UNIT-1

Introduction to python

Python is a general-purpose interpreted, interactive, object-oriented, and high-level programming language. It was created by Guido van Rossum in February 1991.

Why to Learn Python?

Python is a high-level, interpreted, interactive and object-oriented scripting language. Python is designed to be highly readable. It uses English keywords frequently where as other languages use punctuation, and it has fewer syntactical constructions than other languages.

Python is a MUST for students and working professionals to become a great Software Engineer specially when they are working in Web Development Domain.

Advantages of python:

- **Python is Interpreted** − Python is processed at runtime by the interpreter. You do not need to compile your program before executing it. This is similar to PERL and PHP
- **Python is Interactive** − You can actually sit at a Python prompt and interact with the interpreter directly to write your programs.
- **Python is Object-Oriented** − Python supports Object-Oriented style or technique of programming that encapsulates code within objects.
- **Python is a Beginner's Language** − Python is a great language for the beginner-level programmers and supports the development of a wide range of applications from simple text processing to WWW browsers to games.

Characteristics of Python

Following are important characteristics of **Python Programming** −

- It supports functional and structured programming methods as well as OOP.
- It can be used as a scripting language or can be compiled to byte-code for building large applications.
- It provides very high-level dynamic data types and supports dynamic type checking.
- It supports automatic garbage collection.
- It can be easily integrated with C, C++, COM, ActiveX, CORBA, and Java.

History of Python

- Python was developed by Guido van Rossum at the National Research Institute for Mathematics and Computer Science in the Netherlands.
- Python is derived from many other languages, including ABC, Modula-3, C, C++, Algol-68, SmallTalk, and Unix shell and other scripting languages.
- Python is copyrighted. Like Perl, Python source code is now available under the GNU General Public License (GPL).
- Python is now maintained by a core development team at the institute, although Guido van Rossum still holds a vital role in directing its progress.

Modes of Python Program :

We can develop **a python program** in 2 different styles.

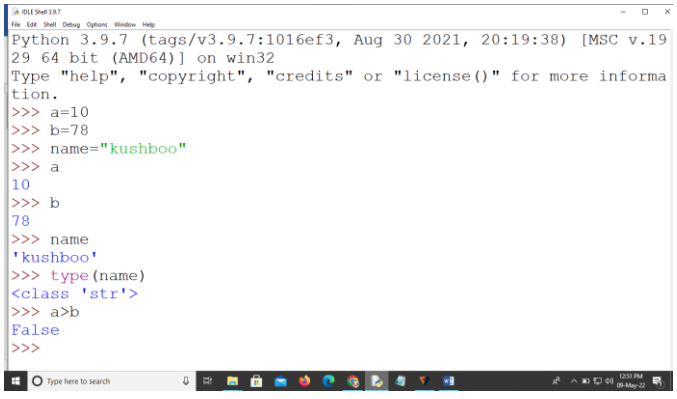
- Interactive Mode and
- Script Mode.

Interactive Mode

Interactive mode is a command line shell. Typically the interactive mode is used to test the features of the python, or to run a smaller script that may not be reusable.

Example for Interactive Mode :

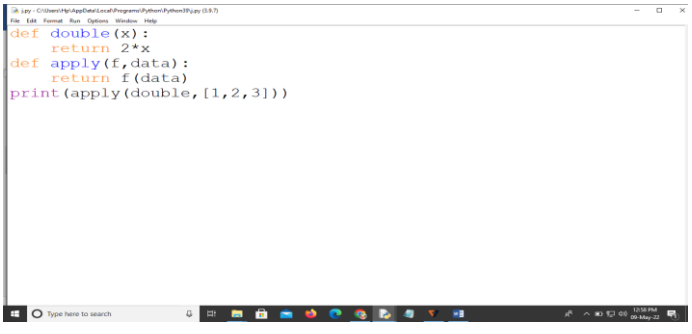
Open the command prompt, and go to the location which your python has been installed and hit the python command.



Script Mode:

Script mode is mainly used to develop business applications. In script mode, we can write a group of python statements in any one of the following editors or IDEs

Example for script mode:



Python Features

Python's features include −

- **Easy-to-learn** − Python has few keywords, simple structure, and a clearly defined syntax. This allows the student to pick up the language quickly.
- **Easy-to-read** − Python code is more clearly defined and visible to the eyes.

- **Easy-to-maintain** − Python's source code is fairly easy-to-maintain.
- **A broad standard library** − Python's bulk of the library is very portable and cross-platform compatible on UNIX, Windows, and Macintosh.
- **Interactive Mode** − Python has support for an interactive mode which allows interactive testing and debugging of snippets of code.
- **Portable** − Python can run on a wide variety of hardware platforms and has the same interface on all platforms.
- **Extendable** − You can add low-level modules to the Python interpreter. These modules enable programmers to add to or customize their tools to be more efficient.
- **Databases** − Python provides interfaces to all major commercial databases.
- **GUI Programming** − Python supports GUI applications that can be created and ported to many system calls, libraries and windows systems, such as Windows MFC, Macintosh, and the X Window system of Unix.
- **Scalable** − Python provides a better structure and support for large programs than shell scripting.
- **Free**- Python can be downloaded from internet freely without paying money.
- **Opensource**- Python is open source language and we can see the python code.
- **Advance features**- Python provides advanced features to develop machine learning programs easily.

Python - Variable Types

- Variables are nothing but reserved memory locations to store values. This means that when you create a variable you reserve some space in memory.
- Based on the data type of a variable, the interpreter allocates memory and decides what can be stored in the reserved memory.
- Therefore, by assigning different data types to variables, you can store integers, decimals or characters in these variables.

Assigning Values to Variables

- Python variables do not need explicit declaration to reserve memory space. The declaration happens automatically when you assign a value to a variable. The equal sign (=) is used to assign values to variables.
- The operand to the left of the = operator is the name of the variable and the operand to the right of the = operator is the value stored in the variable.

For example :

```
counter = 100          # An integer assignment
miles   = 1000.0       # A floating point
name    = "John"       # A string
```

Here, 100, 1000.0 and "John" are the values assigned to *counter*, *miles*, and *name* variables, respectively.

Multiple Assignment

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

For example :

a = b = c = 1

Here, an integer object is created with the value 1, and all three variables are assigned to the same memory location. You can also assign multiple objects to multiple variables.

For example :

a,b,c = 1,2,"john"

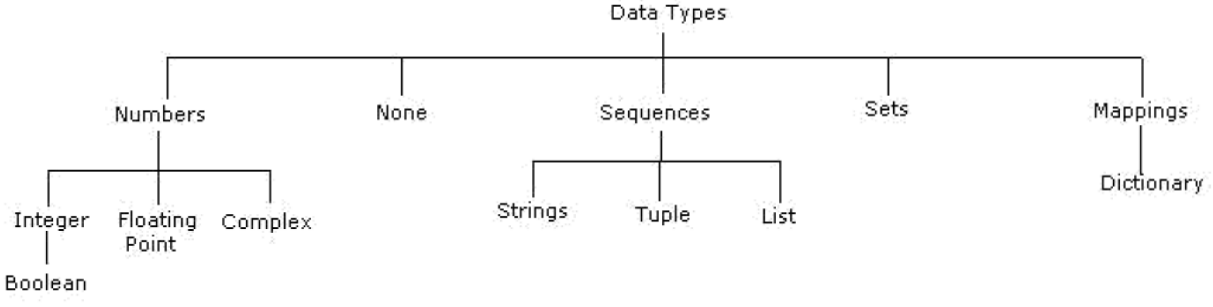
Here, two integer objects with values 1 and 2 are assigned to variables a and b respectively, and one string object with the value "john" is assigned to the variable c.

Standard Data Types of python:

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

Python has five standard data types :

- Numbers
- String
- List
- Tuple
- Dictionary
- Sets



Python Numbers

Number data types store numeric values. Number objects are created when you assign a value to them.

For example:

var1 = 1
var2 = 10

You can also delete the reference to a number object by using the del statement. The syntax of the del statement is:

del var1[,var2[,var3[....,varN]]]]

You can delete a single object or multiple objects by using the del statement. For

example :

del var
del var_a, var_b

Python supports four different numerical types −

- int (signed integers)
- long (long integers, they can also be represented in octal and hexadecimal)
- float (floating point real values)
- complex (complex numbers)

Examples

Here are some examples of numbers :

Int	Long	float	Complex
10	51924361L	0.0	3.14j
100	-0x19323L	15.20	4j
-786	0122L	-21.9	9.322-36j
-0490	535633629843L	-90.	-.6545+0j
-0x260	-052318172735L	-32.54e100	3+26j

- Python allows you to use a lowercase l with long, but it is recommended that you use only an uppercase L to avoid confusion with the number 1. Python displays long integers with an uppercase L.
- A complex number consists of an ordered pair of real floating-point numbers denoted by x + yj, where x and y are the real numbers and j is the imaginary unit.

Booleans

These represent the truth values False and True. Boolean values False and True behave like the values 0 and 1, respectively. To get the Boolean Equivalent of 0 or 1, you can type bool(0) or bool(1), python will return False or True respectively.

Python Dictionary:

- Python's dictionaries are kind of hash table type. They work like associative arrays or hashes found in Perl and consist of key-value pairs. A dictionary key can be any immutable Python type, usually numbers,strings and tuples.

- Dictionaries are enclosed by curly braces ({ }) and values can be assigned and accessed using square braces ([]).
- For example :

```
dict = {}
dict['one'] = "This is one"
```

```
dict[2]     = "This is two"
tinydict = {'name': 'john','code':6734, 'dept': 'sales'}
```

Sets

Sets are used to store multiple items in a single variable. A set is a collection which is *unordered* and *unindexed*.

Example:

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

Python Strings

- Strings in Python are identified as a contiguous set of characters represented in the quotation marks. Python allows for either pairs of single or double quotes.
- Subsets of strings can be taken using the slice operator ([ ] and [:]) with indexes starting at 0 in the beginning of the string and working their way from -1 at the end.
- The plus (+) sign is the string concatenation operator and the asterisk (*) is the repetition operator.

For example :

```
str = 'Hello World!'
```

Python Lists

- Lists are the mutable datatype of Python. A list contains items separated by commas and enclosed within square brackets ([]).
- To some extent, lists are similar to arrays in C. One difference between them is that all the items belonging to a list can be of different data type.
- The values stored in a list can be accessed using the index ([ ]).Indexes starting at 0 in the beginning of the list and working their way to end -1.
- Example :

```
list = [ 'abcd', 786 , 2.23, 'john', 70.2 ]
tinylist = [123, 'john']
```

Python Tuples

- A tuple is another sequence data type that is similar to the list. A tuple consists of a number of values separated by commas. Unlike lists, however, tuples are enclosed within parentheses.
- The main differences between lists and tuples are: Lists are enclosed in brackets ([ ]) and their elements and size can be changed, while tuples are enclosed in parentheses (( )) and cannot be updated. Tuples can be thought of as **read-only** lists.

For example :

```
tuple = ( 'abcd', 786 , 2.23, 'john', 70.2  )
tinytuple = (123, 'john')
```

Python - Basic Operators

Operators are the constructs which can manipulate the value of operands.

Consider the expression 4 + 5 = 9. Here, 4 and 5 are called operands and + is called operator.

Types of Operators:

Python language supports the following types of operators.

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

Arithmetic Operators

Assume variable a holds 10 and variable b holds 20, then :

Operator	Description	Example
tion	alues on either side of the operator.	30
action	cts right hand operand from left hand operand.	-10
iplication	lies values on either side of the operator	200
ion	s left hand operand by right hand operand	2
lulus	s left hand operand by right hand operand and retu der	0
onent	ns exponential (power) calculation on operator	10 to the power 20
Division	vision of operands where the result is the quotie the digits after the decimal point are removed. of the operands is negative, the result is floored d away from zero (towards negative infinity) −	4 and 9.0//2.0 = 4.0, -4, -11.0//3 = -4.0

Comparison Operators

These operators compare the values on either sides of them and decide the relation among them. They are also called Relational operators.

Assume variable a holds 10 and variable b holds 20, then −

Operator	Description	Example
	alues of two operands are equal, then the condition becomes tr	) is not true.
	es of two operands are not equal, then condition becomes true.	) is true.
	es of two operands are not equal, then condition becomes true.	) is true. This i to != operator.
	alue of left operand is greater than the value of right operand, t on becomes true.	is not true.
	alue of left operand is less than the value of right operand, ther on becomes true.	is true.
	alue of left operand is greater than or equal to the value of righ d, then condition becomes true.	) is not true.
	alue of left operand is less than or equal to the value of right d, then condition becomes true.	) is true.

Assignment Operators

Operator	Description	Example
	s values from right side operands to left side ope	b assigns value of a +
l AND	right operand to the left operand and assign the o left operand	is equivalent to c = c +
tract AND	acts right operand from the left operand and ass ılt to left operand	s equivalent to c = c –
ltiply AND	iplies right operand with the left operand and ass ılt to left operand	is equivalent to c = c *

de AND	es left operand with the right operand and assig o left operand	s equivalent to c = c / a
odulus AND	modulus using two operands and assign the res erand	is equivalent to c = c %
ponent AND	ns exponential (power) calculation on operators value to the left operand	is equivalent to c = c **
or Division	rms floor division on operators and assign value operand	is equivalent to c = c //

Bitwise Operators

Bitwise operator works on bits and performs bit by bit operation. Assume if a = 60; and b = 13; Now in the binary format their values will be 0011 1100 and 0000 1101 respectively. Following table lists out the bitwise operators supported by Python language with an example each in those, we use the above two variables (a and b) as operands .

a = 0011 1100

b = 0000 1101

---

a&b = 0000 1100

a|b = 0011 1101

a^b = 0011 0001

~a = 1100 0011

There are following Bitwise operators supported by Python language

Operator	Description	Example
ry AND	or copies a bit to the result if it exists in both ds	(means 0000 1100)
y OR	s a bit if it exists in either operand.	= 61 (means 0011 1101
y XOR	s the bit if it is set in one operand but not both	= 49 (means 0011 0001
y Ones Complemer	ary and has the effect of 'flipping' bits.	-61 (means 1100 0011 plement form due to a binary number.

ary Left Shift	t operands value is moved left by the number ecified by the right operand.	= 240 (means 1111 000
ary Right Shift	t operands value is moved right by the numbe ecified by the right operand.	= 15 (means 0000 1111

Logical Operators

There are following logical operators supported by Python language. Assume variable a holds 10 and variable b holds 20 then

Operator	Description	Example
gical AND	the operands are true then condition becomes tru	b) is true.
ical OR	of the two operands are non-zero then condition es true.	is true.
gical NOT	reverse the logical state of its operand.	nd b) is false.

Membership Operators

Python's membership operators test for membership in a sequence, such as strings, lists, or tuples. There are two membership operators as explained below −

Operator	Description	Example
	tes to true if it finds a variable in the specifi ce and false otherwise.	here in results in a 1 if x is a mem ence y.
	tes to true if it does not finds a variable in th ed sequence and false otherwise.	y, here not in results in a 1 if x is ber of sequence y.

Identity Operators

Identity operators compare the memory locations of two objects. There are two Identity operators explained below −

Operator	Description	Example
	tes to true if the variables on either side of t or point to the same object and false otherw	here **is** results in 1 if id(x) equals
	tes to false if the variables on either side of or point to the same object and true otherwi	y, here **is not** results in 1 if id(x) al to id(y).

Python - Files I/O

Printing to the Screen

- The simplest way to produce output is using the *print* statement where you can pass zero or more expressions separated by commas.
- This function converts the expressions you pass into a string and writes the result to standard output as follows:

```
print ("Python is really a great language,", "isn't it?")
```

This produces the following result on your standard screen .

Reading Keyboard Input

Developers often have a need to interact with users, either to get data or to provide some sort of result. Most programs today use a dialog box as a way of asking the user to provide some type of input. While Python provides us with two inbuilt functions to read the input from the keyboard.

1.input (prompt)
2.raw_input (prompt)

input() function

Python input() function is used to take the values from the user. This function is called to tell the program to stop and wait for the user to input the values. It is a built-in function.

The input() function is used in both the version of Python 2.x and Python 3.x. In Python 3.x, the input function explicitly converts the input you give to type string. But Python 2.x input function takes the value and type of the input you enter as it is without modifying the type.

Example program in Python3.x

```
on program to demonstrate# input() function in Python3.x

 input("Enter the name: ")

ıt the type of input value

ype(val1))
```

```
al1)

 input("Enter the number: ")

ype(val2))

 int(val2)

ype(val2))

al2)
```

Input and Output

Here, the value “python3” take from the user and store it in the **val1** variable. The type of the value stored is always string for input function only for Python 3.x. The value “1997” take from the user and store it in the variable **val2**. Now, the type of variable val2 is a string and we have to convert the type to an integer using int() function. The val2 variable stores the value “1997” as an integer type.

Example program in Python2.x

```
on program to demonstrate

:() function in Python2.x

 input("Enter the name: ")

ype(val1))

al1)

 input("Enter the number: ")

ype(val2))

al2)
```

raw_input() function

Python raw_input function is usedo get the values from the user. We call this function to tell the program to stop and wait for the user to input the values. It is a built-in function. The input function is **used only in Python 2.x** version. The Python 2.x has two functions to take the value from the user. The first one is input function and another one is raw_input() function.
The raw_input() function is similar to input() function in Python 3.x. Developers are recommended to use raw_input function in Python 2.x. Because there is a vulnerability in input function in Python 2.x version.

Example program in Python2

```
on program to demonstrate

:() function in Python2.x

 raw_input("Enter the name: ")

ype(val1))

al1)

 raw_input("Enter the number: ")

ype(val2))

 int(val2)

ype(val2))

al2)
```

Python Indentation

Python uses indentation to highlight the blocks of code. Whitespace is used for indentation in Python. All statements with the same distance to the right belong to the same block of code. If a block has to be more deeply nested, it is simply indented further to the right. You can understand it better by looking at the following lines of code.

```
n program showing
ation

ls'

'ppds':
t('welcome to the python langauge...')

t('retype .')
ll set !')
```

Output:

```
Logging on to geeksforgeeks...
All set !
```

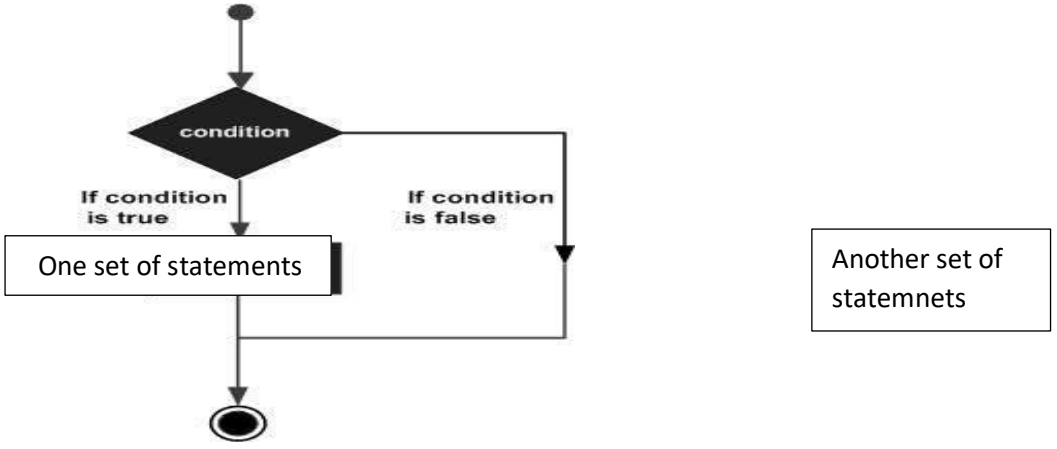
Control Statements

Control statements are classified into three type in python. They are:

1. Decision making statements
2. Looping statements
3. Jumping statements/loop control statements

1.Decision Making Statement:

• Conditional Statement in Python performs different computations or actions depending on whether a specific Boolean
• constraint evaluates to true or false. Conditional statements are handled by IF statements in Python.



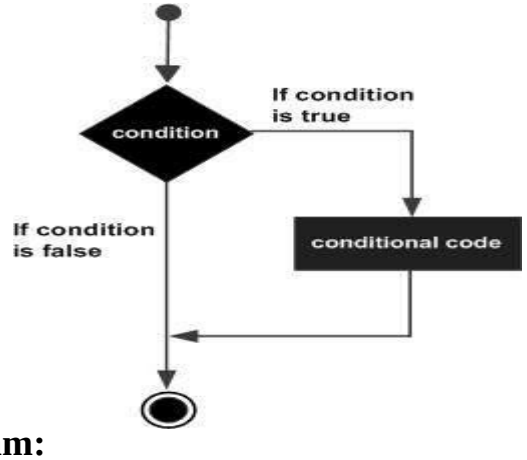
(a).if statement:

In Python if statement is a statement which is used to test specified condition. The if statement executes only when specified condition is true.

general Python **syntax** for a simple if statement is:

```
if condition :
    indented Statement Block
```

Flow chart



Example program:

```
Age=int(input("enter your age"))
if Age>60:
    print("senior citizen")
print("entered age is",Age)
```

If the condition is true, then do the indented statements. If the condition is not true, then skip the indented statements.

(b) if else statement:

The if statement accepts an expression and then executes the specified statements If the condition is true. If the condition is not true, then executes the specified statements indented below else.

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

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

```
   else:
      statement(s)
```
Example program:
```
num=int(input("enter a number:"))
if num%2==0:
      print("entered number is even number")
else:
      print("entered number is odd number")
```
(c) if-elif statement:

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

Syntax:
```
if expression1:
    statement(s)
elif expression2:
    statement(s)
elif expression3:
    statement(s)
else:
     statements(s)
```
Example:
```
marks=int(input("Enter the marks"))
if marks>=75:
    print("Grade=A")
elif marks>=60:
    print("Grade=B")
elif marks>=50:
    print("Garde=C")
else:
    print("fail")
```
Python range() Function:

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

Syntax

range*(start, stop, step*)

Parameter Values

neter	**ption**
	al. An integer number specifying at which position to start. Default i
	ed. An integer number specifying at which position to stop (not :d).

al. An integer number specifying the incrementation. Default is 1

Example1:

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

output:

```
3
4
5
```

Example2:

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

output:

```
3
5
7
9
11
13
15
17
19
```

2.Loops:

Loops are used to iterate over elements of a sequence, it is often used when a piece of code which you want to repeat
"n" number of time. There are two types of loops in python. They are:

1. For loop
2. While loop

(a) For Loop

A for loop is used for iterating over a sequence (i.e., is either a list, a tuple or a string).With the for loop we can execute a set of statements, once for each item in a list, tuple, set etc.

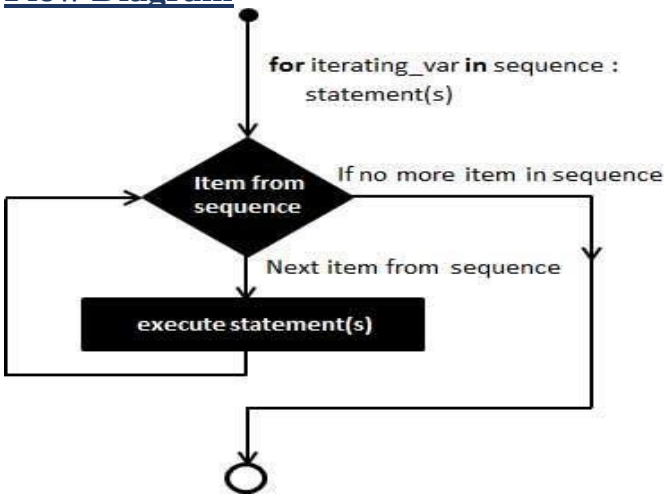
Syntax:

```
for var in sequence:
    statements(s)
else:
    statements(s)
```

Here, var is the variable that takes the value of the item inside the sequence on each iteration. Else statement(s) will be executed when loop terminated normally(without break statement)

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.

Flow Diagram



Example

```
for letter in 'Python':     # First Example
    print( 'Current Letter :', letter")

fruits = ['banana', 'apple',  'mango']
for fruit in fruits:        # Second Example
    print ('Current fruit :', fruit)

print ("Good bye!")
```

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

```
Current Letter : P
Current Letter : y
Current Letter : t
Current Letter : h
Current Letter : o
Current Letter : n
Current fruit : banana
Current fruit : apple
Current fruit : mango
Good bye!
```

Iterating by Sequence Index

An alternative way of iterating through each item is by index offset into the sequence itself. Following is a simple example −

```
fruits = ['banana', 'apple',  'mango']
for index in range(len(fruits)):
```

```
    print( 'Current fruit :', fruits[index]

print( "Good bye!")
```

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

Current fruit : banana
Current fruit : apple
Current fruit : mango
Good bye!

Here, we took the assistance of the len() built-in function, which provides the total number of elements in the tuple as well as the range() built-in function to give us the actual sequence to iterate over.

Using else Statement with For Loop

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.

The following example illustrates the combination of an else statement with a for statement that searches for prime numbers from 10 through 20.

```
for num in range(10,20):     #to iterate between 10 to 20
   for i in range(2,num):    #to iterate on the factors of the number
      if num%i == 0:         #to determine the first factor
      j=num/i                #to calculate the second factor
      print (num,"equals",i,'*',j)
      break #to move to the next number, the #first FOR
   else:                     # else part of the loop
      print(num, 'is a prime number')
```

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

10 equals 2 * 5
11 is a prime number
12 equals 2 * 6
13 is a prime number
14 equals 2 * 7
15 equals 3 * 5
16 equals 2 * 8
17 is a prime number
18 equals 2 * 9
19 is a prime number

While loop:

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 −

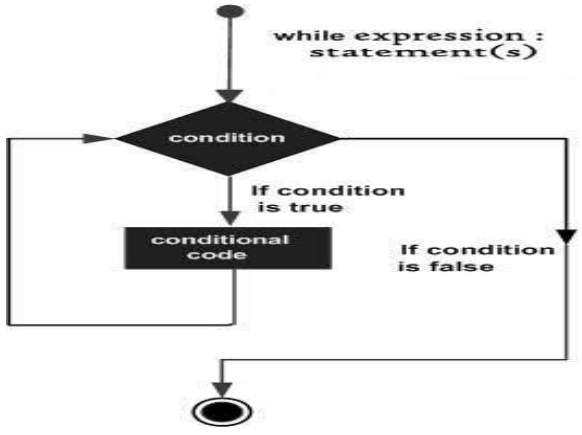
```
while expression:
    statement(s)
```

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.

In Python, all the statements indented by the same number of character spaces after a programming construct are considered to be part of a single block of code. Python uses indentation as its method of grouping statements.

Flow Diagram



Here, key point of the while loop is that the loop might not ever run. When the condition is tested and the result is false, the loop body will be skipped and the first statement after the while loop will be executed.

Example

```
count = 0
while (count < 5):
   print('The count is:', count)(
   count = count + 1

print ("Good bye!")
```

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

```
The count is: 0
The count is: 1
The count is: 2
The count is: 3
```

```
The count is: 4
Good bye!
```

The block here, consisting of the print and increment statements, is executed repeatedly until count is no longer less than 9. With each iteration, the current value of the index count is displayed and then increased by

1.The Infinite Loop

A loop becomes infinite loop if a condition never becomes FALSE. You must use caution when using while loops because of the possibility that this condition never resolves to a FALSE value. This results in a loop that never ends. Such a loop is called an infinite loop.

An infinite loop might be useful in client/server programming where the server needs to run continuously so that client programs can communicate with it as and when required.

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

print ("Good bye!")
```

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

```
Enter a number  :20
You entered:  20
Enter a number  :3
You entered:  3
Enter a number between :Traceback (most recent call last):
  File "test.py", line 5, in <module>
    num =input("Enter a number :")
KeyboardInterrupt
```

Above example goes in an infinite loop and you need to use CTRL+C to exit the program.

Using else Statement with While Loop:

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

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

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.

```
count = 0
while count < 5:
    print (count, " is  less than 5")
  count = count + 1
```

```
else:
    print( count, " is not less than 5")
```

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

0 is less than 5
1 is less than 5
2 is less than 5
3 is less than 5
4 is less than 5

Loop control statements:

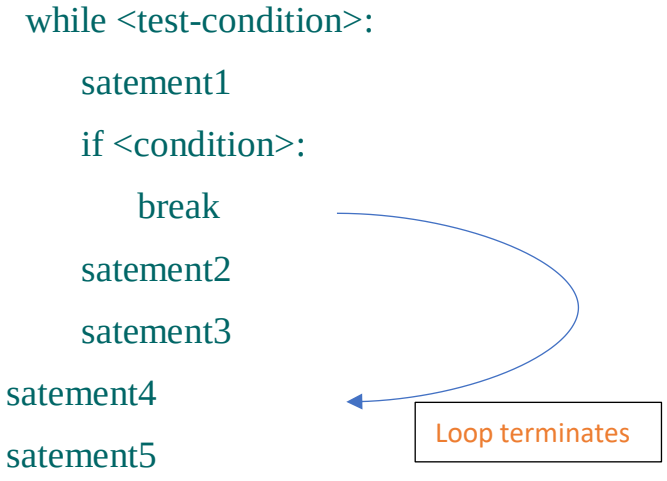
Loop control statements are:

1. Break statement
2. Continue statement

Break statement:

The break statement enables a program to skip over a part of the code. A break statement terminates the very loop it lies within. Execution resumes at the statement immediately following the body of the terminated statement.

The following fig explains the working of the break statement:



example:

```
a=b=c=0

for i in range(1,21):

    a=int(input("Enter number1:"))
```

```
    b=int(input("Enter number2:"))
    if b==0:
        print("division by zero error!Aborting!")
        break
    else:
        c=a//b
        print("quotient=",c)
print("program error")
```

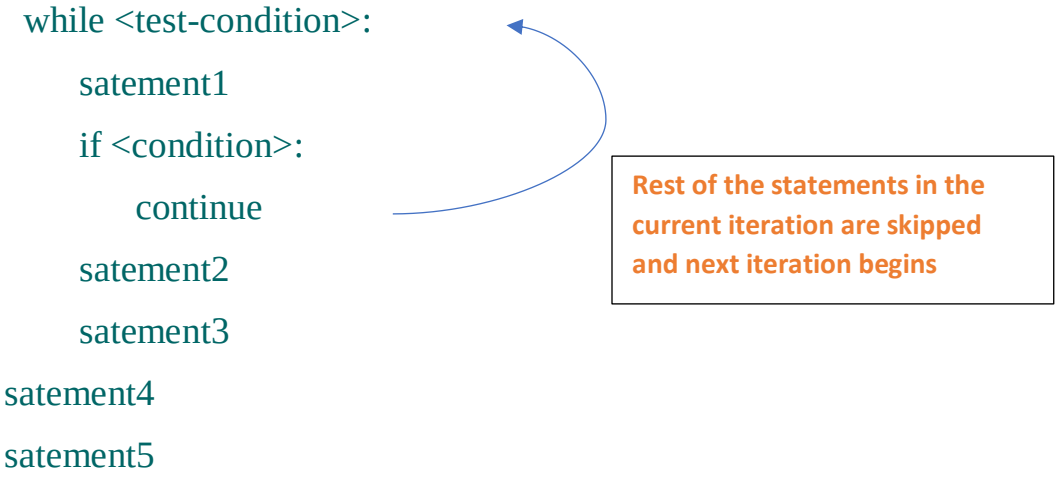
output:

```
Enter number1:5
Enter number2:6
quotient= 0
Enter number1:8
Enter number2:9
quotient= 0
Enter number1:70
Enter number2:10
quotient= 7
Enter number1:7
Enter number2:0
division by zero error!Aborting!
program error
```

Continue statement

The continue statement is another jump statement like break as both the statements skip over a part of the code. But the continue statement is somewhat different from break. Instead of termination, the continue statement "**forces the next iteration of the loop to take place**", skipping any code in between.

The following fig explains the working of the break statement:



Example:

```
a=b=c=0
for i in range(1,21):
    a=int(input("Enter number1:"))
    b=int(input("Enter number2:"))
    if b==0:
        print("division by zero error!Aborting!")
        continue
    else:
        c=a//b
        print("quotient=",c)
print("program error")
```

Output:

Enter number1:8

Enter number2:7

quotient= 1

Enter number1:9

Enter number2:2

quotient= 4

Enter number1:8

Enter number2:0

division by zero error!Aborting!

Enter number1:4

Enter number2:2

quotient= 2

Enter number1:

Strings

String is a sequence which is made up of one or more UNICODE characters. Here the character can be a letter, digit, whitespace or any other symbol. A string can be created by enclosing one or more characters in single, double or triple quote.

Example :

```
>>> str1 = 'Hello World!'
>>> str2 = "Hello World!"
>>> str3 = """Hello World!"""
>>> str4 = '''Hello World!''
```

str1, str2, str3, str4 are all string variables having the same value 'Hello World!'. Values stored in str3 and str4 can be extended to multiple lines using triple codes as can be seen in the following example:

```
>>> str3 = """Hello World! welcome to the world of Python"""
>>> str4 = '''Hello World! welcome to the world of Python'''
```

Accessing Characters in a String:

Each individual character in a string can be accessed using a technique called indexing. The index specifies the character to be accessed in the string and is written in square brackets ([ ]).

The index of the first character (from left) in the string is 0 and the last character is n-l where n is the length of the string. If we give index value out of this range then we get an *IndexError*. The index must be an integer (positive, zero or negative).

```
#initializes a string strl
>>> strl = 'Hello World!'

#gives the first character of strl

>>>strl[0] 'H'

#gives seventh character of strl

>>> strl[6]

        #gives last character of strl

>>> strl[11]


#gives error as index is out of range

>>> strl[15]

IndexError: string index out of range
```

The index can also be an expression including variables and operators but the expression must evaluate to an integer.

```
#an expression resulting in an integer index #so gives 6x character of strl
>>> strl[2+4]

#gives error as index must be an integer
>>> strl[1.5]
TypeError: string indices must be integers
```

Python allows an index value to be negative also. Negative indices are used when we want to access the characters of the string from right to left.

Starting from right hand side, the first character has the index as -1 and the last character has the index —n where n is the length of the string.

```
>>> strl[-1] #gives flrst character from right
⋙ strl[-12]#gives last character fromright'H'
```

Positive Indices	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**		**9**	**10**	**11**
String	H	E	1	1	o		W	o		l	d	
Negative Indices	-12	-11	-10	-9	-8	-7	-6	-5		-3	-2	-1

String is Immutable:

A string is an immutable data type. It means that the contents of the string cannot be changed after it has been created. An attempt to do this would lead to anerror.

```
>>> str1 = "Hello World!"
#if we try to replace character 'e' with 'a'
>>> str1[1] = 'a'

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

STRING OPERATIONS

As we know that string is a sequence of characters. Python allows certain operations on string data type, such as concatenation, repetition, membership and slicing. These operations are explained in the following subsections with suitable examples.

Concatenation:

To concatenate means to join. Python allows us to join two strings using concatenation operator plus which is denoted by symbol +.

```
>>> strl = 'Hello'
>>> str2 = 'World!'
>>> strl + str2 'HelloWorld!'
>>> strl 'Hello'
>>> str2 'World!'
```

Replication :

Python allows us to repeat the given string using repetition operator which is denoted by symbol *

```
#assign string 'Hello' to strl
>>> strl = 'Hello'
#repeat the value of strl 2 times
>>> strl * 2 'HelloHello'
#repeat the value of strl 5 times
>>> strl * 5 'HelloHelloHelloHelloHello'
```

Note: str1 still remains the same after the use of repetition operator.

Membership:

Python has two membership operators 'in' and 'not in'. The 'in' operator takes two strings and returns True if the first string appears as a substring in the second string, otherwise it returns False.

```
>>> str1 = 'Hello World!'
 >>> 'W' in str1 True
 >>> 'Wor' in str1 True
  >>> 'My' in str1 False
```

The 'not in' operator also takes two strings and returns True if the first string does not appear as a substring in the second string, otherwise returns False.

```
 >>> str1 = 'Hello World!'
 >>> 'My' not in str1    True
   >>> 'Hello' not in str1 False
```

Slicing:

In Python, to access some part of a string or substring, we use a method called slicing. This can be done by specifying an index range. Given a string str1, the slice operation str1[n:m] returns the part of the string str1 starting from index n (inclusive) and ending at m (exclusive). In other words, we can say that str1[n:m] returns all the characters starting from str1[n] till str1[m-1]. The numbers of characters in the substring will always be equal to difference of two indices m and n, i.e., (m-n).

```
>>> str1 = 'Hello World!'
#gives substring starting from index 1 to 4
                >>> str1[1:5]
```

'ello'
#gives substring starting from 7 to 9
>>> str1[7:10]
'orl'
#index that is too big is truncated down to
#the end of the string
>>> str1[3:20]
'lo World!'
#first index > second index results in an
#empty " string
>>> str1[7:2]

If the first index is not mentioned, the slice starts from index.

#gives substring from index 0 to 4
>>> str1[:5]
'Hello'

If the second index is not mentioned, the slicing is done till the length of the string.

#gives substring from index 6 to end
>>> str1[6:]
'World!'

The slice operation can also take a third index that specifies the 'step size'. For example, str1[n:m:k], means every k^{th} character has to be extracted from the string str1 starting from n and ending at m-1. By default, the step size is one.

>>> str1[0:10:2]
'HloWr'
>>> str1[0:10:3]
'HlWl'

Negative indexes can also be used for slicing.

#characters at index -6,-5,-4,-3 and -2 are sliced
>>> str1[-6:-1]
'World'

TRAVERSING A STRING:

We can access each character of a string or traverse a string using for loop and while loop.

(A) String Traversal Using for Loop:

```
>>> str1 = 'Hello World!'
>>> for ch in str1:
        print(ch,end = '')
```

Hello World! #output of for loop

In the above code, the loop starts from the first character of the string str1 and automatically ends when the last character is accessed.

(B) String Traversal Using while Loop:

```
>>> str1 = 'Hello World!'
>>> index = 0
#len(): a function to get length of string
>>> while  index < len(str1):
        print(str1[index],end = '')

        index += 1
```

Hello World! #output of while loop

Here while loop runs till the condition index < len(str) is True, where index varies from 0 to len(str1) -1.

STRING METHODS AND BUILT-IN FUNCTIONS:

Python has several built-in functions that allow us to work with strings. The following table describes some of the commonly used built-in functions for string manipulation.

Built-in functions for string manipulations

Method	Description	Example
len()	s the length of the tring	1 = 'Hello World!' (str1)
title()	s the string with first letter of every w ng in uppercase and rest in lowercase	1 = 'hello WORLD!' 1.title() World!'
lower()	the string with all uppercase ed case	= 'hello WORLD!' .lower() orld!'
upper()	the string with all lowercase ed to uppercase	= 'hello WORLD!' .upper()) WORLD!'
nt(str, start, en	number of times substring str occurs ring. If we do not give start index ar hen searching starts from index 0 and e f the string	= 'Hello World! Hello Hello' .count('Hello',12,25) .count('Hello')
d(str,start, end	the first occurrence of index of substr g in the given string. If we do not giv then searching starts from index 0 an n of the string. If the substring is not p ven string, then the function returns -	= 'Hello World! Hello Hello' .find('Hello',10,20) .find('Hello',15,25) .find('Hello') .find('Hee') -1
x(str, start, en	s find() but raises an exception g is not present in the given string	= 'Hello World! Hello .index('Hello') .index('Hee') ror: substring not found
endswith()	True if the given string ends wi substring otherwise returns False	= 'Hello World!' .endswith('World!') .endswith('!') .endswith('lde')

startswith()	s True if the given string starts wit d substring otherwise returns False	1 = 'Hello World!' 1.startswith('He') 1.startswith('Hee')
isalnum()	s True if characters of the given stri alphabets or numeric. If whitespa symbols are part of the given string s empty it returns False	1 = 'HelloWorld' 1.isalnum() 1 = 'HelloWorld2' 1.isalnum() 1 = 'HelloWorld!!' 1.isalnum()
islower()	s True if the string is non-empty and h ase alphabets, or has at least one cha ercase alphabet and rest are non-alp ers	1 = 'hello world!' 1.islower() 1 = 'hello 1234' 1.islower() 1 = 'hello ??' 1.islower() 1 = '1234' 1.islower() 1 = 'Hello World!' 1.islower()
isupper()	s True if the string is non-empty and h ase alphabets, or has at least one cha ercase character and rest are non-alp ers	1 = 'HELLO WORLD!' 1.isupper() 1 = 'HELLO 1234' 1.isupper() 1 = 'HELLO ??' 1.isupper() 1 = '1234' 1.isupper() 1 = 'Hello World!' 1.isupper()

isspace()	s True if the string is non-empty a ers are white spaces (blank, tab, ne e return)	1 = ' \n \t \r' 1.isspace() 1 = 'Hello \n' 1.isspace()
istitle()	s True if the string is non-empty an e., the first letter of every word in the rcase and rest in lowercase	1 = 'Hello World!' 1.istitle() 1 = 'hello World!' 1.istitle()
lstrip()	s the string after removing the space eft of the string	1 = ' Hello World! 1.lstrip() World! '
rstrip()	s the string after removing the space ight of the string	1 = ' Hello World!' 1.rstrip() ello World!'
strip()	s the string after removing the space eft and the right of the string	1 = ' Hello World!' 1.strip() World!'
ce(oldstr, new	es all occurrences of old string with th	1 = 'Hello World!' 1.replace('o','*') W*rld!' 1 = 'Hello World!' 1.replace('World','Country') Country!' 1 = 'Hello World! Hello' 1.replace('Hello','Bye') orld! Bye'

join()	s a string in which the characters
	ave been joined by a separator
		1 = ('HelloWorld!')
		2 = '-' #separator
		2.join(str1)
		-o-W-o-r-l-d-!'

partition()	ns the given string at the first occurre bstring (separator) and returns the ned into three parts. 1. Substring befo or 2. Separator string after the separator If the separa nd in the string, it returns the whole nd two empty strings	1 = 'India is a GreatCountry' 1.partition('is') , 'is', ' a Great Country') 1.partition('are') is a Great Country',' ',' ')

split()	s a list of words delimited by the spe ng. If no delimiter is given then wor ed by space.	1 = 'India is a Great Country' 1.split() ,'is','a','Great', 'Country'] 1 = 'India is a Great y' 1.split('a') ' is ', ' Gre', 't y']

Write a program with a user defined function to count the number of times a character (passed as argument) occurs in the given string.

```
def charCount(ch,st):
    count = 0

  for character in st:

          if character == ch:

              count += 1

               return count #end of function

st = input("Enter a string: ")
ch = input("Enter the character to be searched: ")
count = charCount(ch,st)
print("Number of times character",ch,"occurs in the string is:",count)
```

Program: check whether the given string is palindrome or not

```
str1=input("enter a string:")
str2=str1[::-1]
print(str1)
print(str2)
if str1==str2:
   print("Entered string is palindrome")
else:
   print("Entered string is not a palindrome")
```

Python – Lists

The most basic data structure in Python is the **sequence**. Each element of a sequence is assigned a number - its position or index. The first index is zero, the second index is one, and so forth.

