Experiment 1

AIM: Introduction to Python for machine learning.

Data types

Numbers

```
1+1 # Mathemetical operators : +,-,*,/,**,%
```

```
2
```

```
x = 1
y = 1.0
```

```
type(x)
```

```
int
```

```
type(y)
```

```
float
```

```
a = 1
b = 1
c = a/b
print(c)
type(c)
```

```
1.0
float
```

```
a = 1
b = 1
c = a*b
print(c)
type(c)
```

```
1
int
```

boolean

```
a = True
b = False
```

```
a = 10
b = 5
c = 7
```

```
a > b
```

```
True
```

```
a > b or b > c
```

```
True
```

```
a > b and b > c
```

```
False
```

Strings

```
'hello world'
```

```
'hello world'
```

```
a = 'hello world'
```

```
print(a)
```

```
hello world
```

```
name = 'xyz'
age = 33
```

```
name.upper()
```

```
'XYZ'
```

```
a.split()
```

```
['hello', 'world']
```

```
a.split('e')
```

```
['h', 'llo world']
```

Print

```
print('My name is {} and I am {} years old'.format(name,age))
```

```
My name is xyz and I am 33 years old
```

Exercise 1 :

1. Create two variables, assign any value to them, find their data types, perform some mathematical operations.
2. Create a variable with any string and perform string manipulation.

List

```
list_no = [1,3,2]
list_str = ['a','b','c']
list_mix = [4,'d',list_no,list_str]
```

```
list_no.sort()
list_no
```

```
[1, 2, 3]
```

```
'a' in list_no
```

```
False
```

```
'a' in list_str
```

```
True
```

```
print(list_mix)
```

```
[4, 'd', [1, 2, 3], ['a', 'b', 'c']]
```

```
list_mix[1]
```

```
'd'
```

```
list_mix[-1]
```

```
['a', 'b', 'c']
```

```
list_mix.append('hello')
print(list_mix)
```

```
[4, 'd', [1, 2, 3], ['a', 'b', 'c'], 'hello']
```

```
list_mix[2][2]
```

```
3
```

```
list_mix[4][0]#[0:2]
```

```
'h'
```

Dictionaries

```
dict_me = {'name':'xyz','age':33}
```

```
dict_me['name']
```

```
'xyz'
```

```
dict_me.keys()
```

```
dict_keys(['name', 'age'])
```

```
dict_me.values()
```

```
dict_values(['xyz', 33])
```

```
a = dict_me.items()
a
```

```
dict_items([('name', 'xyz'), ('age', 33)])
```

```
dict_me['name'] = 'JRP'
```

```
a
```

```
dict_items([('name', 'JRP'), ('age', 33)])
```

Tuples

```
a = (1,2,3)
```

```
a[2]
```

```
3
```

```
a[-1]
```

```
3
```

```
#a[2]=10
```

```
a
```

```
(1, 2, 3)
```

Sets

```
{1,'hi',2,'ok',2,1,1,1,}
```

```
{1, 2, 'hi', 'ok'}
```

If-else

```
a = 10
b = 7
c = 50
```

```
if a>b:
    if a>c:
        print('a is larget')
    else:
        print('c is largest')
elif b>c:
    print('b is largest')
else:
    print('c is largest')
```

```
c is largest
```

```
if a > b and a > c:
    print('a is largest')
elif b > c and b > a:
    print('b is largest')
else:
    print('c is largest')
```

```
c is largest
```

for loop

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

```
0
1
2
3
4
```

```
a = range(5)
```

