Python Beginners Guide

STUDY GUIDE

SUBIN VASANTHAN

This guide is prepared by Subin Vasanthan as a personal knowledge reference and learning companion Python.

Table of Contents

1. Extracting a char form a string

Start_index=0 : → This means the counting starts from 0, unlike 1 for excel

a) Index

Explanation: Indexing Helps you to extract a character

Syndax:

String_Object[Index]

Example

```
[In]    mystring= "this is python 3 intro class"
[In]    mystring [15]
[out]   ‘3’
```

b) Slicing

Explanation: Slicing helps you to extract a set of characters

Syndax:

String_Object[start index:end_index]

Example

```
[In]    mystring= "this is python 3 intro class"
[In]    print(mystring[0:4])
[In]    print(mystring[:22])
[out]   this
[out]   this is python intro
```

c) Concatenation

Explanation:

Syndax:

String_Object[start index:end_index] + String_Object[start index:end_index]

Example

```
[In]    mystring= "this is python 3 intro class"
[In]    print(mystring[8:15] + mystring[17:22])
[In]    print(mystring[0:8] + "Java"+ mystring[16:])
[out]   python intro
[out]   this is Java intro class
```

2. Functions- In-Built Methods

Explanation: There are inbuilt functions that helps us to determine several computations on the string
Functions can be opened by entering "." and pressing **Tab** on your keyboard
Also you can get the explanation of the function by pressing **Shift+Tab**

Syndax:

String_Object.Function()

a) Example- replace

```
[In]    print(mystring.replace("python","Java",-1))
        print(mystring.replace("python","Java",1))
        print(mystring.replace("python","Java",2))
        print(mystring.replace("python","Java",3))
        print(mystring.index("python"))
        print(mystring.index("python",9))
        mylist= ["A","B","C"]
        print('-'.join(mylist))
[out]   this is Java 3 intro class and Java and Java
        this is Java 3 intro class and python and python
        this is Java 3 intro class and Java and python
        this is Java 3 intro class and Java and Java
        8
        33
        A-B-C
```

3. Typcasting using format method

Typecasting uses {} to typecast

a) Example. format

```
[In]    a=5
        b=10
        c=a+b
        print('Sum of the {} and {} is {}'.format(a,b,c))
        print('Sum of the {n1} and {n2} is {n3}'.format(n2=a,n1=b,n3=c))
[out]   Sum of the 5 and 10 is 15
        Sum of the 10 and 5 is 15
```

4. Inputs

Explanation: This function connects your python to your keyboard.
Note: The input takes the values always as a string. So we have to type cast the values into other types line int or float.

Syndax:

Input()

Example

```
[In]    a=input('Enter 1st no.')
        b=input('Enter 1st no.')
        c=int(a)+int(b)
        print('Sum of the {} and {} is {}'.format(a,b,c))
```

```
[out]    Enter 1st no.1
         Enter 1st no.2
         Sum of the 1 and 2 is 3
```

5. Arithmetic Operators (Important only)

a) Integer Division

Example

```
Print(41//2)  # Output type will be Integer
Print(41.0//2)  # Output type will be Floating Point
```

b) Exponentiation (raised to the power)

Example

```
Print(41**2)
```

c) Replication

Example

```
Print('Hello' * 2)
Output : HelloHello
Print ('2' *2)
Output: 22
```

d) Reminder

Example

```
Print(50 % 2)
```

6. Comparison Operators

```
==
!=
>
>=
<
<=
```

Example

```
Print('abc' == 'ABC'.lower())
Output: True
```

7. Logical /Bitwise Operators

a) Logical

```
and
or
not
```

b) Bitwise

```
&
|
~
```

8. Assignment Operators

= → initializing a value
+= → Adding a value and assigning it to the same variable
-= → Subtracting value and assigning it to the same variable
*= → Multiplying a value and assigning it to the same variable
/= → Dividing a value and assigning it to the same variable
** → Exponentiation of a value and assigning it to the same variable

Example

```
[In]    val = 10
        val += 10
        print(val)
[out]   20
```

9. Identity operators (is, is not)

It helps to understand if 2 objects share the same memory location
Id() → is the method used to retrieve the memory location

Example

10. Identity operators (in, not in)

Returns true/false if an item is a member of iterables
Eg of iterables → string, list, tupple, set, array, etc.

Example

```
[In]    str1= “python 3”
        City = [‘Bangalore’, ‘pune’, ‘chennai’, ‘mumbai’]  # Example of a list
        Print(‘p’ in str1)
        Print (‘x’ in str1)
        Print (‘x’ not in str1)
        Print (‘Bangalore’ in City)
[out]   True
        False
        True
        True
```

11. Conditional Statements

Simple if statement
If-else statements
If – elseif – else ladder

Syndax:

```
If <expression == True>:                    # always rememeber to use “:”
        <Task1>
Else:
        <Task2>
```

Example

To check if a number is even or odd

```
n=input('enter the number=')
if n.isdigit():
   if int(n)%2 == 0:
      print('{} is even'.format(n))
   elif int(n)%2 != 0:
      print('{} is odd'.format(n))
else:
   print('unknown')
```

To check if an input represents valid integer

```
n=input('enter the number=')
if n.isdigit():
   if int(n)%2 == 0:
      print('{} is even'.format(n))
   elif int(n)%2 != 0:
      print('{} is odd'.format(n))
else:
   print('unknown')
```

Calculator

```
a=input('Enter the first number:')
b=input('Enter the second number:')
c=input('Enter the Operator -+*/:')

if (a.isdigit() and b.isdigit() and (c=='+' or c=='-' or c=='*' or c=='/')):
   if c == '+':
      d=int(a)+int(b)
      print ('{} {} {} = {}'.format(a,c,b,d))
   elif c == '-':
      d=int(a)-int(b)
      print ('{} {} {} = {}'.format(a,c,b,d))
   elif c == '*':
      d=int(a)*int(b)
      print ('{} {} {} = {}'.format(a,c,b,d))
   elif c == '/':
    d=int(a)/int(b)
    print ('{} {} {} = {}'.format(a,c,b,d))
else:
  print('Invalid Inputs')
```

WAP to validate format a number and print ‘correct’ and ‘incorrect’ for the below cases

- 0.99 → correct
- 1 → correct
- 1,000.99 → correct
- 1,00.09 → incorrect
- 1,111,333 → correct
- 1,11,222 → Incorrect

12. Loops

2 types of loops that for and while

a) While Loop

Syndax:

```
### initialize the control variable
While  <expression == True>:
           <Tasks>
### update control variables
```

i. Example- Occurance of a letter in a string

```
### WAP to display index of every occurance of 'a' in string 'abaabijab'
indx =0
mystr = 'abaabijab'

while indx < len(mystr):
   if mystr[indx] == 'a':
      print(indx)
   indx +=1
```

ii. Example- Game with Pass

```
### game
     while True:
        n= input("Enter the Intiger:")

        if n.isdigit():
           pass
        else:
           print("Oops..Invalid Integrer")
           print("Game Terminated".center(100))
           break
```

b) For

Syndax:

```
For variable in <iterable>
           <Tasks>
```

i. For with range eg:1

```
for i in range(1,11):      #range is a function syndax is range(start,end,step) where end is exclusive
   print(i)

for i in range(10,0,-1):      #range is a function syndax is range(start,end,step) where end is exclusive
   print(i)
```

ii. for with range eg:2

### WAP to display index of every occurance of 'a' in string 'abaabijab'

```
indx =0
mystr = 'abaabijaba'
for indx in range(0,len(mystr)):
   if mystr[indx] == 'a':
      print(indx+1)
```

iii. for with range eg:3

```
for 1 in [1,2,3,4]
           print(i)
```

iv. for with char

```
for char in mystr:
           print(char)
```

v. for with continue

```
for i in range (1,11):
   if i==6:
      continue    # once the continue is hit the loop goes back tot e beginning of the for loop without the print
   print(i)
```

vi. for with break

```
for i in range (1,11):
   if i==6:
      break                  # once the break is hit it breaks the for loop
   print(i)
```

13. Collection objects

a) List

Properties of a list

1. Ordered and Indexable heterogeneous data structure
2. Duplicate members are allowed
3. Mutable object

Eg:

```
Salary = [30000,40000,50000]
Mix = [34,34.7,’66’, ‘Hello’, True]
Lst =list ((3,5,5,9))    # this is a tuple used to create a list
```

i. Fetch using indexing

Example

```
[In]      List = [2,3,4]
          List[0]
[out] 2
```

ii. Fetch using negative indexing

Example

```
[In]      List = [2,3,4]
          List[-1]
[out] 4
```

iii. Fetch list within a list

Example

```
[In]      List = [2,3,4,['Hello','World']]
          List[-1][0]
[out] 'Hello'
```

iv. Fetch using slicing

Example

```
[In]      List = [2,3,4,['Hello','World']]
          List[0:2]
[out]     [2, 3]
```

v. Fetch using negative slicing

Example

```
 [In]       List = [2,3,4,['Hello','World']]
            List[-4:-2]

 [out]      [2, 3]
```

vi. Fetching and concatenation using string

Example

```
[In]
[out]
```

vii. Replacing an items in the list

Example

```
[In]     List = [2,3,4,['Hello','World']]
         List[-1][-1]='Java'
         print(List)
[out]    [2, 3, 4, ['Hello', 'Java']]
```

viii. Concatenation and Replication

Example

```
[In]     l1=[4,'7']
         l2=['3',2]
         print(l1+l2)
         print(l1*2)
[out]    [4, '7', '3', 2]
         [4, '7', 4, '7']
```

ix. Adding new elements to a existing list

a. Append

Example

```
[In]     list=[200,300]
         list.append(500)
         list.append('Python')
         list.append(89.5)
         print(list)
[out]    [200, 300, 500, 'Python', 89.5]
```

b. Extend

Example

```
[In]     list=[200,300]
         list.extend([500])
         list.extend('Python')
         list.extend([89.5])
         print(list)
```

```
[out]   [200, 300, 500, 'P', 'y', 't', 'h', 'o', 'n', 89.5]
```

c. Insert

Example

```
[In]    list=[200,300]
        list.insert(1,5)
        print(list)
[out]   [200, 5, 300]
```

x. Removing from a list

a. Pop()

Removes an item from an index position

Example

```
[In]    list=[200,300,500]
        list.pop(-1)
        print(list)
[out]   [200, 300]
```

b. Remove()

Removes an item of the first occurrence

Example

```
[In]    list=[200,300,500,300,500]
        list.remove(500)
        print(list)
[out]   [200, 300, 300, 500]
```

c. Replace()

Removes an item of the first occurrence

Example

```
[In]    list=[200,300,500,300,500]
        list[3] = 600
        print(list)
[out]   [200, 300, 500, 600, 500]
```

xi. Adding 2 lists using for loops

Example

```
[In]    l1=[2,3,7,9,2,1]
        l2=[20,89,30,12,45,45]
        new_list=[]
        for i in range(0,len(l1)):
          for j in range(0,len(l2)):
            if i==j:
                new_list.append(l1[i]+l2[j])
        print(new_list)
[out]   [22, 92, 37, 21, 47, 46]
```

xii. Adding 2 lists using zip function

Zip function takes the first value from the list

Example

```
[In]    l1=[2,3,7,9,2,1]
        l2=[20,89,30,12,45,45]
        new_list=[]
        for i,j in zip(l1,l2):
           new_list.append(i+j)
        new_list
[out]   [22, 92, 37, 21, 47, 46]
```

```
a. ['A', 1, 4.5]
b. ['A', 1, 4.5]
c. ['A', 1, 4.5]
d. ['A', 1, 4.5, 'x']
```

e. Copying by reference

Example

```
[In]    l1=['A',1,4.5]
        l2=l1
        print(l1)
        print(l2)
        l2.append('x')
        print(l1)
        print(l2)
[out]   ['A', 1, 4.5]
        ['A', 1, 4.5]
        ['A', 1, 4.5, 'x']
        ['A', 1, 4.5, 'x']
```

f. Copying by value

Example

```
[In]    l1=['A',1,4.5]
        l2=l1.copy()
        print(l1)
        print(l2)
        l2.append('x')
        print(l1)
        print(l2)
[out]   ['A', 1, 4.5]
        ['A', 1, 4.5]
        ['A', 1, 4.5]
        ['A', 1, 4.5, 'x']
```

xiii. Counting the number of variables in a list

Use count() method

xiv. Reversing a list

Use reverse() method

xv. Sorting a list

Use sort() method

b) List Comprehension

- Are elegant form of for loops for list manipulation
- Returns a list object
- Faster than conventional loop for working on big data

Syntax

[output_expression for control_varable in (conditional statements if any)]

Example

[In]

```
# int_lst =[]
# for i in range(1,11):
#        int_lst.append(i)
# print(int_lst)

# instead of the above we can use list comprehension

# without conditional statements
int_lst2 = [i for i in range(1,11)]
print(int_lst2)

# adding 10 without conditional statements
int_lst2 = [i+10 for i in range(1,11)]
print(int_lst2)

# Even numbers with conditional statements
int_lst3 = [i for i in range(1,11) if item%2 ==0]
print(int_lst3)
```

[out]

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

Example

[In]

```
# e_lst=[]
# for i in range(-10,11,1):
#    if i>0:
#        e_lst.append('p')
#    elif i<0:
```

```
#       e_lst.append('N')
#     else:
#       e_lst.append('Z')
# print(e_lst)

# in the above example the code is written to filter out the original data. So conditional statement
should be written at the conditional statements

# in the below example the output is generated to change the value of each of input ie no, of inputs is
always equal to outputs. So conditional statements should be part of the output expression
l_comp= ['P' if i>0 else 'N' if i<0 else 'Z' for i in range(-10,11,1)]
print(l_comp)
```