Python has six built-in types of sequences, but the most common ones are lists and tuples. There are certain things you can do with all sequence types. These operations include indexing, slicing, adding, multiplying, and checking for membership. In addition, Python has built-in functions for finding the length of a sequence and for finding its largest and smallest elements.

Lists:

The list is a sequence datatype 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"]
```

Similar to string indices, list indices start at 0, and lists can be sliced, concatenated and so on.

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 −

```
#!/usr/bin/python

list1 = ['physics', 'chemistry', 1997, 2000];
list2 = [1, 2, 3, 4, 5, 6, 7 ];
print( "list1[0]: ", list1[0])
```

```
print ("list2[1:5]: ", list2[1:5])
```

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

```
list1[0]:  physics
list2[1:5]:  [2, 3, 4, 5]
```

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 −

```
#!/usr/bin/python

list = ['physics', 'chemistry', 1997, 2000];
print( "Value available at index 2 : ")
print (list[2])
list[2] = 2001;
print ("New value available at index 2 : ")
print (list[2])
```

Note − append() method is discussed in subsequent section.

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

```
Value available at index 2 :
1997
New value available at index 2 :
2001
```

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 −

```
#!/usr/bin/python

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

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

```
['physics', 'chemistry', 1997, 2000]
After deleting value at index 2 :
['physics', 'chemistry', 2000]
```

Note − remove() method is discussed in subsequent section.

Basic List Operations:

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

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

Python Expression	Results	Description
len([1, 2, 3])	3	Length

[1, 2, 3] + [4, 5, 6]	[1, 2, 3, 4, 5, 6]	Concatenation
[1,2] * 4	[1,2,1,2,1,2,1,2]	Repetition
3 in [1, 2, 3] 4 not in [1,2,3]	True True	Membership
for x in [1, 2, 3]: print(x)	1 2 3	Iteration

Indexing, Slicing, and Matrixes

Because lists are sequences, indexing and slicing work the same way for lists 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 List Functions & Methods

Python includes the following list functions –
list1=[1,5,7]
list2=[2,4,6]

Sr.No.	on with description	Example
1	) he total length of the list.	t1=[1,4,6,7.0,'hai'] n(list1)
2	t) s maximum value in the list value	t1=[1,4,6,7.0,'hai'] t2=[3,5,7,8] ax(list2)

3	t) s minimum value in the list value	t2=[3,5,7,8] in(list2)
4	) Converts a sequence into list.	1="program" t(str1) 'o', 'g', 'r', 'a', 'm'] =(1,5,6,"hi") t(tup) , 'hi']

Python includes following list methods:

	ds with Description	Example
	end(item) ds single item to list	lors=["red","blue"] lors.append("green") lors lue', 'green']
	nt(item) s count of how many times item occurs in list	m=[1,1,3,4,5,6,2,3,4] m.count(1)
	end(seq) ds the contents of seq to list	lors=["red","blue"] lors.extend("green") lors lue', 'g', 'r', 'e', 'e', 'n']
	ex(item) s the lowest index in list that item appears	m=[1,1,3,4,5,6,2,3,4] m.index(4)
	rt(index, item) item into list at offset index	m=[1,1,3,4,5,6,2,3,4] m.insert(7,'hi') m , 4, 5, 6, 2, 'hi', 3, 4]
	() es and returns last item from list (2)	m=[1,1,3,4,5,6,2,3,4] m.pop()

	es the item at index 2.	m.pop(7)
	ove(item) es item from list	m=[1,1,3,4,5,6,2,3,4] m.remove(5) m , 4, 6, 2]
	erse() es items of list in place	=[23,56,12,32,10] .reverse() , 12, 56, 23]
	() e the items in sorted order(ascending)	=[23,56,12,32,10] .sort() , 23, 32, 56]

Python - Tuples

A tuple is a collection of objects which ordered and immutable. 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 );
tup3 = "a", "b", "c", "d";
```

The empty tuple is written as two parentheses containing nothing −

```
tup1 = ();
```

To write a tuple containing a single value you have to include a comma, even though there is only one value −

```
tup1 = (50,);
```

Like string indices, tuple indices start at 0, and they can be sliced, concatenated, and so on.

Accessing Values in Tuple:

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 –

```
tup1 = ('physics', 'chemistry', 1997, 2000)
tup2 = (1, 2, 3, 4, 5, 6, 7 )
```

```
print( “tup1[0]: “, tup1[0]”)
print (tup2[1:5]: “, tup2[1:5])
```

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

```
tup1[0]:  physics
tup2[1:5]:  [2, 3, 4, 5]
```

Updating Tuples:

Tuples are immutable which means you cannot update or change the values of tuple elements. You are able to take portions of existing tuples to create new tuples as the following example demonstrates −

```
tup1 = (12, 34.56)
tup2 = ('abc', 'xyz')

# Following action is not valid for tuples
# tup1[0] = 100;

# So let's create a new tuple as follows
tup3 = tup1 + tup2
print (tup3)
```

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

```
(12, 34.56, 'abc', 'xyz')
```

Delete Tuple Elements:

Removing individual tuple elements is not possible. There is, of course, nothing wrong with putting together another tuple with the undesired elements discarded.

To explicitly remove an entire tuple, just use the **del** statement. For example −

```
#!/usr/bin/python

tup = ('physics', 'chemistry', 1997, 2000)
print (tup)
del tup;
print ("After deleting tup : ")
print( tup)
```

This produces the following result. Note an exception raised, this is because after **del tup** tuple does not exist any more −

```
('physics', 'chemistry', 1997, 2000)
After deleting tup :
Traceback (most recent call last):
  File "test.py", line 9, in <module>
    print tup;
NameError: name 'tup' is not defined
```

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

len((1, 2, 3))	3	Length
(1, 2, 3) + (4, 5, 6)	(1, 2, 3, 4, 5, 6)	Concatenation
('Hi!',) * 4	('Hi!', 'Hi!', 'Hi!', 'Hi!')	Repetition
3 in (1, 2, 3)	True	Membership
for x in (1, 2, 3): print (x)	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'	legative: count from the right
L[1:]	['Spam', 'SPAM!']	Slicing fetches sections

Built-in Tuple Functions:

Python includes the following tuple functions −

	on with Description	Example
	le) he total length of the tuple.	m=(1,1,3,4,5,6,2,3,4) n(num)
	ple) s item from the tuple with max value.	m=(1,1,3,4,5,6,2,3,4) ax(num)
	ple) s item from the tuple with min value.	m=(1,1,3,4,5,6,2,3,4) in(num)

	eq) ts a list into tuple.	1="program" ple(str1) 'o', 'g', 'r', 'a', 'm') t1=[1,5,6,"hi"] ple(list1) , 'hi')
	ount(item) s count of how many times item occurs i	m=(1,1,3,4,5,6,2,3,4) m.count(1)
	dex(item) s the lowest index in list that item appea	m=(1,1,3,4,5,6,2,3,4) m.index(4)

Python - Dictionary

Each key is separated from its value by a colon (:), the items are separated by commas, and the whole thing is enclosed in curly braces. An empty dictionary without any items is written with just two curly braces, like this: {}.

Keys are unique within a dictionary while values may not be. The values of a dictionary can be of any type, but the keys must be of an immutable data type such as strings, numbers, or tuples.

Accessing Values in Dictionary

To access dictionary elements, you can use the familiar square brackets along with the key to obtain its value. Following is a simple example −

```
dict = {'Name': 'Zara', 'Age': 7, 'Class': 'First'}
print( "dict['Name']: ", dict['Name'])
print( "dict['Age']: ", dict['Age'])
```

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

```
dict['Name']:  Zara
dict['Age']:  7
```

If we attempt to access a data item with a key, which is not part of the dictionary, we get an error as follows −

```
dict = {'Name': 'Zara', 'Age': 7, 'Class': 'First'}
print ("dict['Alice']: ", dict['Alice'])
```

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

```
dict['Alice']:
Traceback (most recent call last):
  File "test.py", line 4, in <module>
    Print( "dict['Alice']: ", dict['Alice'])
KeyError: 'Alice'
```

Updating Dictionary

You can update a dictionary by adding a new entry or a key-value pair, modifying an existing entry, or deleting an existing entry as shown below in the simple example –

```
dict = {'Name': 'Zara', 'Age': 7, 'Class': 'First'}
dict['Age'] = 8; # update existing entry
dict['School'] = "DPS School"; # Add new entry
print( "dict['Age']: ", dict['Age'])
print ("dict['School']: ", dict['School'])
```

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

```
dict['Age']:  8
dict['School']:  DPS School
```

Delete Dictionary Elements

You can either remove individual dictionary elements or clear the entire contents of a dictionary. You can also delete entire dictionary in a single operation.

To explicitly remove an entire dictionary, just use the **del** statement. Following is a simple example −

```
dict = {'Name': 'Zara', 'Age': 7, 'Class': 'First'}
del dict['Name']; # remove entry with key 'Name'
dict.clear();     # remove all entries in dict
del dict ;        # delete entire dictionary

print ("dict['Age']: ", dict['Age'])
print ("dict['School']: ", dict['School'])
```

This produces the following result. Note that an exception is raised because after **del dict** dictionary does not exist any more −

```
dict['Age']:
Traceback (most recent call last):
  File "test.py", line 8, in <module>
    Print( "dict['Age']: ", dict['Age'])
TypeError: 'type' object is unsubscriptable
```

Note − del() method is discussed in subsequent section.

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.

There are two important points to remember about dictionary 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. For example −

```
dict = {'Name': 'Zara', 'Age': 7, 'Name': 'Manni'}
print( "dict['Name']: ", dict['Name'])
```

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

```
dict['Name']:  Manni
```

(b) Keys must be immutable. Which means you can use strings, numbers or tuples as dictionary keys but something like ['key'] is not allowed. Following is a simple example −

```
dict = {['Name']: 'Zara', 'Age': 7}
print( "dict['Name']: ", dict['Name'])
```

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

```
Traceback (most recent call last):
  File "test.py", line 3, in <module>
    dict = {['Name']: 'Zara', 'Age': 7};
TypeError: unhashable type: 'list'
```

Python includes following dictionary methods :

	ds with Description	Example
	ar() es all the key-value pairs from the diction	c={1:'a',2:'b',3:'c',4:'d'} c.clear() c
	py() s a shallow copy of the dictionary.	c={1:'a',2:'b',3:'c',4:'d'} =dic.copy() c 2: 'b', 3: 'c', 4: 'd'} 2: 'b', 3: 'c', 4: 'd'}
	ary.fromkeys(sequence, value) ct.fromkeys() method creates a ary from the given iterable (string, list as keys and with the specified value	ys = ['Mumbai','Bangalore','Chicago','New York'] ue = 'city' tionary = dict.fromkeys(keys, value) ctionary bai': None, 'New York': None, 'Bangalore': None, 'Chic
	() s the value of the specified key.	manNums = {'I':1, 'II':2, 'III':3, 'IV':4, 'V':5 } ue = romanNums.get('I') ue
	pitem() l removes and returns a tuple of (key, v om the dictionar.	Nums = {'I':1, 'II':2, 'III':3 } manNums.popitem()) manNums.popitem() manNums.popitem() manNums)

	ms() s a dictionary view object that provic ic view of dictionary elements as a list of pairs. This view object changes wher ary changes.	nanNums = {'I':1,'II':2,'III':3,'IV':4} tview = romanNums.items() tview ms([('I', 1), ('II', 2), ('III', 3), ('IV', 4)])
	ys() ethod returns a view object that contains keys of the dictionary. s list of dictionary dict's keys	nanNums = {'I':1, 'II':2, 'III':3, 'IV':4, 'V':5 } ys = romanNums.keys() ys ys(['I', 'II', 'III', 'IV', 'V'])
	default(key, default=None) s the value of the key if the key is pre s the defaultValue if the key is not prese f defaultvalue is not specified.	nanNums = {'I':1, 'II':2, 'III':3 } ue = romanNums.setdefault('VI', 4) nanNums ue II': 2, 'III': 3, 'IV': 4 }
	date(iterable) s the dictionary with the key-value pairs r dictionary or another iterable such as key-value pairs.	nanNums = {'I':1,'III':3,'V':5} nanNums.update((II=2,IV=4)) nanNums III': 3, 'V': 5, 'II': 2, 'IV': 4}
	lues() s a view object with the list of all valu tionary.	nanNums={'I':1,'II':2,'III':3,'IV':4,'V':5} ues = romanNums.values() nanNums lues([1, 2, 3, 4, 5])

UNITT-2

Python - Functions

Definition :

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

As you already know, Python gives you many built-in functions like print(), etc. but you can also create your own functions. These functions are called *user-defined functions.*

Defining a Function:

You can define functions to provide the required functionality. Here are simple rules to define a function in Python.

- Function blocks begin with the keyword **def** followed by the function name and parentheses (( )).
- Any input parameters or arguments should be placed within these parentheses. You can also define parameters inside these parentheses.
- The first statement of a function can be an optional statement - the documentation string of the function or *docstring*.
- The code block within every function starts with a colon (:) and is indented.
- The statement return [expression] exits a function, optionally passing back an expression to the caller. A return statement with no arguments is the same as return None.

Syntax:

```
def functionname( parameters ):
    "documentation string"
     Body of the function
     return statement
```

Calling a Function:

Defining a function only gives it a name, specifies the parameters that are to be included in the function and structures the blocks of code.

Once the basic structure of a function is finalized, you can execute it by calling it from another function or directly from the Python prompt. Following is the example:

Syntax:

```
Function_name(parameters_list)
```

Example:

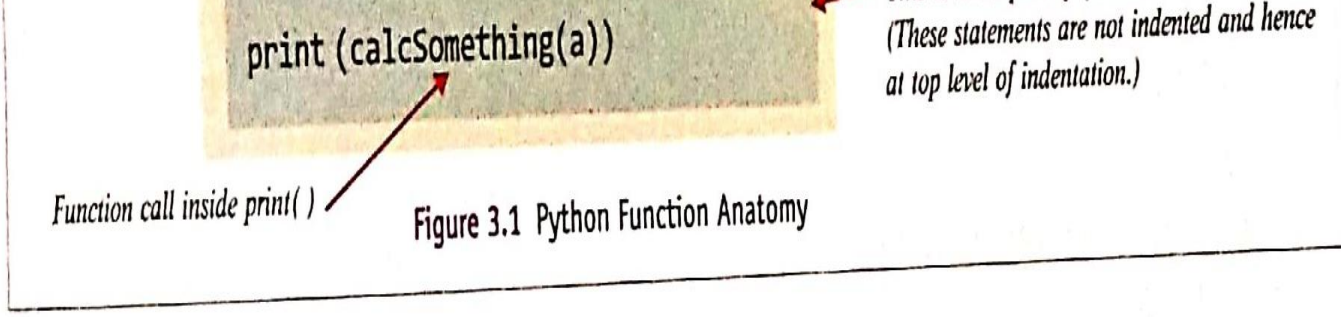


Figure 3.1 Python Function Anatomy

```
#Function definition is here
def printme( str ):
   print(str)
   return       #no value
# Now you can call printme function
printme("I'm first call to user defined function!")
printme("Again second call to the same function")
```

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

I'm first call to user defined function!
Again second call to the same function

Document string or Docstring :

The first string after the function is called the Document string or **Docstring** in short. This is used to describe the functionality of the function. The use of docstring in functions is optional but it is considered a good practice.

Syntax: print(function_name.__doc__)

Example: Adding Docstring to the function

```
# A simple Python function to check whether x is even oı
 def evenOdd(x):
```

```
    """Function to check if the number is even or odd"""
        if (x % 2 == 0):
            print("even")
        else:
            print("odd")
    print(evenOdd.__doc__) # code to call the function
```

Output

Function to check if the number is even or odd

The return statement:

The function return statement is used to exit from a function and go back to the function caller and return the specified value or data item to the caller.

Syntax: return [expression_list]

The return statement can consist of a variable, an expression, or a constant which is returned to the end of the function execution. If none of the above is present with the return statement a None object is returned.

Example: Python Function Return Statement

```
def square_value(num):
    """This function returns the square value of the ente
mber"""
    return num**2
print(square_value(2).__doc__)
```

```
print(square_value(4))
```

Output:

4

16

Arguments of a Function:

Arguments are the values passed inside the parenthesis of the function. A function can have any number of arguments separated by a comma.

Example: Python Function with arguments

In this example, we will create a simple function to check whether the number passed as an argument to the function is even or odd.

```
# A simple Python function to check whether x is even or
def evenOdd(x):
    if (x % 2 == 0)
        print("even")
    else:
        print("odd")
# call the function
evenOdd(2)
evenOdd(3)
```

Output:

even

odd

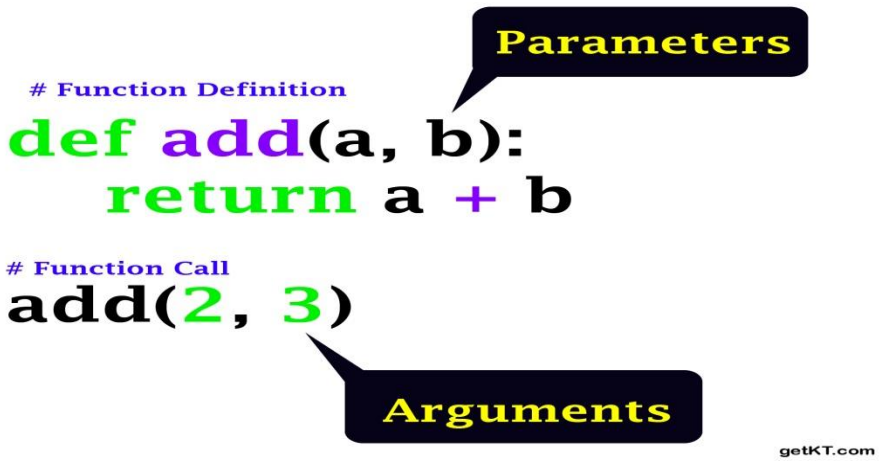
Parameters vs Arguments:

It is recommended to understand what are arguments and parameters before proceeding further. Vocabulary parameters and arguments are not limited to python but they are same across different programming languages.

- **Arguments** are values that are passed into function(or method) when the calling function
- **Parameters** are variables(identifiers) specified in the (header of) function definition

Following image shows difference between parameters and arguments.

Function Parameters VS Arguments



Formal arguments are identifiers used in the function definition to represent corresponding actual arguments.
Actual arguments are values(or variables)/expressions that are used inside the parentheses of a function call.

Above definition and terminology of Formal/Actual arguments is same across different programming languages.

Types of Arguments:

Python supports various types of arguments they are:

1. Default arguments
2. Keyword arguments
3. Positional arguments
4. Arbitrary positional arguments
5. Arbitrary keyword arguments

1.Default arguments:

A default argument is a parameter that assumes a default value, if a value is not provided in the function call for that argument. The following example illustrates Default arguments.

```
# Python program to demonstrate default arguments
 def myFun(x, y=50):
      print("x: ", x)
```

```
    print("y: ", y)
myFun(10)   #  We call myFun() with only one argumen
```

Output

('x: ', 10)

('y: ', 50)

Any number of arguments in a function can have a default value. But once we have a default argument, all the arguments to its right must also have default values.

2.Keyword arguments:

The idea is to allow the caller to specify the argument name with values so that caller does not need to remember the order of parameters.

```
# Python program to demonstrate Keyword Arguments
def student(firstname, lastname):
    print(firstname, lastname)
student(firstname='Geeks', lastname='Practice')  # Keywo
guments
student(lastname='Practice', firstname='Geeks')  # Keywo
guments
```

Output

('Geeks', 'Practice')

('Geeks', 'Practice')

3.Positional arguments:

During a function call, values passed through arguments should be in the order of parameters in the function definition. This is called **positional arguments.**

Keyword arguments should follow positional arguments only.

The above function can be called in two ways:

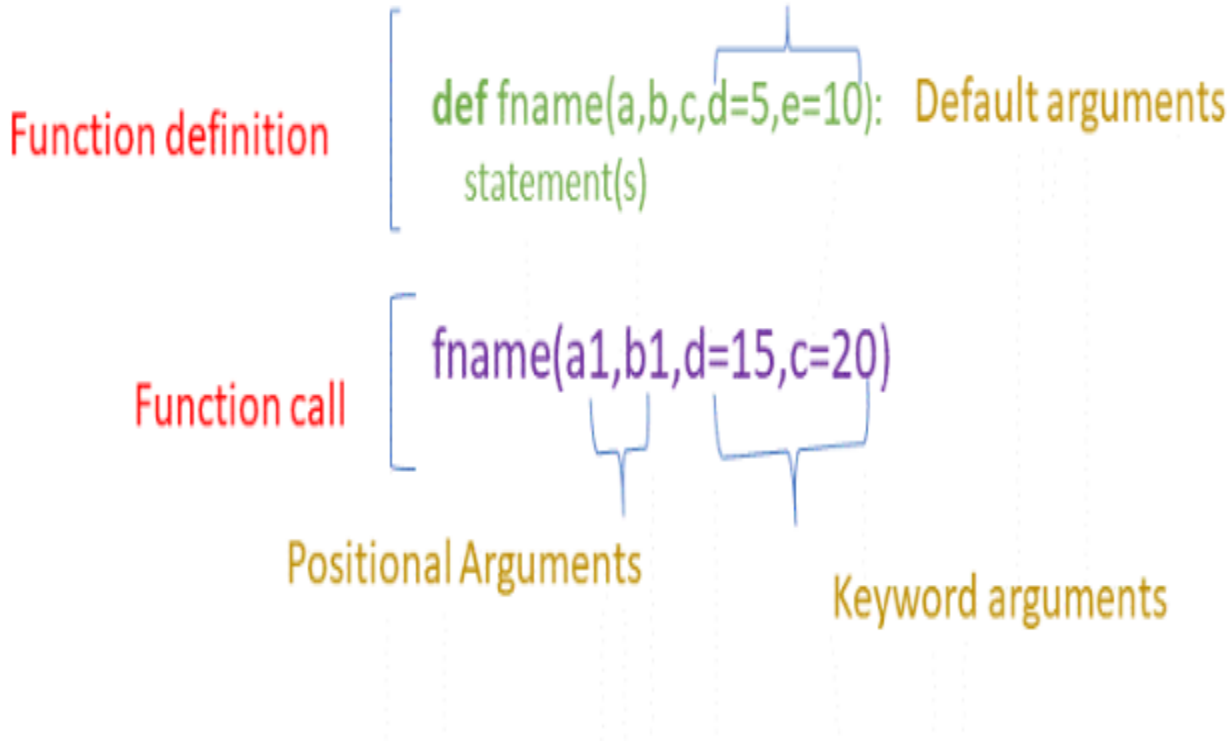
1. During the function call, all arguments are given as positional arguments. Values passed through arguments are passed to parameters by their position. 10 is assigned to a, 20 is assigned to b and 30 is assigned to c.

```
print (add(10,20,30))
#Output:60
```

2. Giving a mix of positional and keyword arguments, keyword arguments should always follow positional arguments

```
print (add(10,c=30,b=20))
#Output:60
```

default vs positional vs keyword arguments:



4.Variable-length arguments:

In Python, we can pass a variable number of arguments to a function using special symbols. There are two special symbols:

- *args (Non-Keyword Arguments)/Arbitrary positional arguments
- **kwargs (Keyword Arguments)/ Arbitrary keyword arguments

Example 1: Variable length non-keywords arguments

```
def myFun(*argv):       # *args for variable number of
arguments
    for arg in argv:
        print(arg)
myFun('Hello', 'Welcome', 'to', 'pyhton')
```

Output

Hello
Welcome
to
python

Example 2: Variable length keyword arguments

```
def myFun(**kwargs):    # *kargs for variable number of keyword arguments
    for key, value in kwargs.items():
        print( (key, value))
myFun(first='python', mid='program', last='functions')
```

Output

('first', 'Geeks')
('mid', 'for')
('last', 'Geeks')

Is Python Function Pass by Reference or pass by value? :

One important thing to note is, in Python every variable name is a reference. When we pass a variable to a function, a new reference to the object is created. Parameter passing in Python is the same as reference passing in C.

Example:

```
def myFun(x):
```

```
        x[0] = 20
 # Driver Code (Note that lst is modified after function ca
lst = [10, 11, 12, 13, 14, 15]
myFun(lst)
print(lst)
```

Output

[20, 11, 12, 13, 14, 15]

When we pass a reference and change the received reference to something else, the connection between the passed and received parameter is broken. For example, consider the below program.

```
 def myFun(x):
        x = [20, 30, 40]
 # Driver Code (Note that lst is not modified after funct
ll.
 lst = [10, 11, 12, 13, 14, 15]
 myFun(lst)
 print(lst)
```

Output

[10, 11, 12, 13, 14, 15]

Another example to demonstrate that the reference link is broken if we assign a new value (inside the function).

```
def myFun(x):
    x = 20
x = 10
```

```
myFun(x)
print(x)
```

Output

10

Exercise: Try to guess the output of the following code.

```
def swap(x, y):
    temp = x
    x = y
    y = temp
x = 2
y = 3
swap(x, y)
print(x)
print(y)
```

Output

2
3

Python Function within another Functions:

A function that is defined inside another function is known as the inner function or nested function. Nested functions are able to access variables of the enclosing scope. Inner functions are used so that they can be protected from everything happening outside the function.

```
  # Python program to demonstrate accessing of variables o
sted functions
   def f1():
      s = 'I love GeeksforGeeks'
      def f2():
          print(s)
      f2()  #nested function calling
   f1()
```

Output

I love GeeksforGeeks

Functions are first class objects :

Python's functions are first-class objects. You can assign them to variables, store them in data structures, pass them as arguments to other functions, and even return them as values from other functions.

Properties of first class functions:

- A function is an instance of the Object type.
- You can store the function in a variable.
- You can pass the function as a parameter to another function.
- You can return the function from a function.
- You can store them in data structures such as hash tables, lists, …

Examples illustrating First Class functions in Python:

1.Functions are objects:

Python functions are first class objects. we are assigning function to a variable. This assignment doesn't call the function. It takes the function object referenced by fun1 and creates a second name pointing to it, fun2.

```
# Python program to illustrate functions can be treated as
jects
def  fun1(text):
    return text.upper()
print(fun1('Hello'))
fun2 = fun1
print(fun2('Hello'))
```

Output:

```
HELLO
HELLO
```

2.Functions can be passed as arguments to other functions:

Because functions are objects we can pass them as arguments to other functions. Functions that can accept other functions as arguments are also called higher-order functions. In the example below, we have created a function **greet** which takes a function as an argument.

```
# Python program to illustrate functions can be passed as
guments to other functions
def simple(text):
```

```
    return text.upper()

def word(text):
    return text.lower()

def greet(func):
    # storing the function in a variable
    greeting = func("""Hi, I am created by a function  passe
argument.""")
    print (greeting)

greet(simple)
greet(word)
```

Output

HI, I AM CREATED BY A FUNCTION PASSED AS AN ARGUMENT.
hi, I am created by a function passed as an argument.

3.Functions can return another function:

Because functions are objects we can return a function from another function. In the below example, the create_adder function returns adder function.

```
# Functions can return another function
  def create_adder(x):
      def adder(y):
          return x+y
```

```
        return adder

    add_15 = create_adder(15)

    print (add_15(10))
```

Output:

25

4.Functions Can Be Stored In Data Structures:

As functions are first-class citizens you can store them in data structures, just like you can with other objects. For example, you can add functions to a list:

```
>>> funcs = [str.lower(), str.capitalize()]

>>> funcs
```

['hi welcome', 'Hi welcome']

5.Functions Can Be Nested:

Python allows functions to be defined inside other functions. These are often called *nested functions* or *inner functions*. Here's an example:

```
def speak(text):

   def whisper(t):
```

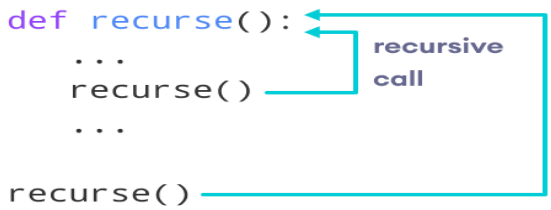
```
        return t.lower() + '...'

    return whisper(text)

speak('Hello, World')

output:    'hello, world...'
```

Recursive function:

Recursion is a programming method, in which a function calls itself one or more times in its body. Usually, it is returning the return value of this function call. If a function definition follows recursion, we call this function a recursive function.

A recursive function has to terminate to be used in a program. It terminates, if with every recursive call the solution of the problem is becomes smaller and moves towards a base case, where the problem can be solved without further recursion. A recursion can lead to an infinite loop, if the base case is not met in the calls.

The following image shows the working of a recursive function called recurse.



Example

The following code returns the sum of first n natural numbers using a recursive python function.

```
def sum_n(n):
    if n== 0:
        return 0
    else:
        return n + sum_n(n-1)
```

This prints the sum of first 100 natural numbers and first 500 natural numbers

```
print(sum_n(100))
print(sum_n(500))
```

Output

5050

125250

Recursive Function in Python:

Following is an example of a recursive function to find the factorial of an integer.

Factorial of a number is the product of all the integers from 1 to that number. For example, the factorial of 6 (denoted as 6!) is 1*2*3*4*5*6 = 720.

Example of a recursive function:

```
def factorial(x):
    """This is a recursive function
```

```
    to find the factorial of an integer"""

    if x == 1:
        return 1
    else:
        return (x * factorial(x-1))

num = 3
print("The factorial of", num, "is", factorial(num))
```

Output

```
The factorial of 3 is 6
```

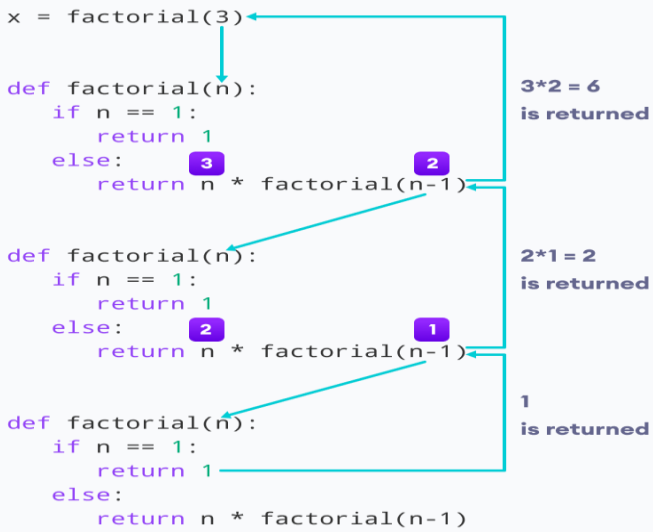
In the above example, factorial() is a recursive function as it calls itself.
When we call this function with a positive integer, it will recursively call itself by decreasing the number.

Each function multiplies the number with the factorial of the number below it until it is equal to one. This recursive call can be explained in the following steps.

```
factorial(3)          # 1st call with 3
3 * factorial(2)      # 2nd call with 2
3 * 2 * factorial(1)  # 3rd call with 1
3 * 2 * 1             # return from 3rd call as number=1
```

```
3 * 2              # return from 2nd call
6                  # return from 1st call
```

Let's look at an image that shows a step-by-step process of what is going on:



Working of a recursive factorial function:

- Our recursion ends when the number reduces to 1. This is called the base condition.
- Every recursive function must have a base condition that stops the recursion or else the function calls itself infinitely.
- The Python interpreter limits the depths of recursion to help avoid infinite recursions, resulting in stack overflows.

- By default, the maximum depth of recursion is 1000. If the limit is crossed, it results

Advantages of Recursion:

- Recursive functions make the code look clean and elegant.
- A complex task can be broken down into simpler sub-problems using recursion.
- Sequence generation is easier with recursion than using some nested iteration.

Disadvantages of Recursion:

- Sometimes the logic behind recursion is hard to follow through.
- Recursive calls are expensive (inefficient) as they take up a lot of memory and time.
- Recursive functions are hard to debug.

Pass by reference vs value:

All parameters (arguments) in the Python language are passed by reference. It means if you change what a parameter refers to within a function, the change also reflects back in the calling function. For example −

```
# Function definition is here
def changeme( mylist ):
   "This changes a passed list into this function"
   mylist.append([1,2,3,4]);
   print( "Values inside the function: ", mylist)
   return
```

```
# Now you can call changeme function
mylist = [10,20,30];
changeme( mylist );
prin(t "Values outside the function: ", mylist)
```

Here, we are maintaining reference of the passed object and appending values in the same object. So, this would produce the following result −

```
Values inside the function:  [10, 20, 30, [1, 2, 3, 4]]
Values outside the function:  [10, 20, 30, [1, 2, 3, 4]]
```

There is one more example where argument is being passed by reference and the reference is being overwritten inside the called function.

```
# Function definition is here
def changeme( mylist ):
      "This changes a passed list into this function"
       mylist = [1,2,3,4]; # This would assig new reference in mylist
        print( "Values inside the function: ", mylist)
        return
# Now you can call changeme function
mylist = [10,20,30];
changeme( mylist );
print( "Values outside the function: ", mylist)
```

The parameter *mylist* is local to the function changeme. Changing mylist within the function does not affect *mylist*. The function

accomplishes nothing and finally this would produce the following result −

Values inside the function: [1, 2, 3, 4]
Values outside the function: [10, 20, 30]

Python – Exceptions

Types of errors in python:

Errors are the problems or mistakes in a program due to which the program will stop the execution.

Two types of Errors occurs in python.

1.Syntax errors

2.Logical errors (Exceptions)

Syntax errors

When the proper syntax of the language is not followed then a syntax error is thrown.

Example

```
amount = 10000

if(amount>2999)
   print("You are eligible to purchase Dsa Self Paced")
```

Output:

```
  File "/home/ac35380186f4ca7978956ff46697139b.py", line 4
    if(amount>2999)
                  ^
SyntaxError: invalid syntax
```

It returns a syntax error message because after the if statement a colon: is missing. We can fix this by writing the correct syntax.

logical errors(Exception):

When in the runtime an error that occurs after passing the syntax test is called exception or logical type. For example, when we divide any number by zero then the ZeroDivisionError

```
marks = 10000

a = marks / 0

print(a)
```

exception is raised, or when we import a module that does not exist then ImportError is raised.

Example 1:

Output:

```
Traceback (most recent call last):
  File "/home/f3ad05420ab851d4bd106ffb04229907.py", line 4, in <module>
    a=marks/0
ZeroDivisionError: division by zero
```

In the above example the ZeroDivisionError as we are trying to divide a number by 0.

Example 2: When indentation is not correct.

```
if(a<3):
print("gfg")
```

Output:

```
  File "/home/959e778cc1b15563df98d2e1e26f92e6.py", line 2
    print("gfg")
        ^
IndentationError: expected an indented block
```

The following table lists important built-in exceptions in Python.

Exception	Description
AssertionError	Raised when the assert statement fails.
AttributeError	Raised on the attribute assignment or reference fails.

Exception	Description
EOFError	Raised when the input() function hits the end-of-file condition.
FloatingPointError	Raised when a floating point operation fails.
GeneratorExit	Raised when a generator's close() method is called.
ImportError	Raised when the imported module is not found.
IndexError	Raised when the index of a sequence is out of range.
KeyError	Raised when a key is not found in a dictionary.
KeyboardInterrupt	Raised when the user hits the interrupt key (Ctrl+c or delete).
MemoryError	Raised when an operation runs out of memory.
NameError	Raised when a variable is not found in the local or global scope.
NotImplementedError	Raised by abstract methods.
OSError	Raised when a system operation causes a system-related error.
OverflowError	Raised when the result of an arithmetic operation is too large to be represented.
ReferenceError	Raised when a weak reference proxy is used to access a garbage collected referent.

Exception	Description
RuntimeError	Raised when an error does not fall under any other category.
StopIteration	Raised by the next() function to indicate that there is no further item to be returned by the iterator.
SyntaxError	Raised by the parser when a syntax error is encountered.
IndentationError	Raised when there is an incorrect indentation.
TabError	Raised when the indentation consists of inconsistent tabs and spaces.
SystemError	Raised when the interpreter detects internal error.
SystemExit	Raised by the sys.exit() function.
TypeError	Raised when a function or operation is applied to an object of an incorrect type.
UnboundLocalError	Raised when a reference is made to a local variable in a function or method, but no value has been bound to that variable.
UnicodeError	Raised when a Unicode-related encoding or decoding error occurs.
UnicodeEncodeError	Raised when a Unicode-related error occurs during encoding.
UnicodeDecodeError	Raised when a Unicode-related error occurs during decoding.

Exception	Description
UnicodeTranslateError	Raised when a Unicode-related error occurs during translation.
ValueError	Raised when a function gets an argument of correct type but improper value.
ZeroDivisionError	Raised when the second operand of a division or module operation is zero.

Exception Handling in Python:

The cause of an exception is often external to the program itself. For example, an incorrect input, a malfunctioning IO device etc. Because the program abruptly terminates on encountering an exception, it may cause damage to system resources, such as files.

Hence, the exceptions should be properly handled so that an abrupt termination of the program is prevented.

Python uses try and except keywords to handle exceptions. Both keywords are followed by indented blocks.

```
try :
   #statements in try block
except :
   #executed when error in try block
```

try block:

The try: block contains one or more statements which are likely to encounter an exception. If the statements in this block are executed without an exception, the subsequent except: block is skipped.

except block:

If the exception does occur, the program flow is transferred to the except: block. The statements in the except: block are meant to handle the cause of the exception appropriately. For example, returning an appropriate error message.

You can specify the type of exception after the except keyword. The subsequent block will be executed only if the specified exception occurs. There may be multiple except clauses with different exception types in a single try block. If the type of exception doesn't match any of the except blocks, it will remain unhandled and the program will terminate.

The rest of the statements after the except block will continue to be executed, regardless if the exception is encountered or not.

The following example will throw an exception when we try to divide an integer by a string.

Example: try...except blocks

Copy

```
try:
```

```
    a=5
    b='0'
    print(a/b)
except:
    print('Some error occurred.')
print("Out of try except blocks.")
```

Output

```
Some error occurred.
Out of try except blocks.
```

You can mention a specific type of exception in front of the except keyword. The subsequent block will be executed only if the specified exception occurs. There may be multiple except clauses with different exception types in a single try block. If the type of exception doesn't match any of the except blocks, it will remain unhandled and the program will terminate.

Example: Catch Specific Error Type

```
try:
    a=5
    b='0'
    print (a+b)
except TypeError:
    print('Unsupported operation')
print ("Out of try except blocks")
```

Output:

Unsupported operation
Out of try except blocks

As mentioned above, a single try block may have multiple except blocks. The following example uses two except blocks to process two different exception types:

Example: Multiple except Blocks

Copy

```
try:
    a=5
    b=0
    print (a/b)
except TypeError:
    print('Unsupported operation')
except ZeroDivisionError:
    print ('Division by zero not allowed')
print ('Out of try except blocks')
```

Output:

Division by zero not allowed
Out of try except blocks

However, if variable b is set to '0', TypeError will be encountered and processed by corresponding except block.

Python - Assert Statement:

In Python, the assert statement is used to continue the execute if the given condition evaluates to True. If the assert condition evaluates to False, then it raises the AssertionError exception with the specified error message.

Syntax

assert condition [, Error Message]

The following example demonstrates a simple assert statement.

```
x = 10
assert x > 0
print('x is a positive number.')
```

Output

x is a positive number.

In the above example, the assert condition, x > 0 evalutes to be True, so it will continue to execute the next statement without any error.

The assert statement can optionally include an error message string, which gets displayed along with the AssertionError. Consider the following assert statement with the error message.

Example: Assert Statement with Error Message

Copy

```
x = 0
assert x > 0, 'Only positive numbers are allowed'
print('x is a positive number.')
```

Output:

```
Traceback (most recent call last):
   assert x > 0, 'Only positive numbers are allowed'
AssertionError: Only positive numbers are allowed
```

Above, x=0, so the assert condition x > 0 becomes False, and so it will raise the AssertionError with the specified message 'Only positive numbers are allowed'. It does not execute print('x is a positive number.') statement.

The following example uses the assert statement in the function.

Example:

```
def square(x):
   assert x>=0, 'Only positive numbers are allowed'
   return x*x

n = square(2) # returns 4
n = square(-2) # raise an AssertionError
```

Output

```
Traceback (most recent call last):
  assert x > 0, 'Only positive numbers are allowed'
AssertionError: Only positive numbers are allowed
```

Above, square(2) will return 4, whereas square(-2) will raise an AssertionError because we passed -2.

The AssertionError is also a built-in exception that can be handled using try-except construct as shown below:

```
def square(x):
  assert x>=0, 'Only positive numbers are allowed'
  return x*x


try:
  square(-2)
except AssertionError as msg:
  print(msg)
```

Output:

```
Only positive numbers are allowed
```

Above, calling square(-2) will raise an AssertionError, which will be handled by the except block. The error message in the assert statement will be passed as an argument to the exception argument msg, using as keyword.

Thus, the assert statement should generally be used to prevent possible errors and specify appropriate error messages.

User Defined Exceptions in Python:

- When we write programs in python for real life applications, there are many constraints on the values which the variables can take in the program. For example, age cannot have a negative value.

- When a person inputs a negative value for age, the program should show an error message. But these types of constraints cannot be applied in the python programs automatically.

- To handle these types of errors and force constraints on values, we use user defined exceptions in python. In this article we will look at different methods by which we can implement user defined exceptions in python.

What are user defined exception in python?

- User defined exceptions in python are created by programmers to enforce constraints on the values which the variables in the program can take.

- Python has many built in exceptions which are raised when there is some error in the program. A program automatically terminates after showing which inbuilt exception has occurred while executing the program when it reaches into an undesired state. We can stop the program from entering into an undesired state by enforcing constraints using user defined exceptions.

- User defined exceptions can be implemented by raising an exception explicitly, by using assert statement or by defining custom classes for user defined exceptions. We will study each method one by one in this article.

.

For example, suppose we have to calculate the year of birth of a person from its age, we can do it as following:

```
age= 10
print("Age is:")
print(age)
yearOfBirth= 2021-age
print("Year of Birth is:")
print(yearOfBirth)
```

Output:

```
Age is:
10
Year of Birth is:
2011
```

The program has given proper output in the case by subtracting age from current year. Now suppose we give a negative value for age in input, say -10.

