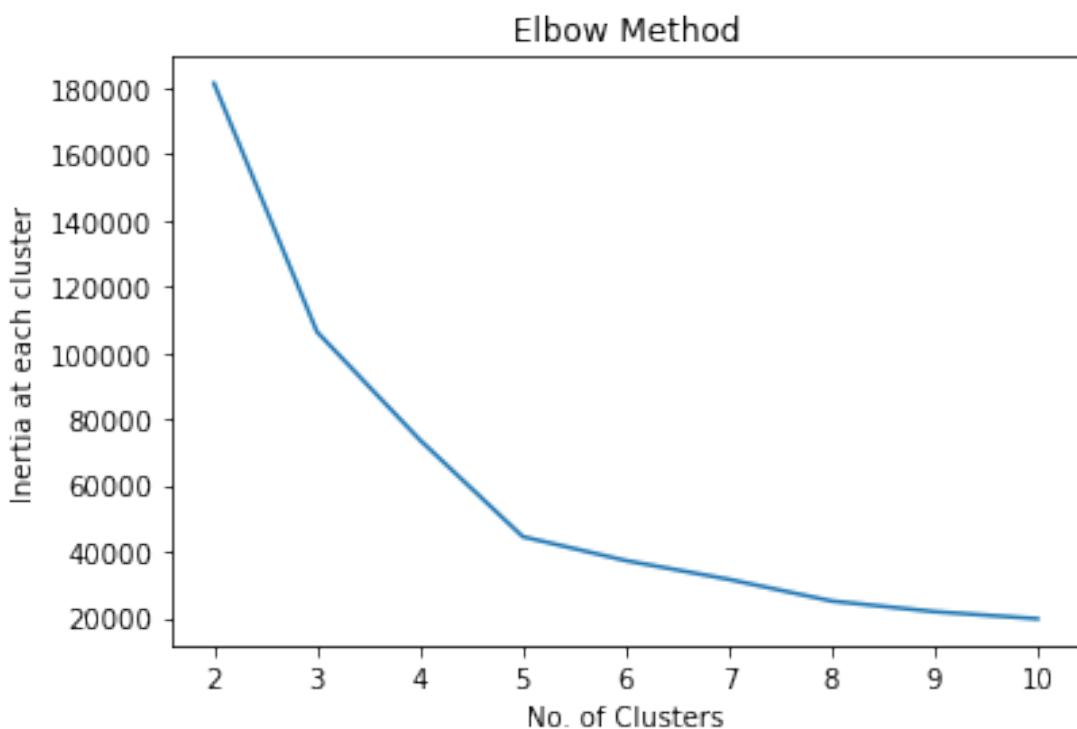
#Elbow method: used to find the optimal no. of clusters for a given data

```
E=[]
#Creating 2 to 10 clusters on the data
for i in range(2,11):
    km_i=KMeans(n_clusters=i)
    km_i.fit(X)
```

```
E.append(km_i.inertia_)
print(E)
[181363.59595959596, 106348.37306211118, 73679.78903948834,
44448.45544793371, 37233.81451071001, 31573.960664122747,
25018.576334776335, 21806.812998695455, 19664.685196005543]
```

#Plotting the graph for Elbow method

```
plt.plot(range(2,11), E)
plt.title('Elbow Method')
plt.xlabel('No. of Clusters')
plt.ylabel('Inertia at each cluster')
plt.show()
```



#The elbow shape ends at cluster=5

#So, the optimal no. of cluster for this data is 5



- [Feature Importance](#)
- [Advantages](#)
- [The Correlation matrix](#)
- [Displaying correlation matrix as heatmap using seaborn](#)
- [Linear Regression Feature Importance](#)
- [Code Snippet illustrating feature importances](#)
- [Decision Tree Feature Importance](#)

Feature Importance



- Feature selection is the process of identifying and selecting a subset of input features that are most relevant to the target variable.
- The scores are useful and can be used in a range of situations in a predictive modeling problem, such as:
 - Better understanding the data.
 - Better understanding a model.
 - Reducing the number of input features.

Advantages



- Feature importance scores can provide insight into the dataset.
- Feature importance scores can provide insight into the model.
- Feature importance can be used to improve a predictive model.

The Correlation matrix



- The correlation matrix can be used to estimate the linear historical relationship between the returns of multiple assets. You can use the built-in .corr() method on a pandas DataFrame to easily calculate the correlation matrix.
- Example: Correlation matrix for Admission Prediction problem

```
import pandas as pd

df = pd.read_csv("Admission_Predict.csv",sep = ",")

#printing correlation matrix

df.corr()
```

Displaying correlation matrix as heatmap using seaborn



```
import seaborn as sns  
  
import matplotlib.pyplot as plt  
  
plt.figure(figsize=(10, 10))  
  
sns.heatmap(df.corr(), annot=True, linewidths=0.05, fmt=' .2f', cmap="magma")  
  
plt.show()
```

Coefficients as Feature Importance



- Linear machine learning algorithms fit a model where the prediction is the weighted sum of the input values. Such as Linear Regression and Logistic Regression.

- All of these algorithms find a set of coefficients to use in the weighted sum in order to make a prediction. These coefficients can be used directly as a crude type of feature importance score.

Linear Regression Feature Importance



- The main job of feature selection is to reduce the number of input variables when developing a predictive model. Feature selection for numerical data can be easily obtained by using the Pearson's correlation coefficient.
- Example: Calculating `model.coef_` for admission prediction problem

```
#feature importance  
  
importance = model.coef_  
  
print('Coefficient:', importance)
```



- `Model.coef_`: It is used to estimate the coefficients for the linear regression problem.
- We can fit a `LinearRegression` model on the regression dataset and retrieve the `coeff_` property that contains the coefficients found for each input variable.
- These coefficients can provide the basis for a crude feature importance score.
- This assumes that the input variables have the same scale or have been scaled prior to fitting a model.

Code Snippet illustrating feature importances



- Printing coefficient for Linear Regression column wise

```
important_feature= pd.DataFrame({'Feature Value': lr.coef_ }, index=X.columns). sort_values  
(by='Feature Value', ascending=False)
```

```
important_feature
```

- Note: coef_ can also be applied to Logistic Regression model



□ **Fitting the LinearRegression usings four features having higher feature value**

```
lr.fit(X_train[['CGPA', 'LOR ', 'TOEFL Score', 'GRE Score']], y_train)
```

```
lr.score(X_test[['CGPA', 'LOR ', 'TOEFL Score', 'GRE Score']], y_test)
```

□ **Fitting the LinearRegression usings five features having higher feature value**

```
lr.fit(X_train[['CGPA', 'LOR ', 'TOEFL Score', 'GRE Score', 'University Rating']], y_train)
```

```
lr.score(X_test[['CGPA', 'LOR ', 'TOEFL Score', 'GRE Score', 'University Rating']], y_test)
```

Decision Tree Feature Importance



- Decision tree algorithms like classification and regression trees (CART) offer importance scores based on the reduction in the criterion used to select split points, like Gini or entropy.
- The model provides a `feature_importances_` property that can be accessed to retrieve the relative importance scores for each input feature.



- Printing feature importance for decision tree classifier column wise

```
important_feature= pd.DataFrame ({'Feature Value': dtf.feature_importances_}, index=X.columns).  
sort_values (by='Feature Value', ascending=False)
```

important_feature

- Note: We can obtain the feature_importances_ for Random Forest Regression and Classification, XGBoost Regression and Classification and KNeighborsClassifier algorithms.

```
#Classification Problem
```

```
#Loan Prediction Problem
```

```
#Reading historical data
```

```
import pandas as pd
```

```
df=pd.read_csv(r'D:\BigData_All_content\csv\train_loan.csv')  
df.head()
```

```
   Loan_ID  Gender Married Dependents      Education Self_Employed \
0  LP001002    Male     No        0       Graduate           No
1  LP001003    Male    Yes        1       Graduate           No
2  LP001005    Male    Yes        0       Graduate          Yes
3  LP001006    Male    Yes        0    Not Graduate          No
4  LP001008    Male     No        0       Graduate          No
```

```
  ApplicantIncome CoapplicantIncome  LoanAmount  Loan_Amount_Term \
0            5849                 0.0        NaN          360.0
1            4583                1508.0      128.0         360.0
2            3000                 0.0        66.0         360.0
3            2583                2358.0      120.0         360.0
4            6000                 0.0        141.0        360.0
```

```
  Credit_History Property_Area Loan_Status
0             1.0      Urban          Y
1             1.0      Rural          N
2             1.0      Urban          Y
3             1.0      Urban          Y
4             1.0      Urban          Y
```

```
df.describe()
```

```
   ApplicantIncome  CoapplicantIncome  LoanAmount  
Loan_Amount_Term \
count      614.000000      614.000000      592.000000  
600.000000
mean      5403.459283     1621.245798     146.412162  
342.000000
std       6109.041673     2926.248369     85.587325  
65.12041
min       150.000000      0.000000      9.000000  
12.000000
25%      2877.500000      0.000000     100.000000  
360.000000
50%      3812.500000     1188.500000     128.000000  
360.000000
75%      5795.000000     2297.250000     168.000000  
360.000000
max      81000.000000    41667.000000     700.000000
```

```
480.00000
```

```
Credit_History
count      564.000000
mean       0.842199
std        0.364878
min        0.000000
25%        1.000000
50%        1.000000
75%        1.000000
max        1.000000
```

```
df.dtypes
```

```
Loan_ID          object
Gender          object
Married         object
Dependents     object
Education       object
Self_Employed   object
ApplicantIncome int64
CoapplicantIncome float64
LoanAmount      float64
Loan_Amount_Term float64
Credit_History  float64
Property_Area   object
Loan_Status     object
dtype: object
```

```
#Dropping 'Loan_ID' column from the df
```

```
df.drop(['Loan_ID'], axis=1, inplace=True)
df.head()
```