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

```
0
1
2
3
4
```

```
a = ['ab','bc','cd','de','ef']
```

```
for i in enumerate(a):
    print(i)
```

```
(0, 'ab')
(1, 'bc')
(2, 'cd')
(3, 'de')
(4, 'ef')
```

```
for i in enumerate(a):
    print(i[1][0])
```

```
a
b
c
d
e
```

```
[i**2 for i in range(5) ]
```

```
[0, 1, 4, 9, 16]
```

```
[i%2 for i in range(5) ]
```

```
[0, 1, 0, 1, 0]
```

while loop

```
i = 0
while i<5:
    print(i**2)
    i+=1
```

```
0
1
4
9
16
```

```
# Example using break
numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

for num in numbers:
    if num == 5:
        break  # Exit the loop if num equals 5
    print(num)

# Output: 1 2 3 4

print()
```

```
1
2
3
4
```

```
# Example using continue
for num in numbers:
    if num % 2 == 0:
        continue  # Skip even numbers and continue to the next iteration
    print(num)

# Output: 1 3 5 7 9
```

```
1
3
5
7
9
```

functions

```
def def_print(para='hello'):
    print(para)


def_print()
```

```
hello
```

```
def_print('hello world')
```

```
hello world
```

```
def sqr(no=0):
    no2 = no**2
    print(no2)
    return no2


a = range(5)


[sqr(i) for i in a]
```

```
0
1
4
9
16
[0, 1, 4, 9, 16]
```

```
sqr2 = lambda a:a**2


[sqr2(i) for i in a]
```

```
[0, 1, 4, 9, 16]
```

```
y = [0,2,4,6,8]
x = [1,3,5,7,9]


list(map(lambda a,b:a+b,x,y))
```

```
[1, 5, 9, 13, 17]
```

```
z = [1,2,3,4,5,6,7,8,9]


list(filter(lambda x:x%2 == 0,z))
```

```
[2, 4, 6, 8]
```

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


a.sort()
a
```

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

Exercise 2:

1. Write a python program to check if a given number is prime.
2. Create a list containing numbers 1 to 100. One by one check if the element is even or odd. Create a dictionary with keys "even" and "odd", Store the respective values in the dictionary.

File handling

```
# Open the file in write mode.
file = open("my_file.txt", "w")

# Write a string to the file.
file.write("This is my file.")

# Close the file.
file.close()



# Open the file in read mode.
file = open("my_file.txt", "r")

# Read the entire file.
data = file.read()

# Close the file.
file.close()

# Print the data.
print(data)
```

```
This is my file.
```

Numpy

```
!pip install numpy
```

```
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (1.22.4)
```

```
import numpy as np
```

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

```
Start coding or generate with AI.
```

```
b = np.array([[11,12,13],[21,22,23],[31,32,33]])
```

```
b[1,2] #b[1][2]
```

```
23
```

```
np.arange(1,10,1)
```

```
array([1, 2, 3, 4, 5, 6, 7, 8, 9])
```

```
np.linspace(1,9,10)
```

```
array([1.        , 1.88888889, 2.77777778, 3.66666667, 4.55555556,
       5.44444444, 6.33333333, 7.22222222, 8.11111111, 9.        ])
```

```
np.zeros(5)
```

```
array([0., 0., 0., 0., 0.])
```

```
np.zeros((5,5))
```

```
array([[0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0.]])
```

```
np.ones((5))
```

```
array([1., 1., 1., 1., 1.])
```

```
np.ones((5,5))
```

```
array([[1., 1., 1., 1., 1.],
       [1., 1., 1., 1., 1.],
       [1., 1., 1., 1., 1.],
       [1., 1., 1., 1., 1.],
       [1., 1., 1., 1., 1.]])
```

```
np.eye(5)
```

```
array([[1., 0., 0., 0., 0.],
       [0., 1., 0., 0., 0.],
       [0., 0., 1., 0., 0.],
       [0., 0., 0., 1., 0.],
       [0., 0., 0., 0., 1.]])
```

Random

```
np.random.rand()
```

```
0.8660763193333921
```

```
np.random.rand(5,5)
```

```
array([[0.62169611, 0.84540513, 0.9986008 , 0.42699757, 0.53562037],
       [0.03741053, 0.33915061, 0.98532901, 0.8492632 , 0.89357144],
       [0.51074526, 0.38807363, 0.07456923, 0.16311325, 0.53640711],
       [0.70250673, 0.48158497, 0.55917168, 0.04986905, 0.17372366],
       [0.72655321, 0.83289816, 0.54997803, 0.83568961, 0.17104645]])
```