```
age= -10
print("Age is:")
print(age)
yearOfBirth= 2021-age
print("Year of Birth is:")
print(yearOfBirth)
```

Output:

```
Age is:
-10
Year of Birth is:
2031
```

When we provide a negative number for age, the program still works fine but gives an output which is logically incorrect because no person can have his year of birth in future.

To prevent this kind of year, we can check if the value given in age as input is negative and then we can force the program to raise an exception using raise keyword as follows.

The syntax for raise statement is `raise ExceptionName` . When an error is raised, the code in the except block should handle the exception otherwise it will cause error in the program.

```
try:
```

```
    age= -10
    print("Age is:")
    print(age)
    if age<0:
        raise ValueError
    yearOfBirth= 2021-age
    print("Year of Birth is:")
    print(yearOfBirth)
except ValueError:
    print("Input Correct age.")
```

Output:

```
Age is:

-10
Input Correct age.
```

Here we can see that for age -10, the program handles the case successfully. Lets check if it gives correct year of birth when given correct value of age.

```


try:
    age= 10
```

```
    print("Age is:")
    print(age)
    if age<0:
        raise ValueError
    yearOfBirth= 2021-age
    print("Year of Birth is:")
    print(yearOfBirth)
except ValueError:
    print("Input Correct age.")
```

Output:

```
Age is:
10
Year of Birth is:
2011
```

So, we can see that after using raise statement in try except block, the program gives correct output in both the cases. Now we will see how we can use assert statements to enforce the same constraint.

UNIT-3

NumPy Introduction

What is NumPy?

- NumPy is a Python library used for working with arrays.It also has functions for working in domain of linear algebra, fourier transform, and matrices.
- NumPy was created in 2005 by Travis Oliphant. It is an open source project and you can use it freely.NumPy stands for Numerical Python.

Use of NumPy:

- In Python we have lists that serve the purpose of arrays, but they are slow to process.NumPy aims to provide an array object that is faster than traditional Python lists.
- The array object in NumPy is called ndarray, it provides a lot of supporting functions that make working with ndarray very easy.Arrays are very frequently used in data science, where speed and resources are very important.

Why is NumPy Faster Than Lists?

- NumPy arrays are stored at one continuous place in memory unlike lists, so processes can access and manipulate them very efficiently.
- This behavior is called locality of reference in computer science.This is the main reason why NumPy is faster than lists. Also it is optimized to work with latest CPU architectures.

Installation of NumPy:

- If you have Python and PIP already installed on a system, then installation of NumPy is very easy.Install it using this command:

  C:\Users*Your Name*>pip install numpy

  If this command fails, then use a python distribution that already has NumPy installed like, Anaconda, Spyder etc.

Import NumPy:

- Once NumPy is installed, import it in your applications by adding the import keyword:

  import numpy

  Now NumPy is imported and ready to use.

Example:

```
import numpy
arr = numpy.array([1, 2, 3, 4, 5])
print(arr)
```

NumPy as np:

NumPy is usually imported under the np alias.

alias: In Python alias are an alternate name for referring to the same thing.Create an alias with the as keyword while importing:

```
import numpy as np
```

Now the NumPy package can be referred to as np instead of numpy.

Checking NumPy Version:

The version string is stored under __version__ attribute.

Example

```
import numpy
print(numpy.__version__)
```

Create a NumPy ndarray Object

NumPy is used to work with arrays. The array object in NumPy is called ndarray.We can create a NumPy ndarray object by using the array() function.

Example

```
import numpy as np
arr = np.array([1, 2, 3, 4, 5])
print(arr)
print(type(arr))
```

type(): This built-in Python function tells us the type of the object passed to it. Like in above code it shows that arr is numpy.ndarray type.

To create an ndarray, we can pass a list, tuple or any array-like object into the array() method, and it will be converted into an ndarray:

Example

Use a tuple to create a NumPy array:

```
import numpy as np
arr = np.array((1, 2, 3, 4, 5))
print(arr)
```

Dimensions in Arrays

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

nested array: are arrays that have arrays as their elements.

0-D Arrays:

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

Example

Create a 0-D array with value 42

```
import numpy as np
arr = np.array(42)
print(arr)
```

1-D Arrays:

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

These are the most common and basic arrays.

Example

Create a 1-D array containing the values 1,2,3,4,5:

```
import numpy as np
arr = np.array([1, 2, 3, 4, 5])
print(arr)
```

2-D Arrays:

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

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

NumPy has a whole sub module dedicated towards matrix operations called numpy.mat

Example

Create a 2-D array containing two arrays with the values 1,2,3 and 4,5,6:

```
import numpy as np
arr = np.array([[1, 2, 3], [4, 5, 6]])
print(arr)
```

3-D arrays:
An array that has 2-D arrays (matrices) as its elements is called 3-D array.
These are often used to represent a 3rd order tensor.
Example
Create a 3-D array with two 2-D arrays, both containing two arrays with the values 1,2,3 and 4,5,6:

```
import numpy as np
arr = np.array([[[1, 2, 3], [4, 5, 6]], [[1, 2, 3], [4, 5, 6]]])
print(arr)
```

Check Number of Dimensions:
NumPy Arrays provides the ndim attribute that returns an integer that tells us how many dimensions the array have.

Example
Check how many dimensions the arrays have:

```
import numpy as np
a = np.array(42)
b = np.array([1, 2, 3, 4, 5])
c = np.array([[1, 2, 3], [4, 5, 6]])
d = np.array([[[1, 2, 3], [4, 5, 6]], [[1, 2, 3], [4, 5, 6]]])
print(a.ndim)
print(b.ndim)
print(c.ndim)
print(d.ndim)
```

Access Array Elements

Array indexing is the same as accessing an array element.
You can access an array element by referring to its index number.
The indexes in NumPy arrays start with 0, meaning that the first element has index 0, and the second has index 1 etc.
Example
Get the first element from the following array:

```
import numpy as np
arr = np.array([1, 2, 3, 4])
print(arr[0])
```

Example
Get the second element from the following array.

```
import numpy as np
arr = np.array([1, 2, 3, 4])
print(arr[1])
```

Example
Get third and fourth elements from the following array and add them.

```
import numpy as np
arr = np.array([1, 2, 3, 4])
print(arr[2] + arr[3])
```

Access 2-D Arrays:
To access elements from 2-D arrays we can use comma separated integers representing the dimension and the index of the element.

Example
Access the 2nd element on 1st dim:

```
import numpy as np
arr = np.array([[1,2,3,4,5], [6,7,8,9,10]])
print('2nd element on 1st dim: ', arr[0, 1])
```

Example
Access the 5th element on 2nd dim:

```
jimport numpy as np
arr = np.array([[1,2,3,4,5], [6,7,8,9,10]])
print('5th element on 2nd dim: ', arr[1, 4])
```

Access 3-D Arrays:
To access elements from 3-D arrays we can use comma separated integers representing the dimensions and the index of the element.
Example
Access the third element of the second array of the first array:

```
import numpy as np
arr = np.array([[[1, 2, 3], [4, 5, 6]], [[7, 8, 9], [10, 11, 12]]])
print(arr[0, 1, 2])
```

Example Explained
arr[0, 1, 2] prints the value 6.

NumPy Array Slicing:

- Slicing in python means taking elements from one given index to another given index.
- We pass slice instead of index like this: [*start*:*end*].
- We can also define the step, like this: [*start*:*end*:*step*].
- If we don't pass start its considered 0
- If we don't pass end its considered length of array in that dimension
- If we don't pass step its considered 1

Example: Slice elements from index 1 to index 5 from the following array:

```
import numpy as np
arr = np.array([1, 2, 3, 4, 5, 6, 7])
print(arr[1:5])
```

output: [2 3 4 5]

Note: The result *includes* the start index, but *excludes* the end index.
Example: Slice elements from index 4 to the end of the array:

```
import numpy as np
arr = np.array([1, 2, 3, 4, 5, 6, 7])
print(arr[4:])
```

output: [5 6 7]

Negative Slicing:
Use the minus operator to refer to an index from the end:
Example: Slice from the index 3 from the end to index 1 from the end:

```
import numpy as np
arr = np.array([1, 2, 3, 4, 5, 6, 7])
print(arr[-3:-1])
```

output: [5 6]

Use the step value to determine the step of the slicing:

Example:Return every other element from index 1 to index 5:

```
import numpy as np
arr = np.array([1, 2, 3, 4, 5, 6, 7])
print(arr[1:5:2])
```

output: [2 4]

Example:Return every other element from the entire array:

```
import numpy as np
arr = np.array([1, 2, 3, 4, 5, 6, 7])
print(arr[::2])
```

output: [1 3 5 7]

Slicing 2-D Arrays:

Example: From the second element, slice elements from index 1 to index 4 (not included)

```
import numpy as np
arr = np.array([[1, 2, 3, 4, 5], [6, 7, 8, 9, 10]])
print(arr[1, 1:4])
```

output: [7 8 9]

Example:From both elements, return index 2:

```
import numpy as np
arr = np.array([[1, 2, 3, 4, 5], [6, 7, 8, 9, 10]])
print(arr[0:2, 2])
```

output: [3 8]

Example:From both elements, slice index 1 to index 4 (not included), this will return a 2-D array

```
import numpy as np
arr = np.array([[1, 2, 3, 4, 5], [6, 7, 8, 9, 10]])
print(arr[0:2, 1:4])
```

output:[[2 3 4]
[7 8 9]]

Slicing 3-D Arrays:

```
import numpy as np
arr = np.array([[[1, 2, 3, 4, 5], [6, 7, 8, 9, 10]],[[11, 12, 13,14, 15], [16, 17,18, 19, 110]]])
print(arr[0:2,0:2,1:4])
```

output: [[[2 3 4]
[7 8 9]]

[[12 13 14]
[17 18 19]]]

Data Types in Python:

By default Python have these data types:

- strings - used to represent text data, the text is given under quote marks. e.g. "ABCD"
- integer - used to represent integer numbers. e.g. -1, -2, -3
- float - used to represent real numbers. e.g. 1.2, 42.42
- boolean - used to represent True or False.
- complex - used to represent complex numbers. e.g. 1.0 + 2.0j, 1.5 + 2.5j

Data Types in NumPy:

NumPy has some extra data types, and refer to data types with one character, like i for integers, u for unsigned integers etc.

Below is a list of all data types in NumPy and the characters used to represent them.

- i - integer
- b - boolean
- u - unsigned integer
- f - float

- c - complex float
- m - timedelta
- M - datetime
- O - object
- S - string
- U - unicode string
- V - fixed chunk of memory for other type (void)

Checking the Data Type of an Array:

The NumPy array object has a property called dtype that returns the data type of the array:

Example: Get the data type of an array object:

```
import numpy as np
arr = np.array([1, 2, 3, 4])
print(arr.dtype)
```

Example: Get the data type of an array containing strings:

```
import numpy as np
arr = np.array(['apple', 'banana'])
print(arr.dtype)
```

Creating Arrays With a Defined Data Type:

We use the array() function to create arrays, this function can take an optional argument: dtype that allows us to define the expected data type of the array elements:

Example: Create an array with data type string:

```
import numpy as np
arr = np.array([1, 2, 3, 4], dtype='S')
print(arr)
print(arr.dtype)
```

For i, u, f, S and U we can define size as well.

Example: Create an array with data type 4 bytes integer:

```
import numpy as np
arr = np.array([1, 2, 3, 4], dtype='i4')
print(arr)
print(arr.dtype)
```

What if a Value Can Not Be Converted?

If a type is given in which elements can't be casted then NumPy will raise a ValueError.

ValueError: In Python ValueError is raised when the type of passed argument to a function is unexpected/incorrect.

Example: A non integer string like 'a' can not be converted to integer (will raise an error):

```
import numpy as np
arr = np.array(['a', '2', '3'], dtype='i')
```

NumPy - Array Attributes/Operations:

ndarray.shape

This array attribute returns a tuple consisting of array dimensions. It can also be used to resize the array.

Example 1

```
import numpy as np
a = np.array([[1,2,3],[4,5,6]])
print( a.shape)
```

The output is as follows −

(2, 3)

ndarray.reshape():

NumPy also provides a reshape function to resize an array.

```
import numpy as np
a = np.array([[1,2,3],[4,5,6]])
b = a.reshape(3,2)
print (b)
```

The output is as follows −

```
[[1, 2]
 [3, 4]
 [5, 6]]
```

```
# this is one dimensional array
import numpy as np
a = np.arange(24)
a.ndim

# now reshape it
b = a.reshape(2,4,3)
print b
# b is having three dimensions
```

The output is as follows −

```
[[[ 0,  1,  2]
  [ 3,  4,  5]
  [ 6,  7,  8]
  [ 9, 10, 11]]
  [[12, 13, 14]
   [15, 16, 17]
   [18, 19, 20]
   [21, 22, 23]]]
```

NumPy - Array From Numerical Ranges

numpy.arange():

This function returns an **ndarray** object containing evenly spaced values within a given range. The format of the function is as follows −

numpy.arange(start, stop, step, dtype)

The constructor takes the following parameters.

Sr.No.	Parameter & Description
1	**Start :** The start of an interval. If omitted, defaults to 0
2	**Stop:** The end of an interval (not including this number)

3	**Step:** Spacing between values, default is 1
4	**Dtype :**Data type of resulting ndarray. If not given, data type of input is used

The following examples show how you can use this function.

```
import numpy as np
x = np.arange(5)
print(x)
```

Its output would be as follows −

[0 1 2 3 4]

Example

```
import numpy as np
# dtype set
x = np.arange(5, dtype = float)
print(x)
```

Here, the output would be −

[0. 1. 2. 3. 4.]

Example

```
# start and stop parameters set
import numpy as np
x = np.arange(10,20,2,dtype='S')
print(x)
```

```
Traceback (most recent call last):
  File "C:/Users/Hp/AppData/Local/Programs/Python/Python37/inv.py", line 2, in <module>
    x = np.arange(5,100,10,dtype="S")
ValueError: no fill-function for data-type.
```

Numpy-Mathematical functions:

Trigonometric Functions:

NumPy has standard trigonometric functions which return trigonometric ratios for a given angle in radians

```
import numpy as np
a = np.array([0,30,45,60,90])
print( 'Sine of different angles:' )
# Convert to radians by multiplying with pi/180
print (np.sin(a*np.pi/180) )
print ('\n')
print('Cosine values for angles in array:' )
print( np.cos(a*np.pi/180) )
print( '\n' )
print ('Tangent values for given angles:' )
print( np.tan(a*np.pi/180))
```

.

arcsin, arcos, and **arctan** functions return the trigonometric inverse of sin, cos, and tan of the given angle. The result of these functions can be verified by **numpy.degrees() function** by converting radians to degrees.

```
import numpy as np
a = np.array([0,30,45,60,90])
print( 'Array containing sine values:' )
sin = np.sin(a*np.pi/180)
print( sin )
print('\n')
print( 'Compute sine inverse of angles. Returned values are in radians.')
inv = np.arcsin(sin)
print (inv )
print ('\n')
print ('Check result by converting to degrees:')
print(np.degrees(inv))
print( '\n' )
print ('arccos and arctan functions behave similarly:' )
cos = np.cos(a*np.pi/180)
print( cos )
print( '\n' )
print( 'Inverse of cos:' )
inv = np.arccos(cos)
print (inv )
print ('\n'  )

print( 'In degrees:' )
print( np.degrees(inv) )
print ('\n')

print( 'Tan function:' )
tan = np.tan(a*np.pi/180)
print (tan)
print( '\n' )

print( 'Inverse of tan:' )
inv = np.arctan(tan)
print( inv )
print( '\n' )

print( 'In degrees:' )
print(np.degrees(inv))
```

Functions for Rounding:

numpy.around()

This is a function that returns the value rounded to the desired precision. The function takes the following parameters.

```
import numpy as np
a = np.array([1.0,5.55, 123, 0.567, 25.532])
print( 'Original array:' )
print( a )
print( '\n')
print( 'After rounding:')
print (np.around(a))
```

It produces the following output −

```
Original array:
[ 1. 5.55 123. 0.567 25.532]

After rounding:
[ 1. 6. 123. 1. 26. ]
```

numpy.floor()

This function returns the largest integer not greater than the input parameter.

Example

```
import numpy as np
a = np.array([-1.7, 1.5, -0.2, 0.6, 10])
print( 'The given array:' )
print (a )
print( '\n' )
print( 'The modified array:' )
print( np.floor(a))
```

It produces the following output −

```
The given array:
[ -1.7 1.5 -0.2 0.6 10. ]

The modified array:
[ -2. 1. -1. 0. 10.]
```

numpy.ceil()

This function returns the largest integer not less than the input parameter.

Example

```
import numpy as np
a = np.array([-1.7, 1.5, -0.2, 0.6, 10])
print( 'The given array:' )
print( a )
print( '\n')
print ('The modified array:' )
print( np.ceil(a))
```

It will produce the following output −

```
The given array:
[ -1.7 1.5 -0.2 0.6 10. ]
The modified array:
[ -1. 2. -0. 1. 10.]
```

NumPy - Arithmetic Operations:

Input arrays for performing arithmetic operations such as add(), subtract(), multiply(), and divide() must be either of the same shape or should conform to array broadcasting rules

```
import numpy as np
a= np.array([20,30,40])
b = np.array([10,10,10])
print('Add the two arrays:' )
print(np.add(a,b))
print ('\n')
print( 'Subtract the two arrays:' )
print (np.subtract(a,b))
print ('\n')
```

```
print ('Multiply the two arrays:')
print (np.multiply(a,b))
print ('\n')
print ('Divide the two arrays:')
print (np.divide(a,b))
print ('Applying mod() function:')
print (np.mod(a,b))
print ('\n')
```

It will produce the following output –

```
Add the two arrays:
[30 40 50]


Subtract the two arrays:
[10 20 30]


Multiply the two arrays:
[200 300 400]


Divide the two arrays:
[2. 3. 4.]
Applying mod() function:
[0 0 0]
```

NumPy Statistical Functions:

NumPy contains various statistical functions that are used to perform statistical data analysis. These statistical functions are useful when finding a maximum or minimum of elements. It is also used to find basic statistical concepts like standard deviation, variance, etc.

Finding maximum and minimum of array in NumPy:

NumPy **np.amin()**and **np.amax()**functions are useful to determine the minimum and maximum value of array elements along a specified axis.

```
import numpy as np
arr= np.array([[1,23,78],[98,60,75],[79,25,48]])
print(arr)
#Minimum Function
print(np.amin(arr))
#Maximum Function
print(np.amax(arr))
```

Finding Mean, Median, Standard Deviation and Variance in NumPy:

Mean:

Mean is the sum of the elements divided by its sum and given by the following formula:

$$\bar{x} = \frac{1}{n}(x_1 + x_2 + \ldots + x_n)$$

It calculates the mean by adding all the items of the arrays and then divides it by the number of elements. We can also mention the axis along which the mean can be calculated.

```
import numpy as np
a = np.array([5,6,7])
print(a)
print(np.mean(a))
```

Output

```
[5 6 7]
6.0
```

Median:

Median is the middle element of the array. The formula differs for odd and even sets.

Odd	**Even**
$\frac{n+1}{2}$	$\frac{n}{2}, \frac{n}{2}+1$

It can calculate the median for both one-dimensional and multi-dimensional arrays. Median separates the higher and lower range of data values.

```
import numpy as np
a = np.array([5,6,7])
print(a)
print(np.median(a))
```

Output

```
[5 6 7]
6.0
```

Standard Deviation:

Standard deviation is the square root of the average of square deviations from mean. The formula for standard deviation is:

$$\mathrm{SD} = \sqrt{\frac{\sum |x - \bar{x}|^2}{n}}$$

```
import numpy as np
a = np.array([5,6,7])
print(a)
print(np.std(a))
```

Output

[5 6 7]
0.816496580927726

Variance:

Variance is the average of the square deviations. Following is the formula for the same:

$$\sigma^2 = \frac{\sum (x_r - \overline{x})^2}{n}$$

```
import numpy as np
a = np.array([5,6,7])
print(a)
print(np.var(a))
```

Output

```
[5 6 7]
0.6666666666666666
```

Python Pandas Introduction

- Pandas is defined as an open-source library that provides high-performance data manipulation in Python. The name of Pandas is derived from the word **Panel Data**, which means **an Econometrics from Multidimensional data**. It is used for data analysis in Python and developed by **Wes McKinney** in **2008**.
- Data analysis requires lots of processing, such as **restructuring, cleaning** or **merging**, etc. There are different tools are available for fast data processing, such as **Numpy, Scipy, Cython**, and **Pandas**. But we prefer Pandas because working with Pandas is fast, simple and more expressive than other tools.
- Pandas is built on top of the **Numpy** package, means **Numpy** is required for operating the Pandas.
- Standard Python distribution doesn't come bundled with Pandas module. A lightweight alternative is to install NumPy using popular Python package installer, **pip.**

pip install pandas

Before Pandas, Python was capable for data preparation, but it only provided limited support for data analysis. So, Pandas came into the picture and enhanced the capabilities of data analysis. It can perform five significant steps required for processing and analysis of data irrespective of the origin of the data, i.e., **load, manipulate, prepare, model, and analyze**.

Key Features of Pandas

- It has a fast and efficient DataFrame object with the default and customized indexing.
- Used for reshaping and pivoting of the data sets.
- Group by data for aggregations and transformations.
- It is used for data alignment and integration of the missing data.
- Provide the functionality of Time Series.
- Process a variety of data sets in different formats like matrix data, tabular heterogeneous, time series.
- Handle multiple operations of the data sets such as subsetting, slicing, filtering, groupBy, re-ordering, and re-shaping.
- It integrates with the other libraries such as SciPy, and scikit-learn.
- Provides fast performance, and If you want to speed it, even more, you can use the **Cython**.

Benefits of Pandas:

The benefits of pandas over using other language are as follows:

- **Data Representation:** It represents the data in a form that is suited for data analysis through its DataFrame and Series.
- **Clear code:** The clear API of the Pandas allows you to focus on the core part of the code. So, it provides clear and concise code for the user.

Python Pandas Data Structure

Pandas deals with the following three data structures −

- Series
- DataFrame
- Panel

These data structures are built on top of Numpy array, which means they are fast.

Dimension & Description

The best way to think of these data structures is that the higher dimensional data structure is a container of its lower dimensional data structure. For example, DataFrame is a container of Series, Panel is a container of DataFrame.

Data Structure	Dimensions	Description
Series	1	1D labeled homogeneous array, size-immutable.
Data Frames	2	General 2D labeled, size-mutable tabular structure with potentially heterogeneously typed columns.
Panel	3	General 3D labeled, size-mutable array.

Mutability:

All Pandas data structures are value mutable (can be changed) and except Series all are size mutable. Series is size immutable.

Note − DataFrame is widely used and one of the most important data structures. Panel is used much less.

1) **Series:**

Series is a one-dimensional array like structure with homogeneous data. For example, the following series is a collection of integers 10, 23, 56, …

10	23	56	17	52	61	73	90	26	72

Key Points

2) Homogeneous data
3) Size Immutable
4) Values of Data Mutable

It is defined as a **one-dimensional array** that is capable of storing various data types. The row labels of series are called the **index**. We can easily convert the list, tuple, and dictionary into series using **"series'** method. A Series cannot contain multiple columns. It has one parameter:

Data: It can be any list, dictionary, or scalar value.

Creating an empty Series :

The empty series can be created by using Series function is as follows:

```
1. import pandas as pd
2. import numpy as np
3. info = np.array(['P','a','n','d','a','s'])
4. a = pd.Series(info)
5. print(a)
```

Creating Series from List:

Creating a series by using list is as follows:

```
import pandas as pd
import numpy as np
L=[1,2,3,4,5]
a = pd.Series(L)
```

```
print(a)
```
output:
```
0   1
1   2
2   3
3   4
4   5
dtype: int64
```

Creating Series from 1-D Array:

Before creating a Series, Firstly, we have to import the numpy module and then use array() function in the program.

```
import pandas as pd
import numpy as np
info = np.array(['P','a','n','d','a','s'])
Series(info)
print(a)
```

Output
```
0   P
1   a
2   n
3   d
4   a
5   s
dtype: object
```

Creating Series from dictionary:

We can create a series by using dictionary is as follows:

```
import pandas as pd
import numpy as np
info = {'name':'Rsihmi','age':8,'class':3}
s=pd.Series(info)
print(s)
```

Output:
```
name    Rsihmi
age          8
class        3
```

Accessing elements of a series:

We can access the data elements of a series by using various methods. We will continue to use the series created above to demonstrate the various methods of accessing.

Accessing the First Element:

The first element is at the index 0 position. So it is accessed by mentioning the index value in the series. We can use both 0 or the custom index to fetch the value.

Example

```
import pandas as pd
s = pd.Series([11,8,6,14,25],index = ['a','b','c','d','e'])
print s[0]
print s['a']
```

Output:

Running the above code gives us the following result −

11
11

Accessing the First Three Elements:

In a similar manner as above we get the first three elements by using the : value in front of the index value of 3 or the appropriate custom index value.

Example

```
import pandas as pd
s = pd.Series([11,8,6,14,25],index = ['a','b','c','d','e'])
print s[:3]
print s[:'c']
```

Output

Running the above code gives us the following result −

```
a   11
b   8
c   6
dtype: int64
a   11
b   8
c 6
dtype: int64
```

Accessing the Last Three Elements:

In a similar manner as above, we get the first three elements by using the: value at the end of the index value of 3 with a negative sign or the appropriate custom index value.

Example

```
import pandas as pd
s = pd.Series([11,8,6,14,25],index = ['a','b','c','d','e'])
print s[-3:]
print s['c':]
```

Output

Running the above code gives us the following result −

```
c 6
d 14
e 25
dtype: int64
c 6
d 14
e 25
dtype: int64
```

Accessing Elements using Index Labels

In this case, we use the custom index values to access non-sequential elements of the series.

Example

```
import pandas as pd
s = pd.Series([11,8,6,14,25],index = ['a','b','c','d','e'])
print (s[['c','b','e']])
```

Output

Running the above code gives us the following result −

```
c 6
b 8
e 25
dtype: int64
```

2) DataFrame:

DataFrame is a two-dimensional array with heterogeneous data. For example,

Name	Age	Gender	Rating
Steve	32	Male	3.45
Lia	28	Female	4.6
Vin	45	Male	3.9
Katie	38	Female	2.78

The table represents the data of a sales team of an organization with their overall performance rating. The data is represented in rows and columns. Each column represents an attribute and each row represents a person.

It is a widely used data structure of pandas and works with a two-dimensional array with labeled axes (rows and columns). DataFrame is defined as a standard way to store data and has two different indexes, i.e., row index and column index. It consists of the following properties:

- The columns can be heterogeneous types like int, bool, and so on.
- It can be seen as a dictionary of Series structure where both the rows and columns are indexed. It is denoted as "columns" in case of columns and "index" in case of rows.

Create a DataFrame using List:

We can easily create a DataFrame in Pandas using list.

```
1. import pandas as pd
2. # a list of strings
3. x = ['Python', 'Pandas']
4.   # Calling DataFrame constructor on list
5. df = pd.DataFrame(x)
6. print(df)
```

Create a DataFrame using Nested List:

```
import pandas as pd
# a list of strings
x = [['Rishmitha',20],['Suhaam', 18]]
# Calling DataFrame constructor on list
df = pd.DataFrame(x,index=['student1','student2'],columns=['name','age'])
print(df)
```

Output

```
name  age
student1  Rishmitha   20
student2     Suhaam   18
```

Create a DataFrame using dictionary:

To create Pandas DataFrame in Python, you can follow this generic template:

```
import pandas as pd
data = {'first_column':  ['first_value', 'second_value', ...],
        'second_column': ['first_value', 'second_value', ...],
         ....
```

```
        }
df = pd.DataFrame(data)
print (df)
```

Note that you don't need to use quotes around *numeric* values (unless you wish to capture those values as strings).

You may then use the code below in order to create the DataFrame for our example:

```
import pandas as pd
data = {'product_name': ['laptop', 'printer', 'tablet', 'desk', 'chair'],
        'price': [1200, 150, 300, 450, 200]
        }
df = pd.DataFrame(data)
print (df)
```

Run the code in Python, and you'll get the following DataFrame:

```
  product_name  price
0       laptop   1200
1      printer    150
2       tablet    300
3         desk    450
4        chair    200
```

You may have noticed that each row is represented by a number (also known as the index) starting from 0. Alternatively, you may assign another value/name to represent each row. For example, in the code below,
the **index=['product_1′,'product_2′,'product_3′,'product_4′,'product_5']** was added:

```
import pandas as pd
data = {'product_name': ['laptop', 'printer', 'tablet', 'desk', 'chair'],
        'price': [1200, 150, 300, 450, 200]
        }
df=pd.DataFrame(data,index=['product_1','product_2','product_3  ','product_4','product_5'])
print (df)
```

Output:

```
           product_name  price
product_1       laptop   1200
product_2      printer    150
product_3       tablet    300
product_4         desk    450
product_5        chair    200
```

Accessing elements in DataFrame:

We are creating a DataFrame using the following commands. Then, we will see how we can access its rows, columns, & values and understand how it can be done in different ways.

Example:

```
import pandas as pd
dictObj = {'EmpId' : ['E01','E02','E03','E04'],
           'EmpName' : ['Raj','Atul','Reena','Ayushi'],
           'Department' : ['IT','IT','HR','Accounts']}
df=pd.DataFrame(dictObj, index=['First','Second','Third','Fourth'])
print(df)
```

Output:

	EmpId	EmpName	Department
First	**E01**	**Raj**	**IT**
Second	**E02**	**Atul**	**IT**
Third	**E03**	**Reena**	**HR**
Fourth	**E04**	**Ayushi**	**Accounts**

Accessing single value from DataFrame:

Using the row name and row index number along with the column, we can easily access a single value of a DataFrame.

Syntax : <DataFrame Object> . <column name> [<row name/ row index number>]

Example :

```
Print(df.EmpName['Third'])   ##Access using row name   #Output  Reena
Print(df.EmpName[2])       ## Access using row index    ##Output Reena
```

Output

```
>>> df
       EmpId EmpName Department
First    E01     Raj         IT
Second   E02    Atul         IT
Third    E03   Reena         HR
Fourth   E04  Ayushi   Accounts
>>>
>>> df.EmpName['Third']
'Reena'
>>> df.EmpName[2]
'Reena'
```

Accessing specific column from DataFrame:

Using the column name ,we can access specific column from the DataFrame.

Syntax: <dataframe_object>[column_name]

Example:

```
import numpy as np
import pandas as pd
d= {'EmpId' : ['E01','E02','E03','E04'],
   'EmpName' : ['Raj','Atul','Reena','Ayushi'],
   'Department' : ['IT','IT','HR','Accounts']}
df=pd.DataFrame(d)
print(df['EmpId'])
```

Output:

```
0    E01
1    E02
2    E03
3    E04
Name: EmpId, dtype: object
```

Accessing specific row from DataFrame:

Using the **loc** ,we can access specific row from the DataFrame.

Syntax: <dataframe_object>.loc[rowindex]

Example:

```
import numpy as np
import pandas as pd
d= {'EmpId' : ['E01','E02','E03','E04'],
   'EmpName' : ['Raj','Atul','Reena','Ayushi'],
   'Department' : ['IT','IT','HR','Accounts']}
df=pd.DataFrame(d)
print(df.loc[2])
```

Output:

```
EmpId          E03
EmpName       Reena
Department       HR
Name: 2, dtype: object
```

Python Pandas - Panel

A **panel** is a 3D container of data. The term **Panel data** is derived from econometrics and is partially responsible for the name pandas − **pan(el)-da(ta)**-s.

The names for the 3 axes are intended to give some semantic meaning to describing operations involving panel data. They are −

- **items** − axis 0, each item corresponds to a DataFrame contained inside.
- **major_axis** − axis 1, it is the index (rows) of each of the DataFrames.
- **minor_axis** − axis 2, it is the columns of each of the DataFrames.

pandas.Panel()

A Panel can be created using the following constructor −

pandas.Panel(data, items, major_axis, minor_axis, dtype, copy)

The parameters of the constructor are as follows −

Parameter	Description
Data	Data takes various forms like ndarray, series, map, lists, dict, constants and also another DataFrame
Items	axis=0
major_axis	axis=1
minor_axis	axis=2
Dtype	Data type of each column
Copy	Copy data. Default, **false**

Create Panel

A Panel can be created using multiple ways like −

- From ndarrays
- From dict of DataFrames

From 3D ndarray

```
# creating an empty panel
import pandas as pd
import numpy as np
data = np.random.rand(2,4,5)
p = pd.Panel(data)
print( p)
```

Its **output** is as follows −

```
<class 'pandas.core.panel.Panel'>
Dimensions: 2 (items) x 4 (major_axis) x 5 (minor_axis)
Items axis: 0 to 1
Major_axis axis: 0 to 3
Minor_axis axis: 0 to 4
```

.

Creating panel From dict of DataFrame Objects:

```
#creating an empty panel
import pandas as pd
import numpy as np

data = {'Item1' : pd.DataFrame(np.random.randn(4, 3)),
   'Item2' : pd.DataFrame(np.random.randn(4, 2))}
p = pd.Panel(data)
print (p)
```

Its **output** is as follows −

```
Dimensions: 2 (items) x 4 (major_axis) x 3 (minor_axis)
Items axis: Item1 to Item2
Major_axis axis: 0 to 3
Minor_axis axis: 0 to 2
```

Create an Empty Panel:

An empty panel can be created using the Panel constructor as follows −

```
#creating an empty panel
import pandas as pd
p = pd.Panel()
print (p)
```

Its **output** is as follows −

```
<class 'pandas.core.panel.Panel'>
Dimensions: 0 (items) x 0 (major_axis) x 0 (minor_axis)
Items axis: None
Major_axis axis: None
Minor_axis axis: None
```

Accessing Data from Panel:
Select the data from the panel using −

- Items
- Major_axis
- Minor_axis

Using Items:

```
# creating an empty panel
import pandas as pd
import numpy as np
data = {'Item1' : pd.DataFrame(np.random.randn(4, 3)),
   'Item2' : pd.DataFrame(np.random.randn(4, 2))}
p = pd.Panel(data)
print( p['Item1'])
```

Its **output** is as follows −

```
0 1 2
0 0.488224 -0.128637 0.930817
1 0.417497 0.896681 0.576657
2 -2.775266 0.571668 0.290082
3 -0.400538 -0.144234 1.110535
```

We have two items, and we retrieved item1. The result is a DataFrame with 4 rows and 3 columns, which are the **Major_axis** and **Minor_axis** dimensions.

Using major_axis:
Data can be accessed using the method **panel.major_axis(index)**.