[out]

```
['N', 'N', 'N', 'N', 'N', 'N', 'N', 'N', 'N', 'N', 'Z', 'p', 'p', 'p', 'p',
'p', 'p', 'p', 'p', 'p', 'p']
```

c) List Generator

- Are lazy evaluator slower than list comprehension
- Returns an iterator object (single value)
- Take less memory for the given for working on big data

Syntax- same as list comprehension but only the difference is instead of [] use ()

(output_expression for control_varable in (conditional statements if any))

Example

[In]

```
g = list((e+10 for e in range(1,11)))   # this does not occupy any memory space
g
```

[out]

```
[11, 12, 13, 14, 15, 16, 17, 18, 19, 20]
```

d) tuple

Definition: Collection of values separated by comma enclosed in round bracket

- an item of a tuple can be a python supported data structure.
- Properties of tuple
  - Ordered and Indexable heterogeneous data structure
  - Duplicate members are allowed
  - Immutable object (values in a tuple cannot be modified.
- We can create a tuple by () or calling a method tuple()
- Methods available for tuple are only count and index

Example

[In]

```
t= (3,4,4,5,6,7,2,1,(44,55),['A','B',('AA','BB','CC')])
print(t[0])
print(t[-1])
print(t[-1][-1])
print(t[-1][-1][0])
print(t[-2][-1])
print(t[-2][-2])
print(t[3])
```

```
        print(t[0:3])   # slcing on a tupke always gives the output in a tupleprint(l2)
[out]   3
        ['A', 'B', ('AA', 'BB', 'CC')]
        ('AA', 'BB', 'CC')
        AA
        55
        44
        5
        (3, 4, 4)
```

e) Dictionary

Definition:

Syntax: {key1:value1, key2:value2 …. Keyn:valuen}

Where:

Keys can be any python supported constant (fixed)

Values can be any python supported data structure

Properties:

- Unordered and Indexable; heterogeneous data structure
- Duplicate values are allowed; duplicate keys are not allowed
- Mutable object

Example

```
[In]    d= {'A':1,'B':2}
        print(type(d))
        print(d)
[out]   <class 'dict'>
        {'A': 1, 'B': 2}
```

Example

```
[In]    name_lst =['A','B','C','D']
        age_lst=[23,45,22,34]
        salary_lst=['23k','45k','56k','67k']

        emp={'Name':name_lst,
            'Age':age_lst,
            'Salary':salary_lst}
        Emp

[out]   {'Name': ['A', 'B', 'C', 'D'],
         'Age': [23, 45, 22, 34],
         'Salary': ['23k', '45k', '56k', '67k']}
```

i. Accessing a value

Example

```
[In]    name_lst =['A','B','C','D']
        age_lst=[23,45,22,34]
```

```
salary_lst=['23k','45k','56k','67k']

emp={'Name':name_lst,
    'Age':age_lst,
    'Salary':salary_lst}
print(emp)

emp['Name'][0]
```

[out]

```
'A'
```

ii. Modifying a value

Example

[In]

```
name_lst =['A','B','C','D']
age_lst=[23,45,22,34]
salary_lst=['23k','45k','56k','67k']

emp={'Name':name_lst,
    'Age':age_lst,
    'Salary':salary_lst}

emp['Name'][0]='X'
print(emp)
```

[out]

```
{'Name': ['X', 'B', 'C', 'D'], 'Age': [23, 45, 22, 34], 'Salary': ['23k',
'45k', '56k', '67k']}
```

iii. Adding new set of values and keys

Example

[In]

```
name_lst =['A','B','C','D']
age_lst=[23,45,22,34]
salary_lst=['23k','45k','56k','67k']

emp={'Name':name_lst,
    'Age':age_lst,
    'Salary':salary_lst}

emp['city']=('bangalore','chennai','x','y','z')
emp
```

[out]

```
{'Name': ['A', 'B', 'C', 'D'],
 'Age': [23, 45, 22, 34],
 'Salary': ['23k', '45k', '56k', '67k'],
 'city': ('bangalore', 'chennai', 'x', 'y', 'z')}
```

iv. For loops

Example

[In]

```
name_lst =['A','B','C','D']
age_lst=[23,45,22,34]
salary_lst=['23k','45k','56k','67k']

emp={'Name':name_lst,
    'Age':age_lst,
    'Salary':salary_lst}

emp['city']=('bangalore','chennai','x','y','z')

for key in emp.keys():
   print(key)

for value in emp.values():
   print(value)
```

[out]

```
Name
Age
Salary
city
['A', 'B', 'C', 'D']
[23, 45, 22, 34]
['23k', '45k', '56k', '67k']
('bangalore', 'chennai', 'x', 'y', 'z')
```

f) Set

Definition:

Syntax: {V1,V2,V3…Vn}

Where:

Values can be any python supported data structure

Properties:

- Unordered and Indexable; heterogeneous data structure
- Duplicate values are not allowed
- Mutable object

Example

[In]

```
print(type({}))
print(type({'a':1}))
print(type({'a'}))
```

[out]

```
<class 'dict'>
<class 'dict'>
<class 'set'>
```

i. Intersection for extracting common elements from 2 lists

Example—extracting common elements

[In]

```
l1 = [1,2,4,7,8,9,1,3,4]
l2 = [2,5,6,1,1,4,1,4]
s1 = set(l1)
s2 = set(l2)
s1.intersection(s2)
```

[out]

```
{1, 2, 4}
```

Other functions can be checked by pressing ‘tab’

14. Functions

- A function is a block of organized, reusable code that is used to perform a single, related action.
- Following are the different type of function
    - Built In
        - e.g.: print
    - User-defined
        - Named function
            - Non-parametric
            - Parametric
                - Default parameters
                - User defined parameters
        - Lambda function

Syntax:

```
Def <name_function>(parameter1, parameter2, parameter):
        <tasks>
        Return <values>
```

Syntax to load function from another file

From <file> **import** <function name>

Function with no return or parameter

Example

```
[In]      def greet():
            print('Hello Python')

          greet()
          greet()
[out]     Hello Python
          Hello Python
```

Function with no return but with parameter

Example

```
[In]      def greet_msg(msg):
            print('Hello'+msg)

          greet_msg(' python')
          greet_msg(' world')
[out]     Hello python
          Hello world
```

Function with no return but with parameter

Example

```
[In]      def greet_msg(msg):
            print('Hello'+msg)

          greet_msg(' python')
          greet_msg(' world')
[out]     Hello python
```

```
Hello world
```

Positional Arguments

a) Functions with Dynamic inputs

i. Non-key worded arguments def <function (* param)

Example

[In]

```
def add(* agr):
    '''This function adds all inputs'''
    return sum(agr)

print(add())
print(add(1))
print(add(1,2))
print(add(3,4,5))
print(add(6,3,2, 5, 1))
```

[out]

```
0
1
3
12
17
```

ii. With key worded arguments def function (** param)

Example

[In]

```
# n -> students
# m -> subjects

# WAF to compute average of scores for n students with m subjects.
lst = []
def get_avg (** students):
#    print(students)
    for name, scores in students.items():
        lst.append({name: sum(scores)/len(scores)})

    return lst

get_avg(Peter = [30,20,30], Jack = [10,20,30], Jill = [40,20,30])
```

[out]

```
[{'Peter': 26.666666666666668}, {'Jack': 20.0}, {'Jill': 30.0}]
```

b) Lambda Function or Lamda Expressions

- It’s called anonymous function or Lambda expression

Syntax:

Lambda argument(s) : output _expression
Where argument(s) should be comma separated

Example

[In]
```
l = lambda n1,n2 : n1+n2
l(40,30)
```

[out]
```
70
```

Example – In the below example we can fix the parameters a,b,c and give the flexibility to the user say provide only different values for x

[In]
```
def calculate (a,b,c):
   return lambda x: a*x**2 + b*x + c  # ax^2+bx+c

f=calculate (2,1,-5)

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

[out]
```
-2
5
16
31
50
73
100
131
166
205
```

iii. Lamda with dynamic arguments

Example

[In]
```
l=lambda *n : sum(n)
print(l(1))
print(l(1,5))
```

[out]
```
1
6
```

c) Filter (), map() and reduce()

i. Filter()

- This is slower than the list comprehension function
- So while doing computation in big data it is suggested in scenarios where memory/computation speed is less

Syntax:

filter (function, iterable)

Example

```
[In]     lst= [2,5,8,7,1,10,3,5,7,18,9]

         print (list(filter(lambda x: x%2 ==0,lst)))
         print (list(filter(lambda x: x%2 !=0,lst)))   # here x is each value in the list
[out]    [2, 8, 10, 18]
         [5, 7, 1, 3, 5, 7, 9]
```

ii. map()

- is used for transformation operation like female for F, N for negative etc.

Syntax:

map (function, iterable)

Example

```
[In]     lst= [2,5,8,7,1,10,3,5,7,18,9]

         # generate a list of values I:
         # I = 2*I if I is even
         # I= 5*I if I is odd

         list (map(lambda x: 2*x if x%2 ==0 else 5*x, lst))
[out]    [4, 25, 16, 35, 5, 20, 15, 25, 35, 36, 45]
```

Example

```
[In]
[out]
```

iii. reduce()

from functools **import** reduce   # To import this function the first time

-

Syntax:

reduce (function, iterable, initialization)

Example without initialization

```
[In]     lst= [4,1,8,9]  ## add list elements
         reduce(lambda x,y : x+y, lst)
[out]    22
```

Example with an initialized value

```
[In]     lst= [4,1,8,9]
         reduce(lambda x,y : x+y, lst,2)
[out]    24
```

15. Regular Expressions- Search(), Match(), sub(), findall(), split()

Import re # this is a package that should be downloaded

a) Metacharacters

We have to know how to use metacharacters to find the logic in the below regular expression usage

b) Search()

Example

```
[In]     ## re.search()
         pat = 'Apples'
         text = "all apples are red"
         if re.search(pattern=pat, string= text, flags = re.I):
            print('Pattern Matched')
         else:
            print('Pattern not Matched')
[out]    Pattern Matched
```

Example

```
[In]     ## re.search()
         pat = 'Apples'
         text = "all apples are red"
         if re.search(pattern=pat, string= text, flags = re.I):
            print('Pattern Matched')
         else:
            print('Pattern not Matched')
[out]    Pattern Matched
```

Example

```
[In]     ## re.search()
         pat = 'Apples'
         text = "all apples are red"
         if re.search(pattern=pat, string= text, flags = re.I):
            print('Pattern Matched')
         else:
            print('Pattern not Matched')
[out]    Pattern Matched
```

Example

```
[In]     numbers = ['9916179812',
         '99161798121',
         '991617981',
         '991617981a']

         pattern = '^\d{10}$' ### pattern -  <10 digits number>
         for phone in numbers:
```

```
    if re.search(pattern=pattern, string=phone):
        print(phone,'=>','Valid Number')
    else:
        print(phone,'=>','InValid Number')
```

[out]

```
9916179812 => Valid Number
99161798121 => InValid Number
991617981 => InValid Number
991617981a => InValid Number
```

Example

[In]

```
numbers = ['+91-9916179812','+91 9916179812','+19 9916179812', '91-9916179812','0091-
9916179812', '(+91)99161798121', '991617981','991617981a']
pattern = '^(\+91|0091)(\s|-)\d{10}$' ### pattern-  +91/0091(-|space)<10 digits number>
for phone in numbers:
    if re.search(pattern=pattern, string=phone):
        print(phone,'=>','Valid Number')
    else:
        print(phone,'=>','InValid Number')
```

[out]

```
9916179812 => Valid Number
99161798121 => InValid Number
991617981 => InValid Number
991617981a => InValid Number
```

Example

[In]

```
emails = ['abc@gmail.com','Abc@gmail.com', 'abc@gmail.comm', 'abc1@gmail.com',
'@gmail.com','abc.xyz@gmail.com']

pattern = '^[a-zA-Z]+@(gmail.com)$' # pattern - (username)@gmail.com  where username should only
albhabets/letters (ignoring case)
for email in emails:
    if re.search(pattern=pattern, string=email):
        print(email,'=>','Valid Email')
    else:
        print(email,'=>','InValid Email')
```

[out]

```
abc@gmail.com => Valid Email
Abc@gmail.com => Valid Email
abc@gmail.comm => InValid Email
abc1@gmail.com => InValid Email
@gmail.com => InValid Email
abc.xyz@gmail.com => InValid Email
```

Example

[In]

```
emails = ['abc@gmail.com','Abc@gmail.com','abc123@gmail.com',
'abc@gmail.comm','1abc@gmail.com', 'abc1@gmail.com', '@gmail.com','abc.xyz@gmail.com']

pattern = '^[a-zA-Z]+[1-9]*@(gmail.com)$' # pattern - (username)@gmail.com  where username should
alphnumeric (ignoring case)
for email in emails:
    if re.search(pattern=pattern, string=email):
        print(email,'=>','Valid Email')
    else:
        print(email,'=>','InValid Email')
```

[out]

```
abc@gmail.com => Valid Email
Abc@gmail.com => Valid Email
abc123@gmail.com => Valid Email
abc@gmail.comm => InValid Email
1abc@gmail.com => InValid Email
abc1@gmail.com => Valid Email
@gmail.com => InValid Email
abc.xyz@gmail.com => InValid Email
```

Example

[In]