```
np.random.randn(5,5)
```

```
array([[ 1.5998749 , -0.26966538,  0.70885273, -1.05654689,  1.71704212],
       [-2.1913735 ,  0.21931484,  0.30663489,  0.07968698, -0.81099809],
       [-0.57462191, -0.33291261, -1.08998416, -1.30454375, -0.72668827],
       [-0.82280315,  1.53827156, -1.34259232,  0.63071194,  0.66158338],
       [ 1.28947722,  1.84223011, -0.76282713,  1.79697604,  0.27206656]])
```

```
np.random.randint(1,5)
```

```
4
```

```
np.random.randint(1,5,[5,5])
```

```
array([[2, 1, 3, 3, 3],
       [4, 2, 3, 2, 3],
       [3, 3, 1, 2, 2],
       [3, 2, 4, 2, 3],
       [4, 1, 4, 1, 1]])
```

array methods

```
data = [1,4,3,5]
arr = np.array(data)
print(arr)
print(arr.dtype)
print(arr.shape)
```

```
[1 4 3 5]
int64
(4,)
```

```
a = np.random.rand(1000)
b = np.random.randn(1000)
```

```
a.min()
```

```
0.0018064618617995576
```

```
a.max()
```

```
0.9949392325291694
```

```
a.mean()
```

```
0.4920039074764793
```

```
b.min()
```

```
-3.9197183203094546
```

```
b.max()
```

```
3.0319763683752705
```

```
b.mean()
```

```
0.011416132009365386
```

```
c = np.random.rand(4)
c
```

```
array([0.59815843, 0.39796413, 0.09048515, 0.99624564])
```

```
c.argmin()
```

```
2
```

```
c.argmax()
```

```
3
```

```
d = c.reshape(2,2)
print(d.shape)
d
```

```
(2, 2)
array([[0.59815843, 0.39796413],
       [0.09048515, 0.99624564]])
```

```
c.reshape(4)
```

```
array([0.59815843, 0.39796413, 0.09048515, 0.99624564])
```

```
c.reshape(2,2)
```

```
array([[0.59815843, 0.39796413],
       [0.09048515, 0.99624564]])
```

```
c.flatten()
```

```
array([0.59815843, 0.39796413, 0.09048515, 0.99624564])
```

```
e = np.array([0])
```

```
f = np.array([0.])
```

```
e.dtype
```

```
dtype('int64')
```

```
f.dtype
```

```
dtype('float64')
```

indexing and broadcasting

1d

```
a = np.arange(5)
```

```
a
```

```
array([0, 1, 2, 3, 4])
```

```
a[:]
```

```
array([0, 1, 2, 3, 4])
```

```
a[2]
```

```
2
```

```
a[2:4]
```

```
array([2, 3])
```

```
a[-1]
```

```
4
```

```
a[2:]
```

```
array([2, 3, 4])
```

```
a[:2]
```

```
array([0, 1])
```

```
a[:2]=10
```

```
a
```

```
array([10, 10,  2,  3,  4])
```

```
a = np.arange(10)
```

```
a
```

```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

```
b = a[2:7]
```

```
b
```

```
array([2, 3, 4, 5, 6])
```

```
b[2:5]=10
```

```
b
```

```
array([ 2,  3, 10, 10, 10])
```

```
a
```

```
array([ 0,  1,  2,  3, 10, 10, 10,  7,  8,  9])
```

```
c = a.copy()
```

```
c
```

```
array([ 0,  1,  2,  3, 10, 10, 10,  7,  8,  9])
```

```
c[4:7]=0
```

```
c
```

```
array([0, 1, 2, 3, 0, 0, 0, 7, 8, 9])
```

```
a
```

```
array([ 0,  1,  2,  3, 10, 10, 10,  7,  8,  9])
```