```
# creating an empty panel
import pandas as pd
import numpy as np
data = {'Item1' : pd.DataFrame(np.random.randn(4, 3)),
   'Item2' : pd.DataFrame(np.random.randn(4, 2))}
p = pd.Panel(data)
print (p.major_xs(1))
```

Its **output** is as follows −

```
Item1 Item2
0 -0.128637 -1.047032
1 0.896681 -0.557322
2 0.571668 0.431953
3 -0.144234 1.302466
```

Using minor_axis:
Data can be accessed using the method **panel.minor_axis(index).**

```
# creating an empty panel
import pandas as pd
import numpy as np
data = {'Item1' : pd.DataFrame(np.random.randn(4, 3)),
   'Item2' : pd.DataFrame(np.random.randn(4, 2))}
p = pd.Panel(data)
print( p.minor_xs(1))
```

Its **output** is as follows −
Item1 Item2
0 0.417497 0.748412
1 0.896681 -0.557322
2 0.576657 NaN

Matplotlib - Introduction

- Matplotlib is a low level graph plotting library in python that serves as a visualization utility.
- Matplotlib was created by John D. Hunter in 2002.
- Matplotlib is open source and we can use it freely.
- Matplotlib is mostly written in python, a few segments are written in C, Objective-C and Javascript for Platform compatibility.

Installation of Matplotlib
If you have Python and PIP already installed on a system, then installation of Matplotlib is very easy.
Install it using this command:

C:\Users*Your Name*>pip install matplotlib

If this command fails, then use a python distribution that already has Matplotlib installed, like Anaconda, Spyder etc.

Import Matplotlib
Once Matplotlib is installed, import it in your applications by adding
the import *module* statement:

```
import matplotlib
```

Now Matplotlib is imported and ready to use:

Checking Matplotlib Version
The version string is stored under __version__ attribute.
Example

```
import matplotlib

print(matplotlib.__version__)
```

Output:
2.0.0

Matplotlib Pyplot:
Most of the Matplotlib utilities lies under the pyplot submodule, and are usually imported under the plt alias:

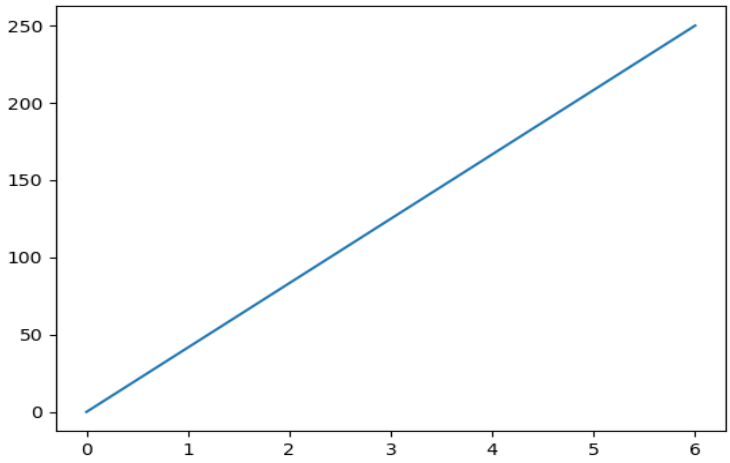
```
import matplotlib.pyplot as plt
```

Now the Pyplot package can be referred to as plt.

Example
Draw a line in a diagram from position (0,0) to position (6,250):

```
import matplotlib.pyplot as plt
import numpy as np
xpoints = np.array([0, 6])
ypoints = np.array([0, 250])
plt.plot(xpoints, ypoints)
plt.show()
```

Result:



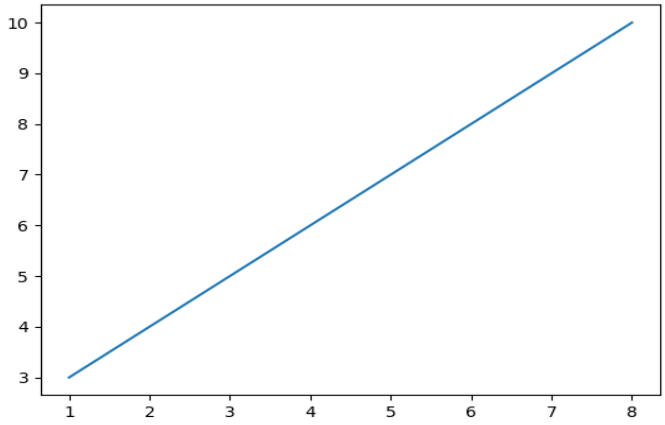
Matplotlib Plotting:

- Plotting x and y points
- The plot() function is used to draw points (markers) in a diagram.
- By default, the plot() function draws a line from point to point.
- The function takes parameters for specifying points in the diagram.
- Parameter 1 is an array containing the points on the **x-axis**.
- Parameter 2 is an array containing the points on the **y-axis**.
- If we need to plot a line from (1, 3) to (8, 10), we have to pass two arrays [1, 8] and [3, 10] to the plot function.

Example
Draw a line in a diagram from position (1, 3) to position (8, 10):

```
import matplotlib.pyplot as plt
import numpy as np
xpoints = np.array([1, 8])
ypoints = np.array([3, 10])
plt.plot(xpoints, ypoints)
plt.show()
```

Result:

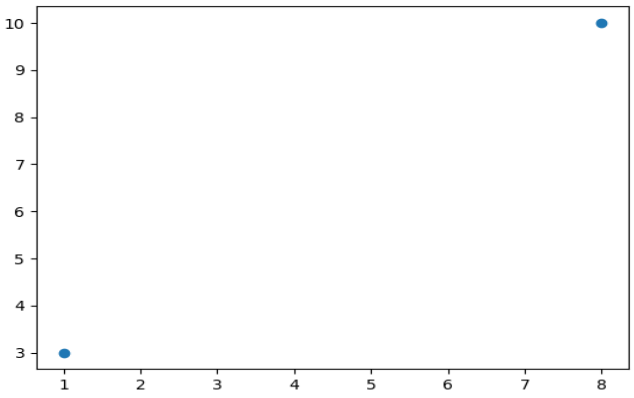


The **x-axis** is the horizontal axis.
The **y-axis** is the vertical axis.

Plotting Without Line:
To plot only the markers, you can use *shortcut string notation* parameter 'o', which means 'rings'.
Example
Draw two points in the diagram, one at position (1, 3) and one in position (8, 10):

```
import matplotlib.pyplot as plt
import numpy as np
xpoints = np.array([1, 8])
ypoints = np.array([3, 10])
plt.plot(xpoints, ypoints, 'o')
plt.show()
```

Result:

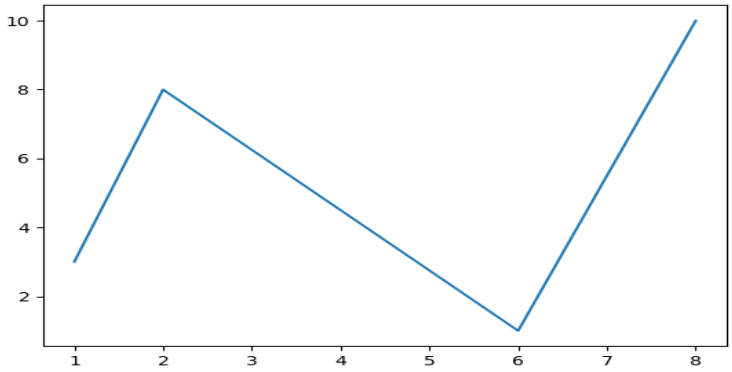


.
Multiple Points:
You can plot as many points as you like, just make sure you have the same number of points in both axis.
Example
Draw a line in a diagram from position (1, 3) to (2, 8) then to (6, 1) and finally to position (8, 10):

```
import matplotlib.pyplot as plt
import numpy as np
xpoints = np.array([1, 2, 6, 8])
ypoints = np.array([3, 8, 1, 10])
```

```
plt.plot(xpoints, ypoints)
plt.show()
```

Result:



Markers:

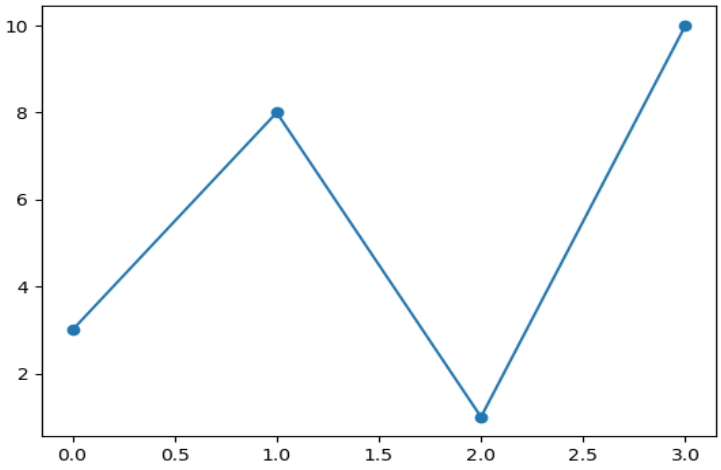
You can use the keyword argument marker to emphasize each point with a specified marker:

Example

Mark each point with a circle:

```
import matplotlib.pyplot as plt
import numpy as np
ypoints = np.array([3, 8, 1, 10])
plt.plot(ypoints, marker = 'o')
plt.show()
```

Result:



Linestyle:

You can use the keyword argument linestyle, or shorter ls, to change the style of the plotted line:

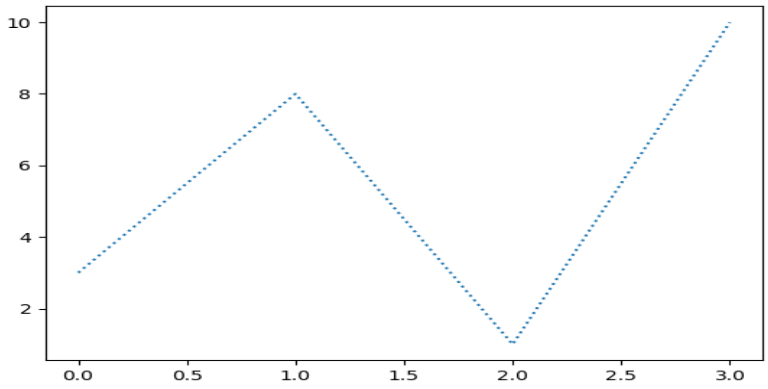
Example

Use a dotted line:

```
import matplotlib.pyplot as plt
import numpy as np
ypoints = np.array([3, 8, 1, 10])
```

```
plt.plot(ypoints, linestyle = 'dotted')
plt.show()
```

Result:



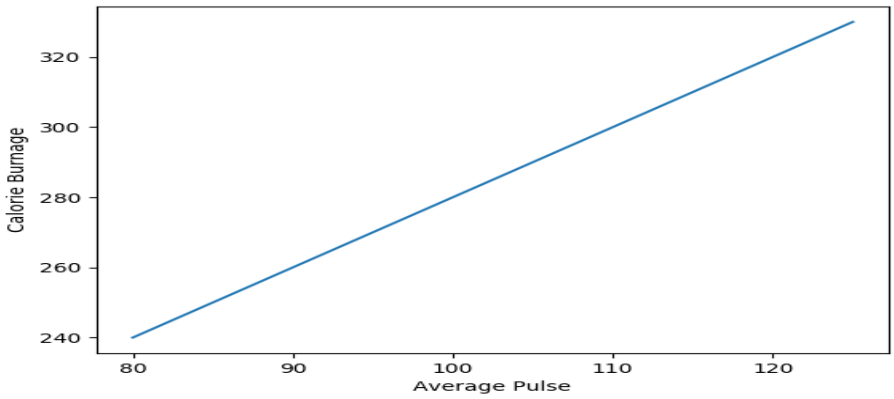
Create Labels for a Plot:

With Pyplot, you can use the xlabel() and ylabel() functions to set a label for the x- and y-axis.

Example
Add labels to the x- and y-axis:

```
import numpy as np
import matplotlib.pyplot as plt
x = np.array([80, 85, 90, 95, 100, 105, 110, 115, 120, 125])
y = np.array([240, 250, 260, 270, 280, 290, 300, 310, 320, 330])
plt.plot(x, y)
plt.xlabel("Average Pulse")
plt.ylabel("Calorie Burnage")
plt.show()
```

Result:



Add Grid Lines to a Plot:

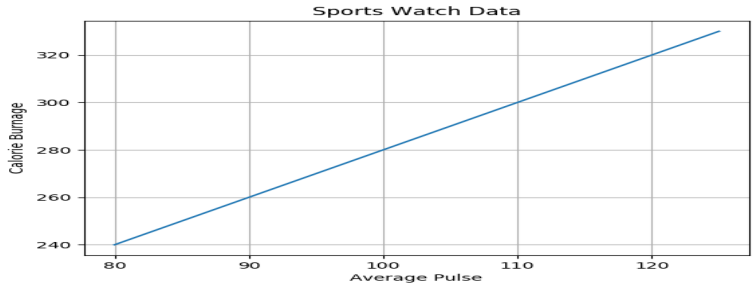
With Pyplot, you can use the grid() function to add grid lines to the plot.

Example
Add grid lines to the plot:

```
import numpy as np
import matplotlib.pyplot as plt
```

```
x = np.array([80, 85, 90, 95, 100, 105, 110, 115, 120, 125])
y = np.array([240, 250, 260, 270, 280, 290, 300, 310, 320, 330])
plt.title("Sports Watch Data")
plt.xlabel("Average Pulse")
plt.ylabel("Calorie Burnage")
plt.plot(x, y)
plt.grid()
plt.show()
```

Result:



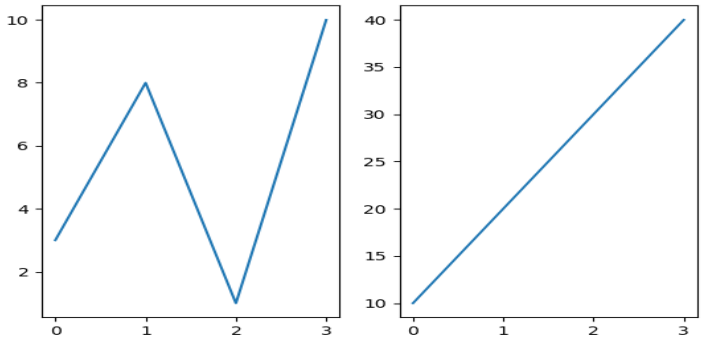
Display Multiple Plots:

With the subplot() function you can draw multiple plots in one figure:

Example
Draw 2 plots:

```
import matplotlib.pyplot as plt
import numpy as np
#plot 1:
x = np.array([0, 1, 2, 3])
y = np.array([3, 8, 1, 10])
plt.subplot(1, 2, 1)
plt.plot(x,y)
#plot 2:
x = np.array([0, 1, 2, 3])
y = np.array([10, 20, 30, 40])
plt.subplot(1, 2, 2)
plt.plot(x,y)
plt.show()
```

Result:



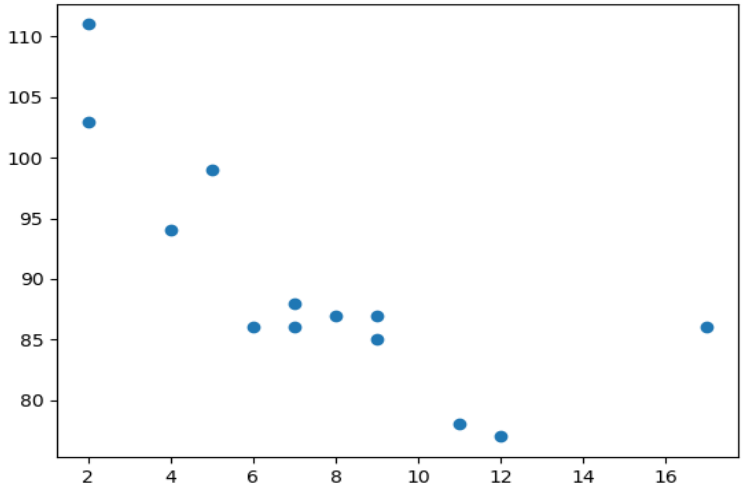
Creating Scatter Plots:

- With Pyplot, you can use the scatter() function to draw a scatter plot.
- The scatter() function plots one dot for each observation. It needs two arrays of the same length, one for the values of the x-axis, and one for values on the y-axis:

Example
A simple scatter plot:

```
import matplotlib.pyplot as plt
import numpy as np
x = np.array([5,7,8,7,2,17,2,9,4,11,12,9,6])
y = np.array([99,86,87,88,111,86,103,87,94,78,77,85,86])
plt.scatter(x, y)
plt.show()
```

Result:

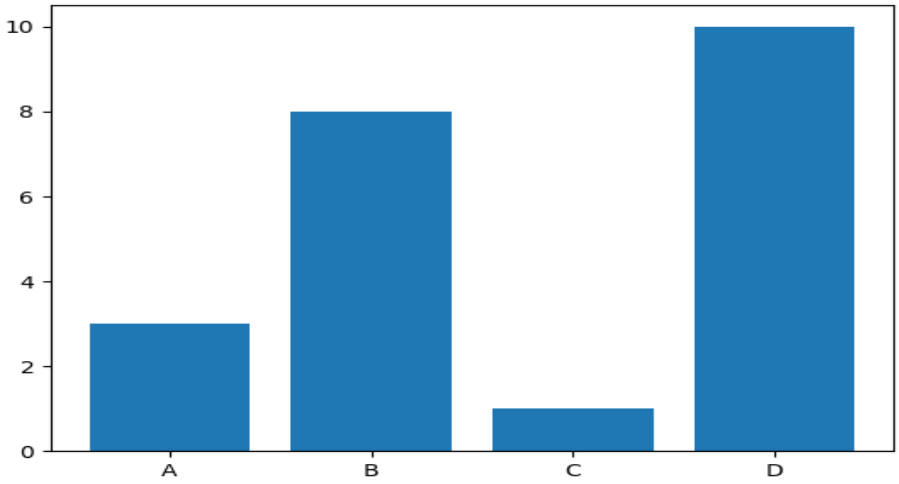


Creating Bars:

With Pyplot, you can use the bar() function to draw bar graphs:
Example
Draw 4 bars:

```
import matplotlib.pyplot as plt
import numpy as np
x = np.array(["A", "B", "C", "D"])
y = np.array([3, 8, 1, 10])
plt.bar(x,y)
plt.show()
```

Result:



Histogram:

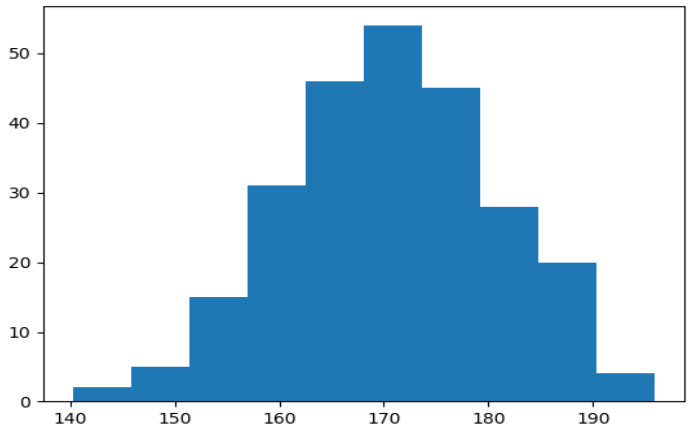
- A histogram is a graph showing *frequency* distributions.
- It is a graph showing the number of observations within each given interval.
- Example: Say you ask for the height of 250 people, you might end up with a histogram like this:

- The hist() function will read the array and produce a histogram:

Example
A simple histogram:

```
import matplotlib.pyplot as plt
import numpy as np
x = np.random.normal(170, 10, 250)
plt.hist(x)
plt.show()
```

Result :



You can read from the histogram that there are approximately:

2 people from 140 to 145cm
5 people from 145 to 150cm
15 people from 151 to 156cm
31 people from 157 to 162cm
46 people from 163 to 168cm
53 people from 168 to 173cm
45 people from 173 to 178cm
28 people from 179 to 184cm
21 people from 185 to 190cm
4 people from 190 to 195cm

Creating Pie Charts:

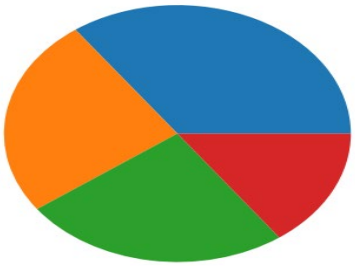
With Pyplot, you can use the pie() function to draw pie charts:

Example

A simple pie chart:

```
import matplotlib.pyplot as plt
import numpy as np
y = np.array([35, 25, 25, 15])
plt.pie(y)
plt.show()
```

Result:

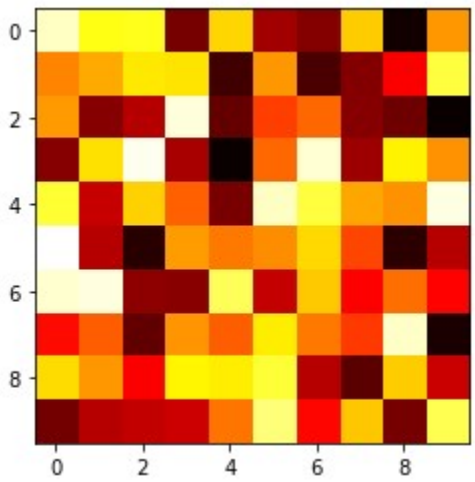


Heatmap:

- heatmap is a graphical representation of numerical data where values are expressed in colors. Plotting matplotlib heatmap provides you an effective summary of the data by converting the data values to pictorial representation. In this entire tutorial, you will know how to plot matplotlib heatmap in python with various methods.
- In Matplotlib there is function ***plt.imshow()*** that accepts cmap argument. It allows you to plot a heatmap for your input data. Execute the lines of code to plot a heatmap on the sample data.

```
import matplotlib.pyplot as plt

import numpy as np


data= np.random.random((10, 10))

plt.imshow(data, cmap='hot', interpolation='nearest')

plt.show()
```

Output:



Exploratory Data Analysis(EDA):

Exploratory Data Analysis (EDA) is an approach to analyze the data using visual techniques. It is used to discover trends, patterns, or to check assumptions with the help of statistical summary and graphical representations.

We will use the employee data for this. It contains 8 columns namely – First Name, Gender, Start Date, Last Login, Salary, Bonus%, Senior Management, and Team.

Dataset Used: Employees.csv

Let's read the dataset using the Pandas module and print the 1st five rows. To print the first five rows we will use the **head()** function.

Example

```
import pandas as pd
import numpy as np

df = pd.read_csv('employees.csv')
df.head()
```

Output:

	First Name	Gender	Start Date	Last Login Time	Salary	Bonus %	Senior Management	Team
0	Douglas	Male	8/6/1993	12:42 PM	97308	6.945	True	Marketing
1	Thomas	Male	3/31/1996	6:53 AM	61933	4.170	True	NaN
2	Maria	Female	4/23/1993	11:17 AM	130590	11.858	False	Finance
3	Jerry	Male	3/4/2005	1:00 PM	138705	9.340	True	Finance
4	Larry	Male	1/24/1998	4:47 PM	101004	1.389	True	Client Services

Steps in Exploratory Data Analysis (EDA) :

1.Getting insights about the dataset
2.Handling Missing Values
3.Data visualization
4.Handling Outliers
5.Removing Outliers

1.Getting insights about the dataset

Let's see the shape of the data using the shape.

Example:

```
df.shape
```

Output:

(1000, 8)

This means that this dataset has 1000 rows and 8 columns.

Let's get a quick summary of the dataset using the **describe()** method. The describe() function applies basic statistical computations on the dataset like extreme values, count of data points standard deviation, etc. Any missing value or NaN value is automatically skipped. describe() function gives a good picture of the distribution of data.

Example:

```
df.describe()
```

Output:

	Salary	Bonus %
count	1000.000000	1000.000000
mean	90662.181000	10.207555
std	32923.693342	5.528481
min	35013.000000	1.015000
25%	62613.000000	5.401750
50%	90428.000000	9.838500
75%	118740.250000	14.838000
max	149908.000000	19.944000

Now, let's also the columns and their data types. For this, we will use the info() method.

```
df.info()
```

Output:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 8 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   First Name         933 non-null    object 
 1   Gender             855 non-null    object 
 2   Start Date         1000 non-null   object 
 3   Last Login Time    1000 non-null   object 
 4   Salary             1000 non-null   int64  
 5   Bonus %            1000 non-null   float64
 6   Senior Management  933 non-null    object 
 7   Team               957 non-null    object 
dtypes: float64(1), int64(1), object(6)
memory usage: 62.6+ KB
```

Till now we have got an idea about the dataset used. Now Let's see if our dataset contains any missing value or not.

2.Handling Missing Values

- You all must be wondering why a dataset will contain any missing value. It can occur when no information is provided for one or more items or for a whole unit. For Example, Suppose different users being surveyed may choose not to share their income, some users may choose not to share the address in this way many datasets went missing.
- Missing Data is a very big problem in real-life scenarios. Missing Data can also refer to as NA(Not Available) values in pandas. There are several useful functions for detecting, removing, and replacing null values in Pandas DataFrame :

- isnull()
- notnull()
- dropna()
- fillna()
- replace()
- interpolate()

Now let's check if there are any missing values in our dataset or not.

Example:

```
df.isnull().sum()
```

Output:

```
First Name           67
Gender              145
Start Date            0
Last Login Time       0
Salary                0
Bonus %               0
Senior Management    67
Team                 43
dtype: int64
```

We can see that every column has a different amount of missing values. Like Gender as 145 missing values and salary has 0. Now for handling these missing values there can be several cases like dropping the rows containing NaN or replacing NaN with either mean, median, mode, or some other value.

Now, let's try to fill the missing values of gender with the string "No Gender".

Example:

```
df["Gender"].fillna("No Gender", inplace = True)

df.isnull().sum()
```

Output:

```
First Name           67
Gender                0
Start Date            0
Last Login Time       0
Salary                0
Bonus %               0
Senior Management    67
Team                 43
dtype: int64
```

We can see that now there is no null value for the gender column. Now, Let's fill the senior management with the mode value.

Example:

```
mode = df['Senior Management'].mode().values[0]
df['Senior Management']= df['Senior Management'].replace(np.nan, mode)
df.isnull().sum()
```

Output:

```
First Name           67
Gender                0
Start Date            0
Last Login Time       0
Salary                0
Bonus %               0
Senior Management     0
Team                 43
dtype: int64
```

Now for the first name and team, we cannot fill the missing values with arbitrary data, so, let's drop all the rows containing these missing values.

Example:

```
df = df.dropna(axis = 0, how ='any')

print(df.isnull().sum())
df.shape
```

Output:

```
First Name           0
Gender               0
Start Date           0
Last Login Time      0
Salary               0
Bonus %              0
Senior Management    0
Team                 0
dtype: int64

(899, 8)
```

We can see that our dataset is now free of all the missing values and after dropping the data the number of also reduced from 1000 to 899.

3.Data visualization

Data Visualization is the process of analyzing data in the form of graphs or maps, making it a lot easier to understand the trends or patterns in the data. There are various types of visualizations –

- **Univariate analysis:** This type of data consists of only one variable. The analysis of univariate data is thus the simplest form of analysis since the information deals with only one quantity that changes. It does not deal with causes or relationships and the main purpose of the analysis is to describe the data and find patterns that exist within it.
- **Bi-Variate analysis:** This type of data involves two different variables. The analysis of this type of data deals with causes and relationships and the analysis is done to find out the relationship among the two variables.
- **Multi-Variate analysis:** When the data involves three or more variables, it is categorized under multivariate.

Histogram

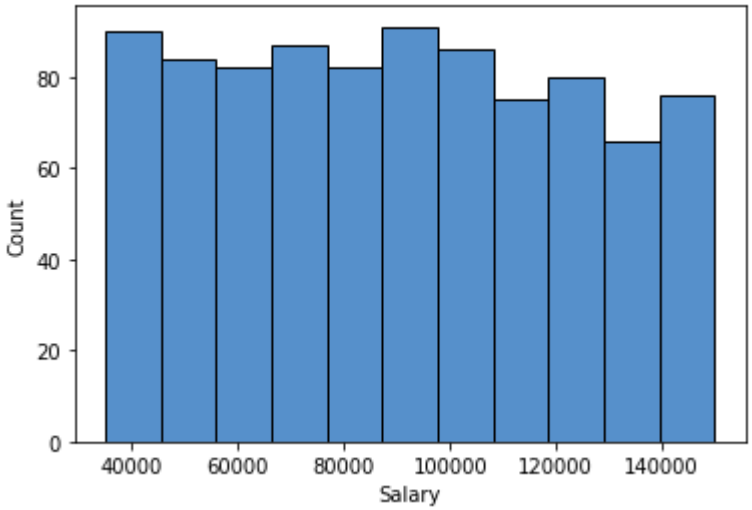
It can be used for both uni and bivariate analysis.

Example:

- Python3

```
# importing packages
import seaborn as sns
import matplotlib.pyplot as plt


sns.histplot(x='Salary', data=df, )
plt.show()
```

Output:



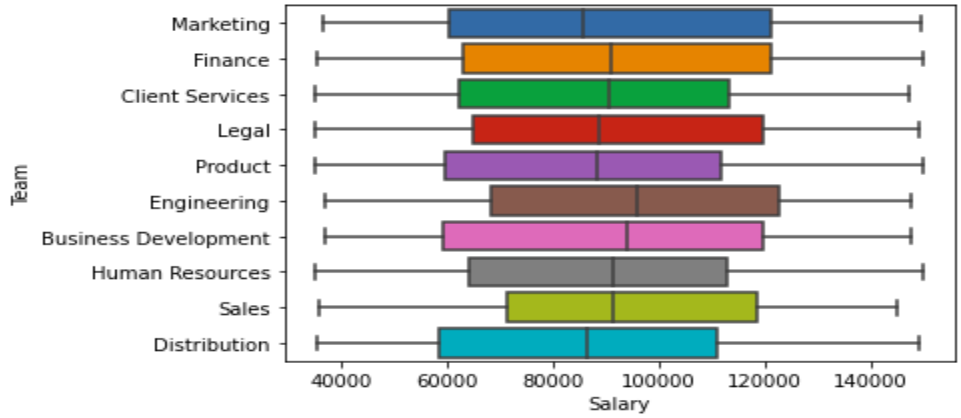
Boxplot

It can also be used for univariate and bivariate analyses.

Example:

```
# importing packages
import seaborn as sns
import matplotlib.pyplot as plt
 sns.boxplot( x="Salary", y='Team', data=df, )
plt.show()
```

Output:



Scatter Plot

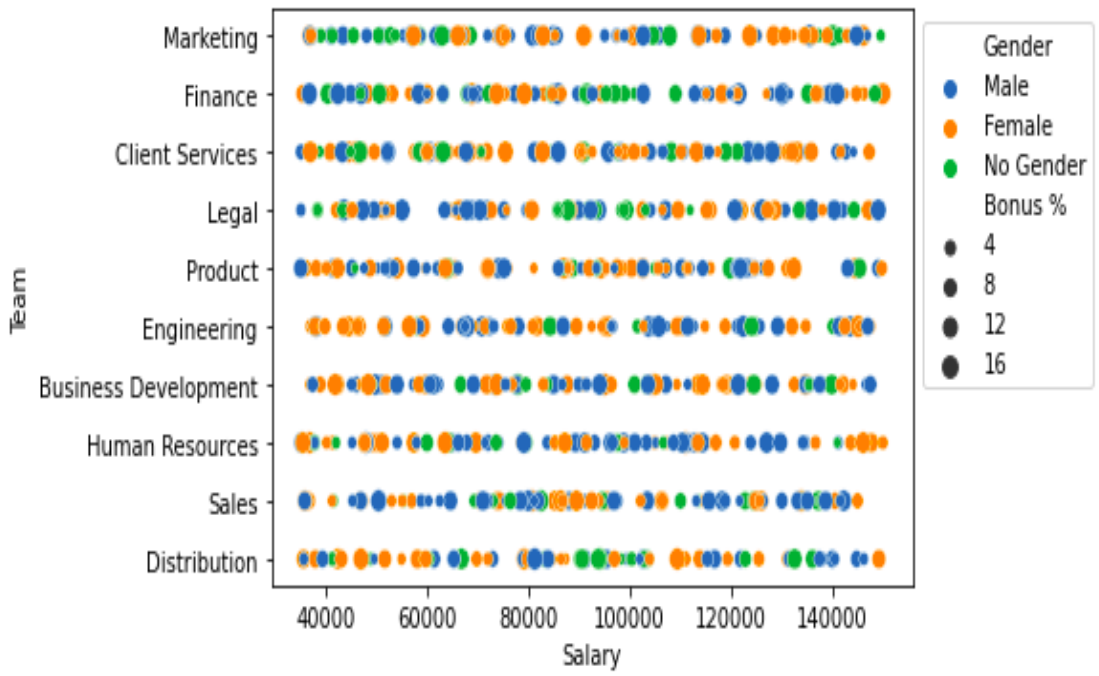
It can be used for bivariate analyses.

Example:

```
# importing packages
import seaborn as sns
import matplotlib.pyplot as plt
sns.scatterplot( x="Salary", y='Team', data=df,
                hue='Gender', size='Bonus %')
```

```
# Placing Legend outside the Figure
plt.legend(bbox_to_anchor=(1, 1), loc=2)
 plt.show()
```

Output:

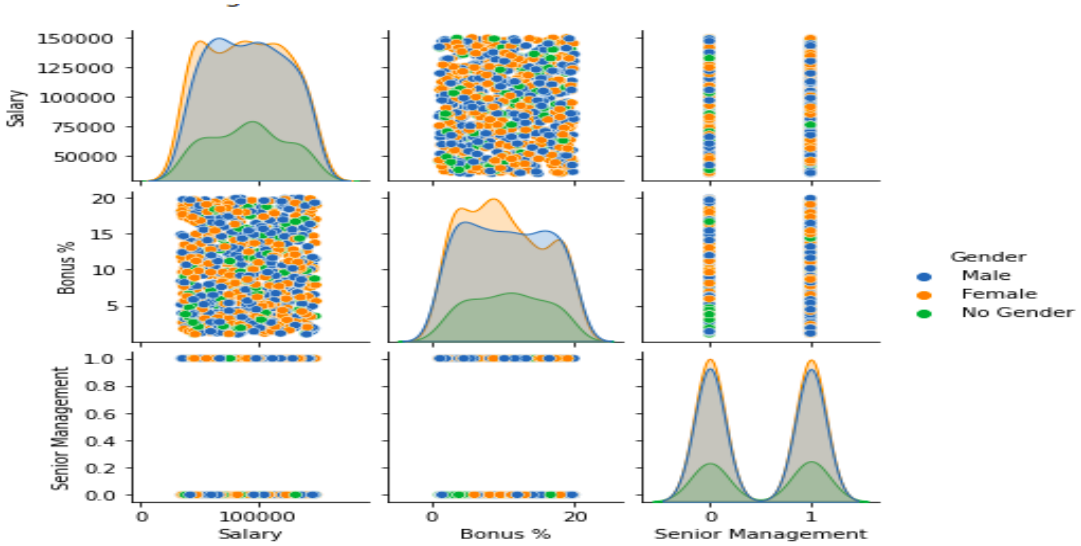


For multivariate analysis, we can the pairplot()method of seaborn module. We can also use it for the multiple pairwise bivariate distributions in a dataset.

Example:

```
# importing packages
import seaborn as sns
import matplotlib.pyplot as plt
sns.pairplot(df, hue='Gender', height=2)
```

Output:



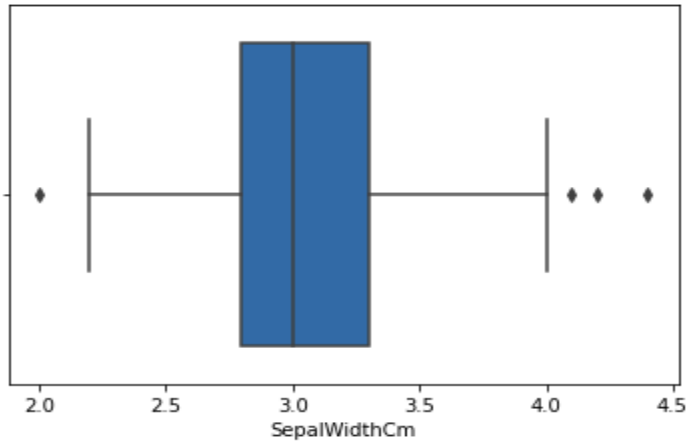
4.Handling Outliers

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

Example:

```
# importing packages
import seaborn as sns
import matplotlib.pyplot as plt

# Load the dataset
df = pd.read_csv('Iris.csv')

sns.boxplot(x='SepalWidthCm', data=df)
```

Output:



In the above graph, the values above 4 and below 2 are acting as outliers.

5.Removing Outliers

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

Example: We will detect the outliers using IQR and then we will remove them. We will also draw the boxplot to see if the outliers are removed or not.

```
# Importing
import sklearn
from sklearn.datasets import load_boston
import pandas as pd
import seaborn as sns
 # Load the dataset
df = pd.read_csv('Iris.csv')
 # IQR
Q1 = np.percentile(df['SepalWidthCm'], 25,
            interpolation = 'midpoint')
 Q3 = np.percentile(df['SepalWidthCm'], 75,
            interpolation = 'midpoint')
IQR = Q3 - Q1

print("Old Shape: ", df.shape)
 # Upper bound
upper = np.where(df['SepalWidthCm'] >= (Q3+1.5*IQR))
 # Lower bound
lower = np.where(df['SepalWidthCm'] <= (Q1-1.5*IQR))
 # Removing the Outliers
df.drop(upper[0], inplace = True)
df.drop(lower[0], inplace = True)
print("New Shape: ", df.shape)
sns.boxplot(x='SepalWidthCm', data=df)
```

Output:

Exploratory Data Analysis (EDA) – Types and Tools:

- Exploratory data analysis was promoted by John Tukey to encourage statisticians to explore data, and possibly formulate hypotheses that might cause new data collection and experiments.
- EDA focuses more narrowly on checking assumptions required for model fitting and hypothesis testing. It also checks while handling missing values and making transformations of variables as needed.filling the counts with
- EDA build a robust understanding of the data, issues associated with either the info or process. it's a scientific approach to get the story of the data.

TYPES OF EXPLORATORY DATA ANALYSIS:

1. Univariate Non-graphical
2. Multivariate Non-graphical
3. Univariate graphical
4. Multivariate graphical

1. Univariate Non-graphical: this is the simplest form of data analysis as during this we use just one variable to research the info. The standard goal of univariate non-graphical EDA is to know the underlying sample distribution/ data and make observations about the population.

Outlier detection is additionally part of the analysis. The characteristics of population distribution include:

- **Central tendency:** The central tendency or location of distribution has got to do with typical or middle values. The commonly useful measures of central tendency are statistics called mean, median, and sometimes mode during which the foremost common is mean. For skewed distribution or when there's concern about outliers, the median may be preferred.
- **Spread:** Spread is an indicator of what proportion distant from the middle we are to seek out the find the info values. the quality deviation and variance are two useful measures of spread. The variance is that the mean of the square of the individual deviations and therefore the variance is the root of the variance
- **Skewness and kurtosis:** Two more useful univariates descriptors are the skewness and kurtosis of the distribution. Skewness is that the measure of asymmetry and kurtosis may be a more subtle measure of peakedness compared to a normal distribution

2. Multivariate Non-graphical: Multivariate non-graphical EDA technique is usually wont to show the connection between two or more variables within the sort of either cross-tabulation or statistics.

- For categorical data, an extension of tabulation called cross-tabulation is extremely useful. For 2 variables, cross-tabulation is preferred by making a two-way table with column headings that match the amount of one-variable and row headings that match the amount of the opposite two variables, then filling the counts with all subjects that share an equivalent pair of levels.
- For each categorical variable and one quantitative variable, we create statistics for quantitative variables separately for every level of the specific variable then compare the statistics across the amount of categorical variable.
- Comparing the means is an off-the-cuff version of ANOVA and comparing medians may be a robust version of one-way ANOVA.

3. Univariate graphical: Non-graphical methods are quantitative and objective, they are doing not give the complete picture of the data; therefore, graphical methods are more involve a degree of subjective analysis, also are required. Common sorts of univariate graphics are:

- **Histogram:** The foremost basic graph is a histogram, which may be a barplot during which each bar represents the frequency (count) or proportion (count/total count) of cases for a variety of values. Histograms are one of the simplest ways to quickly learn a lot about your data, including central tendency, spread, modality, shape and outliers.
- **Stem-and-leaf plots:** An easy substitute for a histogram may be stem-and-leaf plots. It shows all data values and therefore the shape of the distribution.
- **Boxplots:** Another very useful univariate graphical technique is that the boxplot. Boxplots are excellent at presenting information about central tendency and show robust measures of location and spread also as providing information about symmetry and outliers, although they will be misleading about aspects like multimodality. One among the simplest uses of boxplots is within the sort of side-by-side boxplots.
- **Quantile-normal plots:** The ultimate univariate graphical EDA technique is that the most intricate. it's called the quantile-normal or QN plot or more generally the quantile-quantile or QQ plot. it's wont to see how well a specific sample follows a specific theoretical distribution. It allows detection of non-normality and diagnosis of skewness and kurtosis

4. Multivariate graphical: Multivariate graphical data uses graphics to display relationships between two or more sets of knowledge. The sole one used commonly may be a grouped barplot with each group representing one level of 1 of the variables and every bar within a gaggle representing the amount of the opposite variable.

Other common sorts of multivariate graphics are:

- **Scatterplot:** For 2 quantitative variables, the essential graphical EDA technique is that the scatterplot , sohas one variable on the x-axis and one on the y-axis and therefore the point for every case in your dataset.
- **Run chart:** It's a line graph of data plotted over time.
- **Heat map:** It's a graphical representation of data where values are depicted by color.
- **Multivariate chart:** It's a graphical representation of the relationships between factors and response.
- **Bubble chart:** It's a data visualization that displays multiple circles (bubbles) in two-dimensional plot.

In a nutshell: You ought to always perform appropriate EDA before further analysis of your data. Perform whatever steps are necessary to become more conversant in your data, check for obvious mistakes, learn about variable distributions, and study about relationships between variables. EDA is not an exact science- It is very important are!

TOOLS REQUIRED FOR EXPLORATORY DATA ANALYSIS:

Some of the most common tools used to create an EDA are:

1. R: An open-source programming language and free software environment for statistical computing and graphics supported by the R foundation for statistical computing. The R language is widely used among statisticians in developing statistical observations and data analysis.

2. Python: An interpreted, object-oriented programming language with dynamic semantics. Its high level, built-in data structures, combined with dynamic binding, make it very attractive for rapid application development, also as to be used as a scripting or glue language to attach existing components together. Python and EDA are often used together to spot missing values in the data set, which is vital so you'll decide the way to handle missing values for machine learning.

Apart from these functions described above, EDA can also:

- **Perform k-means clustering:** Perform k-means clustering: it's an unsupervised learning algorithm where the info points are assigned to clusters, also referred to as k-groups, k-means clustering is usually utilized in market segmentation, image compression, and pattern recognition
- EDA is often utilized in predictive models like linear regression, where it's wont to predict outcomes.
- It is also utilized in univariate, bivariate, and multivariate visualization for summary statistics, establishing relationships between each variable, and understanding how different fields within the data interact with one another.

Data Science Lifecycle

- Data Science Lifecycle revolves around the use of machine learning and different analytical strategies to produce insights and predictions from information in order to acquire a commercial enterprise objective.
- The complete method includes a number of steps like data cleaning, preparation, modelling, model evaluation, etc. It is a lengthy procedure and may additionally take quite a few months to complete.
- So, it is very essential to have a generic structure to observe for each and every hassle at hand. The globally mentioned structure in fixing any analytical problem is referred to as a Cross Industry Standard Process for Data Mining or CRISP-DM framework.

Let us understand what is the need for Data Science?

Earlier data used to be much less and generally accessible in a well-structured form, that we could save effortlessly and easily in Excel sheets, and with the help of Business Intelligence tools data can be processed efficiently.

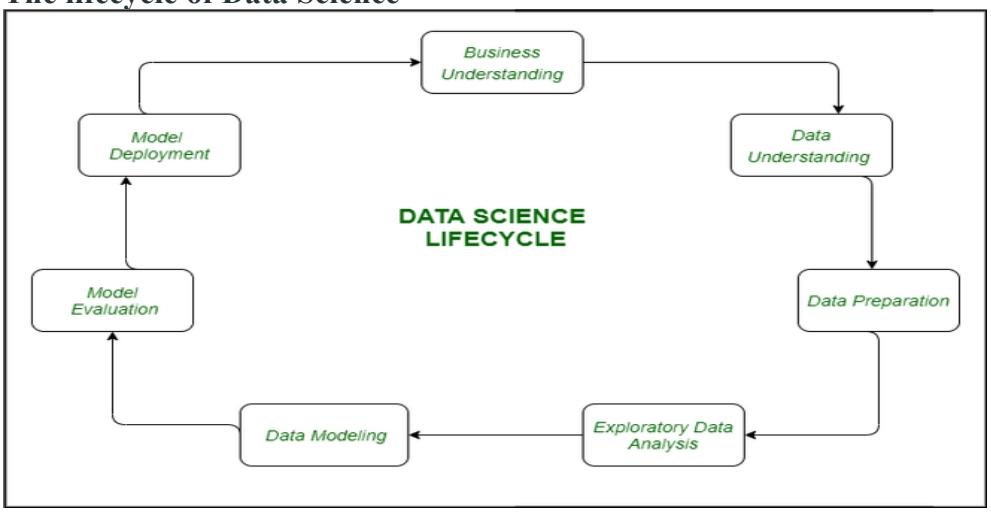
But Today we used to deals with large amounts of data like about 3.0 quintals bytes of records is producing on each and every day, which ultimately results in an explosion of records and data. According to recent researches, It is estimated that 1.9 MB of data and records are created in a second that too through a single individual.

So this a very big challenge for any organization to deal with such a massive amount of data generating every second. For handling and evaluating this data we required some very powerful, complex algorithms and technologies and this is where Data science comes into the picture.

The following are some primary motives for the use of Data science technology:

1. It helps to convert the big quantity of uncooked and unstructured records into significant insights.
2. It can assist in unique predictions such as a range of surveys, elections, etc.
3. It also helps in automating transportation such as growing a self-driving car, we can say which is the future of transportation.
4. Companies are shifting towards Data science and opting for this technology. Amazon, Netflix, etc, which cope with the big quantity of data, are the use of information science algorithms for higher consumer experience.

The lifecycle of Data Science



1. Business Understanding:

- The complete cycle revolves around the enterprise goal. What will you resolve if you do no longer have a specific problem? It is extraordinarily essential to apprehend the commercial enterprise goal sincerely due to the fact that will be your ultimate aim of the analysis.
- After desirable perception only we can set the precise aim of evaluation that is in sync with the enterprise objective. You need to understand if the customer desires to minimize savings loss, or if they prefer to predict the rate of a commodity, etc.

2. Data Understanding:

- After enterprise understanding, the subsequent step is data understanding. This includes a series of all the reachable data.
- Here you need to intently work with the commercial enterprise group as they are certainly conscious of what information is present, what facts should be used for this commercial enterprise problem, and different information.
- This step includes describing the data, their structure, their relevance, their records type. Explore the information using graphical plots. Basically, extracting any data that you can get about the information through simply exploring the data.