```
emails = ['abc@gmail.com','Abc@gmail.com','abc.xyz@gmail.com','abc__xyz@gmail.com','abc_xyz@gmail.com','abc@gmail.com','abc123@gmail.com', 'abc@gmail.comm','1abc@gmail.com', 'abc1@gmail.com', '@gmail.com','abc.xyz@gmail.com']

pattern = '^([a-zA-Z]+\s?\.?_?[1-9]*[a-zA-Z]*)@(gmail.com)$' # pattern - abc(space|.|_)xyz@gmail.com where username should alphnumeric (ignoring case)
for email in emails:
    if re.search(pattern=pattern, string=email):
        print(email,'=>','Valid Email')
#       username = re.search(pattern=pattern, string=email).groups()[0]
    else:
        print(email,'=>','InValid Email')
```

[out]

```
abc@gmail.com => Valid Email
Abc@gmail.com => Valid Email
abc.xyz@gmail.com => Valid Email
abc__xyz@gmail.com => InValid Email
abc_xyz@gmail.com => Valid Email
abc@gmail.com => Valid Email
abc123@gmail.com => Valid Email
abc@gmail.comm => InValid Email
1abc@gmail.com => InValid Email
abc1@gmail.com => Valid Email
@gmail.com => InValid Email
abc.xyz@gmail.com => Valid Email
```

c) Match()

Search, searches any word within a string

Whereas match searches the pattern in the first substring before the first white space.

Example

[In]

```
## re.match()

pat = 'apples'
text = "all apples are red"
if re.match(pattern=pat, string= text):
    print('Pattern Matched')
else:
    print('Pattern not Matched')
```

[out]

```
Pattern not Matched
```

d) sub()

This is used to replace a substring within a string

Example

```
[In]      text = "all black and blue cars are expensive"
          ## replace black with 'green'

          re.sub(pattern='Black', repl='green', string=text, flags=re.I)

[out]     'all green and blue cars are expensive'
```

Example

```
[In]      text = "one two 3 four 444 4 5555 55 222222 "
          ## replace digit by '*'
          # text.replace('3','*').replace('4','*')

          re.sub(pattern='\d+', repl='*', string=text)

[out]     'one two * four * * * * * '
```

Example

```
[In]      re.sub(pattern='\d', repl='*', string=text)
[out]     'one two * four *** * **** ** ****** '
```

Example

```
[In]      ### replace 3 digits numbers
          re.sub(pattern=r'\b\d\d\d\b', repl='*', string=text)
[out]     'one two 3 four * 4 5555 55 222222 '
```

Example

```
[In]      text = "two two two two two two"
          re.sub(pattern='two', repl='*', string=text,count=2)
[out]     '* * two two two two'
```

e) split()

to split your string based on some patter

Example

```
[In]      ### re.split()
          text = "two two two two two two"
          re.split(pattern  = ' ', string= text)
[out]     ['two', 'two', 'two', 'two', 'two', 'two']
```

Example

```
[In]      text = "two two two 4 two two 5 two"
```

```
        ### split on digit
        re.split(pattern  = '\d', string= text)
[out]   ['two two two ', ' two two ', ' two']
```

f) findall()

Example

```
[In]    text = "apple mango banana orange apple grapes"

        ## extract apple
        re.findall(pattern='apple', string=text)
[out]   ['apple', 'apple']
```

Example

```
[In]    text = "pain gain main pencil grapes mango pune"

        ## extract all words starting with 'p' and of length four [pain, pune]
        re.findall(pattern=r'\bp...\b', string=text)

[out]   ['pain', 'pune']
```

Example

```
[In]    ## extract all words of length four [pain, pune, hain, main]
        re.findall(pattern=r'\b....\b', string=text)
[out]   ['pain', 'gain', 'main', 'pune']
```

Example

```
[In]    text = "pain gain main pencil grapes mango pune"
        re.findall(pattern=r'\b.ain\b', string=text)

[out]   ['pain', 'gain', 'main']
```

16. File IO Operations

These File IO operations are mainly used for logging errors on files.

```
f=open(‘log.txt’,mode= ‘w’)
f= writelines([msg1,msg2,msg3..etc])
f.close()
```

a) Opening and closing a file

```
f=open(‘log.txt’,mode= ‘w’)
f.close()
```

b) Writing to a file

Below are the 2 ways of writing to a file

- write()
- writelines()

```
f.write(“This is error”)
f.write(msg1+msg2)
f= writelines([msg1,msg2,msg3..etc])
```

c) appending to a file

Below are the 2 ways of writing to a file

- append()

Open the file in Append mode and then write

```
f=open(‘log.txt’,mode= ‘a’)
f.write(“This is error”)
f.write(msg1+msg2)
f.close()
```

d) Reading from a file

Below are the 3 ways of writing to a file

- read()
- readline()
- readlines()

```
f=open(‘log.txt’,mode= ‘r’)
print(file.read(10))          # 10 is used to read only the 10 chars. If blank it reds everything
print(file.readline())        # read the contents of the first line
print(file.readline())        # read the contents of the second line
f.close()
```

```
f=open(‘log.txt’,mode= ‘r’)
print(file.readlines()[2:3])          # read the contents of the 2nd and 3rd lines
f.close()
```

17. Exception Handling and File IO Operations

Handling the run time error so that the first part of a logic doesn’t affect the remaining code execution.

We can find more types of exceptions in the below link
https://docs.python.org/3/library/exceptions.html#base-classes

a) try:

Any code line that can raise an exception

b) except

what to do when an exception

c) else

what to do when No exception

d) finally

any code that needs to be executed irrespective of any exception

Note:

- else and finally blocks are optional
- try should be before except block

Example without error

```
[In]    try:
           n1,n2 = [int(x) for x in input('Enter 2 number:
        ').split(' ')]
           print('sum of {} and {} = {}'.format(n1,n2,n1+n2))
        except:
           print('An error occured')
        else:
           print('Addition successful')
        finally:
           print('End of Program')
[out]   Enter 2 number: 2 3
        sum of 2 and 3 = 5
        Addition successful
        End of Program
```

Example without error

```
[In]    try:
           n1,n2 = [int(x) for x in input('Enter 2 number:
        ').split(' ')]
           print('sum of {} and {} = {}'.format(n1,n2,n1+n2))
        except:
           print('An error occured')
```

```
else:
   print('Addition successful')
finally:
   print('End of Program')
```

[out]

```
Enter 2 number: 23
An error occured
End of Program
```

e) Using Multiple except

Example

[In]

```
try:
   n=input('Enter a value')
   print(int(n))
   print (4/int(n))
except ValueError:              # this code prints the error message we are providing
   print('Invalid Input for int type casting')
except ZeroDivisionError as e:  # this code print the error description of the actual error
   print(e)
except
   print('Generic Error')
```

[out]

```
Enter a value: a
Invalid Input for int type casting

Enter a value: 0
0
division by zero
```

f) Exception in a defined function

Example

[In]

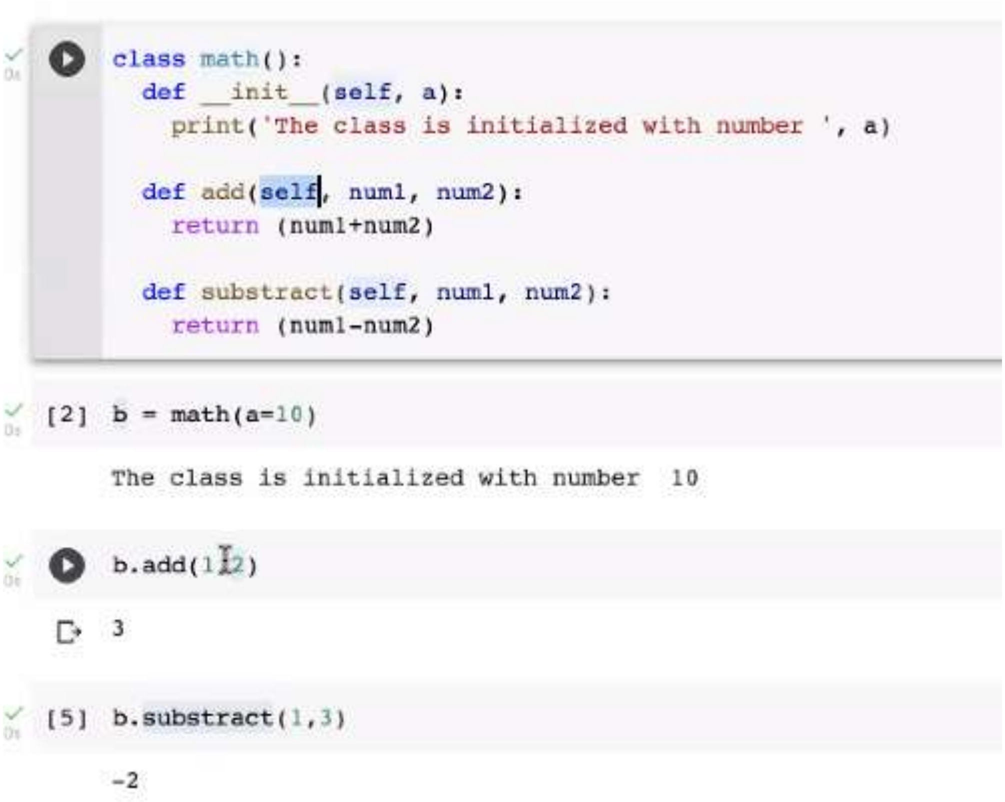
```
def add(n1,n2):
   try:
      return n1+n2
   except:
      return "error occured"

print(add(3,'4'))
print(add(3,4))
```

[out]

```
error occured
7
```

18. Classes



```
class math():
  def __init__(self, a):
    print('The class is initialized with number ', a)

  def add(self, num1, num2):
    return (num1+num2)

  def substract(self, num1, num2):
    return (num1-num2)
```

```
[2] b = math(a=10)
```

```
The class is initialized with number  10
```

```
b.add(1,2)
```

```
3
```

```
[5] b.substract(1,3)
```

```
-2
```

19. Numpy

Is mainly used for Data Analysis

Import numpy as np

a) Arrays in Numpy

Syntax=
Np.array ([["content1"]

```
# creating array object
arr=np.array ([[1,2,3],
            [4,2,3]])

# printing type of arr object
print ("Array is of type: ", type(arr))

# printing array dimensions (axes)
print("No. of dmensions: ", arr.ndim)

# printing shape of array
print("shape of array: ", arr.shape)

# printing size (total number of elements) of array
print("size of array: ", arr.size)

# printing type of elements in array
print("array stores elements of type: ", arr.dtype)
```

b) Arrays creation

i. typecasting in array

Example

```
[In]      b= np.array([[1,2,4],
                  [5,8,7]],dtype='float')
          b
[out]     array([[1., 2., 4.],
                 [5., 8., 7.]])
```

ii. creating array from tuple

Example

```
[In]      c= np.array((1,2,3))
          c
[out]     array([1, 2, 3])
```

iii. creating a 3x4 array with all zeros and ones

Example

```
[In]     e=np.zeros((2,3))
         f=np.ones((2,3))
         print(e)
         print(f)
[out]    [[0.  0.  0.]
          [0.  0.  0.]]
         [[1.  1.  1.]
          [1.  1.  1.]]
```

iv. create a constant value array of complex type

Example

```
[In]     d= np.full((3,3),6,dtype='complex')
         print(d)
[out]    [[6.+0.j  6.+0.j]
          [6.+0.j  6.+0.j  6.+0.j]
          [6.+0.j  6.+0.j  6.+0.j]]
```

v. create an array with random values

Example

```
[In]     e= np.random.random((2,2))
         e
[out]    array([[0.43313833, 0.47476284],
                [0.57883009, 0.22495578]])
```

vi. create a sequence from 1 to b with a step size of x

Example

```
[In]     f=np.arange(0,30,5)
         print(f)
[out]    [ 0  5 10 15 20 25]
```

vii. create a sequence of x in range a to b

Example

```
[In]     g=np.linspace(0,30,5)
         print(g)
[out]    [ 0.   7.5 15.  22.5 30. ]
```

viii. reshaping 3x4 arrray to 2x2x3

Example

```
[In]     arr=np.array([[1,2,3,4],
                 [5,2,4,2],
          [1,2,0,1]])

newarr=arr.reshape(2,2,3)
newarr1=arr.reshape(2,3,2)
print(newarr)
print("------------------------")
print(newarr1)
```

[out]

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

 [[4 2 1]
  [2 0 1]]]
------------------------
[[[1 2]
  [3 4]
  [5 2]]

 [[4 2]
  [1 2]
  [0 1]]]
```

ix. Flatten Array

Example

[In]

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

[out]

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

c) Arrays Indexing

i. Slicing

Example

[In]

```
m = np.array([[-1,2,0,4],
      [4,-0.5,6,0],
      [2.6,0,7,8],
      [3,-7,4,2.0]])
sa=m[:2,:2]
sm=m[:2, :4:2]   # with a step size of 2
print(sa)
print('------')
print(sm)
```