2d

```
a = np.random.rand(5,5)
```

```
a
```

```
array([[0.60046186, 0.66462706, 0.7601304 , 0.16645734, 0.62906346],
       [0.29548582, 0.86576726, 0.55657799, 0.86932634, 0.82810138],
       [0.84177085, 0.81059358, 0.94104169, 0.96364191, 0.04482948],
       [0.84620087, 0.66890516, 0.9715565 , 0.10148009, 0.71185997],
       [0.9260053 , 0.77799478, 0.20622618, 0.08268114, 0.92319813]])
```

```
a[0]
```

```
array([0.60046186, 0.66462706, 0.7601304 , 0.16645734, 0.62906346])
```

```
a[:,0]
```

```
array([0.60046186, 0.29548582, 0.84177085, 0.84620087, 0.9260053 ])
```

```
a[1:4,1:4]=10
```

```
a
```

```
array([[ 0.60046186,  0.66462706,  0.7601304 ,  0.16645734,  0.62906346],
       [ 0.29548582, 10.        , 10.        , 10.        ,  0.82810138],
       [ 0.84177085, 10.        , 10.        , 10.        ,  0.04482948],
       [ 0.84620087, 10.        , 10.        , 10.        ,  0.71185997],
       [ 0.9260053 ,  0.77799478,  0.20622618,  0.08268114,  0.92319813]])
```

logical operator to filter

```
a = np.random.randn(5,5)
```

```
a
```

```
array([[ 1.38915814, -0.98111389, -0.80056555,  0.9731559 , -0.14957839],
       [-0.10479692,  1.68802554, -0.01816371,  1.26656335,  0.19588287],
       [-0.98313464,  0.60928224,  1.20709979, -1.12411333, -0.67453012],
       [-1.02568527,  0.47612887,  0.81841892,  1.11074491, -0.30181466],
       [ 1.57915045,  0.58214521,  1.28142975, -0.32288493, -2.51025102]])
```

```
a>0
```

```
array([[ True, False, False,  True, False],
       [False,  True, False,  True,  True],
       [False,  True,  True, False, False],
       [False,  True,  True,  True, False],
       [ True,  True,  True, False, False]])
```

```
a[a>0]=10
a
```

```
array([[10.        , -0.98111389, -0.80056555, 10.        , -0.14957839],
       [-0.10479692, 10.        , -0.01816371, 10.        , 10.        ],
       [-0.98313464, 10.        , 10.        , -1.12411333, -0.67453012],
       [-1.02568527, 10.        , 10.        , 10.        , -0.30181466],
       [10.        , 10.        , 10.        , -0.32288493, -2.51025102]])
```

Operator

```
a = np.array([0,2,4,6,8])
b = np.array([1,3,5,7,9])
```

```
c = a + b
c
```

```
array([ 1,  5,  9, 13, 17])
```

```
d = b - 1
```

```
c = d==a
```

```
c.any()
```

```
True
```

```
d = np.array([0,5,6,9,9])
```

```
c = d==a
```

```
c.any()
```

```
True
```

```
c.all()
```

```
False
```

functions

array functions: https://numpy.org/doc/stable/reference/ufuncs.html

```
a = np.arange(0,1,0.1)
```

```
a
```

```
array([0. , 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9])
```

```
np.square(a)
```

```
array([0.  , 0.01, 0.04, 0.09, 0.16, 0.25, 0.36, 0.49, 0.64, 0.81])
```

```
np.sin(a)
```

```
array([0.        , 0.09983342, 0.19866933, 0.29552021, 0.38941834,
       0.47942554, 0.56464247, 0.64421769, 0.71735609, 0.78332691])
```

```
np.exp(a)
```

```
array([1.        , 1.10517092, 1.22140276, 1.34985881, 1.4918247 ,
       1.64872127, 1.8221188 , 2.01375271, 2.22554093, 2.45960311])
```