3. Preparation of Data:

- Next comes the data preparation stage. This consists of steps like choosing the applicable data, integrating the data by means of merging the data sets, cleaning it, treating the lacking values through either eliminating them or imputing them, treating inaccurate data through eliminating them, additionally test for outliers the use of box plots and cope with them.
- Constructing new data, derive new elements from present ones. Format the data into the preferred structure, eliminate undesirable columns and features. Data preparation is the most time-consuming but arguably the most essential step in the complete existence cycle. Your model will be as accurate as your data.

4. Exploratory Data Analysis:

- This step includes getting some concept about the answer and elements affecting it, earlier than constructing the real model. Distribution of data inside distinctive variables of a character is explored graphically the usage of bar-graphs, Relations between distinct aspects are captured via graphical representations like scatter plots and warmth maps.
- Many data visualization strategies are considerably used to discover each and every characteristic individually and by means of combining them with different features.

5. Data Modeling:

- Data modeling is the coronary heart of data analysis. A model takes the organized data as input and gives the preferred output. This step consists of selecting the suitable kind of model, whether the problem is a classification problem, or a regression problem or a clustering problem.
- After deciding on the model family, amongst the number of algorithms amongst that family, we need to cautiously pick out the algorithms to put into effect and enforce them. We need to tune the hyperparameters of every model to obtain the preferred performance. We additionally need to make positive there is the right stability between overall performance and generalizability. We do no longer desire the model to study the data and operate poorly on new data.

6. Model Evaluation:

- Here the model is evaluated for checking if it is geared up to be deployed. The model is examined on an unseen data, evaluated on a cautiously thought out set of assessment metrics. We additionally need to make positive that the model conforms to reality.
- If we do not acquire a quality end result in the evaluation, we have to re-iterate the complete modelling procedure until the preferred stage of metrics is achieved. Any data science solution, a machine learning model, simply like a human, must evolve, must be capable to enhance itself with new data, adapt to a new evaluation metric.
- We can construct more than one model for a certain phenomenon, however, a lot of them may additionally be imperfect. The model assessment helps us select and construct an ideal model.

7. Model Deployment:

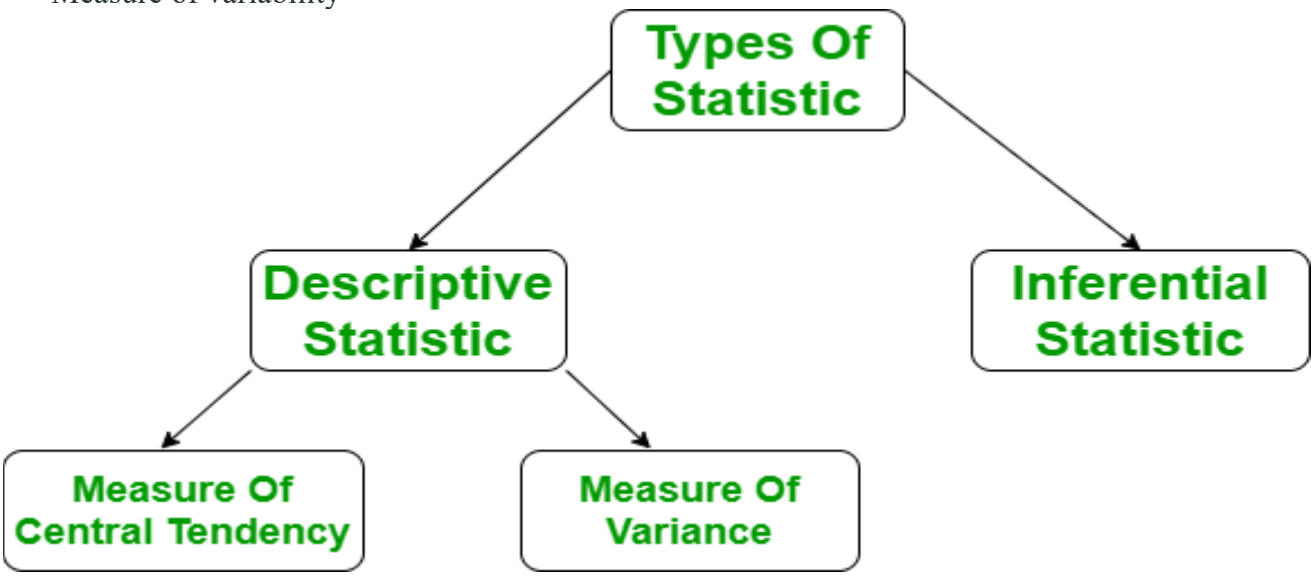
- The model after a rigorous assessment is at the end deployed in the preferred structure and channel. This is the last step in the data science life cycle. Each step in the data science life cycle defined above must be laboured upon carefully.
- If any step is performed improperly, and hence, have an effect on the subsequent step and the complete effort goes to waste. For example, if data is no longer accumulated properly, you'll lose records and you will no longer be constructing an ideal model.
- If information is not cleaned properly, the model will no longer work. If the model is not evaluated properly, it will fail in the actual world. Right from Business perception to model deployment, every step has to be given appropriate attention, time, and effort

Descriptive Statistic

- In Descriptive statistics, we are describing our data with the help of various representative methods like by using charts, graphs, tables, excel files etc. In descriptive statistics, we describe our data in some manner and present it in a meaningful way so that it can be easily understood.
- Most of the times it is performed on small data sets and this analysis helps us a lot to predict some future trends based on the current findings. Some measures that are used to describe a data set are measures of central tendency and measures of variability or dispersion.

Types of Descriptive statistic:

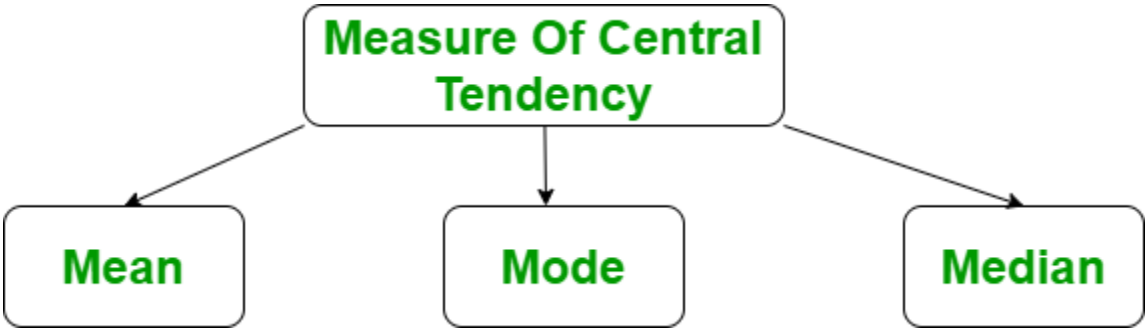
- Measure of central tendency
- Measure of variability



Measure of central tendency:

It represents the whole set of data by single value.It gives us the location of central points. There are three main measures of central tendency:

- Mean
- Mode
- Median



1. **Mean:**
   It is the sum of observation divided by the total number of observations. It is also defined as average which is the sum divided by count.

$$\text{Mean } (\bar{x}) = \frac{\sum x}{n}$$

where, n = number of terms

Python Code to find Mean in python

```
import numpy as np
  # Sample Data
arr = [5, 6, 11]
# Mean
mean = np.mean(arr)
  print("Mean = ", mean)
```

Output :

Mean = 7.333333333333333

2. **Mode:**

It is the value that has the highest frequency in the given data set. The data set may have no mode if the frequency of all data points is the same. Also, we can have more than one mode if we encounter two or more data points having the same frequency.

Code to find Mode in python

```
from scipy import stats
  # sample Data
arr =[1, 2, 2, 3]
  # Mode
mode = stats.mode(arr)
print("Mode = ", mode)
```

Output:

Mode = ModeResult(mode=array([2]), count=array([2]))

3. **Median:**

It is the middle value of the data set. It splits the data into two halves. If the number of elements in the data set is odd then the centre element is median and if it is even then the median would be the average of two central elements.

Odd	**Even**
$\frac{n+1}{2}$	$\frac{n}{2}, \frac{n}{2}+1$

where, n=number of terms

Python code to find Median

```
import numpy as np
  # sample Data
arr =[1, 2, 3, 4]
  # Median
median = np.median(arr)
  print("Median = ", median)
```

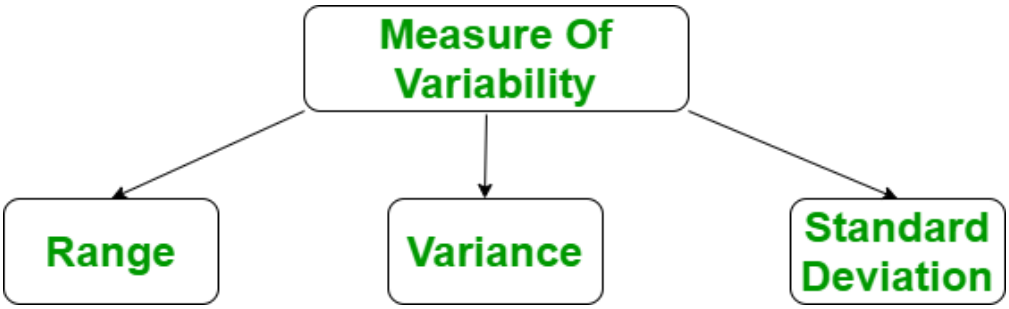
Output:

Median = 2.5

Measure of variability:

Measure of variability is known as the spread of data or how well is our data is distributed. The most common variability measures are:

- Range
- Variance
- Standard deviation



1. **Range:**
   The range describes the difference between the largest and smallest data point in our data set. The bigger the range, the more is the spread of data and vice versa.
   Range = Largest data value – smallest data value
   Python Code to find Range

```
import numpy as np

# Sample Data
arr = [1, 2, 3, 4, 5]

#Finding Max
Maximum = max(arr)
# Finding Min
Minimum = min(arr)

# Difference Of Max and Min
Range = Maximum-Minimum
print("Maximum = {}, Minimum = {} and Range = {}".format(
      Maximum, Minimum, Range))
```

   Output:
   Maximum = 5, Minimum = 1 and Range = 4
2. **Variance:**
   It is defined as an average squared deviation from the mean. It is being calculated by finding the difference between every data point and the average which is also known as the mean, squaring them, adding all of them and then dividing by the number of data points present in our data set.

$$\sigma^2 = \frac{\sum(x - \mu)^2}{N}$$

   where N = number of terms
   u = Mean
   Python code to find Variance

```
import statistics
 # sample data
arr = [1, 2, 3, 4, 5]
# variance
print("Var = ", (statistics.variance(arr)))
```

Output:
Var = 2.5

3. **Standard Deviation:**
It is defined as the square root of the variance. It is being calculated by finding the Mean, then subtract each number from the Mean which is also known as average and square the result. Adding all the values and then divide by the no of terms followed the square root.

$$\sigma = \sqrt{\frac{\Sigma (x - u)^2}{N}}$$

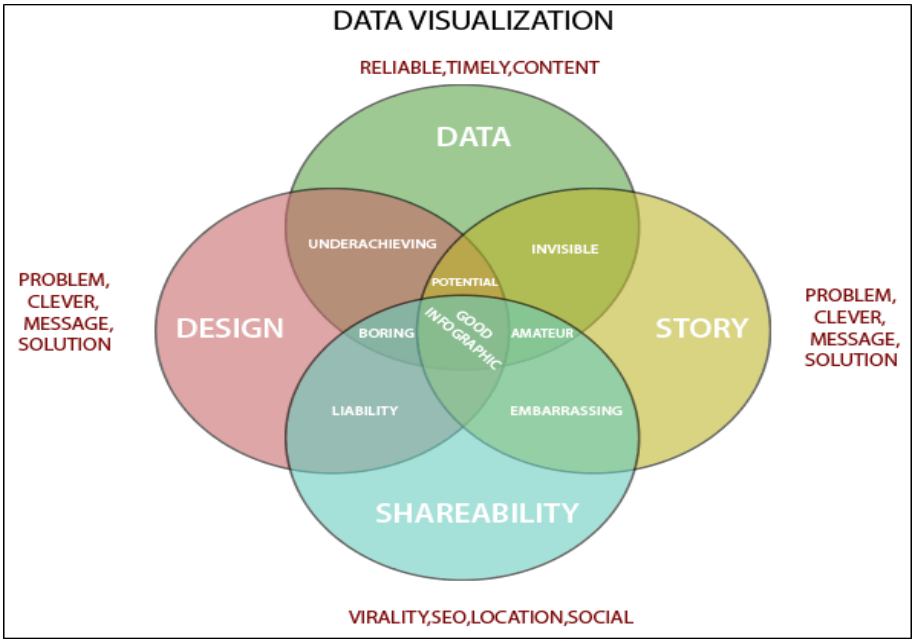
where N = number of terms
u = Mean
Python code to perform Standard Deviation:

```
import statistics

# sample data
arr = [1, 2, 3, 4, 5]
# Standard Deviation
print("Std = ", (statistics.stdev(arr)))
```

Output:
Std = 1.5811388300841898

Data Visualization

- Data visualization is a graphical representation of quantitative information and data by using visual elements like graphs, charts, and maps.
- Data visualization convert large and small data sets into visuals, which is easy to understand and process for humans.
- Data visualization tools provide accessible ways to understand outliers, patterns, and trends in the data.
- In the world of Big Data, the data visualization tools and technologies are required to analyze vast amounts of information.
- Effective data visualization are created by communication, data science, and design collide. Data visualizations did right key insights into complicated data sets into meaningful and natural.



To craft an effective data visualization, you need to start with clean data that is well-sourced and complete. After the data is ready to visualize, you need to pick the right chart.
After you have decided the chart type, you need to design and customize your visualization to your liking. Simplicity is essential - you don't want to add any elements that distract from the data.

UNIT-4

What is Pattern Recognition ?

Pattern recognition is the process of recognizing patterns by using machine learning algorithm. Pattern recognition can be defined as the classification of data based on knowledge already gained or on statistical information extracted from patterns and/or their representation. One of the important aspects of the pattern recognition is its application potential.

Examples: Speech recognition, speaker identification, multimedia document recognition (MDR), automatic medical diagnosis.

In a typical pattern recognition application, the raw data is processed and converted into a form that is amenable for a machine to use. Pattern recognition involves classification and cluster of patterns.

- In classification, an appropriate class label is assigned to a pattern based on an abstraction that is generated using a set of training patterns or domain knowledge. Classification is used in supervised learning.
- Clustering generated a partition of the data which helps decision making, the specific decision making activity of interest to us. Clustering is used in an unsupervised learning.

Features may be represented as continuous, discrete or discrete binary variables. A feature is a function of one or more measurements, computed so that it quantifies some significant characteristics of the object.

Example: consider our face then eyes, ears, nose etc are features of the face.

A set of features that are taken together, forms the **features vector**.

Example: In the above example of face, if all the features (eyes, ears, nose etc) taken together then the sequence is feature vector([eyes, ears, nose]). Feature vector is the sequence of a features represented as a d-dimensional column vector.

Pattern recognition possesses the following features:

- Pattern recognition system should recognise familiar pattern quickly and accurate
- Recognize and classify unfamiliar objects
- Accurately recognize shapes and objects from different angles
- Identify patterns and objects even when partly hidden
- Recognise patterns quickly with ease, and with automaticity.

Pattern Recognition Approaches:

- **Statistical.** This approach is based on statistical decision theory. Pattern recognizer extracts quantitative features from the data along with the multiple samples and compares those features. However, it does not touch upon how those features are related to each other.
- **Structural (a.k.a. syntactic).** This approach is closer to how human perception works. It extracts morphological features from one data sample and checks how those are connected and related.
- **Neural.** In this approach, artificial neural networks are utilized. Compared to the ones mentioned above, it allows more flexibility in learning and is the closest to natural intelligence.

The basic components of pattern recognition systems

Every machine learning-based pattern recognition algorithm includes the following steps.

- **Input of data.** Large amounts of data enter the system through different sensors.
- **Preprocessing or segmentation.** At this stage, the system groups the input data to prepare the sets for future analysis.
- **Feature selection (extraction).** The system searches for and determines the distinguishing traits of the prepared sets of data.
- **Classification.** Based on the features detected in the previous step, data is assigned a class (or cluster), or predicted values are calculated (in the case of regression algorithms).
- **Postprocessing.** According to the outcome of the recognition, the system performs future actions.

Pattern recognition examples

Examples of pattern recognition can be easily found in nature, like humans recognizing faces or pets responding to their names. In technology, pattern recognition algorithms trained through machine learning are applied in various fields, ranging from everyday tasks to highly specialized areas.

Here are some typical examples:

- NLP (natural language processing): virtual assistants, speech-to-text interfaces, automatic captioning;
- OCR scanners (optical character recognition): mobile scanner apps;
- Medical diagnostic software;
- Meteorological forecast software;
- NIDS (Network intrusion detection systems): security systems, which recognize patterns of suspicious activities.

Advantages of Pattern Recognition

Using pattern recognition techniques provides a large number of benefits to an individual. It not only helps in the analysis of trends but also helps in making predictions.

- It helps in the **identification** of objects at varying distances and angles.
- Easy and highly **automated.**
- It is not rocket science and does not require an out of the box thinking ability.
- Highly useful in the finance industry to make **valuable predictions** regarding sales.
- Efficient solutions to **real-time problems**.
- Useful in the medical fields for **forensic analysis** and DNA (Deoxyribonucleic acid) sequencing.

Applications of Pattern **Recognition**

- **Trend Analysis–** Pattern recognition helps in identifying the trend in the given data on which appropriate analysis can be done. For example, looking at the recent trends in the sales made by a particular company or organization, future sales can be predicted.
- **Assistance –** Pattern is an integral part of our daily lives. It provides immense help in our day to day activities. A large number of software and applications are there in the market today that use machine learning algorithms to make predictions regarding the presence of obstacles and alerts the user to void miss happenings.
- **E-Commerce –** Visual search engines recognize the desired item based on its specifications and provide appropriate results. Most of the websites dedicated to online shopping make use of recommender systems. These systems collect data related to each customer purchase and make suggestions. All these tasks are accomplished by analyzing previous trends to make successful predictions.
- **Computer vision–** The user interacts with the system by giving an image or video as the input. The machine compares it with thousands or maybe millions of images stored in its database, to find similar patterns. The drawl of the essential features is done by using an algorithm that is mainly directed for grouping similar looking objects and patterns. This is termed as computer vision. Example, cancer detection.

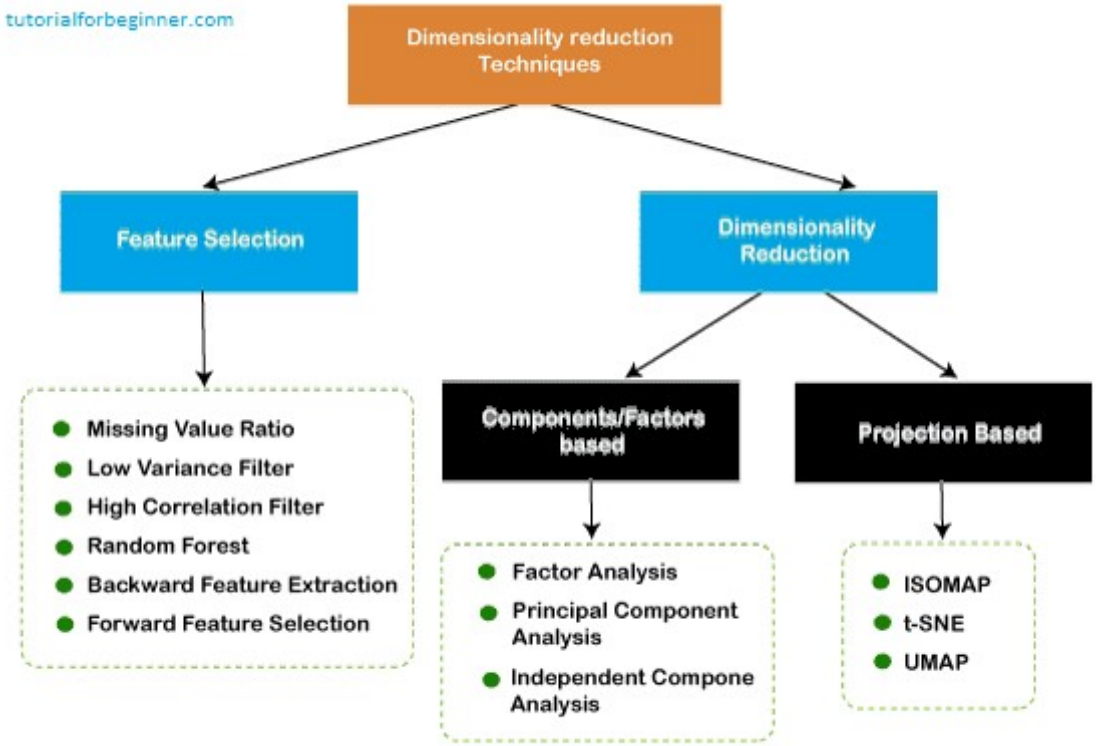
  Biometric devices– These devices secure authentication and security by making using of face recognition and fingerprint detection technologies. On the hidden side, the base that enables the use of technologies like face and fingerprint recognition is machine learning algorithms.

What is Dimensionality Reduction?

- Dimensionality refers to the amount of input characteristics, variables, or columns in a dataset, and dimensionality reduction refers to the process of reducing these features.
- In some circumstances, a dataset has a large number of input features, making predictive modeling more difficult. Because it is difficult to visualize or forecast a training dataset with a large number of characteristics, dimensionality reduction techniques must be used in such circumstances.
- The term "dimensionality reduction technique" refers to the process of reducing the number of dimensions. It can be defined as, ***"It is a way of converting the higher dimensions dataset into lesser dimensions***

dataset ensuring that it provides similar information." These methods are commonly utilized in machine learning to develop a more accurate predictive model while solving classification and regression challenges.

- Speech recognition, signal processing, bioinformatics, and other fields that deal with high-dimensional data use it frequently. It can also be used to visualize data, reduce noise, and do cluster analysis, among other things.



Benefits of applying Dimensionality Reduction

The following are some of the advantages of using a dimensionality reduction technique on a given dataset:

- The space required to store the dataset is lowered by lowering the dimensionality of the features.
- Reduced feature dimensions necessitate less Computation training time.
- The reduced dimensions of the dataset's features aid in quickly displaying the data.
- It takes care of multicollinearity to remove redundant features (if any are present).

Disadvantages of dimensionality Reduction

There are a few drawbacks to using dimensionality reduction, which are listed below:

- Due to the reduction in dimensionality, some data may be lost.
- The principal components that must be considered in the PCA dimensionality reduction technique are occasionally unknown.

What is machine learning:

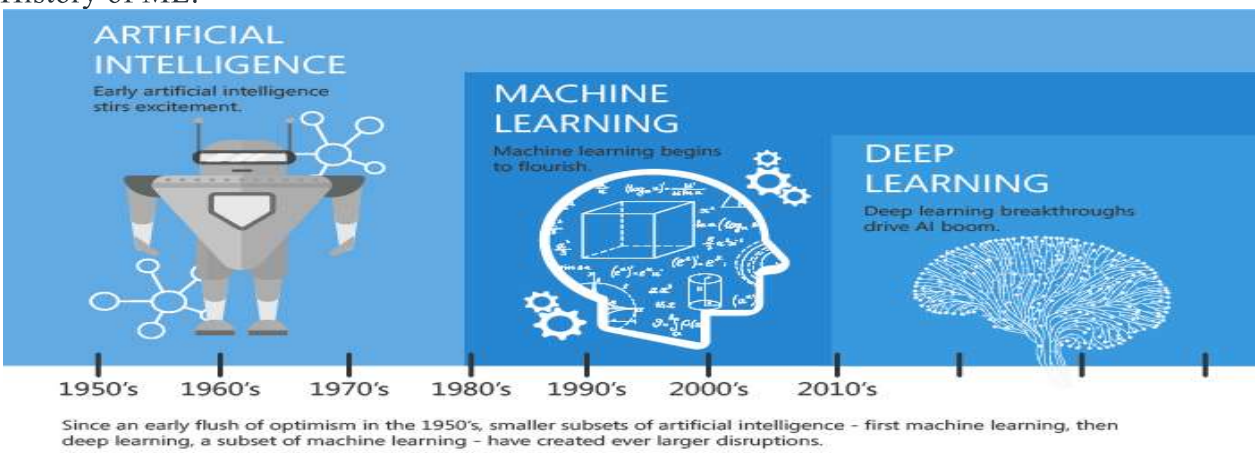
Arthur Samuel, a pioneer in the field of artificial intelligence and computer gaming, coined the term **"Machine Learning"**. He defined machine learning as – **"Field of study that gives computers the capability to learn without being explicitly programmed"**.

In a very layman manner, Machine Learning(ML) can be explained as automating and improving the learning process of computers based on their experiences without being actually programmed i.e. without any human assistance.

Basic Difference in ML and Traditional Programming?

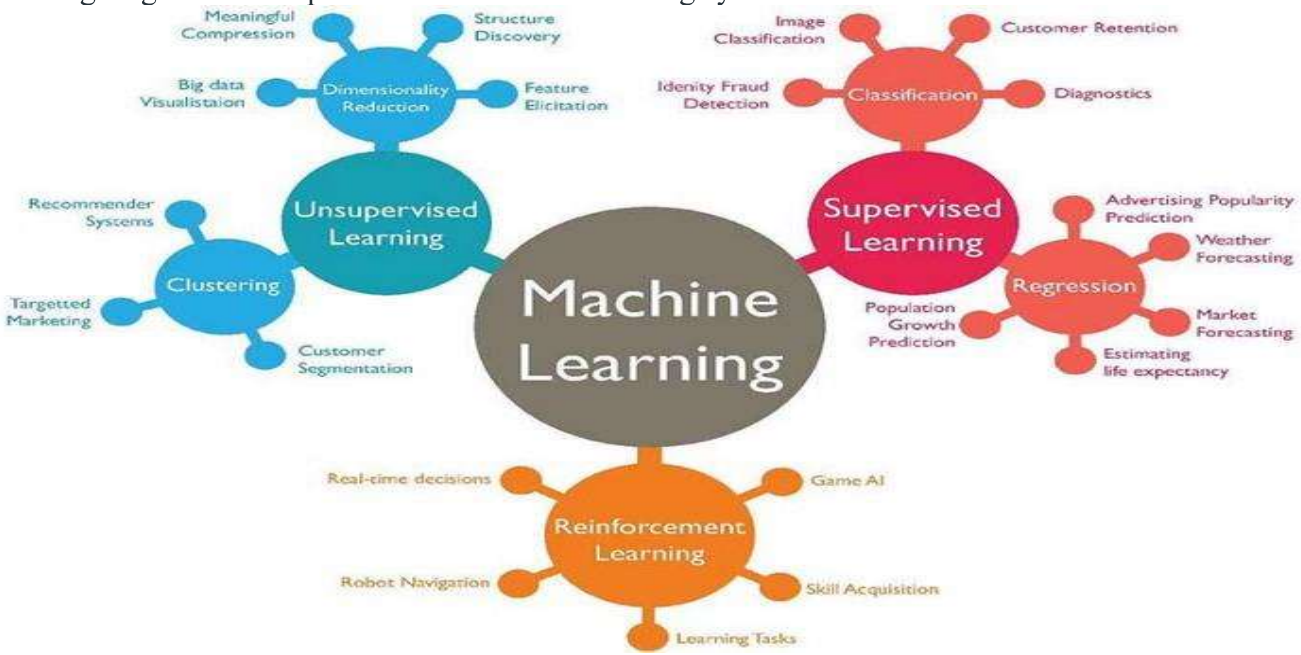
- **Traditional Programming :** We feed in DATA (Input) + PROGRAM (logic), run it on machine and get output.
- **Machine Learning :** We feed in DATA(Input) + Output, run it on machine during training and the machine creates its own program(logic), which can be evaluated while testing.

History of ML:

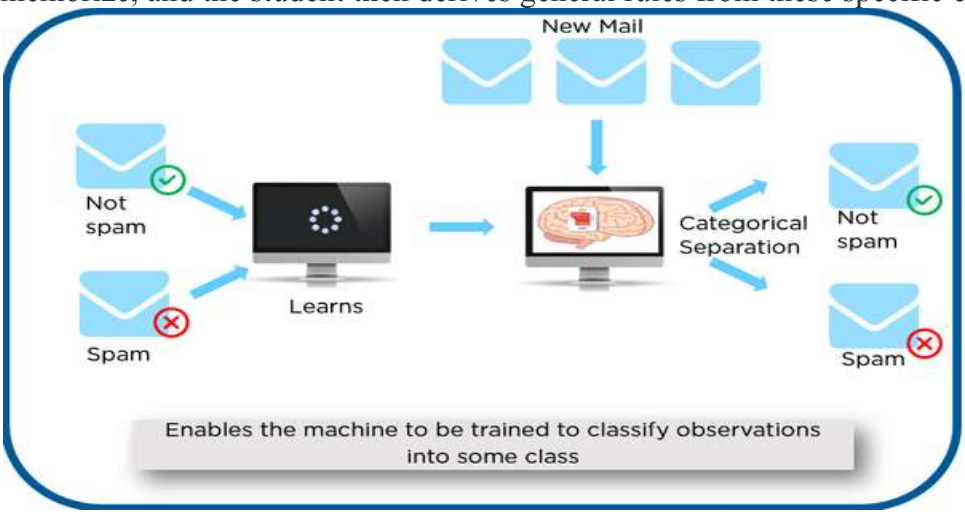


Classification of Machine Learning

Machine learning implementations are classified into three major categories, depending on the nature of the learning "signal" or "response" available to a learning system which is as follows:-

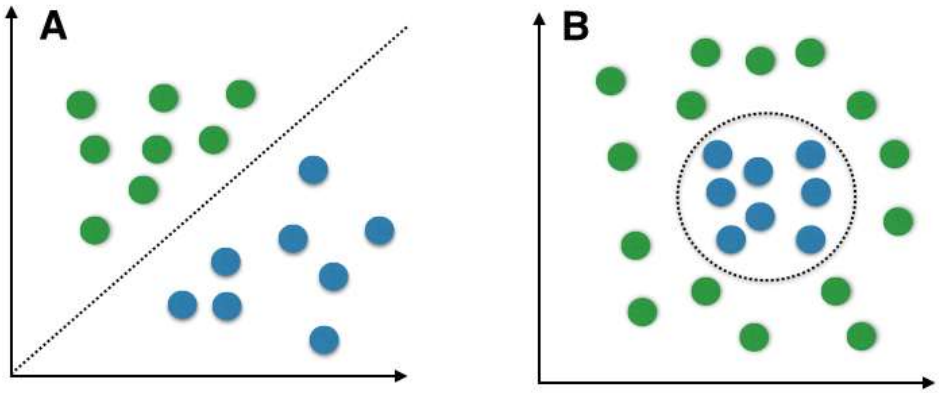


1. **Supervised learning:** When an algorithm learns from example data and associated target responses that can consist of numeric values or string labels, such as classes or tags, in order to later predict the correct response when posed with new examples comes under the category of Supervised learning. This approach is indeed similar to human learning under the supervision of a teacher. The teacher provides good examples for the student to memorize, and the student then derives general rules from these specific examples.



Types of Supervised learning

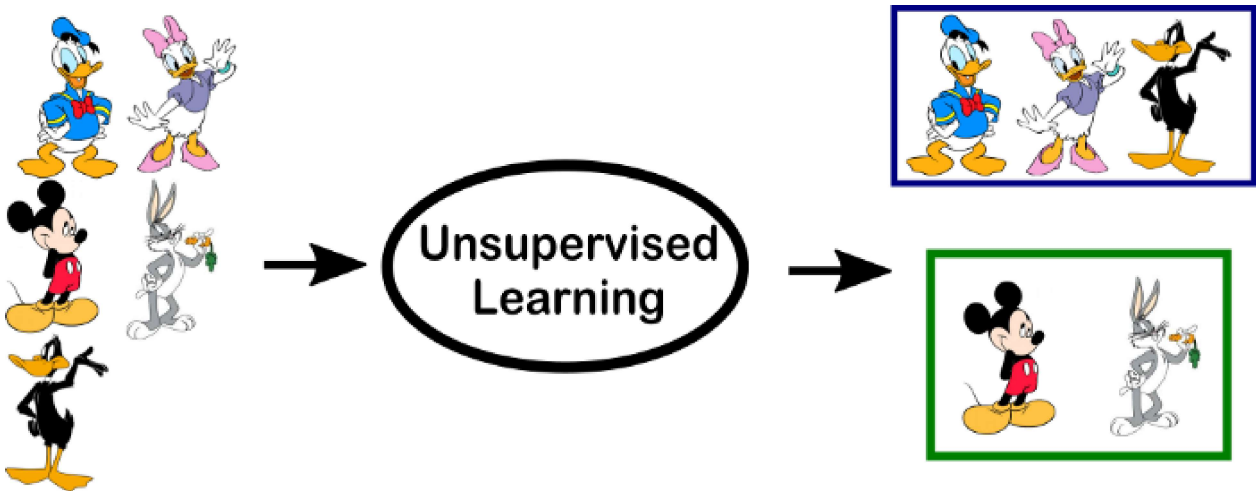
- **Classification**: A classification problem is when the output variable is a category, such as "red" or "blue" or "disease" and "no disease".



- **Regression**: A regression problem is when the output variable is a real value, such as "dollars" or "weight".

2. **Unsupervised learning:** Whereas when an algorithm learns from plain examples without any associated response, leaving to the algorithm to determine the data patterns on its own. This type of algorithm tends to restructure the data into something else, such as new features that may represent a class or a new series of un-correlated values. They are quite useful in providing humans with insights into the meaning of data and new useful inputs to supervised machine learning algorithms.

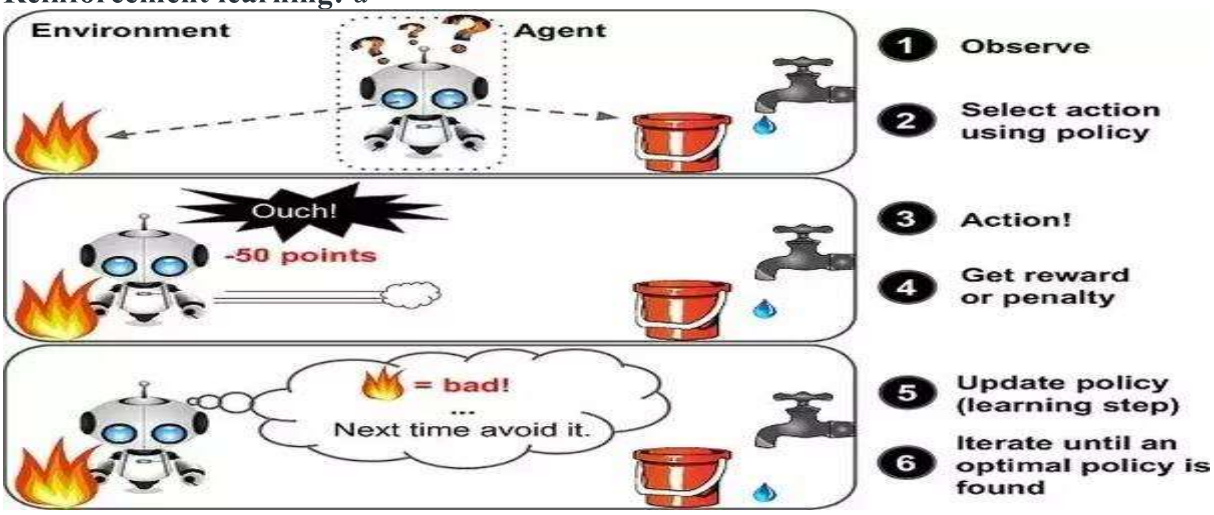
As a kind of learning, it resembles the methods humans use to figure out that certain objects or events are from the same class, such as by observing the degree of similarity between objects. Some recommendation systems that you find on the web in the form of marketing automation are based on this type of learning.



Types of Unsupervised learning

- **Clustering**: A clustering problem is where you want to discover the inherent groupings in the data, such as grouping customers by purchasing behavior.
- **Association**: An association rule learning problem is where you want to discover rules that describe large portions of your data, such as people that buy X also tend to buy Y.

3. **Reinforcement learning:** a



Categorizing on the basis of required Output

Another categorization of machine learning tasks arises when one considers the desired output of a machine-learned system:

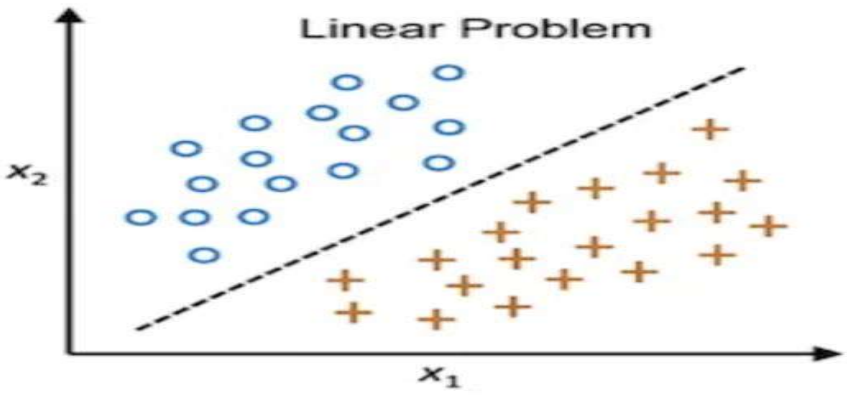
1. **Classification:** When inputs are divided into two or more classes, and the learner must produce a model that assigns unseen inputs to one or more (multi-label classification) of these classes. This is typically tackled in a supervised way. Spam filtering is an example of classification, where the inputs are email (or other) messages and the classes are “spam” and “not spam”.
2. **Regression:** Which is also a supervised problem, A case when the outputs are continuous rather than discrete.
3. **Clustering:** When a set of inputs is to be divided into groups. Unlike in classification, the groups are not known beforehand, making this typically an unsupervised task.

Classification

Linear Classification:

→ Linear Classification refers to categorizing a set of data points into a discrete class based on a linear combination of its explanatory variables.

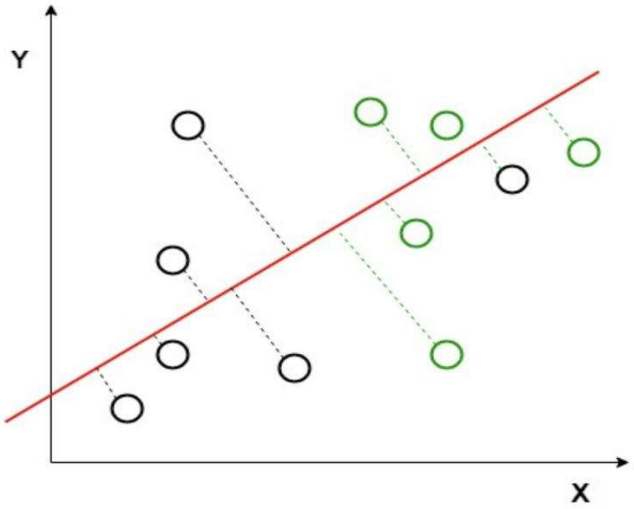
→ Some of the classifiers that use linear functions to separate classes are *Linear Discriminant Classifier, Naive Bayes, Logistic Regression, Perceptron, SVM (linear kernel)*.



→ In the figure above, we have two classes, namely 'O' and '+.' To differentiate between the two classes, an arbitrary line is drawn, ensuring that both the classes are on distinct sides.
→ Since we can tell one class apart from the other, these classes are called 'linearly-separable.'
→ However, an infinite number of lines can be drawn to distinguish the two classes.
→ The exact location of this plane/hyperplane depends on the type of the linear classifier.

Linear Discriminant Classifier :

→ It is a dimensionality reduction technique in the domain of Supervised Machine Learning.
→ It is crucial in modeling differences between two groups, i.e., classes.
→ It helps project features in a high dimensions space in a lower-dimensional space.
→ Technique - Linear Discriminant Analysis (LDA) is used, which reduced the 2D graph into a 1D graph by creating a new axis. This helps to maximize the distance between the two classes for differentiation.



→ In the above graph, we notice that a new axis is created, which maximizes the distance between the mean of the two classes.
→ As a result, variation within each class is also minimized.
→ However, the problem with LDA is that it would fail in case the means of both the classes are the same. This would mean that we would not be able to generate a new axis for differentiating the two.

Naive Bayes:
→ It is based on the Bayes Theorem and lies in the domain of Supervised Machine Learning.
→ Every feature is considered equal and independent of the others during Classification.
→ Naive Bayes indicates the likelihood of occurrence of an event. It is also known as conditional probability.

$$P(A \mid B) = \frac{P(B \mid A) \cdot P(A)}{P(B)}$$

A: event 1
B: event 2
P(A|B): Probability of A being true given B is true - posterior probability
P(B|A): Probability of B being true given A is true - the likelihood
P(A): Probability of A being true - prior
P(B): Probability of B being true - marginalization
However, in the case of the Naive Bayes classifier, we are concerned only with the maximum posterior probability, so we ignore the denominator, i.e., the marginal likelihood. Argmax does not depend on the normalization term.

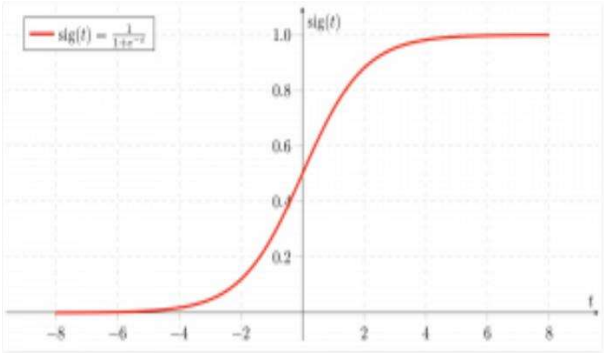
→ **The Naive Bayes classifier is based on two essential assumptions:-**
(i) **Conditional Independence -** All features are independent of each other. This implies that one feature does not affect the performance of the other. This is the sole reason behind the 'Naive' in 'Naive Bayes.'