[out]

```
[[-1.   2. ]
 [ 4.  -0.5]]
------
[[-1.  0.]
 [ 4.  6.]]
```

ii. Integer array indexing

Example

```
[In]     m = np.array([[-1,2,0,4],
                 [4,-0.5,6,0],
                 [2.6,0,7,8],
                 [3,-7,4,2.0]])
         im=m[[0,1,2,3],[3,2,1,0]]    # 1st list will take elements of row and 2nd list the elements of column
         im
[out]    array([4., 6., 0., 3.])
```

iii. Boolean array indexing

It will only take the values with reference to the operator being used

Example

```
[In]     m = np.array([[-1,2,0,4],
                 [4,-0.5,6,0],
                 [2.6,0,7,8],
                 [3,-7,4,2.0]])
         cond=m>2
         bim=m[cond]           # passing the condition to a list
         bim
[out]    array([4. , 4. , 6. , 2.6, 7. , 8. , 3. , 4. ])
```

d) Arrays Operations

i. Basic operation on a single array

It can be done for all operations +,-,*,/

Example

```
[In]     a=np.array([1,2,3,4])
         a=a+1
         a
[out]    array([2, 3, 4, 5])
```

ii. Transpose of an array

Example

```
[In]     m = np.array([[-1,2,0,4],
                 [4,-0.5,6,0],
                 [2.6,0,7,8],
                 [3,-7,4,2.0]])
         m=m.T
         m
[out]    array([[-1. ,  4. ,  2.6,  3. ],
        [ 2. , -0.5,  0. , -7. ],
        [ 0. ,  6. ,  7. ,  4. ],
        [ 4. ,  0. ,  8. ,  2. ]])
```

e) Unary Operators

Example

[In]

```
m = np.array([[-1,2,0,4],
              [4,-0.5,6,0],
              [2.6,0,7,8],
              [3,-7,4,2.0]])

print(m.max())
print('------')
print(m.max(axis=1))    # hint: there are only 2 axis in case of a 2 dimnetional array that 0 and 1
print('------')
print(m.max(axis=0))    # hint: there are only 2 axis in case of a 2 dimnetional array that 0 and 1
print('------')
print(m.min())
print('------')
print(m.min(axis=1))
print('------')
print(m.sum())
print('------')
print(m.sum(axis=0))
print('------')
print(m.cumsum())   # it takes the cumulative sum
print('------')
print(m.cumprod())   # it takes the cumulative sum
```

[out]

```
8.0
------
[4. 6. 8. 4.]
------
[4. 2. 7. 8.]
------
-7.0
------
[-1.  -0.5  0.  -7. ]
------
34.1
------
[ 8.6 -5.5 17.  14. ]
------
[-1.   1.   1.   5.   9.   8.5 14.5 14.5 17.1 17.1 24.1 32.1 35.1 28.1
 32.1 34.1]
------
[-1. -2. -0. -0. -0.  0.  0.  0.  0.  0.  0.  0.  0. -0. -0. -0.]
```

f) Random Array Generator

Syntax:

Out_arr = np.random.randint(low,high,size)

i. Normal random

Example

[In] p=np.random.random((2,3)) # this is random array values ranginf from 0 and 1
p

[out]
```
array([[0.40201091, 0.83372057, 0.69683849],
       [0.64137693, 0.52395948, 0.06359931]])
```

ii. Random of an array

Example

[In] p=np.random.randint(0,30,(2,3)) # this gives a 2d array with the shape given for values between 0 and 30
p

[out]
```
array([[29, 18, 21],
       [10, 27,  3]])
```

Similarly, for 2d array we can give 3d and 4d arrays

Example

[In] p=np.random.randint(0,30,(2,3,2)) # this gives a 2d array with the shape gven for values between 0 and 30
p

[out]
```
array([[[29,  3],
        [28,  2],
        [ 0, 21]],

       [[ 9,  7],
        [10, 20],
        [ 6, 14]]])
```

iii. Random within a range with uniform function

Example

[In] p=np.random.uniform(7,30)
p

[out]
```
28.714452263203622
```

iv. Random within a range with uniform function and rounding

Example

[In] p=round(np.random.uniform(7,30),2)
p

[out]
```
12.77
```

v. Random with choice

Choose a value from the array

Example

```
[In]     p=np.random.choice([1,5,26,7])
         p
[out]    26
```

vi. Random with choice with an array

Example

```
[In]     p=np.random.choice([1,5,2,6,7], (2,3))
         p
[out]    array([[1, 7, 1],
                [6, 2, 2]])
```

g) Binary Operations between arrays

i. Addition

Example

```
[In]     a = np.array([[1,2],
                   [3,4]])
         b = np.array([[4,3],
                   [1,2]])
         c=a+b
         c
[out]    array([[5, 5],
                [4, 6]])
```

ii. Multiplication

Example

```
[In]     a = np.array([[1,2],
                   [3,4]])
         b = np.array([[4,3],
                   [1,2]])
         c=a*b
         c
[out]    array([[4, 6],
                [3, 8]])
```

iii. Matrix Multiplication

Example

```
[In]     a = np.array([[1,2],
                       [3,4]])
         b = np.array([[4,3],
                        [1,2]])
         c=a.dot(b)
         c
[out]    array([[ 6,  7],
                [16, 17]])
```

h) Universal functions (ufunc)

i. sin, cos, tan

Example

```
[In]     a = np.array([[1,2],
                       [3,4]])
         print(np.sin(a))
         print('------')
         print(np.cos(a))
         print('------')
         print(np.tan(a))
[out]    [[ 0.84147098  0.90929743]
          [ 0.14112001 -0.7568025 ]]
         ------
         [[ 0.54030231 -0.41614684]
          [-0.9899925  -0.65364362]]
         ------
         [[ 1.55740772 -2.18503986]
          [-0.14254654  1.15782128]]
```

ii. exponential values

Example

```
[In]     a = np.array([[1,2],
                       [3,4]])
         np.exp(a)   # e raised to the power
[out]    array([[ 2.71828183,  7.3890561 ],
                [20.08553692, 54.59815003]])
```

iii. square root

Example

```
[In]     a = np.array([[1,2],
                       [3,4]])
         np.sqrt(a)
[out]    array([[1.        , 1.41421356],
                [1.73205081, 2.        ]])
```

iv. Power

Example

[In]
```
a = np.array([[1,2],
          [3,4]])
np.power(a,3))
```

[out]
```
array([[ 1,  8],
       [27, 64]], dtype=int32)
```

v. Power

np.pi

it gives the pi value

i) Sorting Array

i. default

Example

[In]
```
q=np.array([[1,4,2],
         [3,4,6],
         [0,-1,5]])
np.sort(q)
```

[out]
```
array([[ 1,  2,  4],
       [ 3,  4,  6],
       [-1,  0,  5]])
```

ii. sort with axis value

Example

[In]
```
q=np.array([[1,4,2],
         [3,4,6],
         [0,-1,5]])
np.sort(q,axis=0)
```

[out]
```
array([[ 0, -1,  2],
       [ 1,  4,  5],
       [ 3,  4,  6]])
```

iii. sort with mergesort

Example

[In]
```
q=np.array([[1,4,2],
         [3,4,6],
         [0,-1,5]])
np.sort(q,axis=0, kind='mergesort')
```

[out]
```
array([[ 0, -1,  2],
       [ 1,  4,  5],
       [ 3,  4,  6]])
```

iv. sort with structured array

Example

[In]

```
d = [('name', 'S10'),('grad',int),('cgpa',float)]

values = [('Hritik', 2009, 8.5), ('Ajay', 2008, 8.7),
        ('Pankaj', 2008,7.9)]

s=np.array(values,dtype=d)
print(s)
print('------')
# to sort this as per the name
print(np.sort(s,order='name'))
print('------')
print(np.sort(s,order='cgpa'))
print('------')
print(np.sort(s,order=['grad','cgpa'])) # sort as per graduation year and then by cgpa
```

[out]

```
[(b'Hritik', 2009, 8.5) (b'Ajay', 2008, 8.7) (b'Pankaj', 2008, 7.9)]
------
[(b'Ajay', 2008, 8.7) (b'Hritik', 2009, 8.5) (b'Pankaj', 2008, 7.9)]
------
[(b'Pankaj', 2008, 7.9) (b'Hritik', 2009, 8.5) (b'Ajay', 2008, 8.7)]
------
[(b'Pankaj', 2008, 7.9) (b'Ajay', 2008, 8.7) (b'Hritik', 2009, 8.5)]
```

20. Pandas

Is mainly used for Data Analysis and is one of the fastest.

Pandas is a fast, powerful, flexible and easy to use open source data analysis and manipulation tool, built on top of the Python programming language.

It provides highly optimized performance with back-end source code is purely written in C or Python.

Import numpy as np
Import pandas as pd

Example

```
[In]     data = np.array (['a','b','c','d'])
         s=pd.Series(data)
         print(s)
         print(type(s))
[out]    0     a
         1     b
         2     c
         3     d
         dtype: object
         <class 'pandas.core.series.Series'>


         Here the first column is Index and 2nd is value column.
```

a) Data frame

Is a 2d array

Example-Type-1

```
[In]     d1={'a':1,'b':2,'c':3}
         d2={'a':4,'b':5,'c':6}

         d={'first':d1,'second':d2}
         df=pd.DataFrame(d)
         print(df)
[out]       first  second
         a      1       4
         b      2       5
         c      3       6
```

Example-Type-2

```
[In]     d1={'first': [2,5,3,1,6],'second':[7,5,3,1,2]}
         df1=pd.DataFrame(d1)
         print(df1)
[out]       first  second
         0      2       7
         1      5       5
         2      3       3
         3      1       1
         4      6       2
```

Example-Type-3 where there is no key. So column name can be passed with the below method

[In]
```
da=[['Alex',10],['Bob',12],['Clarke',13]]
df3=pd.DataFrame(da,columns=['Name','Age'])
print(df3)
```

[out]
```
     Name  Age
0    Alex   10
1     Bob   12
2  Clarke   13
```

i. Extracting data from csv

Example

[In]
```
# d=pd.read_csv('nba.csv')  # if the file is in the same folder as the jupiter notebook file
d=pd.read_csv('E:/PERSONAL/LEARNING/LearnBay/Python/nba.csv')   # here we copy the location and change the \ to /.
d
```

[out]

	Name	Team	Number	Position	Age	Height	Weight	College	Salary
0	Avery Bradley	Boston Celtics	0.0	PG	25.0	6-2	180.0	Texas	7730337.0
1	Jae Crowder	Boston Celtics	99.0	SF	25.0	6-6	235.0	Marquette	6796117.0
2	John Holland	Boston Celtics	30.0	SG	27.0	6-5	205.0	Boston University	NaN
3	R.J. Hunter	Boston Celtics	28.0	SG	22.0	6-5	185.0	Georgia State	1148640.0
4	Jonas Jerebko	Boston Celtics	8.0	PF	29.0	6-10	231.0	NaN	5000000.0
...	...	...	...	...	...	...	...	...	...
452	Trey Lyles	Utah Jazz	41.0	PF	20.0	6-10	234.0	Kentucky	2239800.0
453	Shelvin Mack	Utah Jazz	8.0	PG	26.0	6-3	203.0	Butler	2433333.0
454	Raul Neto	Utah Jazz	25.0	PG	24.0	6-1	179.0	NaN	900000.0
455	Tibor Pleiss	Utah Jazz	21.0	C	26.0	7-3	256.0	NaN	2900000.0
456	Jeff Withey	Utah Jazz	24.0	C	26.0	7-0	231.0	Kansas	947276.0

Example-with separately assign an existing column as index

[In]
```
d=pd.read_csv('E:/PERSONAL/LEARNING/LearnBay/Python/nba.csv', index_col='Name')   # Here we declare the 'Name as the index'
d
```

[out]

	Team	Number	Position	Age	Height	Weight	College	Salary
Name								
Avery Bradley	Boston Celtics	0.0	PG	25.0	6-2	180.0	Texas	7730337.0
Jae Crowder	Boston Celtics	99.0	SF	25.0	6-6	235.0	Marquette	6796117.0
John Holland	Boston Celtics	30.0	SG	27.0	6-5	205.0	Boston University	NaN
R.J. Hunter	Boston Celtics	28.0	SG	22.0	6-5	185.0	Georgia State	1148640.0
Jonas Jerebko	Boston Celtics	8.0	PF	29.0	6-10	231.0	NaN	5000000.0
...	...	...	...	...	...	...	...	...
Trey Lyles	Utah Jazz	41.0	PF	20.0	6-10	234.0	Kentucky	2239800.0
Shelvin Mack	Utah Jazz	8.0	PG	26.0	6-3	203.0	Butler	2433333.0
Raul Neto	Utah Jazz	25.0	PG	24.0	6-1	179.0	NaN	900000.0
Tibor Pleiss	Utah Jazz	21.0	C	26.0	7-3	256.0	NaN	2900000.0

Jeff Withey	Utah Jazz	24.0	C	26.0	7-0	231.0	Kansas	947276.0

457 rows × 8 columns

ii. Extracting columns

Example-show only the age

[In] d['Age']

[out]

```
Name
Avery Bradley    25.0
Jae Crowder      25.0
John Holland     27.0
R.J. Hunter      22.0
Jonas Jerebko    29.0
                 ...
Trey Lyles       20.0
Shelvin Mack     26.0
Raul Neto        24.0
Tibor Pleiss     26.0
Jeff Withey      26.0
Name: Age, Length: 457, dtype: float64
```

Example-for multiple columns

[In] d[['Age','College','Salary']]   # if no double [[]] is present it will be a single value. So we have to pass it as a list

[out]

	Age	**College**	**Salary**
Name			
Avery Bradley	25.0	Texas	7730337.0
Jae Crowder	25.0	Marquette	6796117.0
John Holland	27.0	Boston University	NaN
R.J. Hunter	22.0	Georgia State	1148640.0
Jonas Jerebko	29.0	NaN	5000000.0
...	...	...	...
Trey Lyles	20.0	Kentucky	2239800.0
Shelvin Mack	26.0	Butler	2433333.0
Raul Neto	24.0	NaN	900000.0
Tibor Pleiss	26.0	NaN	2900000.0
Jeff Withey	26.0	Kansas	947276.0

457 rows × 3 columns

iii. Extracting values of a row

a) loc is a label based on indexes

Example-

[In] d.loc[['Avery Bradley','John Holland']]

[out]

	Team	Number	Position	Age	Height	Weight	College	Salary
Name								
Avery Bradley	Boston Celtics	0.0	PG	25.0	6-2	180.0	Texas	7730337.0
John Holland	Boston Celtics	30.0	SG	27.0	6-5	205.0	Boston University	NaN

b) Iloc is integer based location.