	Gender	Married	Dependents	ApplicantIncome	Education	Self_Employed
0	Male	No	0	5849	Graduate	No
1	Male	Yes	1	4583	Graduate	No
2	Male	Yes	0	3000	Graduate	Yes
3	Male	Yes	0	2583	Not Graduate	No
4	Male	No	0	6000	Graduate	No

	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History
0	0.0	NaN	360.0	1.0
1	1508.0	128.0	360.0	1.0
2	0.0	66.0	360.0	1.0

```
3          2358.0      120.0      360.0      1.0
4           0.0       141.0     360.0      1.0
```

```
Property_Area  Loan_Status
0      Urban        Y
1     Rural         N
2      Urban        Y
3      Urban        Y
4      Urban        Y
```

#Prepare X

```
X=df.drop(['Loan_Status'], axis=1)
X.head()
```

```
Gender Married Dependents      Education Self_Employed
ApplicantIncome \
0   Male     No          0      Graduate        No
5849
1   Male    Yes          1      Graduate        No
4583
2   Male    Yes          0      Graduate       Yes
3000
3   Male    Yes          0  Not Graduate        No
2583
4   Male     No          0      Graduate        No
6000
```

```
CoapplicantIncome  LoanAmount  Loan_Amount_Term Credit_History \
0            0.0        NaN        360.0      1.0
1          1508.0      128.0        360.0      1.0
2            0.0        66.0        360.0      1.0
3          2358.0      120.0        360.0      1.0
4            0.0       141.0        360.0      1.0
```

```
Property_Area
0      Urban
1     Rural
2      Urban
3      Urban
4      Urban
```

X.shape

(614, 11)

#Prepare y

```
y=df['Loan_Status']
y.head()
```

```
0    Y  
1    N  
2    Y  
3    Y  
4    Y  
Name: Loan_Status, dtype: object
```

#Identifying missing values in X

```
X.isnull().sum()
```

```
Gender           13  
Married          3  
Dependents      15  
Education         0  
Self_Employed   32  
ApplicantIncome   0  
CoapplicantIncome 0  
LoanAmount       22  
Loan_Amount_Term 14  
Credit_History    50  
Property_Area     0  
dtype: int64
```

#Checking data types of each feature/column of X

```
X.dtypes
```

```
Gender            object  
Married           object  
Dependents        object  
Education          object  
Self_Employed     object  
ApplicantIncome    int64  
CoapplicantIncome  float64  
LoanAmount         float64  
Loan_Amount_Term  float64  
Credit_History     float64  
Property_Area      object  
dtype: object
```

#filling missing data in 'Gender' column/feature

#counting frequency of each category in 'Gender'
X.Gender.value_counts()

```
Male      489  
Female    112  
Name: Gender, dtype: int64
```

```
#Filling missing data in 'Gender' with 'Male'  
X.Gender.fillna('Male', inplace=True)
```

```
X.isnull().sum()
```

```
Gender          0  
Married         3  
Dependents     15  
Education       0  
Self_Employed   32  
ApplicantIncome  0  
CoapplicantIncome 0  
LoanAmount      22  
Loan_Amount_Term 14  
Credit_History   50  
Property_Area    0  
dtype: int64
```

```
#Filling missing data in 'Married'
```

```
X.Married.value_counts()
```

```
Yes    398  
No     213  
Name: Married, dtype: int64
```

```
#Filling missing data in 'Married' with 'Yes'  
X.Married.fillna('Yes', inplace=True)
```

```
X.isnull().sum()
```

```
Gender          0  
Married         0  
Dependents     15  
Education       0  
Self_Employed   32  
ApplicantIncome  0  
CoapplicantIncome 0  
LoanAmount      22  
Loan_Amount_Term 14  
Credit_History   50  
Property_Area    0  
dtype: int64
```

```
#Filling missing data in 'Dependents'
```

```
X.Dependents.value_counts()
```

```
0    345  
1    102  
2    101
```

```
3+      51
Name: Dependents, dtype: int64

#Filling missing data in 'Dependents'

X.Dependents.fillna('0', inplace=True)

X.isnull().sum()

Gender          0
Married         0
Dependents      0
Education        0
Self_Employed   32
ApplicantIncome  0
CoapplicantIncome 0
LoanAmount       22
Loan_Amount_Term 14
Credit_History   50
Property_Area    0
dtype: int64

#Filling missing data in 'Self_Employed'

X.Self_Employed.value_counts()

No      500
Yes     82
Name: Self_Employed, dtype: int64

#Filling missing data in 'Self_Employed'

X.Self_Employed.fillna('No', inplace=True)

X.isnull().sum()

Gender          0
Married         0
Dependents      0
Education        0
Self_Employed   0
ApplicantIncome  0
CoapplicantIncome 0
LoanAmount       22
Loan_Amount_Term 14
Credit_History   50
Property_Area    0
dtype: int64

#filling missing data in 'LoanAmount'

import numpy as np
```

```
X.LoanAmount.fillna(np.mean(X.LoanAmount), inplace=True)
X.isnull().sum()
```

```
Gender          0
Married         0
Dependents      0
Education        0
Self_Employed   0
ApplicantIncome  0
CoapplicantIncome 0
LoanAmount       0
Loan_Amount_Term 14
Credit_History    50
Property_Area     0
dtype: int64
```

#filling missing data in 'Loan_Amount_Term'

```
X.Loan_Amount_Term.fillna(np.mean(X.Loan_Amount_Term), inplace=True)
X.isnull().sum()
```

```
Gender          0
Married         0
Dependents      0
Education        0
Self_Employed   0
ApplicantIncome  0
CoapplicantIncome 0
LoanAmount       0
Loan_Amount_Term  0
Credit_History    50
Property_Area     0
dtype: int64
```

#filling missing data in 'Credit_History'

```
X.Credit_History.value_counts()
```

```
1.0    475
0.0    89
Name: Credit_History, dtype: int64
```

#Filling missing data in 'Credit_History'

```
X.Credit_History.fillna(1.0, inplace=True)
X.isnull().sum()
```

```
Gender          0
Married         0
Dependents      0
```

```
Education          0
Self_Employed     0
ApplicantIncome    0
CoapplicantIncome  0
LoanAmount         0
Loan_Amount_Term   0
Credit_History     0
Property_Area      0
dtype: int64
```

```
X.dtypes
```

```
Gender            object
Married           object
Dependents        object
Education         object
Self_Employed     object
ApplicantIncome   int64
CoapplicantIncome float64
LoanAmount        float64
Loan_Amount_Term  float64
Credit_History    float64
Property_Area     object
dtype: object
```

#Encoding categorical features/columns of X into numerical type using one-hot encoding

```
X=pd.get_dummies(X)
X.head()
```

```
   ApplicantIncome  CoapplicantIncome  LoanAmount  Loan_Amount_Term \
0             5849              0.0  146.412162            360.0
1             4583             1508.0  128.000000            360.0
2             3000              0.0   66.000000            360.0
3             2583             2358.0  120.000000            360.0
4             6000              0.0  141.000000            360.0

   Credit_History  Gender_Female  Gender_Male  Married_No  Married_Yes \
0             1.0              0           1           1           0
1             1.0              0           1           0           1
2             1.0              0           1           0           1
3             1.0              0           1           0           1
4             1.0              0           1           1           0
```

```

Dependents_0 Dependents_1 Dependents_2 Dependents_3+ \
0           1           0           0           0
1           0           1           0           0
2           1           0           0           0
3           1           0           0           0
4           1           0           0           0

Education_Graduate Education_Not Graduate Self_Employed_No \
0                 1                     0           1
1                 1                     0           1
2                 1                     0           0
3                 0                     1           1
4                 1                     0           1

Self_Employed_Yes Property_Area_Rural Property_Area_Semiurban \
0                   0                     0           0
1                   0                     1           0
2                   1                     0           0
3                   0                     0           0
4                   0                     0           0

Property_Area_Urban
0                   1
1                   0
2                   1
3                   1
4                   1

```

#Splitting the data in : Testing set and Training set

```

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test=train_test_split(X,y, test_size=0.2)

print(X_train.shape)
print(y_train.shape)

(491, 20)
(491,)

print(X_test.shape)
print(y_test.shape)

(123, 20)
(123,)

```

#Ensemble Learning: Applying more than one Machine Learning algorithms on given problem.