(ii) **Feature Importance -** All features are equally important. It is essential to know all the features to make good predictions and get the most accurate results.
→ Naive Bayes is classified into three main types: Multinomial Naive Bayes, Bernoulli Naive Bayes, and Gaussian Bayes.

Logistic Regression :

→ It is a very popular supervised machine learning algorithm.
→ The target variable can take only discrete values for a given set of features.
→ The model builds a regression model to predict the probability of a given data entry.
→ Similar to linear regression, logistic regression uses a linear function and, in addition, makes use of the 'sigmoid' function.

$$g(z) = \frac{1}{1+e^{-z}}$$

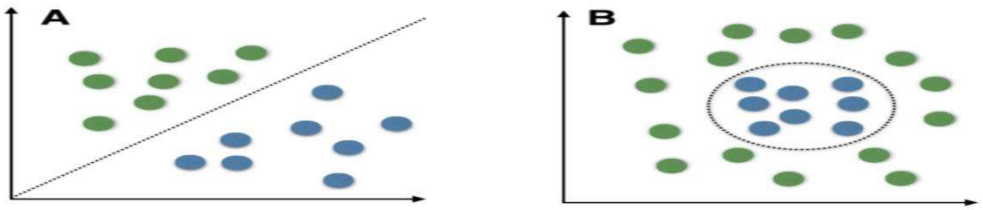


→ **Logistic regression can be further classified into three categories:-**

- Binomial - target variable assumes only two values since binary. Example: ‘0’ or ‘1’.
- Multinomial - target variable assumes >= three unordered values since multinomial. Example: 'Class A,' 'Class B,' and 'Class C.'
- Ordinal - target variable assumes ordered values since ordinal. Example: ‘Very Good’, ‘Good’, ‘Average, ‘poor’, ‘very poor’.

Support Vector Machine (linear kernel)

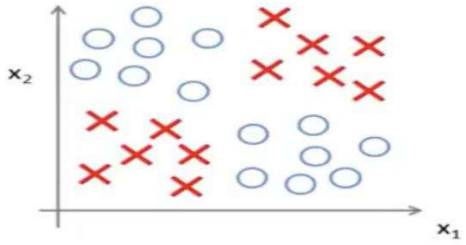
→ It is a straightforward supervised machine learning algorithm used for regression/classification.
→ This model finds a hyper-plane that creates a boundary between the various data types.
→ It can be used for binary Classification as well as multinomial classification problems.
→ A binary classifier can be created for each class to perform multi-class Classification.
→ In the case of SVM, the classifier with the highest score is chosen as the output of the SVM.
→ SVM works very well with linearly separable data but can work for non-linearly separable data as well.



A: Linearly Separable Data B: Non-Linearly Separable Data

Non-Linear Classification:

→ Non-Linear Classification refers to categorizing those instances that are not linearly separable.
→ Some of the classifiers that use non-linear functions to separate classes are Quadratic Discriminant Classifier, Multi-Layer Perceptron (MLP), Decision Trees, Random Forest, and K-Nearest Neighbours (KNN).



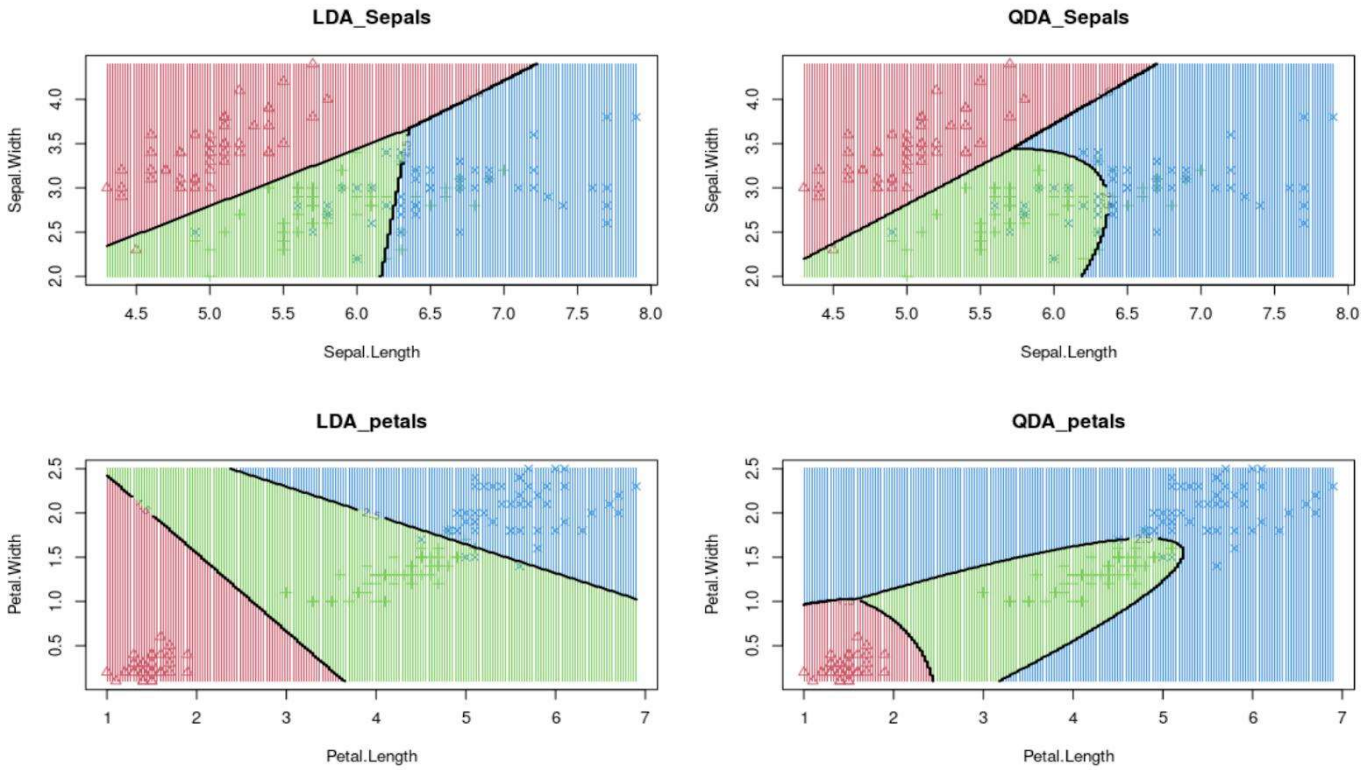
→ In the figure above, we have two classes, namely 'O' and 'X.' To differentiate between the two classes, it is impossible to draw an arbitrary straight line to ensure that both the classes are on distinct sides.

→ We notice that even if we draw a straight line, there would be points of the first-class present between the data points of the second class.
→ In such cases, piece-wise linear or non-linear classification boundaries are required to distinguish the two classes.

Quadratic Discriminant Classifier
→ This technique is similar to LDA(Linear Discriminant Analysis) discussed above.
→ The only difference is that here, we do not assume that the mean and covariance of all classes are the same.
→ We get the quadratic discriminant function as the following:-

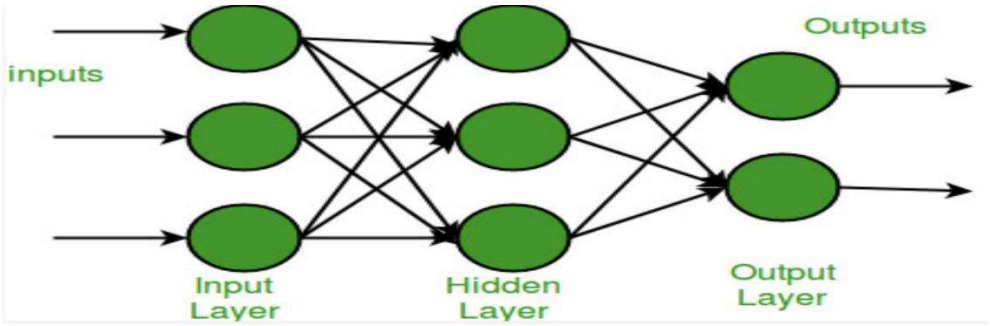
$$\delta_k(x) = \log \pi_k - \frac{1}{2}\mu_k^T \mathbf{\Sigma}_k^{-1} \mu_k + x^T \mathbf{\Sigma}_k^{-1} \mu_k - \frac{1}{2} x^T \Sigma_k^{-1} x - \frac{1}{2} \log |\Sigma_k|$$

→ Now, let us visualize the decision boundaries of both LDA and QDA on the iris dataset. This would give us a clear picture of the difference between the two.



LDA and QDA visualization

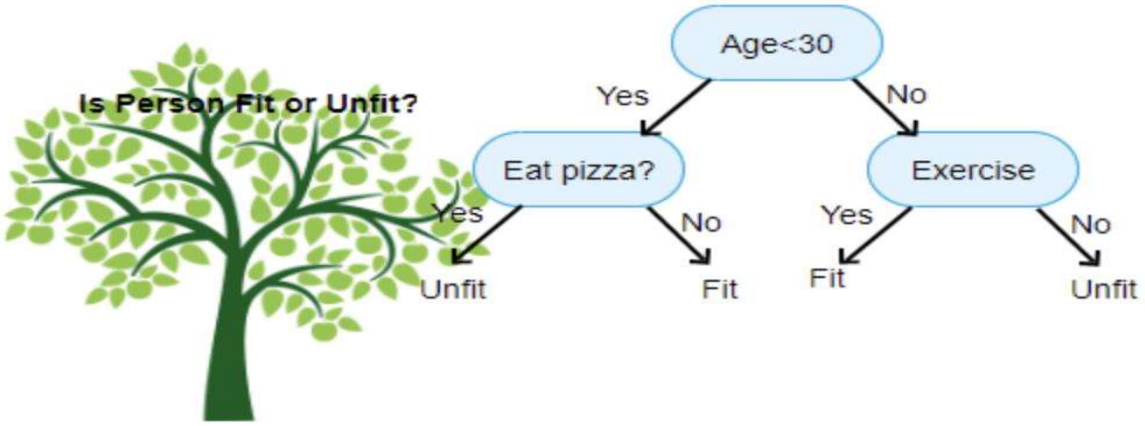
Multi-Layer Perceptron (MLP):
→ This is nothing but a collection of fully connected dense layers. These help transform any given input dimension into the desired dimension.
→ It is nothing but simply a neural network.
→ MLP consists of one input layer(one node belonging to each input), one output layer (one node belonging to each output), and a few hidden layers (>= one node belonging to each hidden layer).



→ In the above diagram, we notice three inputs, resulting in 3 nodes belonging to each input.
→ There is one hidden layer consisting of 3 nodes.
→ There is an output layer consisting of 2 nodes, indicating two outputs.
→ Overall, the nodes belonging to the input layer forward their outputs to the nodes present in the hidden layer. Once this is done, the hidden layer processes the information passed on to it and then further passes it on to the output layer.

Decision Tree:

→ It is considered to be one of the most valuable and robust models.
→ Instances are classified by sorting them down from the root to some leaf node.
→ An instance is classified by starting at the tree's root node, testing the attribute specified by this node, then moving down the tree branch corresponding to the attribute's value, as shown in the above figure.
→ The process is repeated based on each derived subset in a recursive *partitioning* manner.
→ For a better understanding, see the diagram below.



→ The above decision tree helps determine whether the person is fit or not.
→ Similarly, Random Forests, a collection of Decision Trees, is a linear classifier too.

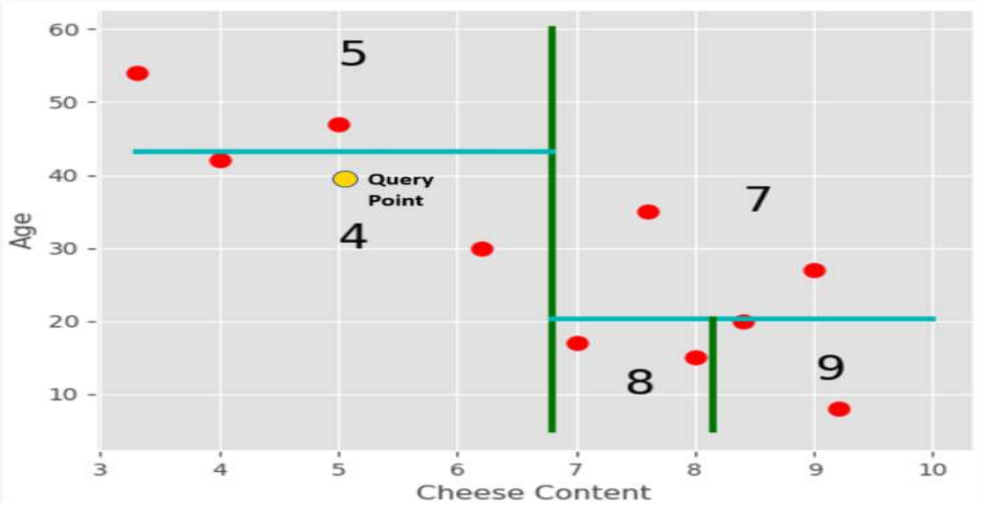
K-Nearest Neighbours:

→ KNN is a supervised machine learning algorithm . It is used for classification problems. Since it is a supervised machine learning algorithm, it uses labeled data to make predictions.
→ KNN analyzes the 'k' nearest data points and then classifies the new data based on the same.

→ In detail, to label a new point, the KNN algorithm analyzes the 'k' nearest neighbors or 'k' nearest data points to the new point. It chooses the label of the new point as the one to which the majority of the 'k' nearest neighbors belong to.

→It is essential to choose an appropriate value of 'K' to avoid the overfitting of our model.

→ For better understanding, have a look at the diagram below.



Comparison

Now, we will briefly sum up all that we've learned and try to compare and contrast Linear Classification and Non-Linear Classification.

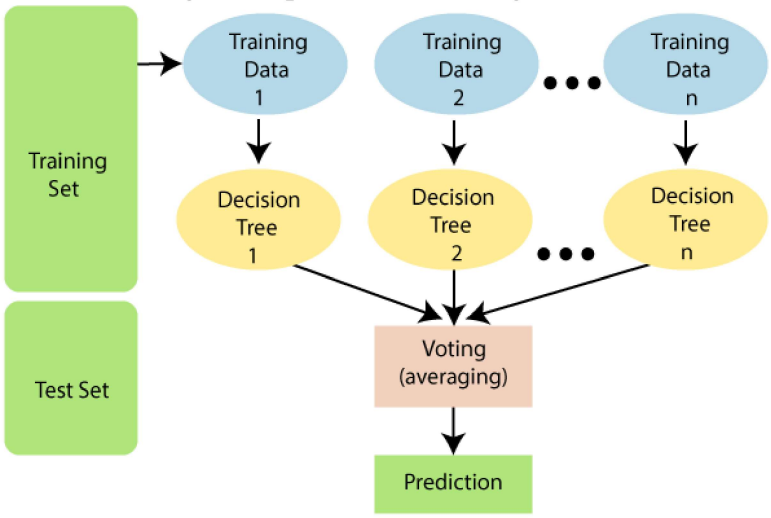
Linear Classification	Non-Linear Classification
Linear Classification refers to categorizing a set of data points into a discrete class based on a linear combination of its explanatory variables.	Non-Linear Classification refers to categorizing those instances that are not linearly separable.
It is possible to classify data with a straight line.	It is not easy to classify data with a straight line.
Data is classified with the help of a hyperplane.	The utilization of kernels is made to transform non-separable data into separable data.

Random Forest Algorithm:

- Random Forest is a popular machine learning algorithm that belongs to the supervised learning technique. It can be used for both Classification and Regression problems in ML. It is based on the concept of **ensemble learning,** which is a process of *combining multiple classifiers to solve a complex problem and to improve the performance of the model.*
- As the name suggests, ***"Random Forest is a classifier that contains a number of decision trees on various subsets of the given dataset and takes the average to improve the predictive accuracy of that dataset."*** Instead of relying on one decision tree, the random forest takes the prediction from each tree and based on the majority votes of predictions, and it predicts the final output.

- **The greater number of trees in the forest leads to higher accuracy and prevents the problem of overfitting.**

The below diagram explains the working of the Random Forest algorithm:



Ensemble methods:

The goal of **ensemble methods** is to combine the predictions of several base estimators built with a given learning algorithm in order to improve generalizability / robustness over a single estimator.

Two families of ensemble methods are usually distinguished:

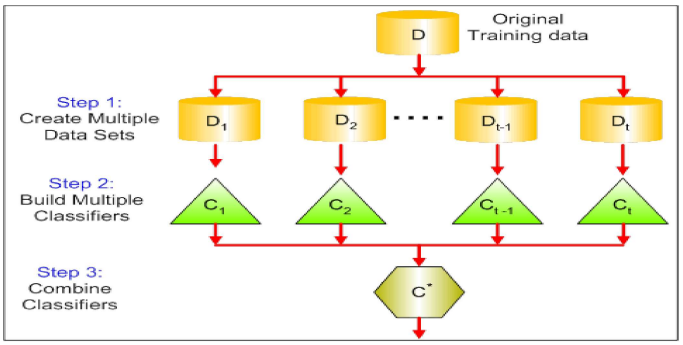
- In **averaging methods**, the driving principle is to build several estimators independently and then to average their predictions. On average, the combined estimator is usually better than any of the single base estimator because its variance is reduced.
  Examples: Bagging methods, Forests of randomized trees, …
- By contrast, in **boosting methods**, base estimators are built sequentially and one tries to reduce the bias of the combined estimator. The motivation is to combine several weak models to produce a powerful ensemble.
  Examples: AdaBoost, Gradient Tree Boosting, …
- Types of ensemble methods in machine learning are:

1. Bagging
2. Boosting
3. Stacking

Bagging based Ensemble learning:

Bagging is one of the Ensemble construction techniques which is also known as ***Bootstrap Aggregation***. Bootstrap establishes the foundation of Bagging technique. Bootstrap is a sampling technique in which we select "n" observations out of a population of "n" observations. But the selection is entirely random, i.e., each observation can be chosen from the original population so that each observation is equally likely to be selected in each iteration of the bootstrapping process. After the bootstrapped samples are formed, separate models are trained with the bootstrapped samples. In real experiments, the bootstrapped samples are drawn from the training set, and the sub-models are tested using the testing set. The final output prediction is combined across the projections of all the sub-models.

The following infographic gives a brief idea of Bagging:



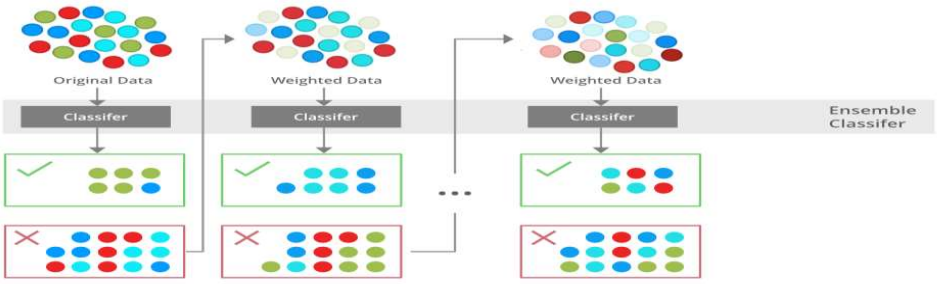
Source

Implementation Steps of Bagging

- **Step 1:** Multiple subsets are created from the original data set with equal tuples, selecting observations with replacement.
- **Step 2:** A base model is created on each of these subsets.
- **Step 3:** Each model is learned in parallel from each training set and independent of each other.
- **Step 4:** The final predictions are determined by combining the predictions from all the models.

Algorithm:

1. *Initialise the dataset and assign equal weight to each of the data point.*
2. *Provide this as input to the model and identify the wrongly classified data points.*
3. *Increase the weight of the wrongly classified data points.*
4. *if (got required results)*
   Goto step 5
   else
   Goto step 2
5. *End*



An illustration presenting the intuition behind the boosting algorithm, consisting of the parallel learners and weighted dataset.

Similarities Between Bagging and Boosting

Bagging and Boosting, both being the commonly used methods, have a universal similarity of being classified as ensemble methods. Here we will explain the similarities between them.

1. Both are ensemble methods to get N learners from 1 learner.
2. Both generate several training data sets by random sampling.
3. Both make the final decision by averaging the N learners (or taking the majority of them i.e Majority Voting).
4. Both are good at reducing variance and provide higher stability.

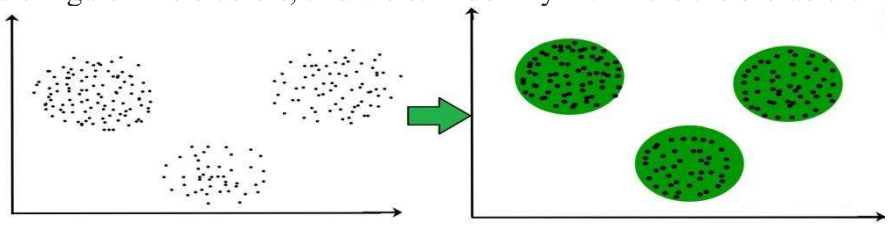
Differences Between Bagging and Boosting

S.NO	Bagging	Boosting
1.	The simplest way of combining predictions that belong to the same type.	A way of combining predictions that belong to the different types.
2.	Aim to decrease variance, not bias.	Aim to decrease bias, not variance.
3.	Each model receives equal weight.	Models are weighted according to their performance.
4.	Each model is built independently.	New models are influenced by the performance of previously built models.
5.	Different training data subsets are randomly drawn with replacement from the entire training dataset.	Every new subset contains the elements that were misclassified by previous models.
6.	Bagging tries to solve the over-fitting problem.	Boosting tries to reduce bias.
7.	If the classifier is unstable (high variance), then apply bagging.	If the classifier is stable and simple (high bias) the apply boosting.
8.	Example: The Random forest model uses Bagging.	Example: The AdaBoost uses Boosting techniques

Clustering :

Clustering : It is basically a type of *unsupervised learning method* . It is the task of dividing the population or data points into a number of groups such that data points in the same groups are more similar to other data points in the same group and dissimilar to the data points in other groups. It is basically a collection of objects on the basis of similarity and dissimilarity between them.

For example– The data points in the graph below clustered together can be classified into one single group. We can distinguish the clusters, and we can identify that there are 3 clusters in the below picture.



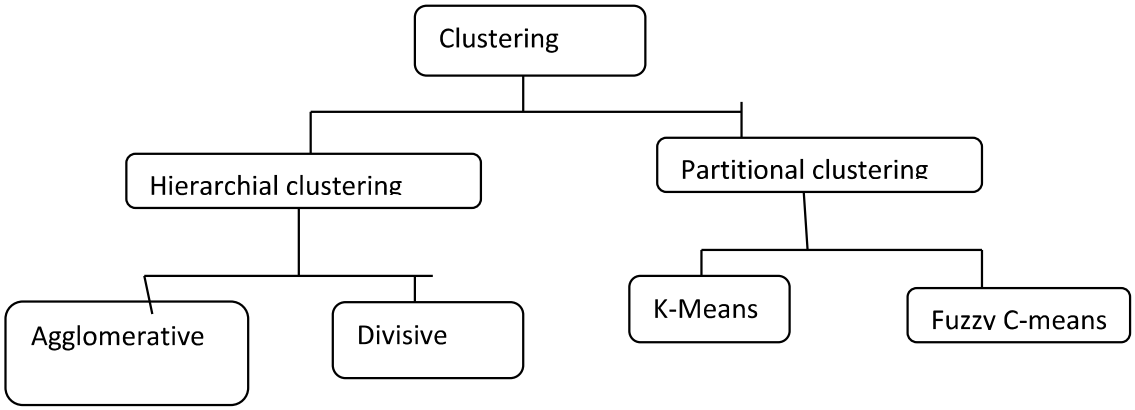
Need for Clustering

Grouping similar entities together helpd to merge the attributes of different clusters. Example 1.Clustering customers to maximize the revenue 2.Grouping books together that belong to similar topics.

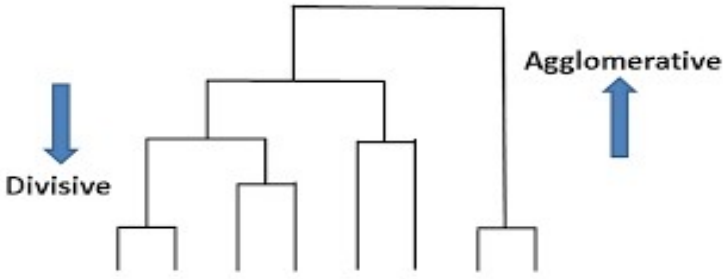
Need:

1. To determine the intrinsic grouping in a set of unlabeled data.
2. To organize data into clusters showing the internal structure of the data.
3. To partition the data points.
4. To understand and extract value from large sets of structured and unstructured data.

Types of Clustering:



- **Hierarchical Based Methods :** The clusters formed in this method forms a tree-type structure based on the hierarchy. New clusters are formed using the previously formed one. It is divided into two category
- **Agglomerative** (*bottom up approach*)
- **Divisive** (*top down approach*)
  examples *CURE (Clustering Using Representatives), BIRCH (Balanced Iterative Reducing Clustering and using Hierarchies)* etc.



- **Partitioning Methods :** These methods partition the objects into k clusters and each partition forms one cluster. This method is used to optimize an objective criterion similarity function such as when the distance is a major parameter example *K-means,Fuzzy C-means*

K-means clustering algorithm – It is the simplest unsupervised learning algorithm that solves clustering problem. K-means algorithm partition n observations into k clusters where each observation belongs to the cluster with the nearest mean serving as a prototype of the cluster .

- ❖ K-means clustering algorithm computes the centroids and iterates until we it finds optimal centroid. It assumes that the number of clusters are already known. It is also called **flat clustering** algorithm. The number of clusters identified from data by algorithm is represented by 'K' in K-means.
- ❖ In this algorithm, the data points are assigned to a cluster in such a manner that the sum of the squared distance between the data points and centroid would be minimum. It is to be understood that less variation within the clusters will lead to more similar data points within same cluster.

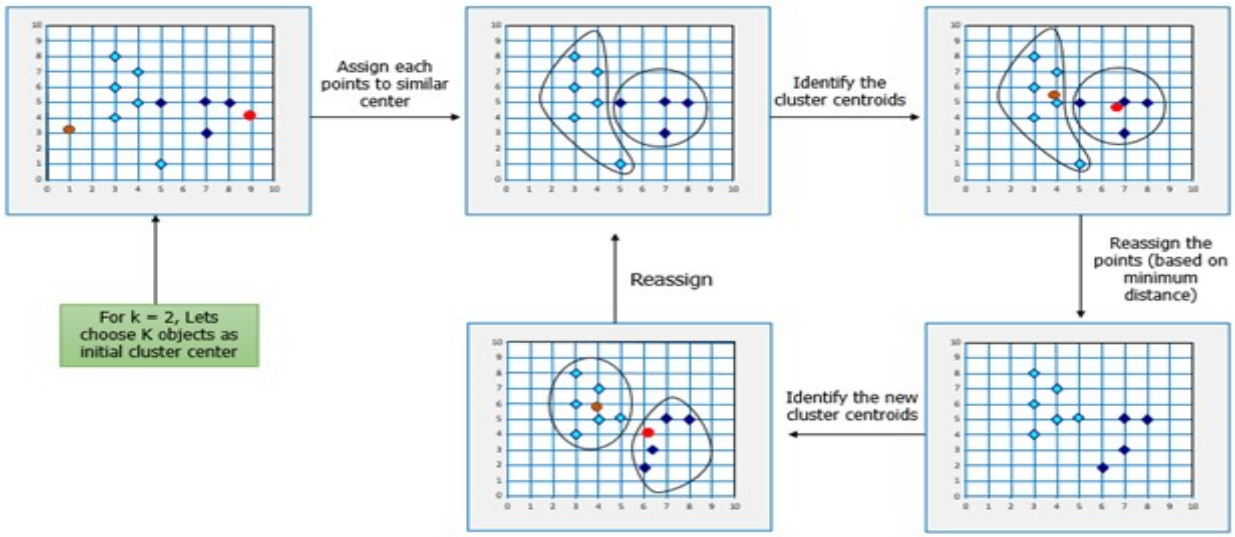
K-means Clustering Method:

If k is given, the K-means algorithm can be executed in the following steps:

- Partition of objects into k non-empty subsets
- Identifying the cluster centroids (mean point) of the current partition.

- Assigning each point to a specific cluster
- Compute the distances from each point and allot points to the cluster where the distance from the centroid is minimum.
- After re-allotting the points, find the centroid of the new cluster formed.

The step by step process:



Advantages

The following are some advantages of K-Means clustering algorithms −

- It is very easy to understand and implement.
- If we have large number of variables then, K-means would be faster than Hierarchical clustering.
- On re-computation of centroids, an instance can change the cluster.
- Tighter clusters are formed with K-means as compared to Hierarchical clustering.

Disadvantages

The following are some disadvantages of K-Means clustering algorithms −

- It is a bit difficult to predict the number of clusters i.e. the value of k.
- Output is strongly impacted by initial inputs like number of clusters (value of k)
- Order of data will have strong impact on the final output.
- It is very sensitive to rescaling. If we will rescale our data by means of normalization or standardization, then the output will completely change.
- It is not good in doing clustering job if the clusters have a complicated geometric shape.

Applications of K-Means Clustering Algorithm

The main goals of cluster analysis are −

- To get a meaningful intuition from the data we are working with.
- Cluster-then-predict where different models will be built for different subgroups.

To fulfill the above-mentioned goals, K-means clustering is performing well enough. It can be used in following applications −

- Market segmentation
- Document Clustering
- Image segmentation
- Image compression
- Customer segmentation
- Analyzing the trend on dynamic data

Applications of Clustering in different fields

- **Marketing :** It can be used to characterize & discover customer segments for marketing purposes.
- **Biology :** It can be used for classification among different species of plants and animals.
- **Libraries :** It is used in clustering different books on the basis of topics and information.
- **Insurance :** It is used to acknowledge the customers, their policies and identifying the frauds.
- **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.

What is Regression?

Regression is a method to determine the statistical relationship between a dependent variable and one or more independent variables. The change independent variable is associated with the change in the independent variables.

Types of Regression

Regression has seven types but, the mainly used are Linear and Logistic Regression. These are the basic and simplest modeling algorithms. We will discuss both of these in detail here.

1. Linear Regression

- The simplest case of linear regression is to find a relationship using a linear model (i.e line) between an input independent variable (input single feature) and an output dependent variable. This is called Bivariate Linear Regression.
- On the other hand, when there is a linear model representing the relationship between a dependent output and multiple independent input variables is called Multivariate Linear Regression.
- The dependent variable is continuous and independent variables may or may not be continuous. We find the relationship between them with the help of the best fit line which is also known as the Regression line. The equation of a line is,

  `y = m * x + b`

  Where,
- **x:** Independent Variable
- **y:** Dependent Variable
- **m:** Slope of Line
- **b:** y Intercept

To evaluate the best fit line, the most common method is the Least Square Method. In this method, the regression line is calculated by minimizing the least squared error between the regression line and the data points. Another method to find this line is also called the R Squared analysis.

It is particularly useful when the relationship between the input variables and the output is not very complex. Also, note that it is very sensitive to outliers.

Syntax in Python:

The Python library named sklearn contains an inbuilt function, we will use LinerRegression from sklearn.

Let us first install the sklearn package.

```
pip install scikit-learn
```

```
from sklearn.linear_model import LinearRegression
linearReg = LinearRegression()
```

To train the model, we will use the fit() function.

```
linearReg.fit(x_train, y_train)
```

2. Logistic Regression:

- It is used when the output is categorical. It is more like a classification problem. The output can be Success / Failure, Yes / No, True/ False or 0/1. There is no need for a linear relationship between the dependent output variable and independent input variables.
- If the output has only two possibilities, then it is called Binary Logistic Regression. If the dependent output has more than two output possibilities and there is no ordering in them, then it is called Multinomial Logistic Regression. If there is order associated with the output and there are more than two output possibilities then it is called Ordinal Logistic Regression.
- Let us take an example, you want to create a model that identifies if the breast cancer is malignant(1) or benign(0). Taking another example if you want to classify if the input email is spam(1) or not spam (0). It can be better explained by Sigmoid function.

  $h\Theta(x) = \text{sigmoid}(Z)$

  Sigmoid Function:

  $\text{sig}(t) = 1 / 1+e^{-t}$

The sigmoid function is the S-shaped curve. If the value goes near positive infinity then the predicted value will be 1. Similarly, if it goes negative infinity then the predicted value will be 0.

Syntax in Python:

For the implementation of logistic regression in Python, there is an inbuilt function available in scikit- learn library of Python. For that first install scikit-learn using pip install.

```
from sklearn.linear_model import LogisticRegression logisticRegr = LogisticRegression()
```

To train the model, we will use the fit() function.

```
logisticRegr.fit(x_train, y_train)
```

Cost Function

A Cost Function is used to measure just how wrong the model is in finding a relation between the input and output. It tells you how badly your model is behaving/predicting

Consider a robot trained to stack boxes in a factory. The robot might have to consider certain changeable parameters, called Variables, which influence how it performs. Let's say the robot comes across an obstacle, like a rock. The robot might bump into the rock and realize that it is not the correct action.

Training and testing a classifier:

This is divided into three parts; they are:

1. Train to the Test Set
2. Train to Test Set for Classification
3. Train to Test Set for Regression

Train to the Test Set

- In applied machine learning, we seek a model that learns the relationship between the input and output variables using the training dataset.
- The hope and goal is that we learn a relationship that generalizes to new examples beyond the training dataset. This goal motivates why we use resampling techniques like k-fold cross-validation to estimate the performance of the model when making predictions on data not used during training.
- In the case of machine learning competitions, like those on Kaggle, we are given access to the complete training dataset and the inputs of the test dataset and are required to make predictions for the test dataset.

- This leads to a possible situation where we may accidentally or choose to train a model to the test set. That is, tune the model behavior to achieve the best performance on the test dataset rather than develop a model that performs well in general, using a technique like k-fold cross-validation.

Train to Test Set for Classification

We will use the diabetes dataset as the basis for exploring training for the test set for classification problems.

Each record describes the medical details of a female and the prediction is the onset of diabetes within the next five years.

The dataset has eight input variables and 768 rows of data; the input variables are all numeric and the target has two class labels, e.g. it is a binary classification task.Below provides a sample of the first five rows of the dataset.

```
1  6,148,72,35,0,33.6,0.627,50,1
2  1,85,66,29,0,26.6,0.351,31,0
3  8,183,64,0,0,23.3,0.672,32,1
4  1,89,66,23,94,28.1,0.167,21,0
5  0,137,40,35,168,43.1,2.288,33,1
6  ...
```

First, we can load the dataset directly from the URL, split it into input and output elements, then split the dataset into train and test sets, holding thirty percent back for the test set. We can then evaluate a KNN model with default model hyperparameters by training it on the training set and making predictions on the test set.

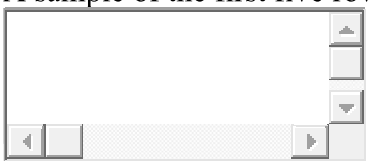
Train to Test Set for Regression

We will use the housing dataset as the basis for exploring training for the test set for regression problems.

The housing dataset involves the prediction of a house price in thousands of dollars given details of the house and its neighborhood.

It is a regression problem, meaning we are predicting a numerical value. There are 506 observations with 13 input variables and one output variable.

A sample of the first five rows is listed below.



```
1  0.00632,18.00,2.310,0,0.5380,6.5750,65.20,4.0900,1,296.0,15.30,396.90,4.98,24.00
2  0.02731,0.00,7.070,0,0.4690,6.4210,78.90,4.9671,2,242.0,17.80,396.90,9.14,21.60
3  0.02729,0.00,7.070,0,0.4690,7.1850,61.10,4.9671,2,242.0,17.80,392.83,4.03,34.70
4  0.03237,0.00,2.180,0,0.4580,6.9980,45.80,6.0622,3,222.0,18.70,394.63,2.94,33.40
5  0.06905,0.00,2.180,0,0.4580,7.1470,54.20,6.0622,3,222.0,18.70,396.90,5.33,36.20
6  ...
```

First, we can load the dataset, split it, and evaluate a KNN model on it directly using the entire training dataset. We will report performance on this regression class using mean absolute error (MAE).

Cross Validation

In machine learning, we couldn't fit the model on the training data and can't say that the model will work accurately for the real data. For this, we must assure that our model got the correct patterns from the data, and it is not getting up too much noise. For this purpose, we use the cross-validation technique.

Cross-validation is a technique in which we train our model using the subset of the data-set and then evaluate using the complementary subset of the data-set.

Methods of Cross Validation(cv)
1.simple cv
2.k-fold cv
3.Time based/series splittting

1.simple validation

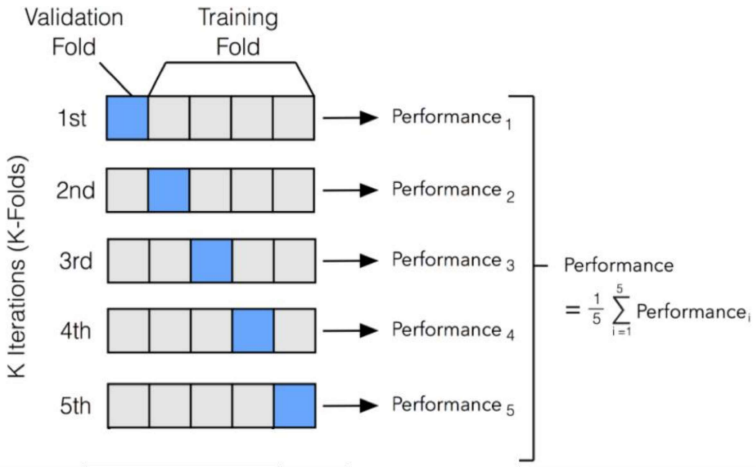
In this method, we perform training on the 50% of the given data-set and rest 50% is used for the testing purpose. The major drawback of this method is that we perform training on the 50% of the dataset, it may possible that the remaining 50% of the data contains some important information which we are leaving while training our model i.e higher bias.

2.K-Fold Cross Validation

In this method, we split the data-set into k number of subsets(known as folds) then we perform training on the all the subsets but leave one(k-1) subset for the evaluation of the trained model. In this method, we iterate k times with a different subset reserved for testing purpose each time.

Example

The diagram below shows an example of the training subsets and evaluation subsets generated in k-fold cross-validation. Here, we have total 25 instances. In first iteration we use the first 20 percent of data for evaluation, and the remaining 80 percent for training([1-5] testing and [5-25] training) while in the second iteration we use the second subset of 20 percent for evaluation, and the remaining three subsets of the data for training([5-10] testing and [1-5 and 10-25] training), and so on.

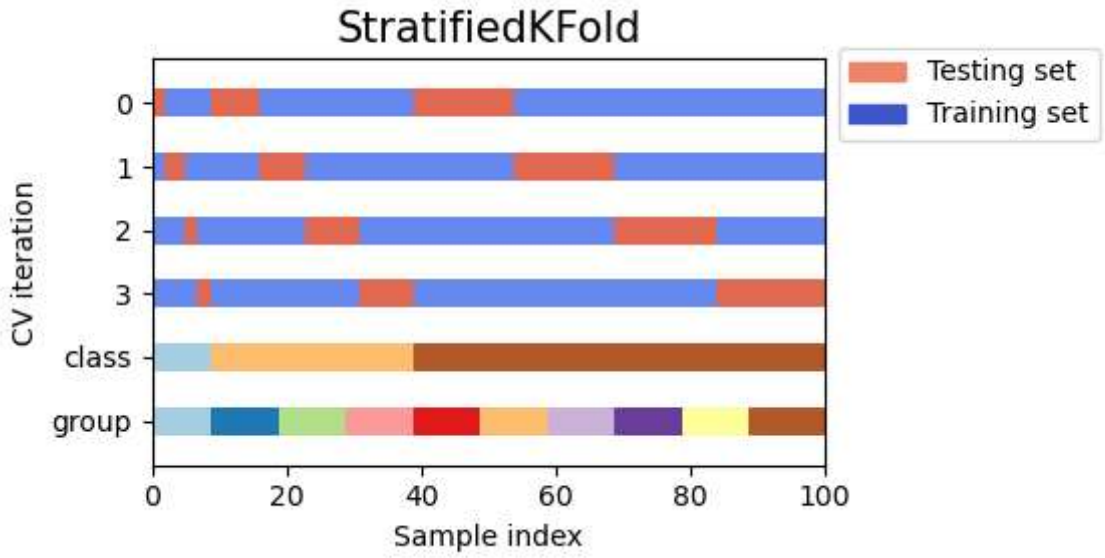


Total instances: 25
Value of k : 5

No. Iteration	Training set observations	Testing set observations
1	[5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24]	[0 1 2 3 4]
2	[0 1 2 3 4 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24]	[5 6 7 8 9]
3	[0 1 2 3 4 5 6 7 8 9 15 16 17 18 19 20 21 22 23 24]	[10 11 12 13 14]
4	[0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 20 21 22 23 24]	[15 16 17 18 19]
5	[0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19]	[20 21 22 23 24]

3.Time series/based splitting:

- ❖ It is a special variation of k-fold cross validation to validate time series ,data samples observed at fixed time intervals.
- ❖ It returns first k-fold as train data set and the (k+1)th set as the test set.
- ❖ The successive training sets are super sets of those that come before subsets.
- ❖ It is also adds the surplus data to the first training set that is always used to train the model.



Advantages of Cross Validation

1. Reduces Overfitting: In Cross Validation, we split the dataset into multiple folds and train the algorithm on different folds. This prevents our model from overfitting the training dataset. So, in this way, the model attains the generalization capabilities which is a good sign of a robust algorithm.

Note: Chances of overfitting are less if the dataset is large. So, Cross Validation may not be required at all in the situation where we have sufficient data available.