Example-

[In] d1.iloc[3]

[out]

```
Team          Boston Celtics
Number                  28.0
Position                  SG
Age                     22.0
Height                   6-5
Weight                 185.0
College        Georgia State
Salary             1148640.0
Name: R.J. Hunter, dtype: object
```

b) Splitting Data in Pandas

Example-

[In] d=pd.read_csv('E:/PERSONAL/LEARNING/LearnBay/Python/nba.csv') # here we copy the location and change the \ to /.
d

[out]

	Name	Team	Number	Position	Age	Height	Weight	College	Salary
0	Avery Bradley	Boston Celtics	0.0	PG	25.0	6-2	180.0	Texas	7730337.0
1	Jae Crowder	Boston Celtics	99.0	SF	25.0	6-6	235.0	Marquette	6796117.0
2	John Holland	Boston Celtics	30.0	SG	27.0	6-5	205.0	Boston University	NaN
3	R.J. Hunter	Boston Celtics	28.0	SG	22.0	6-5	185.0	Georgia State	1148640.0
4	Jonas Jerebko	Boston Celtics	8.0	PF	29.0	6-10	231.0	NaN	5000000.0
...	...	...	...	...	...	...	...	...	...
452	Trey Lyles	Utah Jazz	41.0	PF	20.0	6-10	234.0	Kentucky	2239800.0
453	Shelvin Mack	Utah Jazz	8.0	PG	26.0	6-3	203.0	Butler	2433333.0
454	Raul Neto	Utah Jazz	25.0	PG	24.0	6-1	179.0	NaN	900000.0
455	Tibor Pleiss	Utah Jazz	21.0	C	26.0	7-3	256.0	NaN	2900000.0
456	Jeff Withey	Utah Jazz	24.0	C	26.0	7-0	231.0	Kansas	947276.0

457 rows × 9 columns

Example-

[In] n = d['Name'].str.split(" ",n=1, expand = True) # where " " is the delimiter n=1 is the number of splits and expand=True is for splitting the outputs into separate columns
n

[out]

	0	1
0	Avery	Bradley

1	Jae	Crowder
2	John	Holland
3	R.J.	Hunter
4	Jonas	Jerebko
...	...	...
452	Trey	Lyles
453	Shelvin	Mack
454	Raul	Neto
455	Tibor	Pleiss
456	Jeff	Withey

457 rows × 2 columns

c) Adding the split values into the original table

Example-

[In] d['First Name']=n[0] # here we are defining the new column as First Name and taking the value from n[0]
d['Second name']=n[1]
d

[out]

	Name	Team	Number	Position	Age	Height	Weight	College	Salary	First Name	Second name
0	Avery Bradley	Boston Celtics	0.0	PG	25.0	6-2	180.0	Texas	7730337.0	Avery	Bradley
1	Jae Crowder	Boston Celtics	99.0	SF	25.0	6-6	235.0	Marquette	6796117.0	Jae	Crowder
2	John Holland	Boston Celtics	30.0	SG	27.0	6-5	205.0	Boston University	NaN	John	Holland
3	R.J. Hunter	Boston Celtics	28.0	SG	22.0	6-5	185.0	Georgia State	1148640.0	R.J.	Hunter
4	Jonas Jerebko	Boston Celtics	8.0	PF	29.0	6-10	231.0	NaN	5000000.0	Jonas	Jerebko
...	...	...	...	...	...	...	...	...	...	...	...
452	Trey Lyles	Utah Jazz	41.0	PF	20.0	6-10	234.0	Kentucky	2239800.0	Trey	Lyles
453	Shelvin Mack	Utah Jazz	8.0	PG	26.0	6-3	203.0	Butler	2433333.0	Shelvin	Mack
454	Raul Neto	Utah Jazz	25.0	PG	24.0	6-1	179.0	NaN	900000.0	Raul	Neto
455	Tibor Pleiss	Utah Jazz	21.0	C	26.0	7-3	256.0	NaN	2900000.0	Tibor	Pleiss
456	Jeff Withey	Utah Jazz	24.0	C	26.0	7-0	231.0	Kansas	947276.0	Jeff	Withey

457 rows × 11 columns

d) Dropping columns

Example-

[In] d.drop(columns=['Name'], inplace=True) # inplace=True will make sure that the column Name is removed from the original d dataframe. it is same x+=1
d

[out]

	Team	Number	Position	Age	Height	Weight	College	Salary	First Name	Second name
0	Boston Celtics	0.0	PG	25.0	6-2	180.0	Texas	7730337.0	Avery	Bradley
1	Boston Celtics	99.0	SF	25.0	6-6	235.0	Marquette	6796117.0	Jae	Crowder
2	Boston Celtics	30.0	SG	27.0	6-5	205.0	Boston University	NaN	John	Holland
3	Boston Celtics	28.0	SG	22.0	6-5	185.0	Georgia State	1148640.0	R.J.	Hunter
4	Boston Celtics	8.0	PF	29.0	6-10	231.0	NaN	5000000.0	Jonas	Jerebko
...	...	...	...	...	...	...	...	...	...	...
452	Utah Jazz	41.0	PF	20.0	6-10	234.0	Kentucky	2239800.0	Trey	Lyles
453	Utah Jazz	8.0	PG	26.0	6-3	203.0	Butler	2433333.0	Shelvin	Mack
454	Utah Jazz	25.0	PG	24.0	6-1	179.0	NaN	900000.0	Raul	Neto
455	Utah Jazz	21.0	C	26.0	7-3	256.0	NaN	2900000.0	Tibor	Pleiss

456	Utah Jazz	24.0	C	26.0	7-0	231.0	Kansas	947276.0	Jeff	Withey

457 rows × 10 columns

e) Dropping rows if any value in a row is Nan

Example-

[In] d.dropna()

[out]

	Team	Number	Position	Age	Height	Weight	College	Salary	First Name	Second name
0	Boston Celtics	0.0	PG	25.0	6-2	180.0	Texas	7730337.0	Avery	Bradley
1	Boston Celtics	99.0	SF	25.0	6-6	235.0	Marquette	6796117.0	Jae	Crowder
3	Boston Celtics	28.0	SG	22.0	6-5	185.0	Georgia State	1148640.0	R.J.	Hunter
6	Boston Celtics	55.0	PF	21.0	6-8	235.0	LSU	1170960.0	Jordan	Mickey
7	Boston Celtics	41.0	C	25.0	7-0	238.0	Gonzaga	2165160.0	Kelly	Olynyk
...	...	...	...	...	...	...	...	...	...	...
449	Utah Jazz	5.0	SG	23.0	6-8	206.0	Duke	1348440.0	Rodney	Hood
451	Utah Jazz	23.0	SF	26.0	6-6	206.0	Dayton	981348.0	Chris	Johnson
452	Utah Jazz	41.0	PF	20.0	6-10	234.0	Kentucky	2239800.0	Trey	Lyles
453	Utah Jazz	8.0	PG	26.0	6-3	203.0	Butler	2433333.0	Shelvin	Mack
456	Utah Jazz	24.0	C	26.0	7-0	231.0	Kansas	947276.0	Jeff	Withey

364 rows × 10 columns

f) Combining Data in Pandas

i. Using Combine

Example-

[In]

```
a=[1,2,5,6,3,7,11,0,4]
b = [5,3,2,1,3,9,21,3,1]

a = pd.Series(a)   # creating the series
print(a)
b = pd.Series(b)
print(b)
result= a.combine(b,(lambda x1,x2: x1 if x1 < x2 else x2))   # this compares each value of the corresponding row between a and b and returns the smaller
result
```

[out]

```
0       1
1       2
2       5
3       6
4       3
5       7
6      11
7       0
8       4
dtype: int64
0       5
1       3
2       2
3       1
4       3
5       9
6      21
7       3
8       1
dtype: int64
0       1
1       2
```

```
2      2
3      1
4      3
5      7
6     11
7      0
8      1
dtype: int64
```

ii. Using concat

Example-

[In]

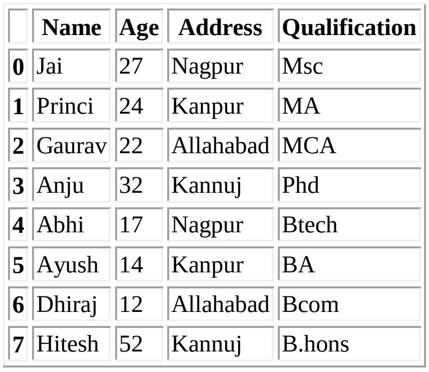
```
data1= {'Name': ['Jai', 'Princi', 'Gaurav', 'Anju'],
      'Age':[27,24,22,32],
      'Address': ['Nagpur','Kanpur', 'Allahabad', 'Kannuj'],
      'Qualification': ['Msc','MA','MCA','Phd']}
data2= {'Name': ['Abhi', 'Ayush', 'Dhiraj', 'Hitesh'],
      'Age':[17,14,12,52],
      'Address': ['Nagpur','Kanpur', 'Allahabad', 'Kannuj'],
      'Qualification': ['Btech','BA','Bcom','B.hons']}


df= pd.DataFrame(data1, index=[0,1,2,3])
df1= pd.DataFrame(data2, index=[4,5,6,7])
print(df, '\n\n', df1)
# frames=[df,df1]
result1=pd.concat([df,df1])
result1
```

[out]

```
     Name  Age    Address Qualification
0     Jai   27     Nagpur           Msc
1  Princi   24     Kanpur            MA
2  Gaurav   22  Allahabad           MCA
3    Anju   32     Kannuj           Phd

       Name  Age    Address Qualification
4     Abhi   17     Nagpur         Btech
5    Ayush   14     Kanpur            BA
6   Dhiraj   12  Allahabad          Bcom
7   Hitesh   52     Kannuj        B.hons
```



	Name	Age	Address	Qualification
0	Jai	27	Nagpur	Msc
1	Princi	24	Kanpur	MA
2	Gaurav	22	Allahabad	MCA
3	Anju	32	Kannuj	Phd
4	Abhi	17	Nagpur	Btech
5	Ayush	14	Kanpur	BA
6	Dhiraj	12	Allahabad	Bcom
7	Hitesh	52	Kannuj	B.hons

iii. Using append

Example-

[In]

```
data1= {'Name': ['Jai', 'Princi', 'Gaurav', 'Anju'],
     'Age':[27,24,22,32],
     'Address': ['Nagpur','Kanpur', 'Allahabad', 'Kannuj'],
     'Qualification': ['Msc','MA','MCA','Phd']}
data2= {'Name': ['Abhi', 'Ayush', 'Dhiraj', 'Hitesh'],
     'Age':[17,14,12,52],
     'Address': ['Nagpur','Kanpur', 'Allahabad', 'Kannuj'],
     'Qualification': ['Btech','BA','Bcom','B.hons']}
r=df.append(df1)
r
```

[out]

	Name	Age	Address	Qualification
0	Jai	27	Nagpur	Msc
1	Princi	24	Kanpur	MA
2	Gaurav	22	Allahabad	MCA
3	Anju	32	Kannuj	Phd
4	Abhi	17	Nagpur	Btech
5	Ayush	14	Kanpur	BA
6	Dhiraj	12	Allahabad	Bcom
7	Hitesh	52	Kannuj	B.hons

g) Filtering Data in Pandas

Example-

[In]

```
x=pd.read_csv('E:/PERSONAL/LEARNING/LearnBay/Python/nba.csv')   # here we copy the location and
change the \ to /.
x
# print(x.loc[x['Age']>25])
x.loc[(x['Age'] == 27) & (x['Salary'] <= 90000)]
```

[out]

	Name	Team	Number	Position	Age	Height	Weight	College	Salary
0	Avery Bradley	Boston Celtics	0.0	PG	25.0	6-2	180.0	Texas	7730337.0
1	Jae Crowder	Boston Celtics	99.0	SF	25.0	6-6	235.0	Marquette	6796117.0
2	John Holland	Boston Celtics	30.0	SG	27.0	6-5	205.0	Boston University	NaN
3	R.J. Hunter	Boston Celtics	28.0	SG	22.0	6-5	185.0	Georgia State	1148640.0
4	Jonas Jerebko	Boston Celtics	8.0	PF	29.0	6-10	231.0	NaN	5000000.0
...	...	...	...	...	...	...	...	...	...
452	Trey Lyles	Utah Jazz	41.0	PF	20.0	6-10	234.0	Kentucky	2239800.0
453	Shelvin Mack	Utah Jazz	8.0	PG	26.0	6-3	203.0	Butler	2433333.0
454	Raul Neto	Utah Jazz	25.0	PG	24.0	6-1	179.0	NaN	900000.0
455	Tibor Pleiss	Utah Jazz	21.0	C	26.0	7-3	256.0	NaN	2900000.0
456	Jeff Withey	Utah Jazz	24.0	C	26.0	7-0	231.0	Kansas	947276.0