#Means, Training the model using More than one ML algorithm

```
#Applying classification ML algorithms
```

```
#Applying Logistic Regression
```

```
from sklearn.linear_model import LogisticRegression

LR=LogisticRegression()

LR.fit(X_train, y_train)

C:\Users\hp\Anaconda3\lib\site-packages\sklearn\linear_model\
logistic.py:432: FutureWarning: Default solver will be changed to
'lbfgs' in 0.22. Specify a solver to silence this warning.
  FutureWarning)

LogisticRegression(C=1.0, class_weight=None, dual=False,
fit_intercept=True,
           intercept_scaling=1, l1_ratio=None, max_iter=100,
           multi_class='warn', n_jobs=None, penalty='l2',
           random_state=None, solver='warn', tol=0.0001,
verbose=0,
           warm_start=False)
```

```
#Applying Support Vector Classifier
```

```
from sklearn.svm import SVC

svc=SVC()

svc.fit(X_train, y_train)

C:\Users\hp\Anaconda3\lib\site-packages\sklearn\svm\base.py:193:
FutureWarning: The default value of gamma will change from 'auto' to
'scale' in version 0.22 to account better for unscaled features. Set
gamma explicitly to 'auto' or 'scale' to avoid this warning.
  "avoid this warning.", FutureWarning)

SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma='auto_deprecated',
    kernel='rbf', max_iter=-1, probability=False, random_state=None,
    shrinking=True, tol=0.001, verbose=False)
```

```
#Applying DecisionTreeClassifier
```

```
from sklearn.tree import DecisionTreeClassifier

dtc=DecisionTreeClassifier()

dtc.fit(X_train, y_train)
```

```

DecisionTreeClassifier(class_weight=None, criterion='gini',
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=False,
           random_state=None, splitter='best')

#Applying RandomForestClassifier

from sklearn.ensemble import RandomForestClassifier

rfc=RandomForestClassifier()

rfc.fit(X_train, y_train)

C:\Users\hp\Anaconda3\lib\site-packages\sklearn\ensemble\
forest.py:245: FutureWarning: The default value of n_estimators will
change from 10 in version 0.20 to 100 in 0.22.
  "10 in version 0.20 to 100 in 0.22.", FutureWarning)

RandomForestClassifier(bootstrap=True, class_weight=None,
criterion='gini',
           max_depth=None, max_features='auto',
max_leaf_nodes=None,
           min_impurity_decrease=0.0,
min_impurity_split=None,
           min_samples_leaf=1, min_samples_split=2,
           min_weight_fraction_leaf=0.0, n_estimators=10,
           n_jobs=None, oob_score=False,
random_state=None,
           verbose=0, warm_start=False)

#Applying GaussianNB

from sklearn.naive_bayes import GaussianNB

gnb=GaussianNB()
gnb.fit(X_train, y_train)

GaussianNB(priors=None, var_smoothing=1e-09)

#Score

print('Score of LogisticRegression:', LR.score(X_test, y_test))
print('Score of Support Vector Classifier:', svc.score(X_test,
y_test))
print('Score of DecisionTreeClassifier:', dtc.score(X_test, y_test))
print('Score of RandomForestClassifier:', rfc.score(X_test, y_test))
print('Score of GaussianNB:', gnb.score(X_test, y_test))

```

```
Score of LogisticRegression: 0.8130081300813008
Score of Support Vector Classifier: 0.6016260162601627
Score of DecisionTreeClassifier: 0.6585365853658537
Score of RandomForestClassifier: 0.7235772357723578
Score of GaussianNB: 0.7967479674796748
```

#Printing the important features for decision tree classifier

```
dtc.feature_importances_
array([0.21387793, 0.1045502 , 0.16122561, 0.06850944, 0.26221736,
       0.0080577 , 0.          , 0.01275678, 0.01138905, 0.03576537,
       0.01716328, 0.00816466, 0.01410417, 0.01476468, 0.00486179,
       0.0081095 , 0.01881304, 0.01374639, 0.00772881, 0.01419423])

imp_dtc=pd.DataFrame({'Feature value':dtc.feature_importances_},
index=X_train.columns)
imp_dtc.sort_values(by='Feature value', ascending=False)

      Feature value
Credit_History           0.262217
ApplicantIncome           0.213878
LoanAmount                0.161226
CoapplicantIncome          0.104550
Loan_Amount_Term           0.068509
Dependents_0               0.035765
Self_Employed_Yes           0.018813
Dependents_1               0.017163
Education_Graduate          0.014765
Property_Area_Urban         0.014194
Dependents_3+               0.014104
Property_Area_Rural          0.013746
Married_No                  0.012757
Married_Yes                  0.011389
Dependents_2               0.008165
Self_Employed_No              0.008110
Gender_Female                 0.008058
Property_Area_Semiurban        0.007729
Education_Not Graduate        0.004862
Gender_Male                  0.000000
```

#Identifying important features for RandomForestclassifier

```
imp_rfc=pd.DataFrame({'Feature value':rfc.feature_importances_},
index=X_train.columns)
imp_rfc.sort_values(by='Feature value', ascending=False)
```

	Feature value
LoanAmount	0.211584
ApplicantIncome	0.196210
Credit_History	0.194369
CoapplicantIncome	0.113777
Loan_Amount_Term	0.047461
Property_Area_Rural	0.028494
Dependents_0	0.026482
Property_Area_Semiurban	0.022842
Property_Area_Urban	0.019794
Dependents_1	0.016982
Married_Yes	0.016415
Education_Not Graduate	0.015595
Education_Graduate	0.014442
Self_Employed_Yes	0.012526
Married_No	0.012477
Dependents_2	0.012312
Self_Employed_No	0.010178
Dependents_3+	0.009840
Gender_Female	0.009203
Gender_Male	0.009016

```
#Regression Problem
```

```
#Admission Prediction Problem
```

```
import pandas as pd
```

```
df=pd.read_csv(r'D:\BigData_All_content\csv\Admission_Predict.csv')
```

```
#head(): will return first 5 records of the dataframe  
df.head()
```

CGPA \ Serial No.	GRE Score	TOEFL Score	University Rating	SOP	LOR
9.65	1	337	118	4	4.5
8.87	2	324	107	4	4.0
8.00	3	316	104	3	3.0
8.67	4	322	110	3	3.5
8.21	5	314	103	2	2.0

	Research	Chance of Admit
0	1	0.92
1	1	0.76
2	1	0.72
3	1	0.80
4	0	0.65

```
df.describe()
```

SOP \ Serial No.	GRE Score	TOEFL Score	University Rating
count	400.000000	400.000000	400.000000
mean	200.500000	316.807500	3.087500
std	115.614301	11.473646	1.143728
min	1.000000	290.000000	1.000000
25%	100.750000	308.000000	2.000000
50%	200.500000	317.000000	3.000000
75%	300.250000	325.000000	4.000000

```

4.000000
max    400.000000  340.000000   120.000000           5.000000
5.000000

```

	LOR	CGPA	Research	Chance of Admit
count	400.000000	400.000000	400.000000	400.000000
mean	3.452500	8.598925	0.547500	0.724350
std	0.898478	0.596317	0.498362	0.142609
min	1.000000	6.800000	0.000000	0.340000
25%	3.000000	8.170000	0.000000	0.640000
50%	3.500000	8.610000	1.000000	0.730000
75%	4.000000	9.062500	1.000000	0.830000
max	5.000000	9.920000	1.000000	0.970000

#Dropping Serial No. feature

```

df.drop(['Serial No.'], axis=1, inplace=True)
df.head()

```

	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research	
0	337	118		4	4.5	4.5	9.65	1
1	324	107		4	4.0	4.5	8.87	1
2	316	104		3	3.0	3.5	8.00	1
3	322	110		3	3.5	2.5	8.67	1
4	314	103		2	2.0	3.0	8.21	0

	Chance of Admit
0	0.92
1	0.76
2	0.72
3	0.80
4	0.65

*#Preparing correlation matrix on df using corr()
#Syntax: dataframe.corr()*

```

df.corr()

```

	GRE Score	TOEFL Score	University Rating	SOP
GRE Score	1.000000	0.835977		0.668976 0.612831
TOEFL Score	0.835977	1.000000		0.695590 0.657981

University Rating	0.668976	0.695590	1.000000	0.734523
SOP	0.612831	0.657981	0.734523	1.000000
LOR	0.557555	0.567721	0.660123	0.729593
CGPA	0.833060	0.828417	0.746479	0.718144
Research	0.580391	0.489858	0.447783	0.444029
Chance of Admit	0.802610	0.791594	0.711250	0.675732

	LOR	CGPA	Research	Chance of Admit
GRE Score	0.557555	0.833060	0.580391	0.802610
TOEFL Score	0.567721	0.828417	0.489858	0.791594
University Rating	0.660123	0.746479	0.447783	0.711250
SOP	0.729593	0.718144	0.444029	0.675732
LOR	1.000000	0.670211	0.396859	0.669889
CGPA	0.670211	1.000000	0.521654	0.873289
Research	0.396859	0.521654	1.000000	0.553202
Chance of Admit	0.669889	0.873289	0.553202	1.000000

#Correlation matrix is analyzed on the basis of outcome/target variable/feature
#of your data

#Basic use of correlation matrix is to identify the key/important features upon
#which outcome/target feature highly depends

#Observation
#On the basis of this matrix, we can say that 'Chance of Admit' highly depends on
#'CGPA'

#Now, we can easily indentify top/key/important features from our data

#Top 5 features are:'CGPA', 'GRE Score', 'TOEFL Score', 'University Rating'
#&'SOP'