2. Hyperparameter Tuning: Cross Validation helps in finding the optimal value of hyperparameters to increase the efficiency of the algorithm.

Disadvantages of Cross Validation

1. Increases Training Time: Cross Validation drastically increases the training time. Earlier you had to train your model only on one training set, but with Cross Validation you have to train your model on multiple training sets.

For example, if you go with 5 Fold Cross Validation, you need to do 5 rounds of training each on different 4/5 of available data. And this is for only one choice of hyperparameters. If you have multiple choice of parameters, then the training period will shoot too high.

2. Needs Expensive Computation: Cross Validation is computationally very expensive in terms of processing power required.

What is class Imbalance in machine learning?

In machine learning class imbalance is the issue of **target class** distribution. Will explain why we are saying it is an issue. If the target classes are not equally distributed or not in an equal ratio, we call the data having an imbalance data issue.

Examples of balanced and imbalanced datasets

Let me give an example of a target class balanced and imbalanced datasets, which helps in understanding about class imbalance datasets.

Balanced datasets:-

- A random sampling of a coin trail
- Classifying images to cat or dog
- Sentiment analysis of movie reviews

Suppose you see in the above examples. For the balanced datasets, the target class distribution is nearly equal. For example, In the random coin trail, even the researchers say the probability of getting head is higher than the tail Still, the **distribution** of head and tail is nearly equal. It is the same with the movie review case too.

Class Imbalance dataset:-

- Email spam
- Credit card fraud detection
- Network failure detections
- Anomaly detections
- Medical diagnosis
- Facial recognition

But when it comes to the imbalanced dataset, the target distribution is not equal. For email spam or ham, distribution is not equal.

Just imagine how many emails we receive every day and how many were classified as spam. Google uses its email classifier to do that.

In general, out of 10 emails, we receive one will go to the spam folder, and the other emails will go to the inbox. Here the ham and spam ration is **9:1** In credit card fraud detection the ration will much lesser like **9.5: 5** By now, we are clear about imbalanced data. Now, let's learn why we need to balance data. In other words, why we need to handle the imbalanced data.

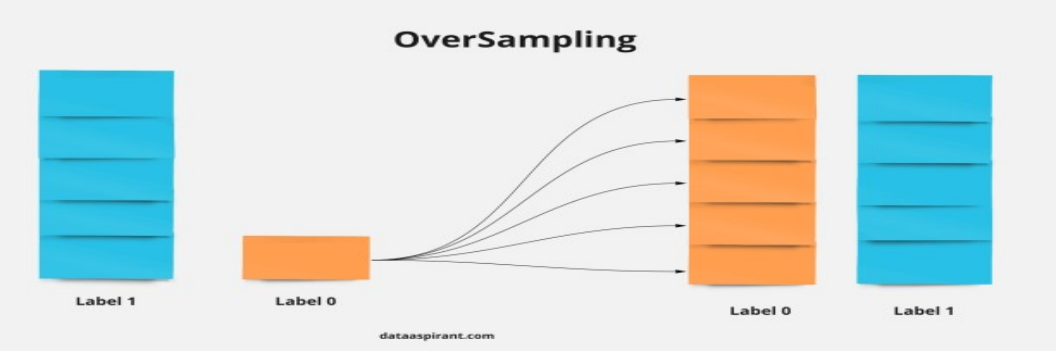
Techniques for handling imbalanced data

For handling imbalance data we are having many other ways, In this article, we will learn about the below techniques along with the code implementation.

1. Oversampling
2. Undersampling
3. Ensemble Techniques

In this article we will be focusing only on the **first 2 methods** for handling imbalance data.

OverSampling



In oversampling, we increase the number of samples in minority class to match up to the number of samples of the majority class.

In simple terms, you take the **minority class** and try to create new samples that could match up to the length of the **majority samples**.

Let me explain in a much better way.

E.g., Suppose we have a data with 100 labels with 0's and 900 labels with 1's, here the minority class 0's, what we do is we increase the data 9:1 ratio, i.e., for everyone data point it will increase 9 times results in creating new 9 data points on that top of one point.

Mathematically:

1 label ----------> 900 data points
0 label ----------> 100 data points
+ 800 points
---
900 data points

Now the data ratio is 1:1 ,
1 label ------>900 data points
0 label ------> 900 data points

Oversampling Implementation

We can implement in two ways,

1. RandomOverSampler method
2. SMOTETomek method

First, we have to install imblearn library, to install enter below command in cmd
Command: **pip install imbalanced-learn**

Oversampling pros and cons

Below are the listed pros and cons of using the oversampling technique.

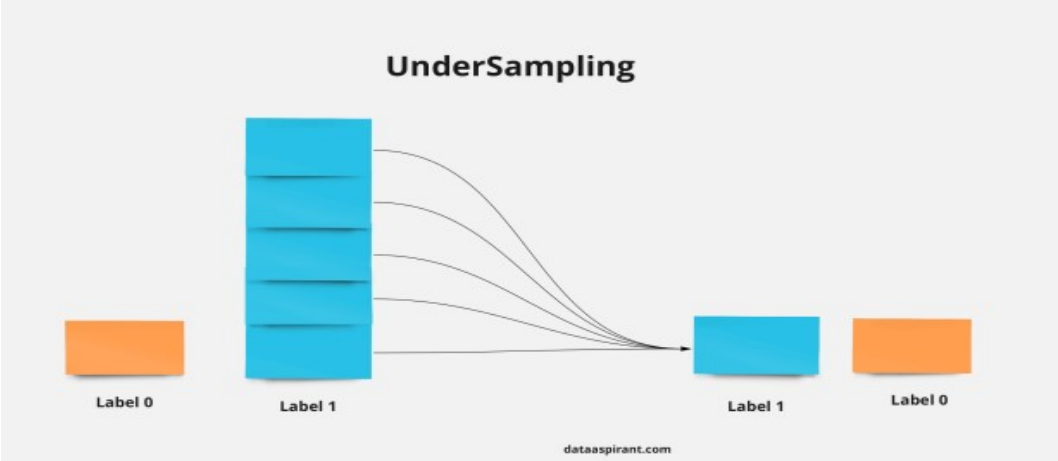
Pros:

- This method doesn't lead to information loss.
- Performs well and gives good accuracy.
- It creates new synthetic data points with the nearest neighbours from existing data.

Cons:

- Increase the size of data takes high time for training.
- It may also lead to overfitting since it is replicating the minor classes.

UnderSampling:



In undersampling, we **decrease** the number of samples in the majority class to match the number of samples of the minority class.

In brief, you take the majority class and try to create new samples that match the length of the minority samples.

Let me explain in a much better way

E.g., Suppose we have a data with 100 labels with 0's and 900 labels with 1's, here the minority class 0's, what we do is we balance the data from 9:1 ratio to 1:1 ratio i.e., We randomly select 100 data points out of 900 data points in majority class. Results in 1: 1 ratio, i.e.,

1 label ----------> 100 data points

0 label ----------> 100 data points

Undersampling Implementation

We can implement in two **different** ways,

1. RandomunderSampler method
2. NearMiss method

Random undersampling Implementation

It simply samples the majority class at random until it reaches a similar number of observations as the minority classes.

NearMiss Implementation

It selects samples from the majority class for which the average distance of the N closet samples of a majority class is smallest.

Undersampling pros and cons

Below are the listed pros and corns of using the undersampling techniques

Pros:

- Reduces storage problems, easy to train
- In most cases it creates a balanced subset that carries the greatest potential for representing the larger group as a whole.
- It produces a simple random sample which is much less complicated than other techniques.

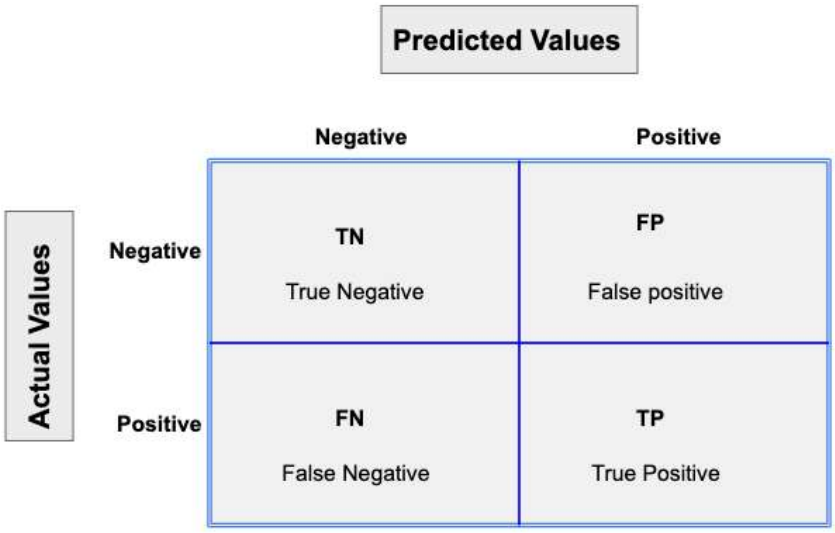
Cons:

- It can ignore potentially useful information which could be important for building classifiers.
- The sample chosen by random under-sampling may be a biased sample, resulting in inaccurate results with the actual test data.
- Loss of useful information of the majority class.

Confusion Matrix

A confusion matrix is a matrix representation of the prediction results of any binary testing that is often used to **describe the performance of the classification model** (or "classifier") on a set of test data for which the true values are known.

The confusion matrix itself is relatively simple to understand, but the related terminology can be confusing.



Confusion matrix with 2 class labels.

Each prediction can be one of the four outcomes, based on how it matches up to the actual value:

- **True Positive (TP):** Predicted True and True in reality.
- **True Negative (TN):** Predicted False and False in reality.
- **False Positive (FP):** Predicted True and False in reality.
- **False Negative (FN):** Predicted False and True in reality.

There are two types of errors

These errors are known as **Type 1 and Type II errors.**

Type I Error:- equivalent to False Positives(FP).
Type II Error:- equivalent to False Negatives(FN).

Accuracy

Overall, how often is the classifier correct?
Accuracy = (TP+TN)/total
When our classes are roughly equal in size, we can use accuracy**,** which will give us correctly classified values. Accuracy is a common evaluation metric for classification problems. It's the number of correct predictions made as a ratio of all predictions made.

Misclassification Rate(Error Rate): Overall, how often is it wrong. Since accuracy is the percent we correctly classified (success rate), it follows that our error rate (the percentage we got wrong) can be calculated as follows:
Misclassification Rate = (FP+FN)/total

Precision

When it predicts yes, how often is it correct?

Precision=TP/predicted yes

When we have a class imbalance, accuracy can become an unreliable metric for measuring our performance. For instance, if we had a 99/1 split between two classes, A and B, where the rare event, B, is our positive class, we could build a model that was 99% accurate by just saying everything belonged to class A. Clearly, we shouldn't bother building a model if it doesn't do anything to identify class B; thus, we need different metrics that will discourage this behavior. For this, we use precision and recall instead of accuracy.

Recall or Sensitivity

When it's actually yes, how often does it predict yes?

True Positive Rate = TP/actual yes

Recall gives us the **true positive rate** (**TPR**), which is the ratio of true positives to everything positive.

Evaluation Metrics for Machine Learning - Accuracy, Precision, Recall, and F1 Defined

After a data scientist has chosen a target variable - e.g. the "column" in a spreadsheet they wish to predict - and completed the prerequisites of transforming data and building a model, one of the final steps is evaluating the model's performance.

It is divided in to two categories.

1. Classification Metrics
2. Regression Metrics

Classofication metrics:

These are classified into different types

a) Confusion matrix
b) Accuracy
c) Precesion
d) Recall
e) F1 score
f) ROC curve

Confusion Matrix

Choosing a performance metric often depends on the business problem being solved. Let's say you have 100 examples in your dataset, and you've fed each one to your model and received a classification. The predicted vs. actual classification can be charted in a table called a confusion matrix

	Negative (predicted)	**Positive (predicted)**
Negative (actual)	98	0
Positive (actual)	1	1

The table above describes an output of negative vs. positive. These two outcomes are the "classes" of each examples. Because there are only two classes, the model used to generate the confusion matrix can be described as a *binary classifier*. (Example of a binary classifier: spam detection. All emails are spam or not spam, just as all food is a hot dog or not a hot dog.)

To better interpret the table, you can also see it in terms of true positives, true negatives, false positives and false negatives.

	Negative (predicted)	Positive (predicted)
Negative (actual)	true negative	false positive
Positive (actual)	false negative	true positive

Accuracy

Overall, how often is our model correct?

$$Accuracy = \frac{truepositives + truenegatives}{totalexamples}$$ [1]

The problem with using accuracy as your main performance metric is that it does not do well when you have a severe class imbalance. Let's use the dataset in the confusion matrix above. Let's say the negatives are normal transactions and the positives are fraudulent transactions. Accuracy will tell you that you're right 99% of the time

Precision

When the model predicts positive, how often is it correct?

$$Precision = \frac{truepositives}{truepositives + falsepositives}$$

Precision helps when the costs of false positives are high. So let's assume the problem involves the detection of skin cancer. If we have a model that has very low precision, then many patients will be told that they have melanoma, and that will include some misdiagnoses. Lots of extra tests and stress are at stake. When false positives are too high, those who monitor the results will learn to ignore them after being bombarded with false alarms.

Recall

$$Recall = \frac{truepositives}{truepositives + falsenegatives}$$

Recall helps when the cost of false negatives is high. What if we need to detect incoming nuclear missiles? A false negative has devastating consequences. Get it wrong and we all die. When false negatives are frequent, you get hit by the thing you want to avoid.

F1 Score

$$F1 = 2 \times \frac{precision \times recall}{precision + recall}$$

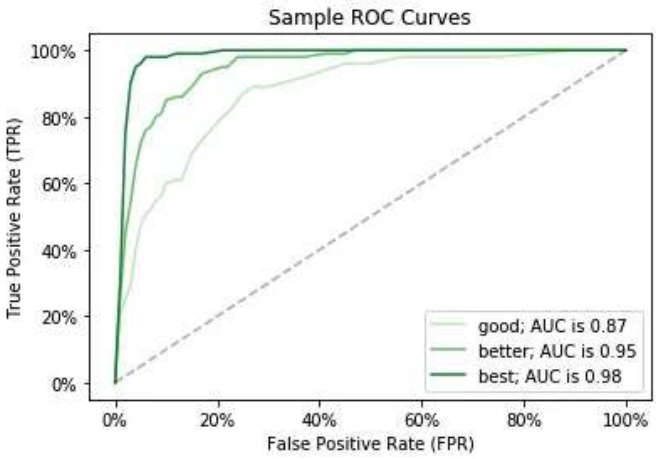
F1 is an overall measure of a model's accuracy that combines precision and recall, in that weird way that addition and multiplication just mix two ingredients to make a separate dish altogether. That is, a good F1 score means that you have low false positives and low false negatives, so you're correctly identifying real threats and you are not disturbed by false alarms. An F1 score is considered perfect when it's 1, while the model is a total failure when it's 0.

Remember: All models are wrong, but some are useful. That is, all models will generate some false negatives, some false positives, and possibly both. While you can tune a model to minimize one or the other, you often face a tradeoff, where a decrease in false negatives leads to an increase in false positives, or vice versa. You'll need to optimize for the performance metrics that are most useful for your specific problem.

Receiver Operating Characteristics (ROC) Curve:

Measuring the area under the ROC curve is also a very useful method for evaluating a model. By plotting the true positive rate (sensitivity) versus the false-positive rate (1 — specificity), we get the **Receiver Operating Characteristic** (**ROC**) **curve**. This curve allows us to visualize the trade-off between the true positive rate and the false positive rate.

The following are examples of good ROC curves. The dashed line would be random guessing (no predictive value) and is used as a baseline; anything below that is considered worse than guessing. We want to be toward the top-left corner:



Introduction to Deep Learning

What is Deep Learning?

Deep learning is a branch of machine learning which is completely based on artificial neural networks, as neural network is going to mimic the human brain so deep learning is also a kind of mimic of human brain. In deep learning, we don't need to explicitly program everything. The concept of deep learning is not new. It has been around for a couple of years now. It's on hype nowadays because earlier we did not have that much processing power and a lot of data. As in the last 20 years, the processing power increases exponentially, deep learning and machine learning came in the picture.

A formal definition of deep learning :

"Deep learning is a particular kind of machine learning that achieves great power and flexibility by learning to represent the world as a nested hierarchy of concepts, with each concept defined in relation to simpler concepts, and more abstract representations computed in terms of less abstract ones".

In human brain approximately 100 billion neurons all together this is a picture of an individual neuron and each neuron is connected through thousand of their neighbors.

The question here is how do we recreate these neurons in a computer. So, we create an artificial structure called an artificial neural net where we have nodes or neurons. We have some neurons for input value and some for output value and in between, there may be lots of neurons interconnected in the hidden layer.

Architectures :

Deep Neural Network :

It is a neural network with a certain level of complexity (having multiple hidden layers in between input and output layers). They are capable of modeling and processing non-linear relationships.

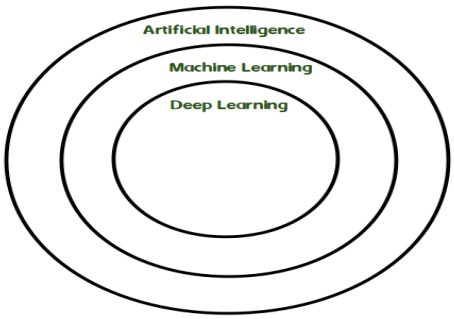
Deep Belief Network(DBN) –

It is a class of Deep Neural Network. It is multi-layer belief networks.

Steps for performing DBN :

a. Learn a layer of features from visible units using Contrastive Divergence algorithm.
b. Treat activations of previously trained features as visible units and then learn features of features.
c. Finally, the whole DBN is trained when the learning for the final hidden layer is achieved.

Recurrent (perform same task for every element of a sequence) Neural Network – Allows for parallel and sequential computation. Similar to the human brain (large feedback network of connected neurons). They are able to remember important things about the input they received and hence enables them to be more precise.

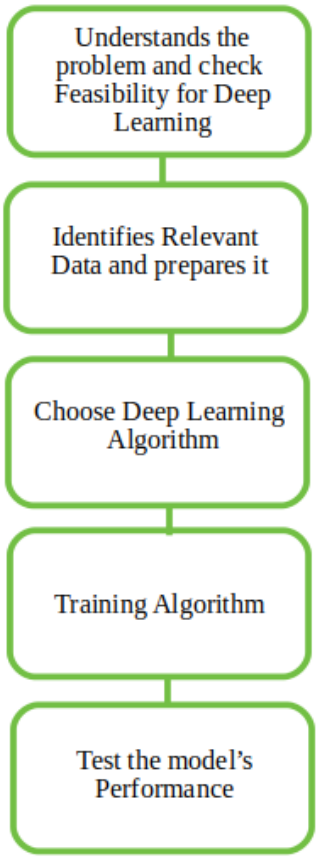


Difference between Machine Learning and Deep Learning :

Machine Learning	Deep Learning
Works on small amount of Dataset for accuracy.	Works on Large amount of Dataset.
Dependent on Low-end Machine.	Heavily dependent on High-end Machine.
Divides the tasks into sub-tasks, solves them individually and finally combine the results.	Solves problem end to end.
Takes less time to train.	Takes longer time to train.
Testing time may increase.	Less time to test the data.

Working :

1. First, we need to identify the actual problem in order to get the right solution and it should be understood, the feasibility of the Deep Learning should also be checked (whether it should fit Deep Learning or not).
2. Second, we need to identify the relevant data which should correspond to the actual problem and should be prepared accordingly.
3. Third, Choose the Deep Learning Algorithm appropriately. Fourth, Algorithm should be used while training the dataset. Fifth, Final testing should be done on the dataset.



Applications :

- Automatic Text Generation – Corpus of text is learned and from this model new text is generated, word-by-word or character-by-character. Then this model is capable of learning how to spell, punctuate, form sentences, or it may even capture the style.
- Healthcare – Helps in diagnosing various diseases and treating it.
- Automatic Machine Translation – Certain words, sentences or phrases in one language is transformed into another language (Deep Learning is achieving top results in the areas of text, images).
- Image Recognition – Recognizes and identifies peoples and objects in images as well as to understand content and context. This area is already being used in Gaming, Retail, Tourism, etc.
- Predicting Earthquakes – Teaches a computer to perform viscoelastic computations which are used in predicting earthquakes.

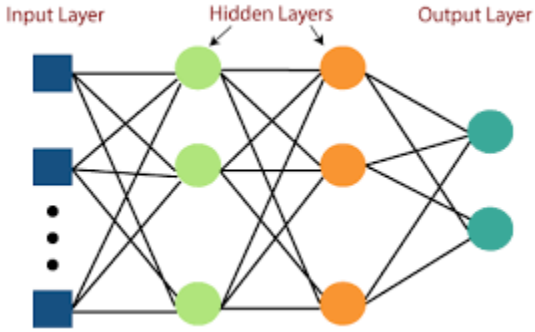
Multi-layered perceptron(MLP)

A **multi-layered perceptron (MLP)** is one of the most common neural network models used in the field of deep learning. Often referred to as a "vanilla" neural network, an MLP is simpler than the complex models .

The multilayer perceptron (MLP) is used for a variety of tasks, such as stock analysis, image identification, spam detection, and election voting predictions.

The Basic Structure:

A multi-layered perceptron consists of interconnected *neurons* transferring information to each other, much like the human brain. Each neuron is assigned a value. The network can be divided into three main layers.



Input Layer

This is the initial layer of the network which takes in an input which will be used to produce an output.

Hidden Layer(s)

The network needs to have at least one hidden layer. The hidden layer(s) perform computations and operations on the input data to produce something meaningful.

Output Layer

The neurons in this layer display a meaningful output.

Backpropagation

- Backpropagation (backward propagation) is an important mathematical tool for improving the accuracy of predictions in data mining and machine learning. Essentially, backpropagation is an algorithm used to calculate derivatives quickly.

- The backpropagation algorithm works by computing the gradient of the loss function with respect to each weight by the chain rule, computing the gradient one layer at a time, iterating backward from the last layer to avoid redundant calculations of intermediate terms in the chain rule; this is an example of dynamic ..

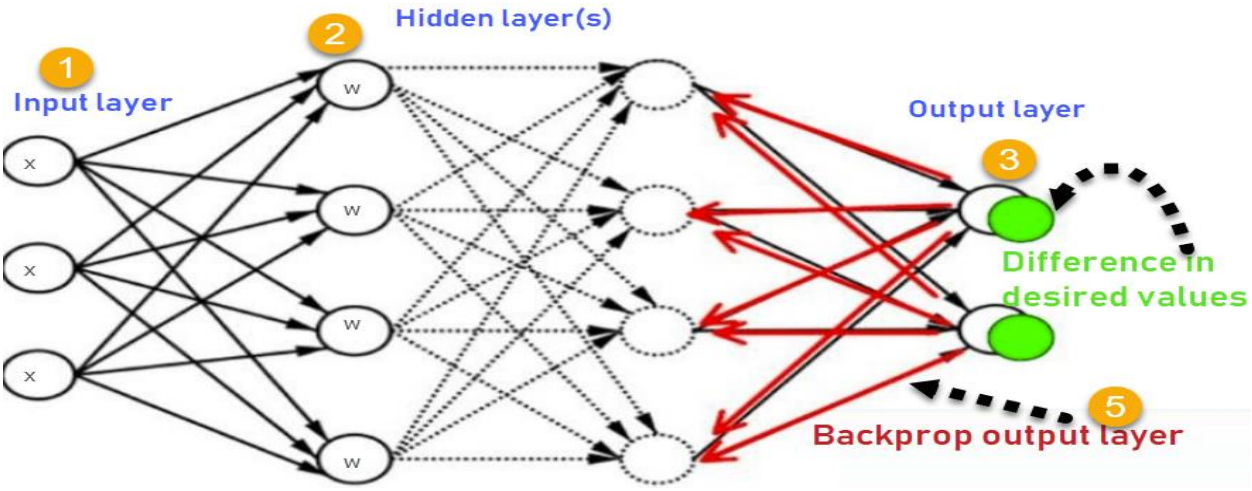
- Backpropagation refers to the process of sending this error back through the network, readjusting the weights automatically so that eventually, the error between the actual and expected output is minimized.

- In this way, the output of the current iteration becomes the input and affects the next output. This is repeated until the correct output is produced. The weights at the end of the process would be the ones on which the neural network works correctly.

How Backpropagation Algorithm Works:

The Backpropagation algorithm in neural network computes the gradient of the loss function for a single weight by the chain rule. It efficiently computes one layer at a time, unlike a native direct computation. It computes the gradient, but it does not define how the gradient is used. It generalizes the computation in the delta rule.

Consider the following Back propagation neural network example diagram to understand:



1. Inputs X, arrive through the preconnected path
2. Input is modeled using real weights W. The weights are usually randomly selected.
3. Calculate the output for every neuron from the input layer, to the hidden layers, to the output layer.
4. Calculate the error in the outputs
   $Error_B$= Actual Output – Desired Output
5. Travel back from the output layer to the hidden layer to adjust the weights such that the error is decreased.

Loss function

What is Loss and Loss function?

- **Loss functions** are used to estimate and understand how well a particular algorithm performs. They are classified into two categories:
- The **loss function** is the function that **computes the distance between the current output of the algorithm and the expected output**.

- ❖ It's a method to evaluate how your algorithm models the data. It can be categorized into two groups. One for **classification** (discrete values, 0,1,2…) and the other for **regression** (continuous values).

The most common losses used in Machine learning and Deep learning is:

1. ***Categorical Crossentropy***
2. ***Binary Crossentropy***
3. ***Mean Absolute Error***
4. ***Mean Squared Error***

1. Categorical Crossentropy:

This loss function work for multiclass, single-label classification. This employ when only one category applies to each data point. It compares the distribution of predictions (the activations in the output layer, one for each class) with the actual distribution, where the probability of the true class is 1and 0 for others.

2. Binary Crossentropy

This loss function work for multiclass, multilabel classification. The Loss tells us how wrong the model predictions are. In multilabel problems, where an example can belong to multiple classes at the same time, the model tries to decide for each category whether the sample belongs to that category or not.

Binary cross-entropy measures how far away from the actual value (either 0 or 1). The prediction is for each of the classes and then averages these class-wise errors to obtain the final Loss.

3. Mean Absolute Error(MAE):

It is the absolute difference between the actual and predicted value.

MAE is not sensitive towards outliers and given several examples with the same input feature values, and the optimal prediction will be their median target value.

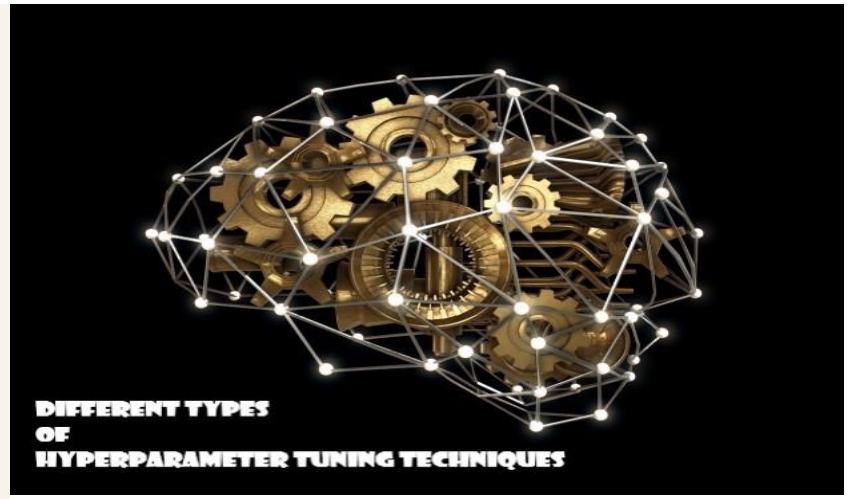
4. Mean Squared Error (MSE):

It is the mean of the squared difference between the actual and predicted value and most commonly used loss function for regression.

MSE is sensitive towards outliers and given several examples with the same input feature values; the optimal prediction will be their mean target value.

Hyperparameter

Hyperparameters are the tunable parameters adjusted before running the training process. These parameters directly affect model performance and help you achieve faster global minima.



List of most used hyperparameters:

- **Learning rate:** step size of each iteration and can be set from 0.1 to 0.0001. In short, it determines the speed at which the model learns.
- **Batch size:** number of samples passed through a neural network at a time.
- **Number of epochs:** an iteration of how many times the model changes weights. Too many epochs can cause models to overfit and too few can cause models to underfit, so we have to choose a medium number.

Different Hyperparameter Tuning Methods:

1. GridSearch
2. RandomSearch
3. Successive Halving
4. Bayesian Optimizers
5. Manual Search

1. GridSearch:

- Grid search picks out hyperparameter values by combining each value passed in the grid to each other, evaluates every one of them, and returns the best.

2. RandomSearch:

- Random Search removes the exhaustive search done by GridSearch by combining the values randomly.
- Since the selection of parameters is completely random; ***it yields high variance during computing*.**

3. Successive Halving:

- Successive halving (SH) is like a tournament among candidate parameter combinations.
- SH is an iterative selection process where all candidates (the parameter combinations) are evaluated with a small amount of resources at the first iteration.
- Only some of these candidates are selected for the next iteration, which will be allocated more resources.
- For parameter tuning, the resource is typically the number of training samples, but it can also be an arbitrary numeric parameter such as **n_estimators** in a random forest.

4. Bayesian Optimizers:

Hyperopt

Hyperopt is a Python library for serial and parallel optimization ,which may include real-valued, discrete, and conditional dimensions.

Optuna

- Eager dynamic search spaces
- Easy integration
- Good visualizations
- Distributed optimization

Scikit-Optimize

- Sequential model-based optimization
- Built on NumPy, SciPy, and Scikit-Learn
- Open source, commercially usable

BayesSearchCV

As of now BayesSearchCV is not compatible with sklearn 0.24 version.
To use BayesSearch downgrade sklearn to 0.23.2

5. Manual Search:

- Manual Search can be done on the basis of our judgment/experience.
- We train the model based on the random values that we assigned manually, evaluate its accuracy and start the process again.
- This loop is repeated until a satisfactory accuracy is scored.

Overview of RNN, CNN and LSTM

What Is a Recurrent Neural Network (RNN)?

RNN works on the principle of saving the output of a particular layer and feeding this back to the input in order to predict the output of the layer.

Below is how you can convert a Feed-Forward Neural Network into a Recurrent Neural Network:

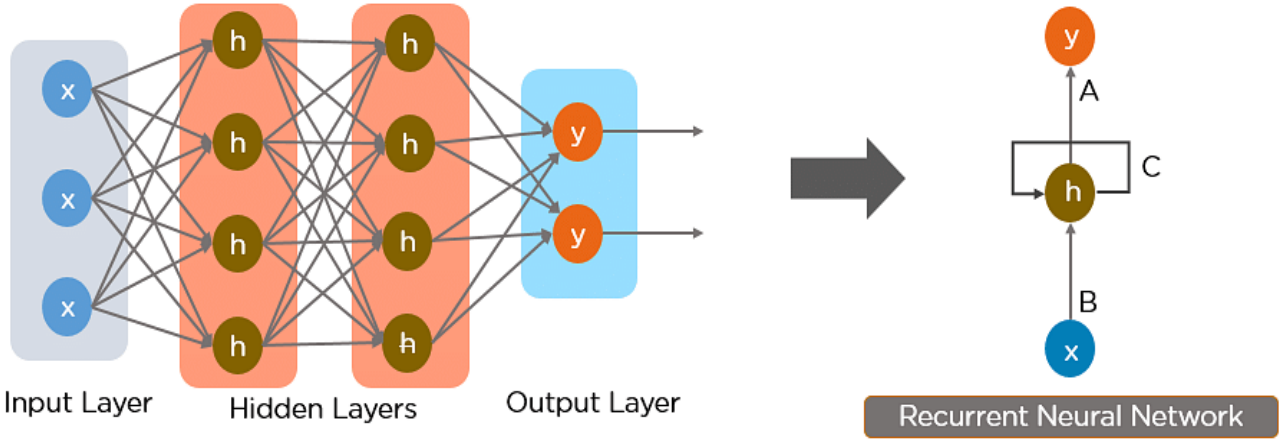


Fig: Simple Recurrent Neural Network

The nodes in different layers of the neural network are compressed to form a single layer of recurrent neural networks. A, B, and C are the parameters of the network.

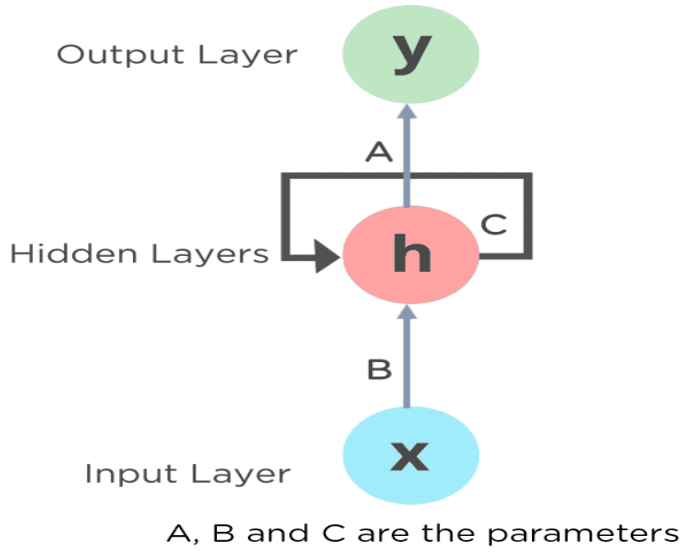


Fig: Fully connected Recurrent Neural Network

Here, “x” is the input layer, “h” is the hidden layer, and “y” is the output layer. A, B, and C are the network parameters used to improve the output of the model. At any given time t, the current input is a combination of input at x(t) and x(t-1). The output at any given time is fetched back to the network to improve on the output.

Why Recurrent Neural Networks?

RNN were created because there were a few issues in the feed-forward neural network:

- Cannot handle sequential data
- Considers only the current input
- Cannot memorize previous inputs

The solution to these issues is the RNN. An RNN can handle sequential data, accepting the current input data, and previously received inputs. RNNs can memorize previous inputs due to their internal memory.

Applications of Recurrent Neural Networks:

Image Captioning:
RNNs are used to caption an image by analyzing the activities present.

"A Dog catching a ball in mid air"

Time Series Prediction:
Any time series problem, like predicting the prices of stocks in a particular month, can be solved using an RNN.

Natural Language Processing:
Text mining and Sentiment analysis can be carried out using an RNN for Natural Language Processing (NLP).

When it rains, look for rainbows.
When it's dark, look for stars.

Positive Sentiment

Natural Language Processing

Machine Translation:
Given an input in one language, RNNs can be used to translate the input into different languages as output.

Here the person is speaking in English and it is getting translated into Chinese, Italian, French, German and Spanish languages

Machine Translation

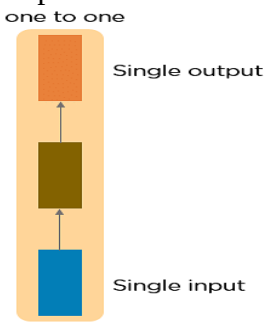
Types of Recurrent Neural Networks:

There are four types of Recurrent Neural Networks:

1. One to One
2. One to Many
3. Many to One
4. Many to Many

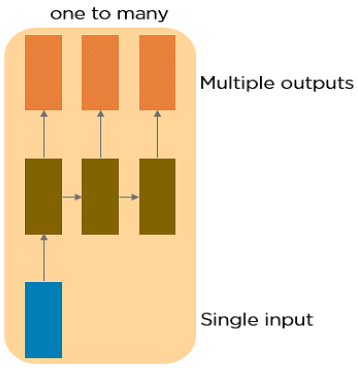
One to One RNN

This type of neural network is known as the Vanilla Neural Network. It's used for general machine learning problems, which has a single input and a single output.



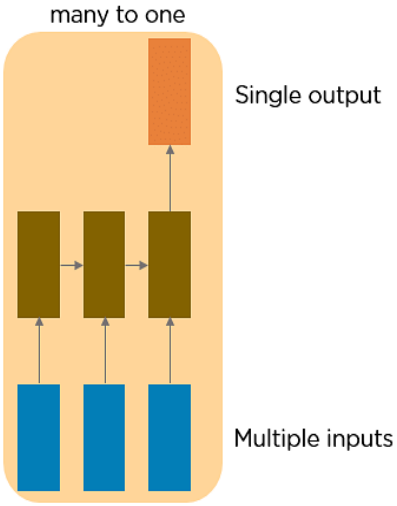
One to Many RNN

This type of neural network has a single input and multiple outputs. An example of this is the image caption.



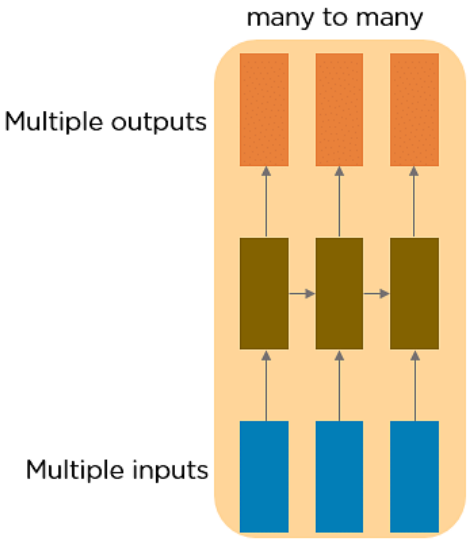
Many to One RNN

This RNN takes a sequence of inputs and generates a single output. Sentiment analysis is a good example of this kind of network where a given sentence can be classified as expressing positive or negative sentiments.



Many to Many RNN

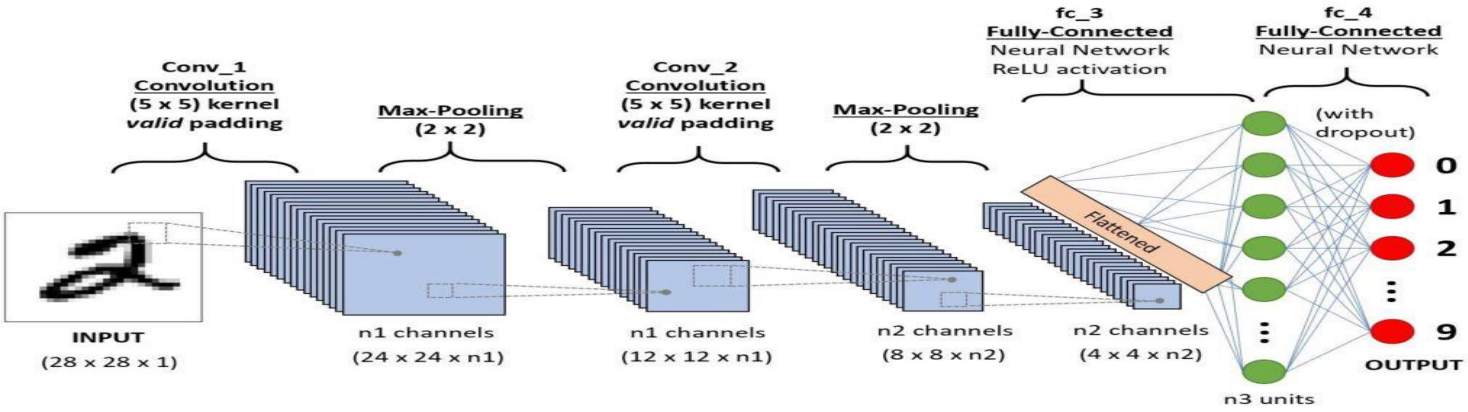
This RNN takes a sequence of inputs and generates a sequence of outputs. Machine translation is one of the examples.



What is Convolution Neural Network?

Convolutional Neural Network (CNN) or ConvNets for short is a class of deep neural networks popularly used for visual data analysis. This visual data can be in the form of images or videos. CNNs are inspired by the mammalian visual cortex. They were developed as a computational model for natural visual perception, similar to the human visual system. The applications of CNN include image classification, object detection, semantic segmentation, medical image analysis, and many more.

A Convolutional Neural Network (CNN) takes an input image, performs a series of operations on it and then classifies it under certain categories (Example: Cat or Dog). The result can vary depending upon the type of problem we are trying to solve. In the case of image classification, we get a label (probability score). In the case of semantic segmentation, we get the segmentation map in which each pixel is labeled with a class label.



A CNN to identify the handwritten digits.