457 rows × 9 columns

	Name	Team	Number	Position	Age	Height	Weight	College	Salary
291	Orlando Johnson	New Orleans Pelicans	0.0	SG	27.0	6-5	220.0	UC Santa Barbara	55722.0

h) Merging Data in Pandas

Example-

```
[In]    l={ 'id':[1,2,3,4,5],
          'Name': ['Alex', 'Amy', 'Allen', 'Alice', 'Ayoung'],
          'subject_id':['sub1','sub2','sub4','sub6','sub5']}
        dl = pd.DataFrame(l)

        r={'id':[1,2,3,4,5],
          'Name': ['Billy', 'Brian', 'Bran', 'Bryce', 'Betty'],
          'subject_id':['sub2','sub4','sub3','sub6','sub5']}
        dr = pd.DataFrame(r)
        print(dl)
        print(dr)
        print('-----------')
        print(pd.merge(dl,dr,on='id'))
        print('-----------')
        print (pd.merge(dl,dr,on=['id','subject_id']))
        print('-----------')
[out]      id    Name subject_id
        0   1    Alex       sub1
        1   2     Amy       sub2
        2   3   Allen       sub4
        3   4   Alice       sub6
        4   5  Ayoung       sub5
           id   Name subject_id
        0   1  Billy       sub2
        1   2  Brian       sub4
        2   3   Bran       sub3
        3   4  Bryce       sub6
        4   5  Betty       sub5
        -----------
           id  Name_x subject_id_x Name_y subject_id_y
        0   1    Alex         sub1  Billy         sub2
        1   2     Amy         sub2  Brian         sub4
        2   3   Allen         sub4   Bran         sub3
        3   4   Alice         sub6  Bryce         sub6
        4   5  Ayoung         sub5  Betty         sub5
        -----------
           id  Name_x subject_id Name_y
        0   4   Alice       sub6  Bryce
        1   5  Ayoung       sub5  Betty
        -----------
```

i. Left join

Example-

```
[In]    l={ 'id':[1,2,3,4,5],
          'Name': ['Alex', 'Amy', 'Allen', 'Alice', 'Ayoung'],
          'subject_id':['sub1','sub2','sub4','sub6','sub5']}
        dl = pd.DataFrame(l)

        r={'id':[1,2,3,7,8],
          'Name': ['Billy', 'Brian', 'Bran', 'Bryce', 'Betty'],
          'subject_id':['sub2','sub4','sub3','sub6','sub5']}
        dr = pd.DataFrame(r)
        print(dl)
        print(dr)
        print('-----------')
        print(pd.merge(dl,dr,on='id',how='left'))
```

[out]

```
   id    Name subject_id
0   1    Alex       sub1
1   2     Amy       sub2
2   3   Allen       sub4
3   4   Alice       sub6
4   5  Ayoung       sub5
   id   Name subject_id
0   1  Billy       sub2
1   2  Brian       sub4
2   3   Bran       sub3
3   7  Bryce       sub6
4   8  Betty       sub5
-----------
   id  Name_x subject_id_x Name_y subject_id_y
0   1    Alex         sub1  Billy         sub2
1   2     Amy         sub2  Brian         sub4
2   3   Allen         sub4   Bran         sub3
3   4   Alice         sub6    NaN          NaN
4   5  Ayoung         sub5    NaN          NaN
```

ii. Right join

Example-

[In]

```
l={ 'id':[1,2,3,4,5],
  'Name': ['Alex', 'Amy', 'Allen', 'Alice', 'Ayoung'],
  'subject_id':['sub1','sub2','sub4','sub6','sub5']}
dl = pd.DataFrame(l)

r={'id':[1,2,3,7,8],
  'Name': ['Billy', 'Brian', 'Bran', 'Bryce', 'Betty'],
  'subject_id':['sub2','sub4','sub3','sub6','sub5']}
dr = pd.DataFrame(r)
print(dl)
print(dr)
print('-----------')
print(pd.merge(dl,dr,on='id',how='right'))
```

[out]

```
   id    Name subject_id
0   1    Alex       sub1
1   2     Amy       sub2
2   3   Allen       sub4
3   4   Alice       sub6
4   5  Ayoung       sub5
   id   Name subject_id
0   1  Billy       sub2
1   2  Brian       sub4
2   3   Bran       sub3
3   7  Bryce       sub6
4   8  Betty       sub5
-----------
   id Name_x subject_id_x Name_y subject_id_y
0   1   Alex         sub1  Billy         sub2
```

```
1   2    Amy          sub2  Brian          sub4
2   3  Allen          sub4   Bran          sub3
3   7    NaN           NaN  Bryce          sub6
4   8    NaN           NaN  Betty          sub5
```

iii. Inner join

Example-

[In]

```
l={ 'id':[1,2,3,4,5],
  'Name': ['Alex', 'Amy', 'Allen', 'Alice', 'Ayoung'],
  'subject_id':['sub1','sub2','sub4','sub6','sub5']}
dl = pd.DataFrame(l)

r={'id':[1,2,3,7,8],
  'Name': ['Billy', 'Brian', 'Bran', 'Bryce', 'Betty'],
  'subject_id':['sub2','sub4','sub3','sub6','sub5']}
dr = pd.DataFrame(r)
print(dl)
print(dr)
print('-----------')
print(pd.merge(dl,dr,on='subject_id',how='inner'))
```

[out]

```
   id    Name subject_id
0   1    Alex       sub1
1   2     Amy       sub2
2   3   Allen       sub4
3   4   Alice       sub6
4   5  Ayoung       sub5
   id   Name subject_id
0   1  Billy       sub2
1   2  Brian       sub4
2   3   Bran       sub3
3   7  Bryce       sub6
4   8  Betty       sub5
-----------
   id_x  Name_x subject_id  id_y Name_y
0     2     Amy       sub2     1  Billy
1     3   Allen       sub4     2  Brian
2     4   Alice       sub6     7  Bryce
3     5  Ayoung       sub5     8  Betty
```

iv. Outer join

Example-

[In]

```
l={ 'id':[1,2,3,4,5],
  'Name': ['Alex', 'Amy', 'Allen', 'Alice', 'Ayoung'],
  'subject_id':['sub1','sub2','sub4','sub6','sub5']}
dl = pd.DataFrame(l)

r={'id':[1,2,3,7,8],
  'Name': ['Billy', 'Brian', 'Bran', 'Bryce', 'Betty'],
  'subject_id':['sub2','sub4','sub3','sub6','sub5']}
dr = pd.DataFrame(r)
print(dl)
print(dr)
print('-----------')
print(pd.merge(dl,dr,on='id',how='outer'))
```

[out]

```
   id    Name subject_id
0   1    Alex       sub1
1   2     Amy       sub2
2   3   Allen       sub4
3   4   Alice       sub6
4   5  Ayoung       sub5
   id   Name subject_id
0   1  Billy       sub2
1   2  Brian       sub4
2   3   Bran       sub3
3   7  Bryce       sub6
4   8  Betty       sub5
-----------
   id  Name_x subject_id_x Name_y subject_id_y
0   1    Alex         sub1  Billy         sub2
1   2     Amy         sub2  Brian         sub4
2   3   Allen         sub4   Bran         sub3
3   4   Alice         sub6    NaN          NaN
4   5  Ayoung         sub5    NaN          NaN
5   7     NaN          NaN  Bryce         sub6
6   8     NaN          NaN  Betty         sub5
```

i) Descriptive Statistics in Pandas

1. count() → Number of non-null observations
2. sum () → Sum of Values
3. mean() → mean of values
4. median() → median of values
5. mode() → Mode of values
6. std() → Standard Deviation of Values
7. min() → Minimum Value
8. max() → Maximum Value
9. abs() → Absolute Value
10. prod() → Product of Values
11. cumsum() → Cumulative Sum
12. cumprod() → Cumulative Product

Example-

[In]

```
d={'Name':pd.Series(['Tom','James','Ricky','Vin','Steve','Smith','Jack','Lee','David','Gasper','Betina','Andres'])
,
   'Age':pd.Series([25,26,25,23,30,29,23,34,40,30,51,46]),
   'Rating':pd.Series([4.23,3.24,3.98,2.56,3.20,4.6,3.8,3.78,2.98,4.80,4.10,3.65]) }

# The above series is provided inside a dictionery to take into account the index
# d={'Name':['Tom','James','Ricky','Vin','Steve','Smith','Jack','Lee','David','Gasper','Betina','Andres'],
#    'Age':[25,26,25,23,30,29,23,34,40,30,51,46],
#    'Rating':[4.23,3.24,3.98,2.56,3.20,4.6,3.8,3.78,2.98,4.80,4.10,3.65]
#   }
df=pd.DataFrame(d)
df
print (df.sum())
```

```
print('-----------')
print (df.mean())
```

[out
]

```
Name      TomJamesRickyVinSteveSmithJackLeeDavidGasperBe...
Age                                                     382
Rating                                                44.92
dtype: object
-----------
Age       31.833333
Rating     3.743333
dtype: float64
```

Example-

[In]

```
d={'Name':pd.Series(['Tom','James','Ricky','Vin','Steve','Smith','Jack','Lee','David','Gasper','Betina','Andres'])
,
  'Age':pd.Series([25,26,25,23,30,29,23,34,40,30,51,46]),
  'Rating':pd.Series([4.23,3.24,3.98,2.56,3.20,4.6,3.8,3.78,2.98,4.80,4.10,3.65]) }
df=pd.DataFrame(d)
df['Age'].cumsum()   # We can also specify the column
```

[out
]

```
0      25
1      51
2      76
3      99
4     129
5     158
6     181
7     215
8     255
9     285
10    336
11    382
Name: Age, dtype: int64
```

j) Summarizing Statistics in Pandas

This function gives the mean, std and IQR values.
It excludes the character columns and gives summary aboutnumeric columns. 'include' is the argument which is used topass necessary information regarding what columns need tobe considered for summarizing.
Takes the list of values; by default, 'number'.
•object –Summarizes String columns
•number –Summarizes Numeric columns
•all –Summarizes all columns together (Should not pass it asa list value)

Example-

[In]

```
d={'Name':pd.Series(['Tom','James','Ricky','Vin','Steve','Smith','Jack','Lee','David','Gasper','Betina','Andres'])
,
  'Age':pd.Series([25,26,25,23,30,29,23,34,40,30,51,46]),
  'Rating':pd.Series([4.23,3.24,3.98,2.56,3.20,4.6,3.8,3.78,2.98,4.80,4.10,3.65]) }

df=pd.DataFrame(d)
df.describe()
```

[out
]

	Age	Rating
count	12.000000	12.000000
mean	31.833333	3.743333
std	9.232682	0.661628
min	23.000000	2.560000

25%	25.000000	3.230000
50%	29.500000	3.790000
75%	35.500000	4.132500
max	51.000000	4.800000

k) Handling missing data in Pandas

Methods to handle missing data:

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

i. isnull(), notnull(), dropna()

Example-

[In]

```
d = {'f':[100, 90, np.nan, 95],
   's':[30,45,56,np.nan],
   't':[np.nan, 40,80, 60]}
df = pd.DataFrame(d)
print(df)
print('-----Returns true for null and false for not null------')
print(df.isnull())
print('-----Returns true for nonnull and false for null------')
print(df.notnull())
print('-----Drops the rows where Nan is present------')
print(df.dropna())
```

[out]

```
       f     s     t
0  100.0  30.0   NaN
1   90.0  45.0  40.0
2    NaN  56.0  80.0
3   95.0   NaN  60.0
-----Returns true for null and false for not null------
       f      s      t
0  False  False   True
1  False  False  False
2   True  False  False
3  False   True  False
-----Returns true for nonnull and false for null------
       f      s      t
0   True   True  False
1   True   True   True
2  False   True   True
3   True  False   True
-----Drops the rows where Nan is present------
      f     s     t
1  90.0  45.0  40.0
```

ii. fillna()

Example-

[In]

```
d = {'f':[100, 90, np.nan, 95],
   's':[30,45,56,np.nan],
   't':[np.nan, 40,80, 60]}
df = pd.DataFrame(d)
print(df)

print('-----Fill a value provided where there is Nan------')
print(df.fillna(10))
print('-----Fill with a function bfill- where this fills a backward value, similary thereis ffill------')
print(df.fillna(method='bfill'))
```

[out]

```
       f     s     t
0  100.0  30.0   NaN
1   90.0  45.0  40.0
2    NaN  56.0  80.0
3   95.0   NaN  60.0
-----Fill a value provided where there is Nan------
       f     s     t
0  100.0  30.0  10.0
1   90.0  45.0  40.0
2   10.0  56.0  80.0
3   95.0  10.0  60.0
-----Fill with a function bfill- where this fills a backward value, similary
thereis ffill------
       f     s     t
0  100.0  30.0  40.0
1   90.0  45.0  40.0
2   95.0  56.0  80.0
3   95.0   NaN  60.0
```

iii. replace()

Example-

[In]

```
d = {'f':[100, 90, np.nan, 95],
   's':[30,45,56,np.nan],
   't':[np.nan, 40,80, 60]}
df = pd.DataFrame(d)
print(df)
print('-----Replaces a value------')
print(df.replace(100,20))
print('-----Replaces a value------')
print(df.replace(to_replace=np.nan,value='No Data'))
```

[out]

```
       f     s     t
0  100.0  30.0   NaN
1   90.0  45.0  40.0
2    NaN  56.0  80.0
3   95.0   NaN  60.0
-----Replaces a value------
      f     s     t
0  20.0  30.0   NaN
1  90.0  45.0  40.0
```

```
2   NaN  56.0  80.0
3  95.0   NaN  60.0
-----Replaces a value------
         f        s        t
0    100.0     30.0  No Data
1     90.0     45.0     40.0
2  No Data     56.0     80.0
3     95.0  No Data     60.0
```

iv. interpolate()

Syntax: DataFrame.interpolate(method='linear', axis=0, limit=None, inplace=False, limit_direction='forward', limit_area=None, downcast=None, **kwargs)

Parameters :
method : {'linear', 'time', 'index', 'values', 'nearest', 'zero', 'slinear', 'quadratic', 'cubic', 'barycentric', 'krogh', 'polynomial', 'spline', 'piecewise_polynomial', 'from_derivatives', 'pchip', 'akima'}

axis : 0 fill column-by-column and 1 fill row-by-row.
limit : Maximum number of consecutive NaNs to fill. Must be greater than 0.
limit_direction : {'forward', 'backward', 'both'}, default 'forward'
limit_area : None (default) no fill restriction. inside Only fill NaNs surrounded by valid values (interpolate). outside Only fill NaNs outside valid values (extrapolate). If limit is specified, consecutive NaNs will be filled in this direction.
inplace : Update the NDFrame in place if possible.
downcast : Downcast dtypes if possible.
kwargs : keyword arguments to pass on to the interpolating function.