#Prepare independent feature X

```
X=df.drop(['Chance of Admit'], axis=1)
X.head()
```

```
GRE Score  TOEFL Score  University Rating  SOP  LOR  CGPA  Research
0          337           118                  4    4.5  4.5  9.65   1
1          324           107                  4    4.0  4.5  8.87   1
2          316           104                  3    3.0  3.5  8.00   1
3          322           110                  3    3.5  2.5  8.67   1
4          314           103                  2    2.0  3.0  8.21   0
```

```
df.head()
```

```
GRE Score  TOEFL Score  University Rating  SOP  LOR  CGPA  Research
\0          337           118                  4    4.5  4.5  9.65   1
1          324           107                  4    4.0  4.5  8.87   1
2          316           104                  3    3.0  3.5  8.00   1
3          322           110                  3    3.5  2.5  8.67   1
4          314           103                  2    2.0  3.0  8.21   0
```

```
Chance of Admit
0          0.92
1          0.76
2          0.72
3          0.80
4          0.65
```

```
#Prepare dependent feature/outcome feature y
```

```
y=df['Chance of Admit']
y.head()

0      0.92
1      0.76
2      0.72
3      0.80
4      0.65
Name: Chance of Admit, dtype: float64
```

```
#Checking missing data in X
```

```
X.isnull().sum()
```

```
GRE Score      0
TOEFL Score    0
University Rating 0
SOP            0
LOR            0
CGPA           0
```

```
Research          0  
dtype: int64
```

```
X.describe()
```

	GRE Score	TOEFL Score	University Rating	SOP
LOR \				
count	400.000000	400.000000	400.000000	400.000000
mean	316.807500	107.410000	3.087500	3.400000
std	11.473646	6.069514	1.143728	1.006869
min	290.000000	92.000000	1.000000	1.000000
25%	308.000000	103.000000	2.000000	2.500000
50%	317.000000	107.000000	3.000000	3.500000
75%	325.000000	112.000000	4.000000	4.000000
max	340.000000	120.000000	5.000000	5.000000

	CGPA	Research
count	400.000000	400.000000
mean	8.598925	0.547500
std	0.596317	0.498362
min	6.800000	0.000000
25%	8.170000	0.000000
50%	8.610000	1.000000
75%	9.062500	1.000000
max	9.920000	1.000000

```
X.head()
```

	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research
0	337	118		4	4.5	4.5	9.65
1	324	107		4	4.0	4.5	8.87
2	316	104		3	3.0	3.5	8.00
3	322	110		3	3.5	2.5	8.67
4	314	103		2	2.0	3.0	8.21

```
#Feature scaling of numerical features using MinMaxScaler
```

```
from sklearn.preprocessing import MinMaxScaler
```

```
mx=MinMaxScaler(feature_range=(0,1))
```

```
#Scaling all features/columns of X
```

```

X[X.columns]=mx.fit_transform(X[X.columns])
X.head()

    GRE Score  TOEFL Score  University Rating      SOP      LOR      CGPA
Research
0           0.94       0.928571                  0.75  0.875  0.875  0.913462
1.0
1           0.68       0.535714                  0.75  0.750  0.875  0.663462
1.0
2           0.52       0.428571                  0.50  0.500  0.625  0.384615
1.0
3           0.64       0.642857                  0.50  0.625  0.375  0.599359
1.0
4           0.48       0.392857                  0.25  0.250  0.500  0.451923
0.0

#splitting the data into training

from sklearn.model_selection import train_test_split

#test_size represents the size of data tobe used for testing

X_train, X_test, y_train, y_test=train_test_split(X,y, test_size=0.20)

#Total data=400

#X_train= 80% data of X= 320
#X_test= 20% data of X= 80

#y_train= 80% data of y= 320
#y_test= 20% data of y= 80

print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)

(320, 7)
(80, 7)
(320,)
(80,)

#Now our data is ready to apply Machine Learning algorithms/models

#We are using LinearRegression algo/mode to solve this regression
problem

from sklearn.linear_model import LinearRegression

```

```
#Creating the object of LinearRegression algo/model
lr=LinearRegression()

#Training the LinearRegression algo/model using X_train, y_train

lr.fit(X_train, y_train)

LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None,
normalize=False)

#Testing the LinearRegression Algo/Model

#To view the score the LinearRegression algo/model

lr.score(X_test, y_test)

#Here, score is ~80%
#It means that the LinearRegression algo predicts the value of y_test
on the basis of X_test with
#80% of accuracy
```

0.7765049784922724

```
#After analyzing the correlation matrix, we have indentified the top 5
key features
```

```
 #'CGPA', 'GRE Score', 'TOEFL Score', 'University Rating' &'SOP'
```

```
#Now, training the LinearRegression model using these 5 key features
```

```
lr1=LinearRegression()
lr1.fit(X_train[['CGPA', 'GRE Score', 'TOEFL Score', 'University
Rating', 'SOP']],y_train)
```

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None,
normalize=False)
```

```
#Score of LinearRegression for top 5 features
```

```
lr1.score(X_test[['CGPA', 'GRE Score', 'TOEFL Score', 'University
Rating', 'SOP']], y_test)
```

0.76124525696099

```
#If score is not satisfactory, you can improve the score by
introducing more features.
```

```
lr2=LinearRegression()
lr2.fit(X_train[['CGPA', 'GRE Score', 'TOEFL Score', 'University
Rating', 'SOP', 'LOR']],y_train)
```

```

lr2.score(X_test[['CGPA', 'GRE Score', 'TOEFL Score', 'University Rating', 'SOP', 'LOR']], y_test)

0.7742513761414436

#You can also access the important features as per LinearRegression

#We can identify the key features of LinearRegression by using coef_property

lr.coef_

array([ 0.06618215,  0.09519411,  0.01707127, -0.00709768,
       0.0887192 ,
       0.38114545,  0.0270981 ])

X_train.columns

Index(['GRE Score', 'TOEFL Score', 'University Rating', 'SOP', 'LOR',
       'CGPA',
       'Research'],
      dtype='object')

feat=lr.coef_

imp=pd.DataFrame({'Feature value':feat}, index=X_train.columns)
imp

          Feature value
GRE Score           0.066182
TOEFL Score          0.095194
University Rating     0.017071
SOP                  -0.007098
LOR                  0.088719
CGPA                 0.381145
Research              0.027098

imp.sort_values(by='Feature value', ascending=False)