The basic architecture of a convolutional neural network (CNN) consists of the following layers:

- Convolutional layer — CONV
- Activation layer — ACT
- Pooling layer — POOL
- Fully-connected layer — FC

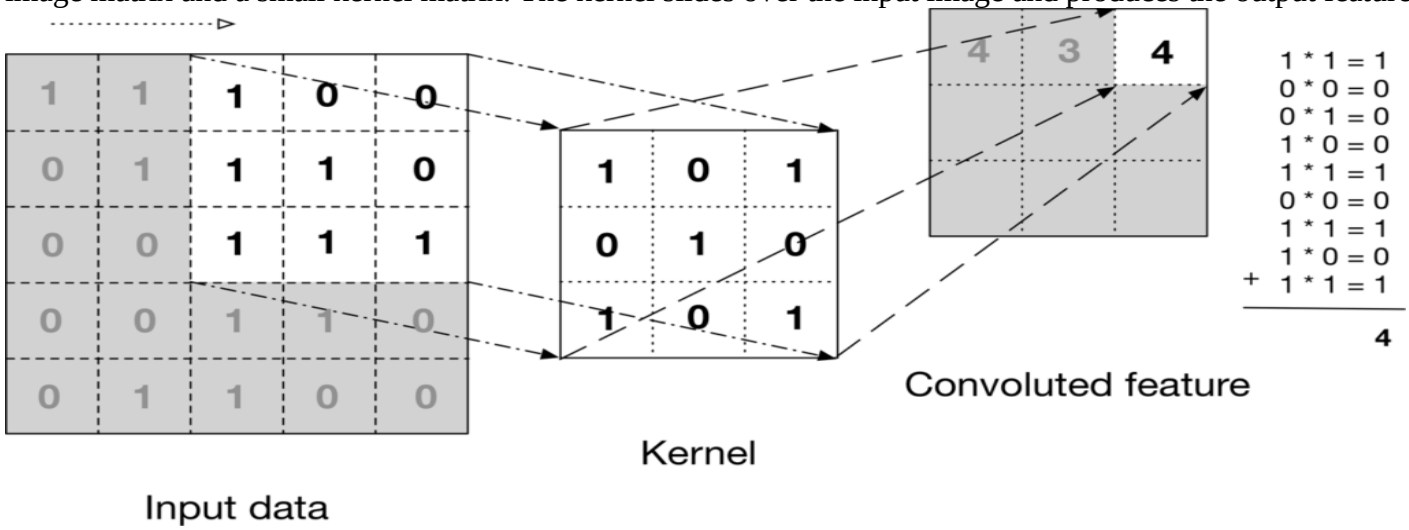
All these layers are used in some combination to build a convolutional neural network. For an image classification task, the general architecture is as follows:

INPUT-> [CONV -> ACT -> POOL] x N times -> [FC -> ACT] x M times -> OUTPUT

Convolutional Layer:

The central building block of convolutional neural networks (CNN) is the convolution layer, which enables networks to construct informative features by fusing both spatial and channel-wise information within local receptive fields at each layer.

The convolutional layer consists of the mathematical convolution operation, which takes two inputs, first the image matrix and a small kernel matrix. The kernel slides over the input image and produces the output feature map.



The basic working of the convolutional layer.

The aim of the convolution operation is to extract high-level features such as edges, lines, curves, etc from the input image. On a broad level, we can say the role of the convolution layer is to reduce the images into a form which is easier to process, without losing the essential features.

Activation Layer:

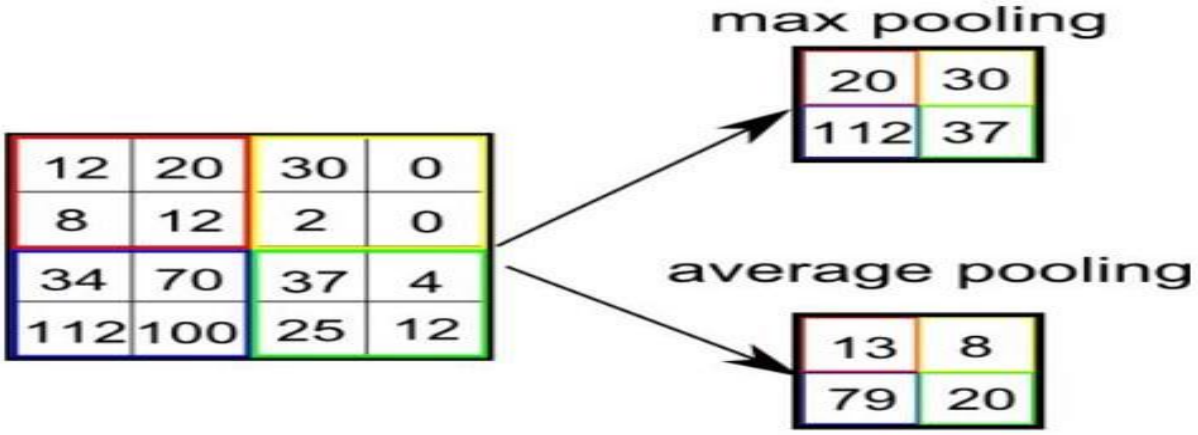
The activation layer introduces non-linearity in the neural network. ReLU (Rectified Linear Unit) is the most popularly used activation function in the convolutional neural network. It is used in all the layers except the output layer. The activation function of the output layer depends upon the type of problem we are trying to solve. Like in the case of classification, we use sigmoid for single class and softmax for multiple classes.

Pooling Layer:

The pooling layer is used to reduce the spatial dimensions i.e., height and width of the input feature maps. It helps to reduce the computational resources required to process the features maps by dimensionality reduction.

There are two major types of pooling:

- **Max pooling**
- **Average pooling**



Fully Connected Layer:

After a series of convolutional, activation, and pooling layers, we flatten the incoming feature maps and feed them as input to the fully connected layers for classification.

The fully connected or dense layers take the flattened input features and the number of output features.

For example:

The Input shape of features from the pooling layer is (B, H, W, N). Now, we flatten the feature maps by reshaping them into (B, H*W*N). Now we feed it in the fully connected layer with the number of output features be the O. The shape of the feature map given by the fully connected layer is (B, O).

After that we apply an activation function on the output of the fully connected layer, we get a probability score, which is used to determine the label for the input image.

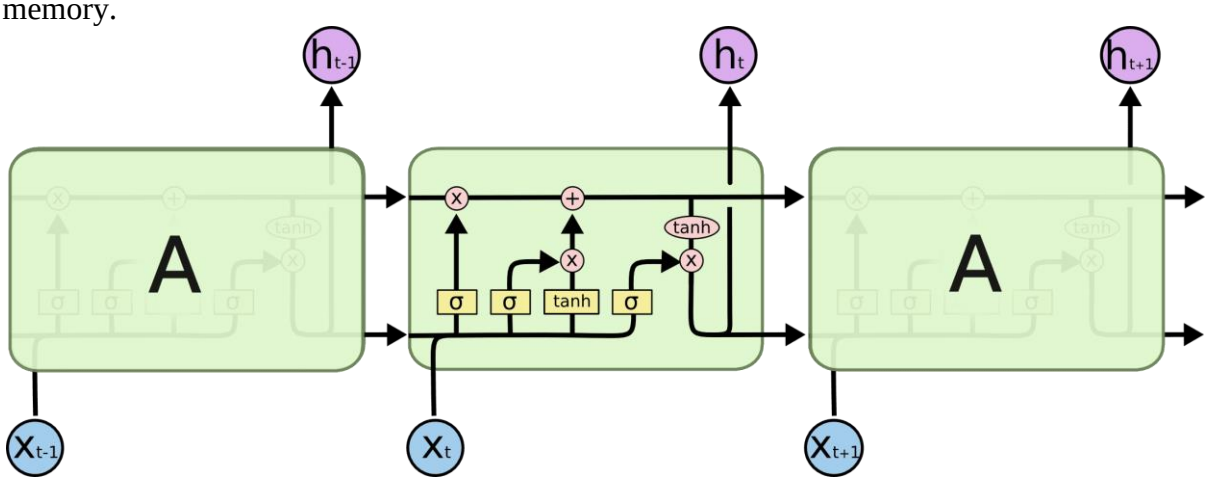
Long Short Term Memory (LSTM):

LSTMs are a special kind of **RNN**, capable of learning long-term dependencies which make **RNN** smart at remembering things that have happened in the past and finding patterns across time to make its next guesses make sense. **LSTMs** broke records for improved Machine Translation, Language Modeling and Multilingual Language Processing.

The LSTM networks are popular nowadays. The LSTM network are called cells and these cells take the input from the previous state *ht-1* and current input *xt*. The main function of the cells is to decide what to keep in mind and what to omit from the memory. The past state, the current memory and the present input work together to predict the next output.

Architecture of LSTM network:

LSTM network have a sequence like structure, but the recurring network has a different module. Instead of having single neural network layer, they have small parts connected to each other which function in storing and removal of memory.



There are various gates in the **LSTM** process. When the cell state is carrying the information, these gates help the new information flow. The gates will indicate which data is useful to keep and which data is not useful, making it okay to throw. So only the relevant data passes through the chain of sequence for easy prediction.

Sigmoid

The gates contain various activations called sigmoid, which contain some values. These values range from zeros to one. These values help in forgetting and keeping the information. If the data multiply by one, the value of that data remains the same. However, if the data multiplies by zero, the value becomes zero and disappears. We can learn more if we closely understand these gates. There are three types of gates:

Forget Gate

The first gate that we will understand is the forget gate. The function of this gate is to decide to keep or forget the information. Only information coming from previously hidden layers and the current input stays with the sigmoid function. Any value that is closer to one will stay, and any value closer to zero will disappear.

Input Gate

The input gate helps in updating the cell state. The current input and previous state information pass through the sigmoid function, which will update the value by multiplying with zero and one. Similarly, for regulating the network, the data also passes through the tanh function. Now, the output of sigmoid multiplies by the output of tanh. The output of sigmoid will identify valuable information to keep from the tanh output.

Cell State

Now, the information we have will help us calculate the cell state. The cell state's value may drop if the multiplied value is near zero after multiplying the forget vector and the previous cell state. Now, we can find the new cell state by adding the output of the input gate pointwise.

Output Gate

The next hidden state is defined in the output gate. To find the hidden state's information, we need to multiply the sigmoid output with the tanh output. You can predict the following information from a hidden state. Now, the new hidden and new cell state will travel to the next step.

Data Science Models

The field of Data Science has been expanding ever since it's onset in the early 2000s. With time, more and more cutting edge technologies are being incorporated into the field. Some of such more recent additions are listed below:

Artificial Intelligence: Machine Learning has been one of the core elements of Data Science. However, with the increased parallel compute capabilities, Deep Learning has been the latest and one of the most significant additions to the Data Science field.

Smart Apps or Intelligent Systems: The development of data-driven intelligent applications and their accessibility in a portable form factor has lead to the inclusion of a part of this field into Data Science. This is primarily because a large portion of Data Science is built around Machine Learning, which is also what Smart Apps and Intelligent Systems are based on.

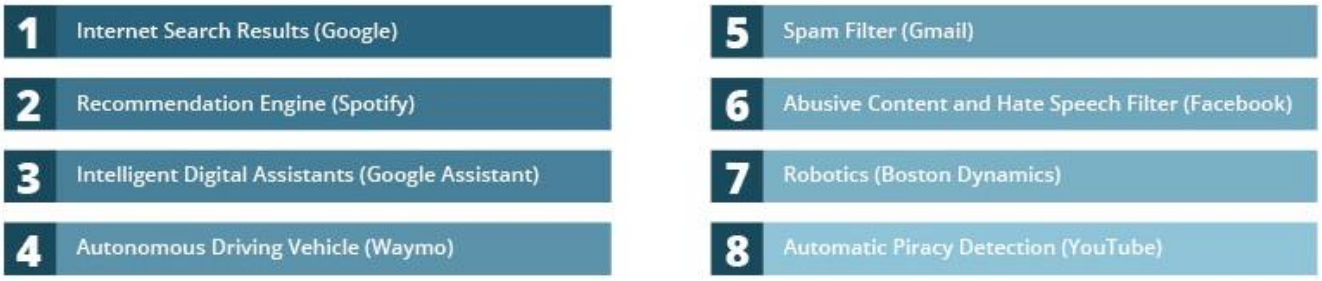
Edge Computing: Edge computing is a recently developed concept and is related to IoT (Internet of Things). Edge computing basically puts the Data Science pipeline of information collection, delivery, and processing closer to the source of information. This is achievable through IoT and has recently been added to be a part of Data Science.

Security: Security has been a major challenge in the digital space. Malware injection and the concept of hacking is quite common and all digital systems are vulnerable to it. Fortunately, there have been few recent technological advancements which apply Data Science techniques to prevent exploitation of digital systems. For example, Machine Learning techniques have proven more capable of detecting computer virus or malware when compared to traditional algorithms.

Applications of Data Science:

Data Science is such a field which can be applied to almost every industry to solve complex problems. Every company applies Data Science to a different application with the view of solving a different problem. Some companies completely depend upon Data Science and Machine Learning techniques to solve a certain set of problems, which,

otherwise, could not have been solved. Some of such applications of Data Science and the companies behind them are listed below.



Internet Search Results (Google): When a user searches for something on Google, complex Machine Learning algorithms determine which are the most relevant results for the search term(s). These algorithms help to rank pages such that the most relevant information is provided to the user at the click of a button.

Recommendation Engine (Spotify): Spotify is a music streaming service which is quite popular for its ability to recommend music as per the taste of the user. This is a very good example of Data Science at play. Spotify's algorithms use the data generated by each user over time to learn the user's taste in music and recommend him/her with similar music in the future. This allows the company to attract more users since it is more convenient for the user to use Spotify as it does not demand much attention.

Intelligent Digital Assistants (Google Assistant): Google Assistant, similar to other voice or text-based digital assistants (also known as chatbots) is one example of advanced Machine Learning algorithms put to use. These algorithms are able to convert the speech of a person (even with different accents and languages) to text, understand the context of the text/command and provide relevant information or perform a desired task, all just by speaking to the device.

Autonomous Driving Vehicle (Waymo): Autonomous Driving vehicles are one of the bleeding edge of technology. Companies like Waymo uses high-resolution cameras and LIDARs to capture live video and 3D maps of the surrounding in order to feed that through Machine Learning algorithms which assist in autonomously driving the car. Here, the data is the videos and 3D maps captured by the sensors.

Spam Filter (Gmail): Another key application of Data Science which we use in our day-to-day life is the spam filters in our emails. These filters automatically separate the spam emails from the rest, effectively giving the user a much cleaner email experience. Just like the other applications, Data Science is the key building block here.

Abusive Content and Hate Speech Filter (Facebook): Similar to the spam filter, Facebook and other social media platforms use Data Science and Machine Learning algorithms to filter out abusive and age-restricted content from the unintended audience.

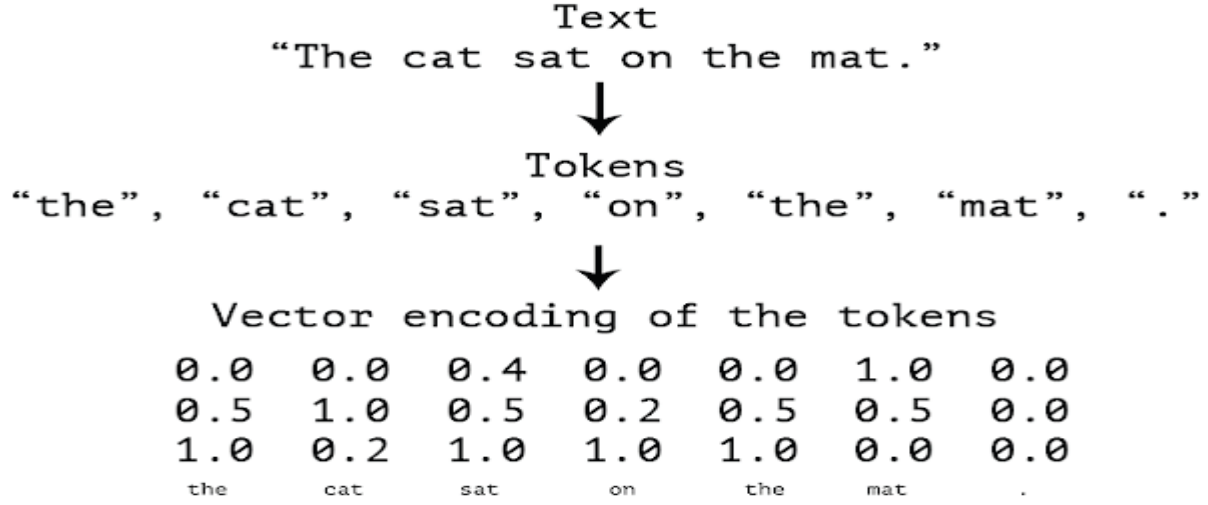
Robotics (Boston Dynamics): A key component of Data Science is Machine Learning, which is exactly what fuels most of the robotics operations. Companies like Boston Dynamics are at the forefront of the robotics industry and develop autonomous robots that are capable of humanoid movements and actions.

Automatic Piracy Detection (YouTube): Most videos that are uploaded to YouTube are original content created by content creators. However, quite often, pirated and copied videos are also uploaded to YouTube, which is against their policy. Due to the sheer volume of daily uploads, it is not possible to manually detect and take down such pirated videos. This is where Data Science is used to automatically detect pirated videos and remove them from the platform.

Applications to text in Deep Learnig:

- Text is one of the most widespread form of sequence data. It can be understood either as a sequence of characters, or a sequence of words, albeit it is most common to work at the level of words.
- The deep learning sequence processing models that we'll introduce can use text to produce a basic form of natural language understanding, sufficient for applications ranging from document classification, sentiment analysis, author identification, or even question answering (in a constrained context).

- Deep learning for natural language processing is pattern recognition applied to words, sentences, and paragraphs, in much the same way that computer vision is pattern recognition applied to pixels. Like all other neural networks, deep learning models don't take as input raw text: they only work with numeric tensors. Vectorizing text is the process of transforming text into numeric tensors. This can be done in multiple ways:
    - By segmenting text into words, and transforming each word into a vector.
    - By segmenting text into characters, and transforming each character into a vector.
    - By extracting "N-grams" of words or characters, and transforming each N-gram into a vector. "N-grams" are overlapping groups of multiple consecutive words or characters.
- Collectively, the different units into which you can break down text (words, characters or N-grams) are called "tokens", and breaking down text into such tokens is called "tokenization".
- All text vectorization processes consist in applying some tokenization scheme, then associating numeric vectors with the generated tokens. These vectors, packed into sequence tensors, are what get fed into deep neural networks. There are multiple ways to associate a vector to a token. In this section we will present two major ones: one-hot encoding of tokens, and token embeddings (typically used exclusively for words, and called "word embeddings").



Understanding N-grams and "bag-of-words".

Word N-grams are groups of N (or fewer) consecutive words that you can extract from a sentence. The same concept may also be applied to characters instead of words. Here's a simple example. Consider the sentence: "The cat sat on the mat". It may be decomposed as the following set of 2-grams:

{"The", "The cat", "cat", "cat sat", "sat", "sat on", "on", "on the", "the", "the mat", "mat"}

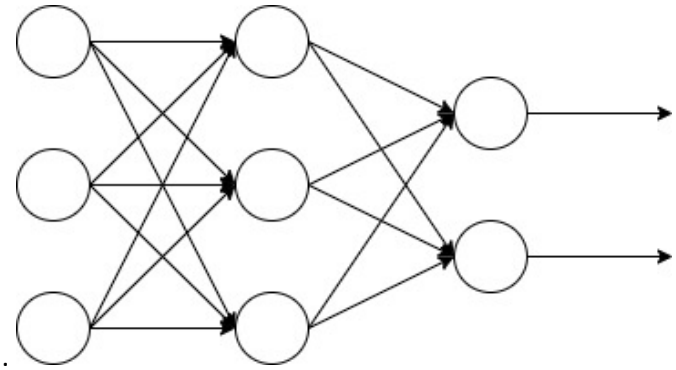
It may also be decomposed as the following set of 3-grams:

{"The", "The cat", "cat", "cat sat", "The cat sat", "sat", "sat on", "on", "cat sat on", "on the", "the", "sat on the", "the mat", "mat", "on the mat"}

Applications to image in Data Science:

In recent years, the use of Deep Learning mechanisms and techniques in image processing applications has become popular. The availability of powerful computing environments on the Internet and the fact that cloud providers offer ready-made machine learning and artificial intelligence laboratories to researchers have accelerated these efforts.

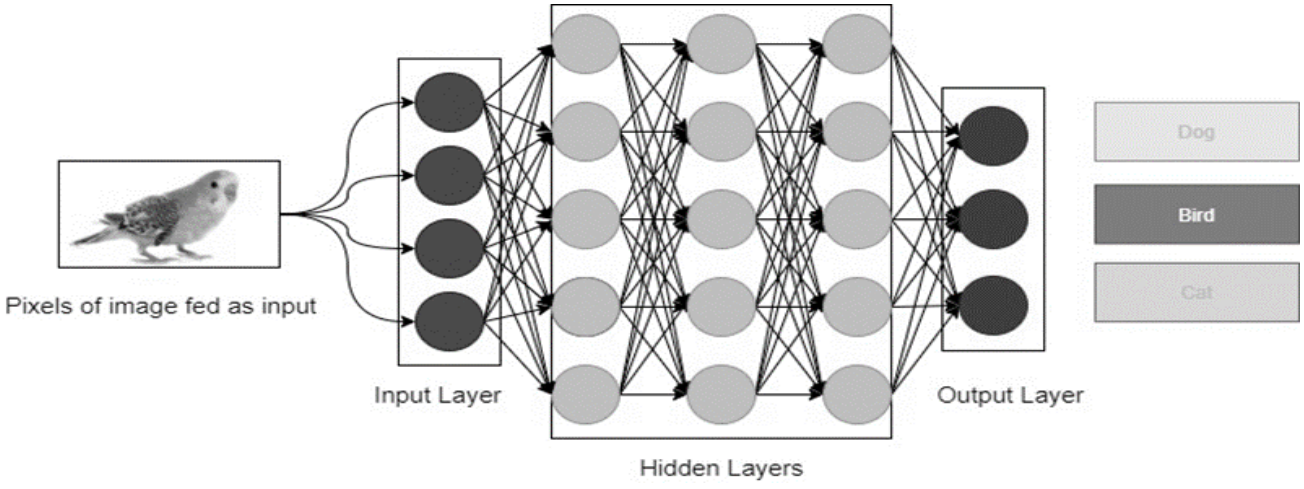
Deep learning is a machine learning method that predicts the results of a given dataset and its structure consists of more than one artificial neural network. It can be termed as a subfield of machine learning. Researchers use Deep Learning software to analyze complex and large data sets and process image, text and audio data more accurately and quickly.



Nowadays, there are many different neural network architectures designed for different purposes. The simple architecture of a neural network is shown in Figure. The names of these architectures are defined by the type of layers used and the way the layers are interconnected.

The deep learning algorithm Convolutional Neural Network (CNN) is still widely used for object recognition. CNN has shown successful results in image processing applications such as image segmentation and classification .

As an example, consider the image of a bird in below Figure. To determine whether it is really a bird or some other object, send the pixels of the image as arrays to the input layer of the neural network (multilayer networks for object classification). Hidden layers implement various computational methods and perform feature extraction by making changes. Convolutional layer performs the process of feature extraction from the image sent to the system. As the last layer, there is a fully connected layer that defines the object in the image.



Convolutional Neural Networks are inspired by biology. CNN consists of three basic layers: Convolutional Layer, Pooling Layer and Fully- connected Layer. Any number of Convolutional and Pooling layers can be applied

sequentially. Then the fully-connected layer is used. If the classification problem is to be solved with multiple labels, the softmax layer is used as the last layer. In the fully connected layer, the three dimensional input is reduced to one dimension . In the literature, there are many deep learning models that use Convolutional Neural Network (CNN). The most commonly used CNN models include LeNet, AlexNet, ZFNet, VGGNet, GoogLeNet and ResNet.

Image Processing Applications:

This section will describe two commonly used image processing applications.

YOLO

It is an algorithm that has been widely used in recent years for computer recognition of objects. Its most important feature is real-time object recognition. The general average accuracy (mAP) values used in object recognition are widely used because they are better than others.

SSD

SSD algorithm is used for real-time object detection. However, it uses optimization methods such as multiscaling and standard boxes to improve the accuracy drop. With these improvements, SSD tries to match the accuracy of Faster R-CNN on low resolution images and further increase the speed. The SSD algorithm consists of 2 stages. In the first stage, feature maps are created. In the second stage, convolutional filters are applied to detect objects.

Applications video in Deep Learning

Deep Neural Networks for video tasks are part of the most complex models, with at least twice the parameters of models for image tasks. Despite the success of deep learning architectures in image classification, progress in architectures for video based AI has been slower.

There are two basic approaches for video-based AI tasks:

1. Single Stream Network:

In this approach the consecutive frames are presented as the input, and there are four different configurations that can be used:

a. Single frame: a single architecture is used to fuse information from all the frames at the last stage.
b. Late fusion: two nets with shared params are used. The nets are spaced 15 frams apart, and combine predictions at the end of the configuration.
c. Early fusion: the combination is performed in the first layer by convolving over 10 frames.
d. Slow fusion: fusion is performed at multiple stages, as a balance between early and late fusion.
Multiple clips are sampled from the entire video and prediction scores are averaged from the sampled clips in order to perform the final predictions.

Important issues that provoked this approach to fail:

- The learnt spatiotemporal features don't capture motion features.
- The used dataset was not diverse enough, causing difficulties to learn such detailed features.

2. Two Stream Networks:

It's an approach ,which tries to overcome the failures of the Single Stream Network approach. Instead of a single network for spatial context, the proposed architecture is composed of two separate networks: a pre-trained network for spatial context, and another network for motion context.

Deep learning-based recommender systems

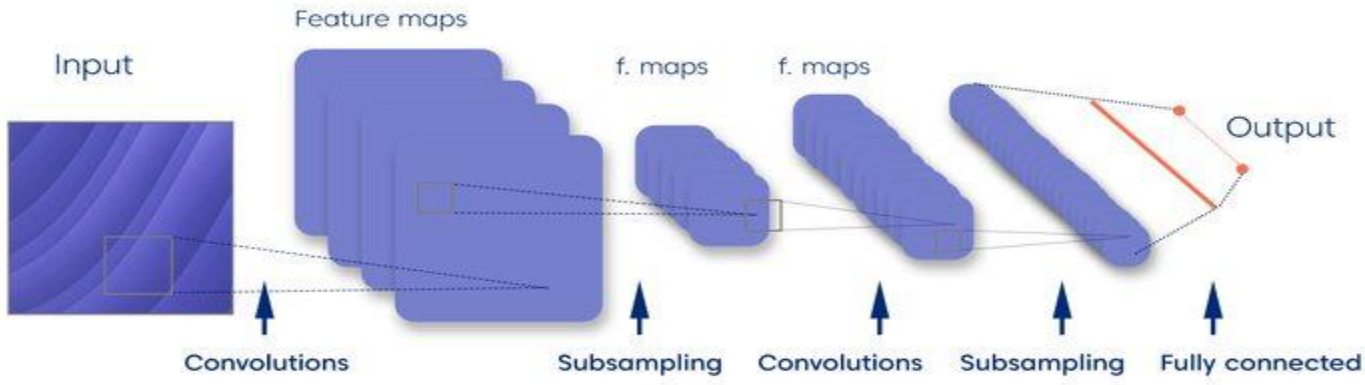
In general, deep learning (DL) is the subfield of ML learning from multiple levels of data representations and abstractions. Some tech companies are already using DL systems drawn on different neural networks (NNs) to make the customer experience better. For instance, YouTube, eBay, Yahoo, and Twitter choose deep neural networks (DNNs), while Spotify prefers convolutional neural networks (CNNs).

Meanwhile, DNNs and CNNs are only a few types of networks applied, as we can continue this list of deep learning algorithms. Why do we need a range of them? The answer is related to the business domain, particular task, or recommender scenario. Owing to the use case, one might leverage different NNs or even a hybrid approach to DL-based recommendation systems.

. **Deep learning for recommendations:**

To explain how each particular RS operates and what kind of benefits it brings, we have developed an overview of the most popular techniques.

Convolutional neural networks based recommender systems

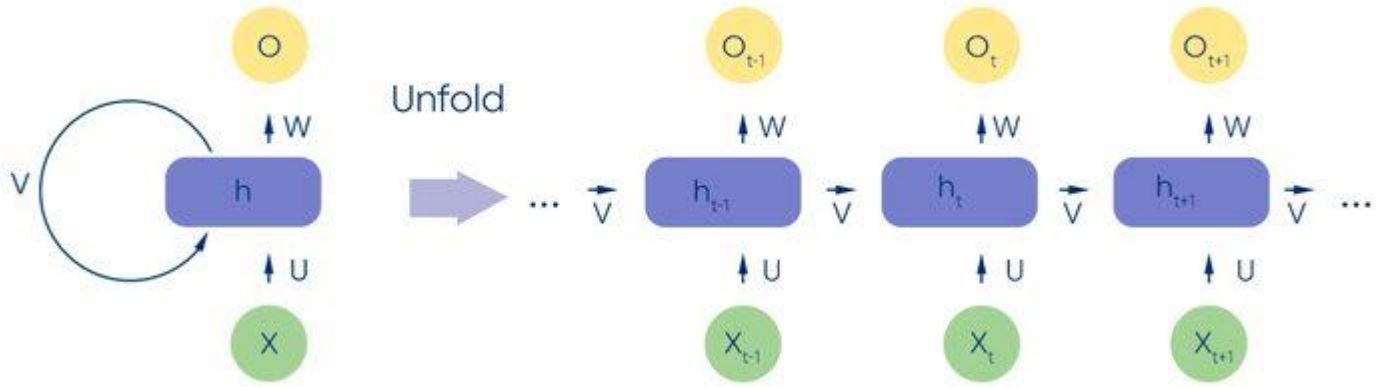


Typical CNN architecture:

Convolutional Neural Networks (CNNs) are a good fit for unstructured multimedia data processing given effective feature extraction. They are processing the data like image, text, audio, and video. CNNs help to eliminate the cold start problem or empower traditional systems like collaborative filtering. This feature is critical for e-commerce, as most customers conclude their decisions by assessing goods' visuals.

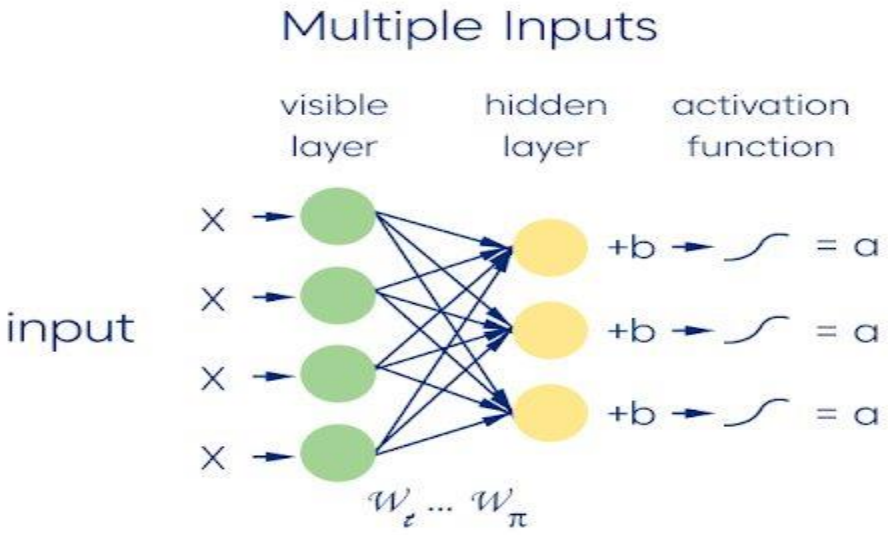
CNNs is also an option for non-Euclidean data (non-ordinal or hierarchical data) like social networks, protein-interaction networks, and knowledge graph. For instance, this kind of system could apply to Pinterest recommendations.

Recurrent neural networks based recommender systems:



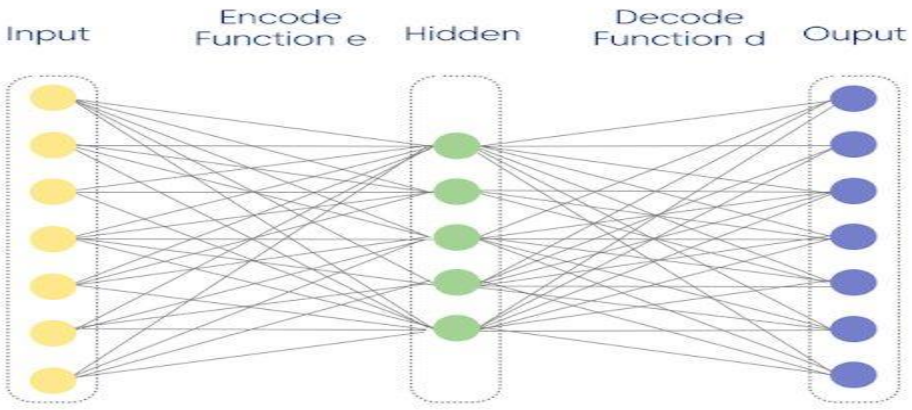
Recurrent Neural Networks (RNNs) could become a killer feature for sequential data processing, defining temporal dynamics of interactions and sequential user behavior patterns. For example, YouTube recommends content for a particular time of the day or predicts the next piece of content drawn on the already watched one.

Restricted Boltzmann machine based recommender systems:



Restricted Boltzmann Machine (RBM) belongs to the most old-fashioned RSs rooted from 2007 but is still in play. Significantly, RBM, combined with collaborative filtering, won the Netflix Prize for better recommendations on the streaming platform in 2009. Moreover, RBM-based techniques are still scalable to large data sets and producing high-quality recommendations of items per particular user.

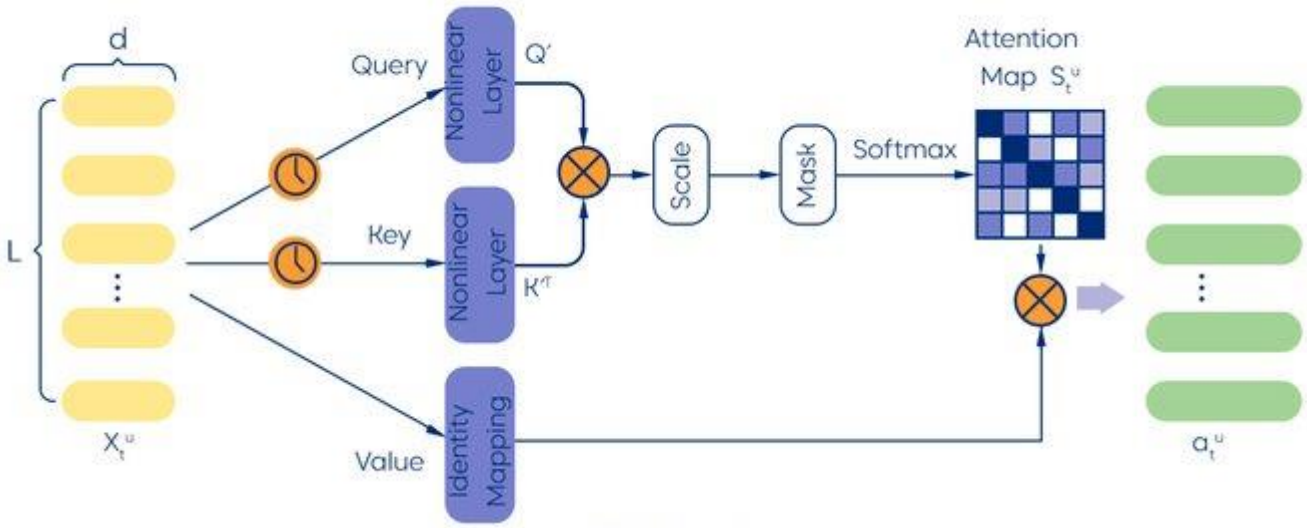
Autoencoder based recommender systems:



In essence, an autoencoder is a neural network that reconstructs its input data in the output layer. It has an internal hidden layer that describes a code used to represent the input. The autoencoder consists of two main parts. They are an encoder that maps the information into the code and a decoder that maps the code to reconstruct the input. The beauty of autoencoder is in its agility in data dimensionality reduction, data reconstruction, and feature extraction.

Neural attention based recommender systems:

Attention mechanism derives from computer vision and natural language processing domains. Simply put, it is a vector of importance weights that predicts the next item. The attention mechanism is based on correlation with other elements (e.g., a pixel in the image or the next word in a sentence). In essence, human visual attention stands as a source of inspiration for this technique.



Social Networking Graph

Social network analysis is an important problem in data mining. A fundamental step for analyzing social networks is to encode network data into low-dimensional representations, i.e., network embeddings, so that the network topology structure and other attribute information can be effectively preserved. Network representation leaning facilitates further applications such as classification, link prediction, anomaly detection, and clustering.

Introduction:

Social networks, such as Facebook, Twitter, and LinkedIn, have greatly facilitated communication between web users around the world. The analysis of social networks helps summarizing the interests and opinions of users (nodes), discovering patterns from the interactions (links) between users, and mining the events that take place in online platforms. The information obtained by analyzing social networks could be especially valuable for many applications.

2. Notations and Problem Definitions:

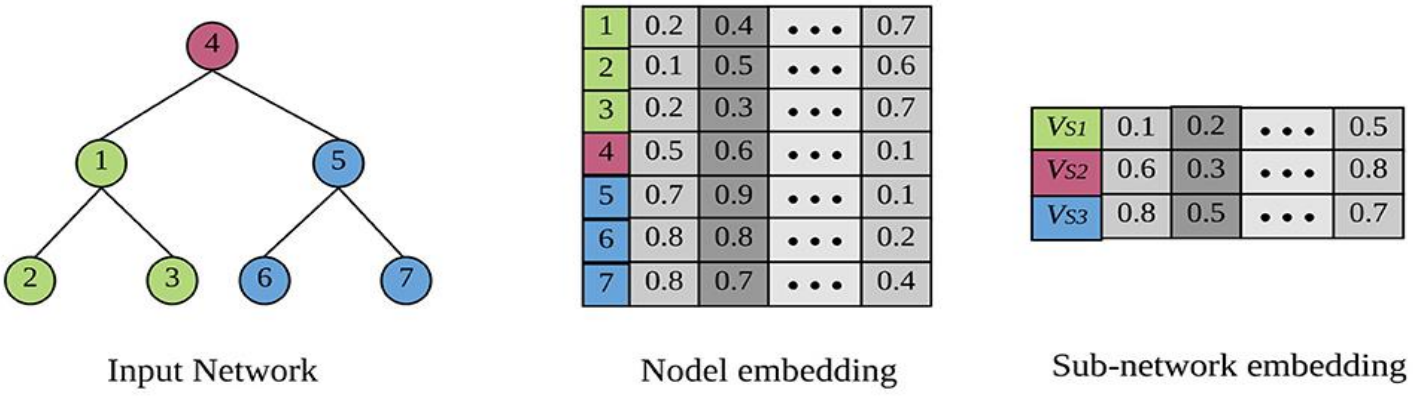
In this section, we define some important terminologies that will be used in later sections

Definition 1 (Network). Let G={V,E,X,Y}G={V,E,X,Y} be a network, where the *i*-th node (or vertex) is denoted as vi∈Vvi∈V and ei,j∈Eei,j∈E denotes the edge between node v_i and v_j. **X** and **Y** are node attributes and labels, if available.
Definition 2 (Homogeneous Network). A homogeneous network is a network in which $|T^v| = |T^e| = 1$. All nodes and edges in *G* belong to one single type.

Definition 3 (Heterogeneous Network). A heterogeneous network is a network with $|T^v| + |T^e| > 2$. There are at least two different types of nodes or edges in heterogeneous networks.

Definition 4 (Node Representation Learning). Suppose $\mathbf{z} \in \mathbb{R}^D$ denotes the latent vector of node *v*, node representation learning aims to build a mapping function *f* so that $\mathbf{z} = f(v)$. It is expected that nodes with similar roles or characteristics, which are defined according to specific application domains, are mapped close to each other in the latent space.

Definition 5 (Subgraph Representation Learning). Let *g* denote a subgraph of GG. The nodes and edges in *g* are denoted as VSVS and ESES, respectively, and we have VS⊂VVS⊂V and ES⊂EES⊂E. The subgraph representation learning aims to learn a mapping function *f* so that $\mathbf{z} = f(g)$, where in this case $\mathbf{z} \in \mathbb{R}^D$ corresponds to the latent vector of *g*.
Figure 1 shows a toy example of network embedding. There are three subgraphs in this network distinguished with different colors: VS1={v1,v2,v3}VS1={v1,v2,v3}, VS2={v4}VS2={v4}, and VS3={v5,v6,v7}VS3={v5,v6,v7}. Given a network as input, the example below generates one representation for each node, as well as for each of the three subgraphs.
Figure 1



Note: write about NNs,CNN,RNN,LSTM

Dynamic Networks:

Social networks are inherently highly dynamic in real-life scenarios. The overall set of nodes, the underlying network structure, as well as attribute information, might evolve over time. As an example, these elements in real world social networks such as Facebook could correspond to users, connections, and personal profiles. This property makes existing static learning techniques fail in working properly. Although several methods have been proposed to tackle dynamic networks, they often rely on certain assumptions, such as assuming that the node set is fixed and only deals with dynamics caused by edge deletion and addition.

Hierarchical Network Structure:

Most of the existing techniques mainly focus on designing advanced encoding or decoding functions trying to capture node pairwise relationships. Nevertheless, pairwise relations can only provide insights about local neighborhoods, and might not infer global hierarchical network structures, which is crucial for more complex networks .

Heterogeneous Networks:

Existing network embedding methods mainly deal with homogeneous networks. However, many relational systems in real-life scenarios can be abstracted as heterogeneous networks with multiple types of nodes or edges. In this case, it is hard to evaluate semantic proximity between different network elements in the low-dimensional space. While some work has investigated the use of metapaths to approximate semantic similarity for heterogeneous network embedding, many tasks on heterogeneous networks have not been fully evaluated. Learning embeddings for heterogeneous networks is still at the early stage, and more comprehensive techniques are required to fully capture the relations between different types of network elements, toward modeling more complex real systems.

Scalability:

Although deep learning based network embedding methods have achieved substantial performances due to their great capacities, they still suffer from the problem of efficiency. This problem will become more severe when dealing with real-life massive datasets with billions of nodes and edges. Designing deep representation learning frameworks that are scalable for real network datasets is another driving factor to advance the research in this domain.

Interpretability:

Despite the superior performances achieved by deep models, one fundamental limitation of them is the lack of interpretability. Different dimensions in the embedding space usually have no specific meaning, thus it is difficult to comprehend the underlying factors that have been preserved in the latent space. Since the interpretability aspect of machine learning models is currently receiving increased attention, it might also be important to explore how to understand the representation learning outcome, how to develop interpretable network representation learning models, as well as how to utilize interpretation to improve the representation models. Answering these questions is helpful to learn more meaningful and task-specific embeddings toward various social network analysis problems.