Example-

[In]

```
d = {'f':[100, 90, np.nan, 95],
   's':[30,45,56,np.nan],
   't':[np.nan, 40,80, 60]}
df = pd.DataFrame(d)
print(df)
print('-----Interpolates with linear -----')
print(df.interpolate(method='linear'))
print('-----Interpolates with linear and forward -----')
print(df.interpolate(method='linear',limit_direction='forward'))
print('-----Interpolates with linear and both -----')
print(df.interpolate(method='linear',limit_direction='both'))
```

[out]

```
       f     s     t
0  100.0  30.0   NaN
1   90.0  45.0  40.0
2    NaN  56.0  80.0
3   95.0   NaN  60.0
-----Interpolates with linear -----
       f     s     t
0  100.0  30.0   NaN
1   90.0  45.0  40.0
2   92.5  56.0  80.0
3   95.0  56.0  60.0
-----Interpolates with linear and forward -----
       f     s     t
```

```
0  100.0  30.0   NaN
1   90.0  45.0  40.0
2   92.5  56.0  80.0
3   95.0  56.0  60.0
-----Interpolates with linear and both -----
       f     s     t
0  100.0  30.0  40.0
1   90.0  45.0  40.0
2   92.5  56.0  80.0
3   95.0  56.0  60.0
```

l) Group Operations in Pandas

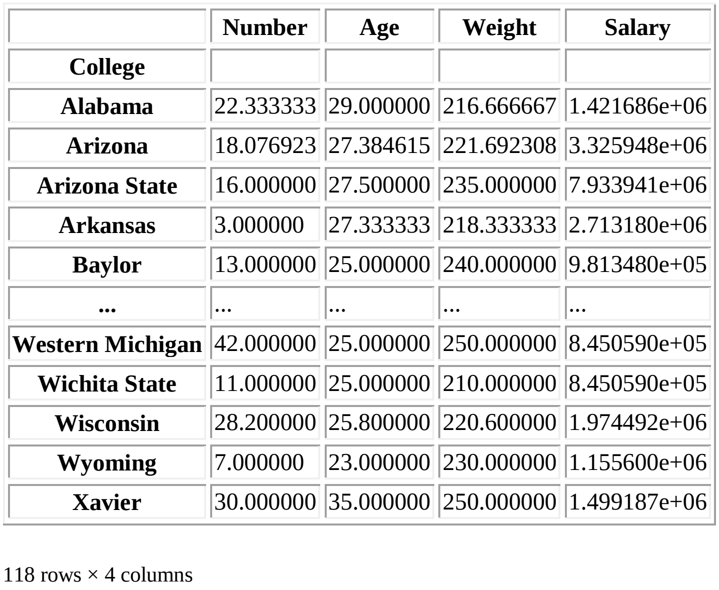
i. Grouping

Group operations are performed to aggregate values based on a selection. Here for example group colleges with their mean salaries, Age etc.

Example-

```
[In]    f=d.groupby('College').mean()
        f
```

[out]



	Number	Age	Weight	Salary
College				
Alabama	22.333333	29.000000	216.666667	1.421686e+06
Arizona	18.076923	27.384615	221.692308	3.325948e+06
Arizona State	16.000000	27.500000	235.000000	7.933941e+06
Arkansas	3.000000	27.333333	218.333333	2.713180e+06
Baylor	13.000000	25.000000	240.000000	9.813480e+05
...	...	...	...	...
Western Michigan	42.000000	25.000000	250.000000	8.450590e+05
Wichita State	11.000000	25.000000	210.000000	8.450590e+05
Wisconsin	28.200000	25.800000	220.600000	1.974492e+06
Wyoming	7.000000	23.000000	230.000000	1.155600e+06
Xavier	30.000000	35.000000	250.000000	1.499187e+06

118 rows × 4 columns

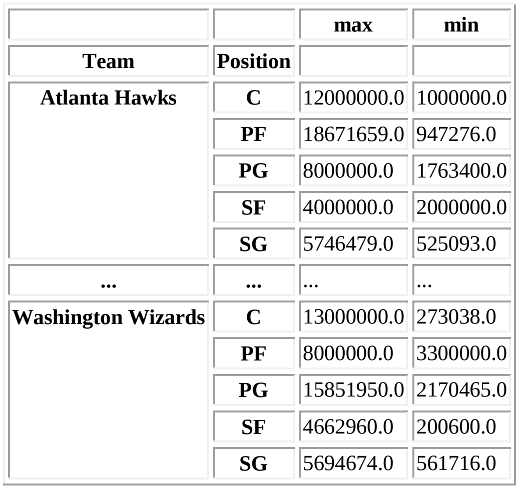
ii. Grouping and aggregating

The grouped values can be aggregated that means we can check the max, min or mean values etc

Example-

```
[In]    d.groupby(['Team', 'Position'])['Salary'].agg(['max', 'min'])   # here they are grouping by Team and
Position, then selecting Salary and aggregating function max and min is used
```

[out]



		max	min
Team	**Position**		
Atlanta Hawks	**C**	12000000.0	1000000.0
	PF	18671659.0	947276.0
	PG	8000000.0	1763400.0
	SF	4000000.0	2000000.0
	SG	5746479.0	525093.0
...	**...**	...	...
Washington Wizards	**C**	13000000.0	273038.0
	PF	8000000.0	3300000.0
	PG	15851950.0	2170465.0
	SF	4662960.0	200600.0
	SG	5694674.0	561716.0

149 rows × 2 columns

21. Matplotlib -Data Visualization

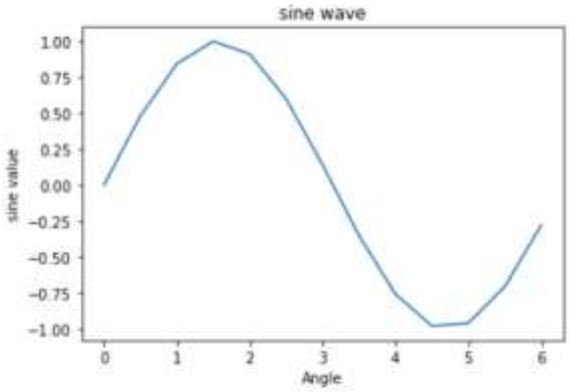
Matplotlib is one of the most popular Python packages used for data visualization. It is a cross platform library for making 2D plots from data in arrays.

matplotlib
pyplot is a collection of command style functions that make Matplotlib work like MATLAB Each Pyplot function makes some change to a figure For example, a function creates a figure, a plotting area in a figure, plots some lines in a plotting area, decorates the plot with labels, etc

a) Plotting Method 1 using xlabel

Example- Method 1

```
[In]     x= np.arange(0, math.pi*2, 0.5)
         print('-----x value -----')
         print(x)
         y= np.sin(x)
         print('-----y value -----')
         print(y)
         plt.plot(x,y)
         plt.xlabel('Angle')
         plt.ylabel('sine value')
         plt.title('sine wave')
         plt.show()
[out]    -----x value -----
         [0.  0.5 1.  1.5 2.  2.5 3.  3.5 4.  4.5 5.  5.5 6. ]
         -----y value -----
         [ 0.          0.47942554  0.84147098  0.99749499  0.90929743  0.59847214
           0.14112001 -0.35078323 -0.7568025  -0.97753012 -0.95892427 -0.70554033
          -0.2794155 ]
```



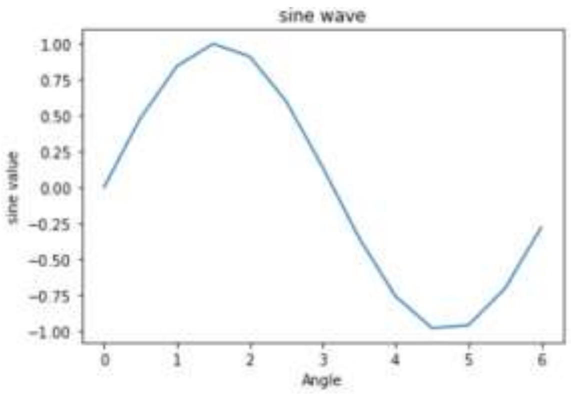
b) Plotting Method 2 using set

Example- method 2

[In]

```
fig=plt.figure()
ax= fig.add_axes([0,0,1,1])   # 0,0 meanes starting from and 1,1 means height and width
ax.plot(x,y)
ax.set_title('Sine Function')
ax.set_xlabel('Angle')
ax.set_ylabel('Sine of Angle')
```

[out]



c) Adding Legends and making 2 plots

Example

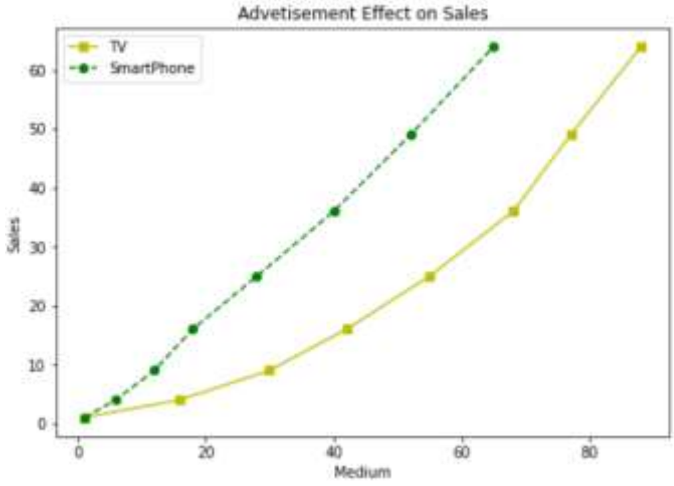
[In]

```
y = [1,4,9,16,25,36,49,64]
x1 = [1,16,30,42,55,68,77,88]
x2 = [1,6,12,18,28,40,52,65]
fig = plt.figure()
ax = fig.add_axes([0,0,1,1])

l1= ax.plot(x1, y, 'ys-')    # ys- stands for y for yellow, s for square, - for solid line
l2= ax.plot(x2, y, 'go--')    # go-- stands for g for green, o for round, -- for dashed

ax.legend(labels = ('TV', 'SmartPhone'),loc='upper left')
ax.set_title("Advetisement Effect on Sales")
ax.set_xlabel("Medium")
ax.set_ylabel('Sales')
plt.show()
```

[out]



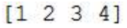
d) Adding multiple plots

Example

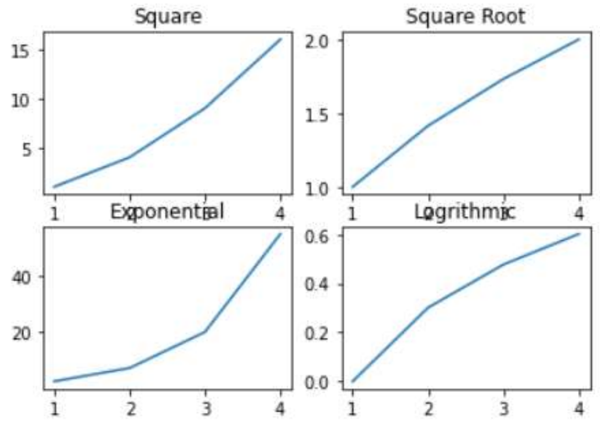
[In]

```
from matplotlib import pyplot as plt
import numpy as np
fig, a = plt.subplots(2,2)
x = np.arange(1,5)
print(x)
a[0][0].plot(x, x*x)
a[0][0].set_title("Square")
a[0][1].plot(x, np.sqrt(x))
a[0][1].set_title("Square Root")
a[1][0].plot(x, np.exp(x))
a[1][0].set_title("Exponential")
a[1][1].plot(x, np.log10(x))
a[1][1].set_title("Logarithmic")
plt.show()
```