          Feature value
CGPA                 0.381145
TOEFL Score          0.095194
LOR                  0.088719
GRE Score             0.066182
Research              0.027098
University Rating     0.017071
SOP                  -0.007098

```



- [Ensemble learning](#)
- [Observation](#)
- [Model Assumption](#)
- [Basic Ensemble Structure](#)
- [Commonly used Ensemble learning techniques](#)
- [Bagging](#)
- [Boosting](#)
- [Stacking](#)

Ensemble learning



- In the world of Statistics and Machine Learning, Ensemble learning techniques attempt to make the performance of the predictive models better by improving their accuracy. Ensemble Learning is a process using which multiple machine learning models (such as classifiers) are strategically constructed to solve a particular problem.
- Ensemble learning is the process by which multiple models, such as classifiers or experts, are strategically generated and combined to solve a particular computational intelligence problem. Ensemble learning is primarily used to improve the classification, prediction, function, approximation, etc.

Ensemble Learning



□ Example: I want to invest in a company XYZ. I am not sure about its performance though. So, I look for advice on whether the stock price will increase more than 6% per annum or not? I decide to approach various experts having diverse domain experience:

□ **Employee of Company XYZ**

This person knows the internal functionality of the company and has the insider information about the functionality of the firm. But he lacks a broader perspective on how are competitors innovating, how is the technology evolving and what will be the impact of this evolution on Company XYZ's product. In the past, he has been right 70% times.

□ **Financial Advisor of Company XYZ**

This person has a broader perspective on how companies strategy will fair of in this competitive environment. However, he lacks a view on how the company's internal policies are fairing off. In the past, he has been right 75% times.



Stock Market Trader

This person has observed the company's stock price over past 3 years. He knows the seasonality trends and how the overall market is performing. He also has developed a strong intuition on how stocks might vary over time. **In the past, he has been right 70% times.**

Employee of a competitor

This person knows the internal functionality of the competitor firms and is aware of certain changes which are yet to be brought. He lacks a sight of company in focus and the external factors which can relate the growth of competitor with the company of subject. **In the past, he has been right 60% of times.**



- **Market Research team in same segment**
- This team analyzes the customer preference of company XYZ's product over others and how is this changing with time. Because he deals with customer side, he is unaware of the changes company XYZ will bring because of alignment to its own goals. In the past, they have been right 75% of times.
- **Social Media Expert**
- This person can help us understand how has company XYZ has positioned its products in the market. And how are the sentiment of customers changing over time towards company. He is unaware of any kind of details beyond digital marketing. In the past, he has been right 65% of times.

Observation



- In a scenario when all the 6 experts/teams verify that it's a good decision (assuming all the predictions are independent of each other), we will get a combined accuracy rate of

$$1 - 30\% * 25\% * 30\% * 40\% * 25\% * 35\%$$

$$= 1 - 0.07875 = 99.92125\%$$

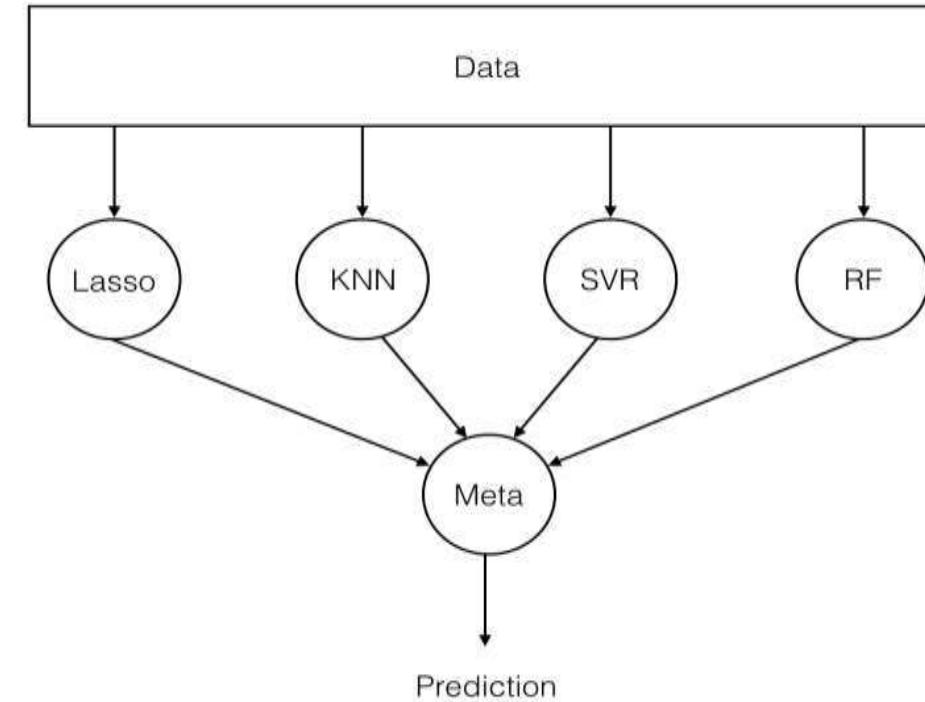
- The assumption used here that all the predictions are completely independent is slightly extreme as they are expected to be correlated. However, you can see how we can be so sure by combining various forecasts together.
- An ensemble is the art of combining a diverse set of learners (individual models) together to improvise on the stability and predictive power of the model. In the above example, the way we combine all the predictions collectively will be termed as Ensemble learning.

Model Assumption



- **Models can be different from each other for a variety of reasons:**
- There can be difference in the population of data.
- There can be a different modeling technique used.
- There can be a different hypothesis.

Basic Ensemble Structure

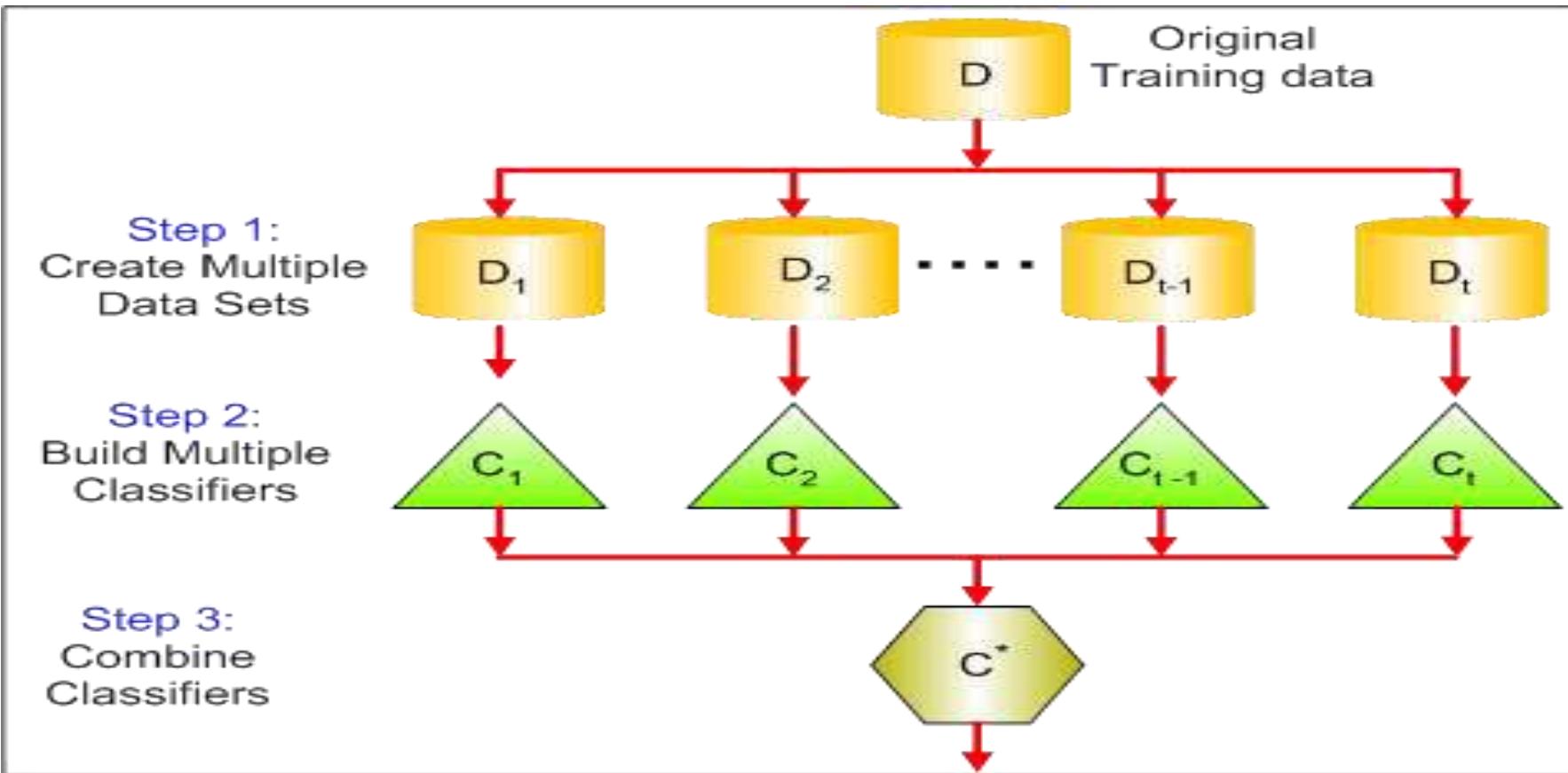


Commonly used Ensemble learning techniques



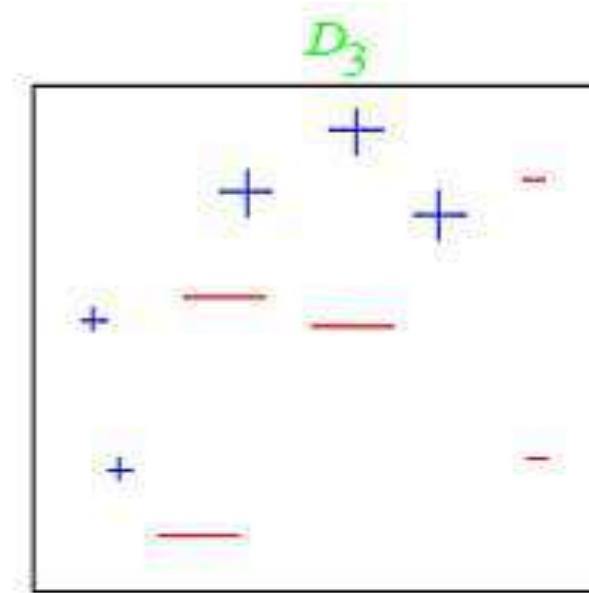
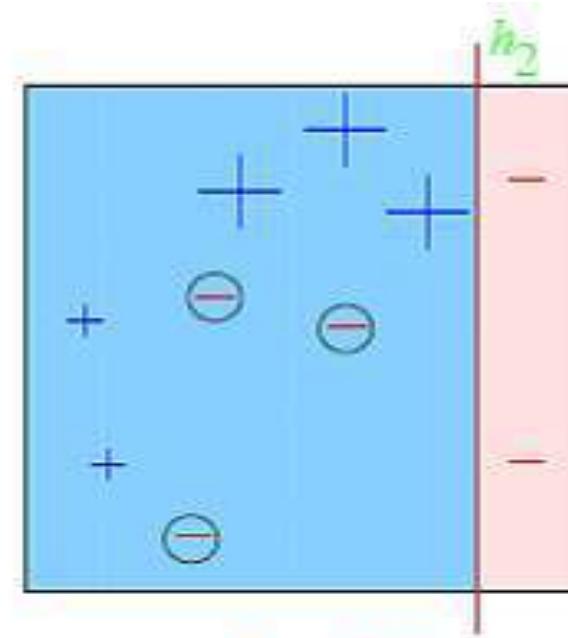
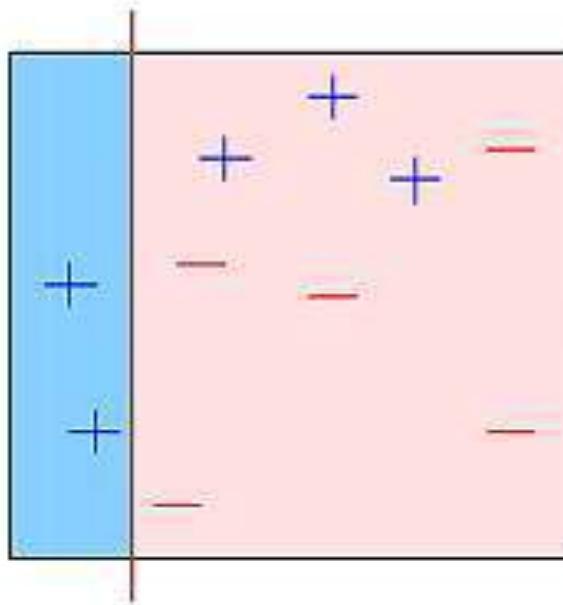
□ Bagging

Bagging tries to implement similar learners on small sample populations and then takes a mean of all the predictions. In generalized bagging, you can use different learners on different population. As you can expect this helps us to reduce the variance error.



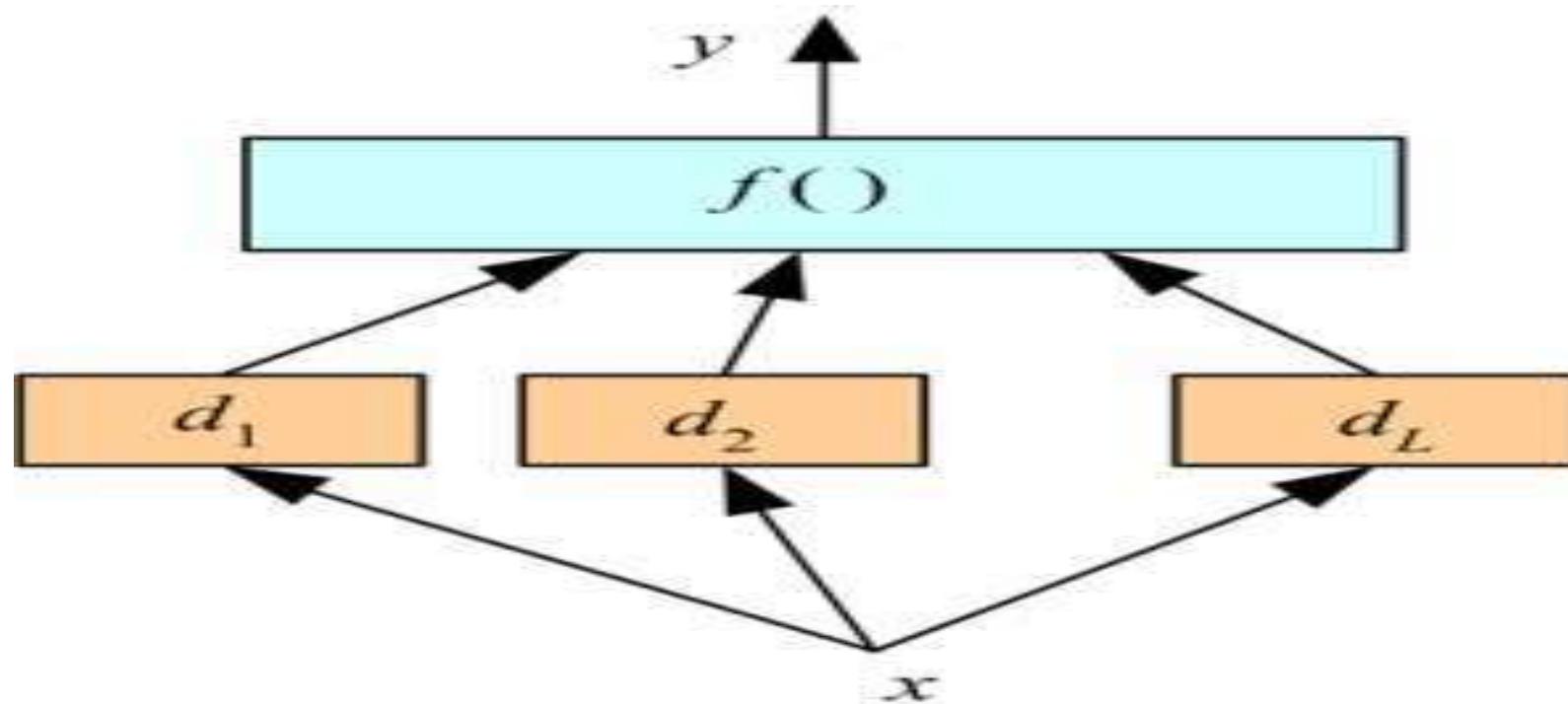


- Boosting is an iterative technique which adjusts the weight of an observation based on the last classification. If an observation was classified incorrectly, it tries to increase the weight of this observation and vice versa. Boosting in general decreases the bias error and builds strong predictive models. However, they may sometimes over fit on the training data.





- This is a very interesting way of combining models. Here we use a learner to combine output from different learners. This can lead to decrease in either bias or variance error depending on the combining learner we use.





- General speaking, ensemble learning is the technique of applying more than one Machine Learning algorithms on given problem.

```
#Machine Learning for classification problem
```

```
#Loan Prediction Problem
```

```
import pandas as pd
```

```
#reading the historical data
```

```
df_loan=pd.read_csv(r'D:\BigData_All_content\csv\train_loan.csv')  
df_loan.head()
```

```
    Loan_ID Gender Married Dependents Education Self_Employed \
0   LP001002   Male     No       0      Graduate        No
1   LP001003   Male    Yes       1      Graduate        No
2   LP001005   Male    Yes       0      Graduate       Yes
3   LP001006   Male    Yes       0  Not Graduate        No
4   LP001008   Male     No       0      Graduate        No

    ApplicantIncome CoapplicantIncome  LoanAmount  Loan_Amount_Term \
0           5849             0.0          NaN            360.0
1           4583            1508.0         128.0            360.0
2           3000             0.0           66.0            360.0
3           2583            2358.0         120.0            360.0
4           6000             0.0           141.0            360.0

    Credit_History Property_Area Loan_Status
0              1.0      Urban          Y
1              1.0      Rural          N
2              1.0      Urban          Y
3              1.0      Urban          Y
4              1.0      Urban          Y
```

```
#To view description of numerical data
```

```
df_loan.describe()
```

```
    ApplicantIncome CoapplicantIncome  LoanAmount
Loan_Amount_Term \
count       614.000000          614.000000  592.000000
600.000000
mean        5403.459283         1621.245798  146.412162
342.000000
std         6109.041673         2926.248369   85.587325
65.12041
min         150.000000          0.000000    9.000000
12.000000
25%        2877.500000          0.000000  100.000000
360.000000
50%        3812.500000         1188.500000  128.000000
360.000000
75%        5795.000000         2297.250000  168.000000
360.000000
max        81000.000000        41667.000000  700.000000
```

480.00000

```
Credit_History
count      564.000000
mean       0.842199
std        0.364878
min        0.000000
25%        1.000000
50%        1.000000
75%        1.000000
max        1.000000
```

#To view the summary of categorical data only
df_loan.describe(include='object')

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	\
count	614	601	611	599	614	582	
unique	614	2	2	4	2	2	
top	LP002556	Male	Yes	0	Graduate	No	
freq	1	489	398	345	480	500	

	Property_Area	Loan_Status
count	614	614
unique	3	2
top	Semiurban	Y
freq	233	422

#To view summary of both numerical and categorical

df_loan.describe(include='all')

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	\
count	614	601	611	599	614	582	
unique	614	2	2	4	2	2	
top	LP002556	Male	Yes	0	Graduate	No	
freq	1	489	398	345	480	500	
mean	NaN	NaN	NaN	NaN	NaN	NaN	
std	NaN	NaN	NaN	NaN	NaN	NaN	
min	NaN	NaN	NaN	NaN	NaN	NaN	
25%	NaN	NaN	NaN	NaN	NaN	NaN	
50%	NaN	NaN	NaN	NaN	NaN	NaN	
75%	NaN	NaN	NaN	NaN	NaN	NaN	
max	NaN	NaN	NaN	NaN	NaN	NaN	

	ApplicantIncome	CoapplicantIncome	LoanAmount
Loan_Amount_Term	\		
count	614.000000	614.000000	592.000000
600.000000			
unique	NaN	NaN	NaN
NaN			
top	NaN	NaN	NaN

NaN			
freq		NaN	
NaN			
mean	5403.459283	1621.245798	146.412162
342.00000			
std	6109.041673	2926.248369	85.587325
65.12041			
min	150.000000	0.000000	9.000000
12.00000			
25%	2877.500000	0.000000	100.000000
360.00000			
50%	3812.500000	1188.500000	128.000000
360.00000			
75%	5795.000000	2297.250000	168.000000
360.00000			
max	81000.000000	41667.000000	700.000000
480.00000			

	Credit_History	Property_Area	Loan_Status
count	564.000000	614	614
unique	NaN	3	2
top	Nan	Semiurban	Y
freq	NaN	233	422
mean	0.842199	NaN	NaN
std	0.364878	NaN	NaN
min	0.000000	NaN	NaN
25%	1.000000	NaN	NaN
50%	1.000000	NaN	NaN
75%	1.000000	NaN	NaN
max	1.000000	NaN	NaN

#Preparing correlation matrix of df_loan
df_loan.corr()

	ApplicantIncome	CoapplicantIncome	LoanAmount	\
ApplicantIncome	1.000000	-0.116605	0.570909	
CoapplicantIncome	-0.116605	1.000000	0.188619	
LoanAmount	0.570909	0.188619	1.000000	
Loan_Amount_Term	-0.045306	-0.059878	0.039447	
Credit_History	-0.014715	-0.002056	-0.008433	

	Loan_Amount_Term	Credit_History
ApplicantIncome	-0.045306	-0.014715
CoapplicantIncome	-0.059878	-0.002056
LoanAmount	0.039447	-0.008433
Loan_Amount_Term	1.000000	0.001470
Credit_History	0.001470	1.000000

df_loan.columns

```

Index(['Loan_ID', 'Gender', 'Married', 'Dependents', 'Education',
       'Self_Employed', 'ApplicantIncome', 'CoapplicantIncome',
       'LoanAmount',
       'Loan_Amount_Term', 'Credit_History', 'Property_Area',
       'Loan_Status'],
      dtype='object')

#Prepare X
X=df_loan.drop(['Loan_ID', 'Loan_Status'], axis=1)
X.head()