[out]



```
[1 2 3 4]
```



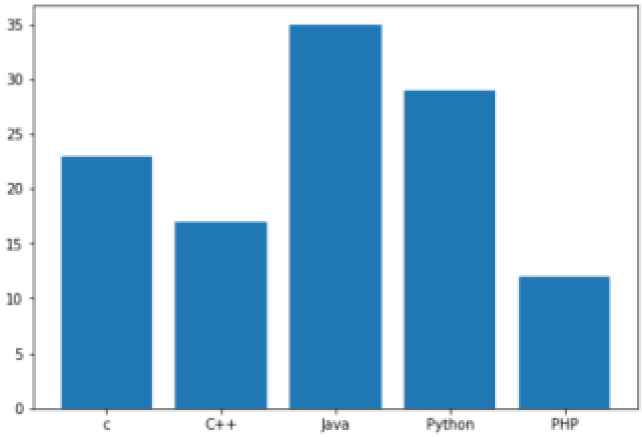
e) Barplots

Example

[In]

```
fig = plt.figure()
ax = fig.add_axes([0,0,1,1])
x=['c','C++','Java','Python', 'PHP']
y= [23,17,35,29,12]
ax.bar(x,y)
plt.show()
```

[out]



f) Multiple Barplots

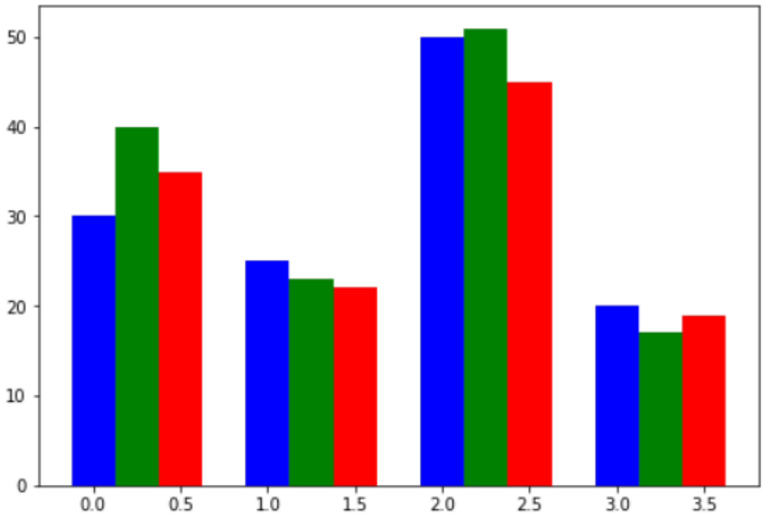
Example

[In]

```
d = [[30,25,50,20],
   [40,23,51,17],
   [35,22,45,19]]
x = np.arange(4)   # x is an array created x= array([0,1,2,3])

fig = plt.figure()
ax = fig.add_axes([0,0,1,1])
ax.bar(x + 0.00, d[0], color = 'b', width = 0.25)
ax.bar(x + 0.25, d[1], color = 'g', width = 0.25)
ax.bar(x + 0.50, d[2], color = 'r', width = 0.25)
plt.show()
```

[out]



g) Random Plots

Example

[In]

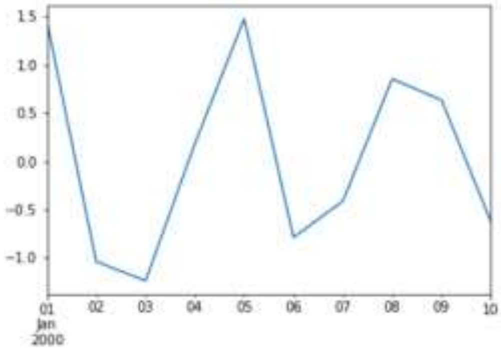
```
print('-----Random values--------')
x=np.random.randn(10)
print(x)
print('-----creating a date range with function date_range--------')
i=pd.date_range('1/1/2000', periods=10)
print(i)
print('-----creating a series with random values and dates--------')
t=pd.Series(x,index=i)
print(t)
print('-----Plotting--------')
t.plot()
plt.show()

print('-----Plotting with cumulative sum--------')
ts=t.cumsum()
ts.plot()
plt.show()
```

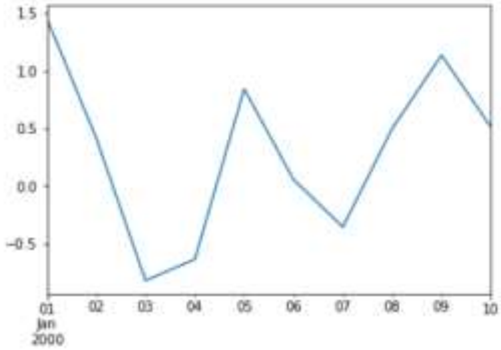
[out]

```
-----Random values--------
[ 1.45463242 -1.03715498 -1.23705757  0.18460466  1.47751847 -0.78673997
 -0.41041084  0.85397213  0.63561826 -0.62277257]
-----creating a date range with function date_range--------
DatetimeIndex(['2000-01-01', '2000-01-02', '2000-01-03', '2000-01-04',
               '2000-01-05', '2000-01-06', '2000-01-07', '2000-01-08',
               '2000-01-09', '2000-01-10'],
              dtype='datetime64[ns]', freq='D')
-----creating a series with random values and dates--------
2000-01-01    1.454632
2000-01-02   -1.037155
2000-01-03   -1.237058
2000-01-04    0.184605
2000-01-05    1.477518
2000-01-06   -0.786740
2000-01-07   -0.410411
2000-01-08    0.853972
2000-01-09    0.635618
2000-01-10   -0.622773
Freq: D, dtype: float64
```

```
-----Plotting--------
```



```
-----Plotting with cumulative sum--------
```



h) Exporting csv data directly from Internet

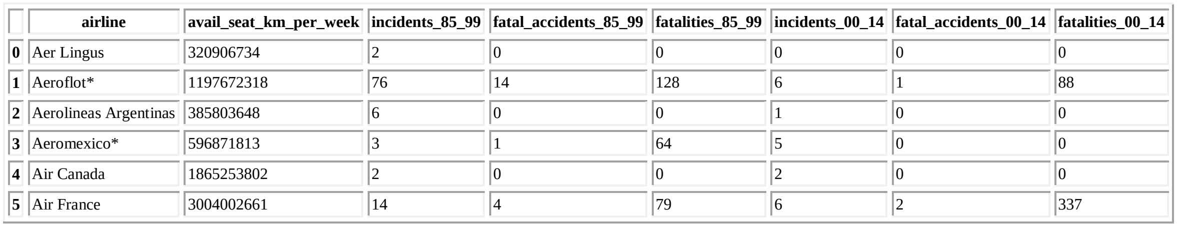
Example

[In]

```
import pandas as pd

df = pd.read_csv("https://raw.githubusercontent.com/fivethirtyeight/data/master/airline-safety/airline-safety.csv")
df
```

[out]



	airline	avail_seat_km_per_week	incidents_85_99	fatal_accidents_85_99	fatalities_85_99	incidents_00_14	fatal_accidents_00_14	fatalities_00_14
0	Aer Lingus	320906734	2	0	0	0	0	0
1	Aeroflot*	1197672318	76	14	128	6	1	88
2	Aerolineas Argentinas	385803648	6	0	0	1	0	0
3	Aeromexico*	596871813	3	1	64	5	0	0
4	Air Canada	1865253802	2	0	0	2	0	0
5	Air France	3004002661	14	4	79	6	2	337

i) Box plot

For understanding box plot we should study IQR (Interquartile range). Read from internet more about this topic to understand the data displayed in the box plot

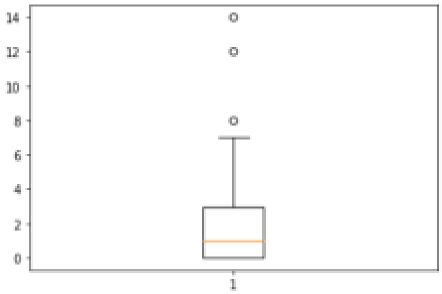
Example

[In]

```
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
import math
```

```
y=df.fatal_accidents_85_99            # this data Is taken from the above table
plt.boxplot(y)
plt.show()
```

[out]



22. Seaborn -Data Visualization

It is a data visualization library for beautifying the plots

a) Distribution plot

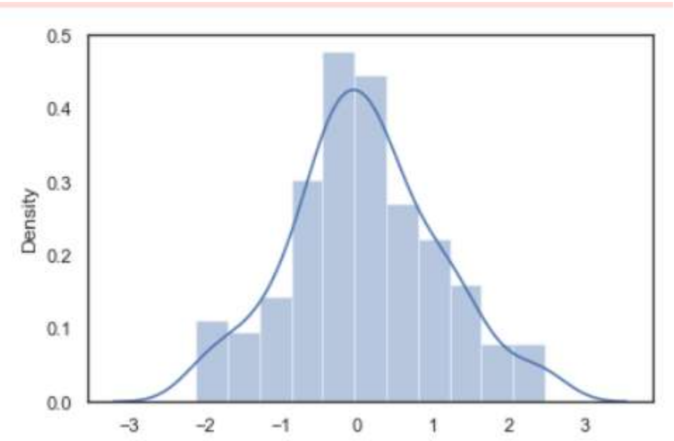
Example

[In]

```
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
import math
import seaborn as sns

sns.set(style = 'white')
rs = np.random.RandomState(10)
e = rs.normal(size = 150)
sns.distplot(e, kde = True, color = 'b')    # kde is the Kernel density estimation
plt.show()
```

[out]

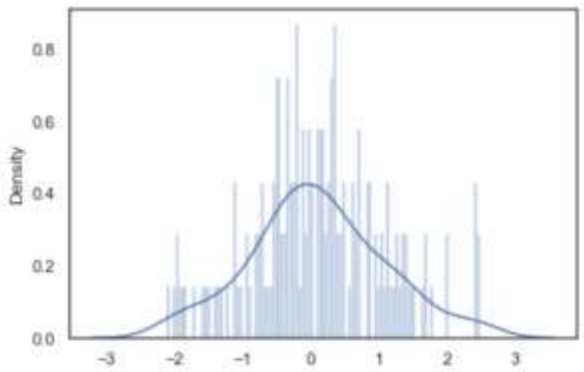


Example

[In]

```
sns.set(style = 'white')
rs = np.random.RandomState(10)
e = rs.normal(size = 150)
sns.distplot(e, kde = True, bins=100,color = 'b')     # here we have introduced bins. Bins increases the values in the density plot
plt.show()
```

[out]



b) Count plot

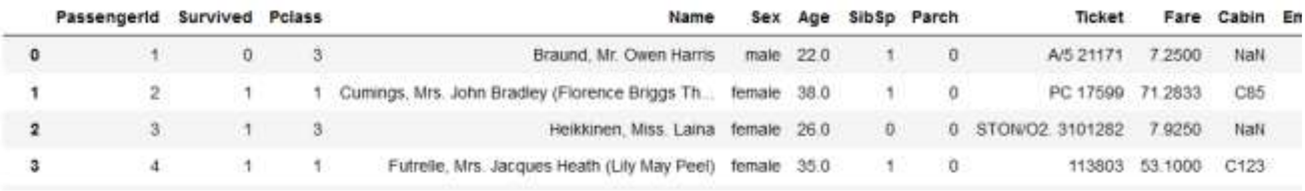
Example

[In]

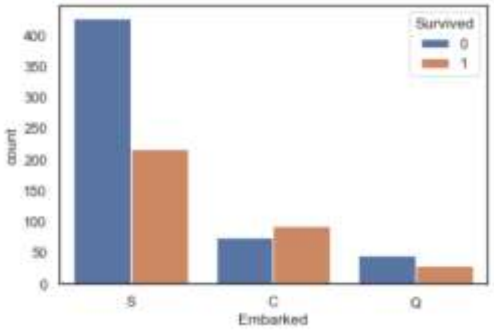
```
df = pd.read_csv("titanic.csv")
df

sns.countplot(df['Embarked'], hue= df['Survived'], dodge = True)
plt.show()\
```

[out
]



	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Em
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	



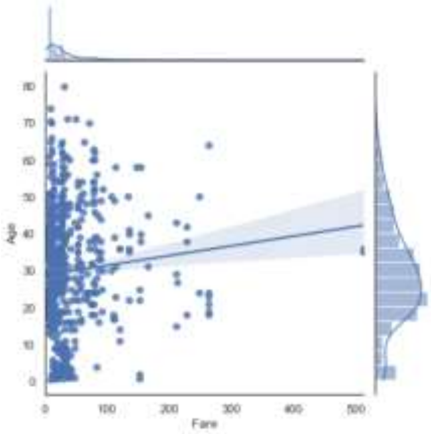
c) Jointplot

Example

[In]

```
sns.jointplot(x=df['Fare'], y= df['Age'], kind= "reg")      # kind` must be one of ['scatter', 'hist', 'hex', 'kde', 'reg', 'resid']
plt.show()
```

[out]



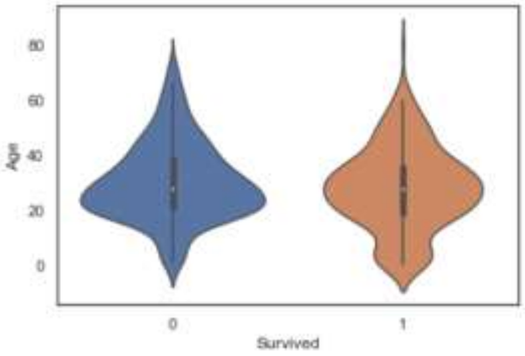
d) Violin

Example

[In]

```
sns.violinplot(x=df['Survived'], y= df['Age'])
plt.show()
```

[out]



23. EDA

Exploration Data Analysis

1. Data Extraction that is importing the data ie importin a csv file
2. Data Cleaning
3. Visualization
4. Pandas Query

24. Important shortcuts and websites

a) Dataset website

Kaggle.com → this site has numerous amounts of Data sets available for download

b) To use help

?sns.stripplot

By running this command, we will get all the parameters that can be passed to the stripplot