   Gender Married Dependents          Education Self_Employed
ApplicantIncome \
0    Male     No         0        Graduate        No
5849
1    Male    Yes         1        Graduate        No
4583
2    Male    Yes         0        Graduate       Yes
3000
3    Male    Yes         0  Not Graduate        No
2583
4    Male     No         0        Graduate        No
6000

   CoapplicantIncome  LoanAmount  Loan_Amount_Term Credit_History \
0            0.0        NaN           360.0          1.0
1          1508.0       128.0           360.0          1.0
2            0.0        66.0           360.0          1.0
3          2358.0       120.0           360.0          1.0
4            0.0       141.0           360.0          1.0

   Property_Area
0      Urban
1      Rural
2      Urban
3      Urban
4      Urban

#prepare y
y=df_loan['Loan_Status']
y.head()

0    Y
1    N
2    Y
3    Y
4    Y
Name: Loan_Status, dtype: object

```

```
#checking missing values
```

```
X.isnull().sum()
```

```
Gender          13
Married         3
Dependents     15
Education       0
Self_Employed  32
ApplicantIncome 0
CoapplicantIncome 0
LoanAmount      22
Loan_Amount_Term 14
Credit_History  50
Property_Area   0
dtype: int64
```

```
#Fill missing values in categorical column
```

```
#Filling missing data in 'Gender'
```

```
#First, count frequency of each category in 'Gender'
```

```
X['Gender'].value_counts()
```

```
Male      489
Female    112
Name: Gender, dtype: int64
```

```
#Replace missing values in 'Gender' with highest frequency data i.e.  
'Male'
```

```
X.Gender.fillna('Male', inplace=True)
```

```
X.isnull().sum()
```

```
Gender          0
Married         3
Dependents     15
Education       0
Self_Employed  32
ApplicantIncome 0
CoapplicantIncome 0
LoanAmount      22
Loan_Amount_Term 14
Credit_History  50
Property_Area   0
dtype: int64
```

```
#Filling values in 'Married'

X.Married.value_counts()

Yes    398
No     213
Name: Married, dtype: int64

#Replacing missing data with 'Yes' in 'Married'

X.Married.fillna('Yes', inplace=True)

X.isnull().sum()

Gender          0
Married         0
Dependents     15
Education       0
Self_Employed   32
ApplicantIncome  0
CoapplicantIncome 0
LoanAmount      22
Loan_Amount_Term 14
Credit_History   50
Property_Area    0
dtype: int64

#Filling values in 'Dependents'

X.Dependents.value_counts()

0    345
1    102
2    101
3+   51
Name: Dependents, dtype: int64

#Replacing missing data with '0' in 'Dependents'

X.Dependents.fillna('0', inplace=True)

X.isnull().sum()

Gender          0
Married         0
Dependents     0
Education       0
Self_Employed   32
ApplicantIncome  0
CoapplicantIncome 0
LoanAmount      22
```

```
Loan_Amount_Term      14  
Credit_History        50  
Property_Area         0  
dtype: int64
```

#Filling values in 'Self_Employed'

```
X.Self_Employed.value_counts()
```

```
No      500  
Yes     82  
Name: Self_Employed, dtype: int64
```

#Replacing missing data with 'No' in 'Self_Employed'

```
X.Self_Employed.fillna('No', inplace=True)
```

```
X.isnull().sum()
```

```
Gender          0  
Married         0  
Dependents     0  
Education       0  
Self_Employed   0  
ApplicantIncome 0  
CoapplicantIncome 0  
LoanAmount      22  
Loan_Amount_Term 14  
Credit_History   50  
Property_Area    0  
dtype: int64
```

#Now, filling missing data in Numerical value columns/features

```
import numpy as np
```

#Filling missing values in 'LoanAmount' with thier mean value

```
X.LoanAmount.fillna(np.mean(X.LoanAmount), inplace=True)
```

```
X.isnull().sum()
```

```
Gender          0  
Married         0  
Dependents     0  
Education       0  
Self_Employed   0  
ApplicantIncome 0  
CoapplicantIncome 0  
LoanAmount      0
```

```
Loan_Amount_Term      14
Credit_History        50
Property_Area         0
dtype: int64

#Filling missing values in 'Loan_Amount_Term' with thier mean value
X.Loan_Amount_Term.fillna(np.mean(X.Loan_Amount_Term), inplace=True)

X.isnull().sum()

Gender                  0
Married                 0
Dependents              0
Education               0
Self_Employed           0
ApplicantIncome          0
CoapplicantIncome         0
LoanAmount               0
Loan_Amount_Term          0
Credit_History            50
Property_Area             0
dtype: int64

#filling missing data in 'Credit History'

X.Credit_History.value_counts()

1.0      475
0.0       89
Name: Credit_History, dtype: int64

#filling missing data in 'Credit_History' with 1

X.Credit_History.fillna(1, inplace=True)

X.isnull().sum()

Gender                  0
Married                 0
Dependents              0
Education               0
Self_Employed           0
ApplicantIncome          0
CoapplicantIncome         0
LoanAmount               0
Loan_Amount_Term          0
Credit_History            0
Property_Area             0
dtype: int64
```

```
#All the features/columns in X must be numeric to apply machine learning algorithm/model
```

```
#Now, encode the categorical feature/columns into numerical one
```

```
#We are using one-hot encoding to encode the categorical features/columns
```

```
X=pd.get_dummies(X)
```

```
X.head()
```

```
ApplicantIncome CoapplicantIncome LoanAmount Loan_Amount_Term \
0 5849 0.0 146.412162 360.0
1 4583 1508.0 128.000000 360.0
2 3000 0.0 66.000000 360.0
3 2583 2358.0 120.000000 360.0
4 6000 0.0 141.000000 360.0
```

```
Credit_History Gender_Female Gender_Male Married_No Married_Yes \
0 1.0 0 1 1 0
1 1.0 0 1 0 1
2 1.0 0 1 0 1
3 1.0 0 1 0 1
4 1.0 0 1 1 0
```

```
Dependents_0 Dependents_1 Dependents_2 Dependents_3+ \
0 1 0 0 0
1 0 1 0 0
2 1 0 0 0
3 1 0 0 0
4 1 0 0 0
```

```
Education_Graduate Education_Not_Graduate Self_Employed_No \
0 1 0 1
1 1 0 1
2 1 0 0
3 0 1 1
4 1 0 1
```

```
Self_Employed_Yes Property_Area_Rural Property_Area_Semiurban \
0 0 0 0
```

```
1          0          1          0
2          1          0          0
3          0          0          0
4          0          0          0
```

Property_Area_Urban

```
0          1
1          0
2          1
3          1
4          1
```

X.dtypes

```
ApplicantIncome           int64
CoapplicantIncome         float64
LoanAmount                float64
Loan_Amount_Term          float64
Credit_History             float64
Gender_Female              uint8
Gender_Male                uint8
Married_No                  uint8
Married_Yes                  uint8
Dependents_0                  uint8
Dependents_1                  uint8
Dependents_2                  uint8
Dependents_3+                  uint8
Education_Graduate          uint8
Education_Not Graduate       uint8
Self_Employed_No             uint8
Self_Employed_Yes             uint8
Property_Area_Rural          uint8
Property_Area_Semiurban        uint8
Property_Area_Urban          uint8
dtype: object
```

X.columns

```
Index(['ApplicantIncome', 'CoapplicantIncome', 'LoanAmount',
       'Loan_Amount_Term', 'Credit_History', 'Gender_Female',
       'Gender_Male',
       'Married_No', 'Married_Yes', 'Dependents_0', 'Dependents_1',
       'Dependents_2', 'Dependents_3+', 'Education_Graduate',
       'Education_Not Graduate', 'Self_Employed_No',
       'Self_Employed_Yes',
       'Property_Area_Rural', 'Property_Area_Semiurban',
       'Property_Area_Urban'],
      dtype='object')
```

#splitting the X into training set and testing set

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test=train_test_split(X,y, test_size=0.25)

print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)

(460, 20)
(154, 20)
(460,)
(154,)

#Applying machine learning classificaton algorithms on given data

#Applying LogisticRegression

from sklearn.linear_model import LogisticRegression

LR=LogisticRegression()

LR.fit(X_train, y_train)

C:\Users\hp\Anaconda3\lib\site-packages\sklearn\linear_model\
logistic.py:432: FutureWarning: Default solver will be changed to
'lbfgs' in 0.22. Specify a solver to silence this warning.
  FutureWarning)

LogisticRegression(C=1.0, class_weight=None, dual=False,
fit_intercept=True,
           intercept_scaling=1, l1_ratio=None, max_iter=100,
           multi_class='warn', n_jobs=None, penalty='l2',
           random_state=None, solver='warn', tol=0.0001,
verbose=0,
           warm_start=False)

#Applying Support Vector Classifier (SVC)

from sklearn.svm import SVC
svc=SVC()

svc.fit(X_train, y_train)

C:\Users\hp\Anaconda3\lib\site-packages\sklearn\svm\base.py:193:
FutureWarning: The default value of gamma will change from 'auto' to
'scale' in version 0.22 to account better for unscaled features. Set
gamma explicitly to 'auto' or 'scale' to avoid this warning.
  "avoid this warning.", FutureWarning)
```

```
SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,  
decision_function_shape='ovr', degree=3, gamma='auto_deprecated',  
kernel='rbf', max_iter=-1, probability=False, random_state=None,  
shrinking=True, tol=0.001, verbose=False)
```

#Applying Decision Tree Classifier

```
from sklearn.tree import DecisionTreeClassifier  
  
dtc=DecisionTreeClassifier()  
  
dtc.fit(X_train, y_train)  
  
DecisionTreeClassifier(class_weight=None, criterion='gini',  
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=False,  
random_state=None, splitter='best')
```

#Applying GaussianNB

```
from sklearn.naive_bayes import GaussianNB  
  
nb=GaussianNB()  
  
nb.fit(X_train, y_train)  
GaussianNB(priors=None, var_smoothing=1e-09)
```

#Applying RandomForestClassifier

```
from sklearn.ensemble import RandomForestClassifier  
  
rfc=RandomForestClassifier()  
  
rfc.fit(X_train, y_train)  
  
C:\Users\hp\Anaconda3\lib\site-packages\sklearn\ensemble\  
forest.py:245: FutureWarning: The default value of n_estimators will  
change from 10 in version 0.20 to 100 in 0.22.  
"10 in version 0.20 to 100 in 0.22.", FutureWarning)  
  
RandomForestClassifier(bootstrap=True, class_weight=None,  
criterion='gini',  
          max_depth=None, max_features='auto',  
max_leaf_nodes=None,  
          min_impurity_decrease=0.0,  
min_impurity_split=None,
```

```

        min_samples_leaf=1, min_samples_split=2,
        min_weight_fraction_leaf=0.0, n_estimators=10,
        n_jobs=None, oob_score=False,
random_state=None,
                verbose=0, warm_start=False)

print('Score of LogisticRegression:', LR.score(X_test, y_test))
print('Score of SVC:', svc.score(X_test, y_test))
print('Score of Decision Tree Classifier:', dtc.score(X_test, y_test))
print('Score of GaussianNB:', nb.score(X_test, y_test))
print('Score of RandomForestClassifier:', rfc.score(X_test, y_test))

Score of LogisticRegression: 0.8116883116883117
Score of SVC: 0.7077922077922078
Score of Decision Tree Classifier: 0.7402597402597403
Score of GaussianNB: 0.8051948051948052
Score of RandomForestClassifier: 0.7597402597402597

from sklearn.metrics import confusion_matrix

yp=LR.predict(X_test)
lr_conf=confusion_matrix(y_test, yp)
lr_conf

array([[ 22,  24],
       [  5, 103]], dtype=int64)

df_loan.dtypes

#Saving the finalized ML model

#We can save the finalized ML model. So that, we can transfer the saved model to another team

#We are using pickle pkg for saving the model

import pickle

#Ex: Saving the LogisticRegression Model because it has highest score and optimal confusion matrix

#1. Specifying the model name and its location

filename='d:\\Logistic_reg.model'

#Here, .model is not mandatory. You can use simple

#filename='d:\\Logistic_reg'

#I am using .model to identify the saved model easily

```

```
#Next, Specify the content of the Logistic_reg.model file  
#content=(object-of-finalized-ML-model, data)  
content=(LR, X)  
  
#Here, LR is the object of finalized ML model i.e. LogisticRegression  
and X is the data.  
  
#Writing the model and data on disk i.e. saving the finalized model  
  
#pickle.dump(content, filename): It is used to save the model on disk.  
It will take two arguments:  
  
#First argument will be your content to be dumped/saved  
#Second argument will be file name in which data is to be  
dumped/saved.  
  
#Note: You can write in any file only when you have opened the file  
#open(file_name, file_mode): used to open a file  
  
#Open the file specified by filename for writing (w) in binary (b)  
mode  
  
pickle.dump(content, open(filename, 'wb'))
```

```

#Loading/Reading the saved model

import pickle

#pickle.load(): used to load/read the saved model
#Opening the file in read mode

m, d=pickle.load(open('d:\\Logistic_reg.model', 'rb'))

#Here, m will store the object of finalized/saved model
#d, will store saved data

#m=LR
#d=X

type(d)

pandas.core.frame.DataFrame

import pandas as pd
import numpy as np
from tkinter import *
from tkinter.ttk import *
import pickle

class App(object):
    def __init__(self):

        self.root=Tk()
        self.root.geometry('900x600')
        self.root.resizable(height=True,width=True)
        self.root.title('Loan Prediction Problem')
        self.root.configure(background='lightblue')

        self.gender_lb=Label(self.root,background='lightblue',foreground='black', text="What is your gender (Male or Female)",width=40, font=('Times New Roman',16))
        self.gender_lb.grid(row=3,column=0, sticky='w')

        self.gender=Entry(self.root)
        self.gender.grid(row=3,column=1)

self.married_lb=Label(self.root,background='lightblue',foreground='black', text="Married (Enter Yes or No)",width=40, font=('Times New

```

```
Roman',16))
    self.married_lb.grid(row=4,column=0,sticky='w')

    self.married=Entry(self.root)
    self.married.grid(row=4,column=1)

self.depends_lb=Label(self.root,background='lightblue',foreground='black', text="Enter dependents value (Enter 0 or 1 or 2 or 3+)",width=40, font=('Times New Roman',16))
    self.depends_lb.grid(row=5,column=0,sticky='w')

    self.depends=Entry(self.root)
    self.depends.grid(row=5,column=1)

self.edu_lb=Label(self.root,background='lightblue',foreground='black', text="Enter Education (Enter Graduate or Not Graduate)",width=40, font=('Times New Roman',16))
    self.edu_lb.grid(row=6,column=0,sticky='w')

    self.edu=Entry(self.root)
    self.edu.grid(row=6,column=1)

self.emp_lb=Label(self.root,background='lightblue',foreground='black', text="Self Employed (Enter Yes or No)",width=40, font=('Times New Roman',16))
    self.emp_lb.grid(row=7,column=0,sticky='w')

    self.emp=Entry(self.root)
    self.emp.grid(row=7,column=1)

self.app_lb=Label(self.root,background='lightblue',foreground='black', text="Enter Applicant Income (150 to 81000)",width=40, font=('Times New Roman',16))
    self.app_lb.grid(row=8,column=0,sticky='w')

    self.app=Entry(self.root)
    self.app.grid(row=8,column=1)

self.coapp_lb=Label(self.root,background='lightblue',foreground='black', text="Enter Co applicant Income",width=40, font=('Times New Roman',16))
    self.coapp_lb.grid(row=9,column=0,sticky='w')

    self.coapp=Entry(self.root)
    self.coapp.grid(row=9,column=1)
```

```
self.loanamt_lb=Label(self.root,background='lightblue',foreground='black', text="Enter Loan Amount",width=40, font=('Times New Roman',16))
    self.loanamt_lb.grid(row=10,column=0,sticky='w')

    self.loanamt=Entry(self.root)
    self.loanamt.grid(row=10,column=1)

self.loanterm_lb=Label(self.root,background='lightblue',foreground='black', text="Enter Loan Amount Term",width=40, font=('Times New Roman',16))
    self.loanterm_lb.grid(row=11,column=0,sticky='w')

    self.loanterm=Entry(self.root)
    self.loanterm.grid(row=11,column=1)

self.history_lb=Label(self.root,background='lightblue',foreground='black', text="Enter Credit History (0 or 1)",width=40, font=('Times New Roman',16))
    self.history_lb.grid(row=12,column=0,sticky='w')

    self.history=Entry(self.root)
    self.history.grid(row=12,column=1)

self.property_lb=Label(self.root,background='lightblue',foreground='black', text="Enter Property Area",width=40, font=('Times New Roman',16))
    self.property_lb.grid(row=13,column=0,sticky='w')

    self.property=Entry(self.root)
    self.property.grid(row=13,column=1)

    self.button3=Button(self.root,text="Submit",style='W.TButton',
command=self.prediction)
    self.button3.grid(row=14,column=1,sticky='w')

self.result_lb=Label(self.root,background='lightblue',foreground='black', text="Loan Status",width=40, font=('Times New Roman',16))
    self.result_lb.grid(row=16,column=0,sticky='w')

entry_text = StringVar()
    self.result=Entry(self.root, textvariable=entry_text,
width=50)
    self.result.grid(row=16,column=1)
```

```

        self.root.mainloop()

    def prediction(self):
        gender=self.gender.get()
        married=self.married.get()
        dependents=self.depends.get()
        Education=self.edu.get()
        SelfEmployed=self.emp.get()
        Applicantincome=int(self.app.get())
        coapplicantincome=float(self.coapp.get())

        loanamount=float(self.loantamt.get())
        loanamountterm=float(self.loanterm.get())
        credithistory=float(self.history.get())
        propertyarea=self.property.get()

        data =
[[gender,married,dependents,Education,SelfEmployed,Applicantincome,co
applicantincome,loanamount,loanamountterm,credithistory,propertyarea]]
        newdf = pd.DataFrame(data, columns =
['Gender','Married','Dependents','Education','Self_Employed','ApplicantIncome','CoapplicantIncome','LoanAmount','Loan_Amount_Term','Credit_History','Property_Area'])

        newdf = pd.get_dummies(newdf)
        missing_cols = set( d.columns ) - set( newdf.columns )
        for c in missing_cols:
            newdf[c] = 0

        newdf = newdf[d.columns]

        new_predict=m.predict(newdf)
        print(new_predict)
        a=""
        if (new_predict[0]=='Y'):
            a="Your Loan is approved, Please contact at HDFC Bank"
        else:
            a="Sorry ! Your Loan is not approved"
        self.result.delete(0, END)
        # insert new_text at position 0
        self.result.insert(0, a)

        #print(type(gre))

```

App()

['Y']

<__main__.App at 0x22609d6a908>