Linear algebra

```
# Define two matrices
A = np.array([[1, 2], [3, 4]])
B = np.array([[5, 6], [7, 8]])

# Matrix multiplication
C = np.dot(A, B)
# Or equivalently: C = A @ B

print(C)
```

```
[[19 22]
 [43 50]]
```

```
# Define a matrix
A = np.array([[1, 2], [3, 4]])

# Compute eigenvalues and eigenvectors
eigenvalues, eigenvectors = np.linalg.eig(A)

print("Eigenvalues:")
print(eigenvalues)

print("\nEigenvectors:")
print(eigenvectors)
```

```
Eigenvalues:
[-0.37228132  5.37228132]

Eigenvectors:
[[-0.82456484 -0.41597356]
 [ 0.56576746 -0.90937671]]
```

```
# Define a matrix
A = np.array([[1, 2, 3], [4, 5, 6]])

# Compute SVD
U, S, V = np.linalg.svd(A)

print("U:")
print(U)

print("\nS:")
print(S)

print("\nV:")
print(V)
```

```
U:
[[-0.3863177  -0.92236578]
 [-0.92236578  0.3863177 ]]

S:
[9.508032   0.77286964]

V:
[[-0.42866713 -0.56630692 -0.7039467 ]
 [ 0.80596391  0.11238241 -0.58119908]
 [ 0.40824829 -0.81649658  0.40824829]]
```

Pandas

```
!pip install pandas
```

```
Requirement already satisfied: pandas in /usr/local/lib/python3.10/dist-packages (1.5.3)
Requirement already satisfied: python-dateutil>=2.8.1 in /usr/local/lib/python3.10/dist-packages (from pandas
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas) (2022.7.
Requirement already satisfied: numpy>=1.21.0 in /usr/local/lib/python3.10/dist-packages (from pandas) (1.22.4
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.8
```

```
import pandas as pd
```

series

```
pd.Series([6,2,3])
```

```
0    6
1    2
2    3
dtype: int64
```

```
pd.Series(data=[6,2,3],index=['abc','def','ghi'])
```

```
abc    6
def    2
ghi    3
dtype: int64
```

```
a = pd.Series(data=[6,2,3],index=['abc','def','ghi'])
a['abc']
```

```
6
```

```
pd.Series([6,2,3],['abc','def','ghi'])
```

```
abc    6
def    2
ghi    3
dtype: int64
```

```
pd.Series(np.array([6,2,3]),['abc','def','ghi'])
```

```
abc    6
def    2
ghi    3
dtype: int64
```

```
dict = {'abc':6,'def':2,'ghi':3}
dict
```

```
{'abc': 6, 'def': 2, 'ghi': 3}
```

```
pd.Series(dict)
```

```
abc    6
def    2
ghi    3
dtype: int64
```

```
a = [2,3,5]
```

```
b = pd.Series([sum,min,max])
```

```
b[0](a)
```

```
10
```

```
b[1](a)
```

```
2
```

```
b[2](a)
```

```
5
```

```
a = {'a':5,'b':3,'c':10,'d':20}
b = {'a':2,'b':3,'c':10}
```

```
#a+b
```

```
pd.Series(a)+pd.Series(b)
```

```
a     7.0
b     6.0
c    20.0
d     NaN
dtype: float64
```

Link for other series function: https://pandas.pydata.org/docs/reference/series.html

DataFrame

Link for other DataFrame function:https://pandas.pydata.org/docs/reference/frame.html

```
pd.DataFrame(data=[[1,2,3],[4,5,6],[7,8,9]])
```

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

```
pd.DataFrame(data=[[1,2,3],[4,5,6],[7,8,9]],columns=['y1','y2','y3'])
```

	y1	y2	y3
0	1	2	3
1	4	5	6
2	7	8	9

```
pd.DataFrame(data=[[1,2,3],[4,5,6],[7,8,9]],index=['x1','x2','x3'])
```

	0	1	2
x1	1	2	3
x2	4	5	6
x3	7	8	9

```
pd.DataFrame(data=[[1,2,3],[4,5,6],[7,8,9]],index=['x1','x2','x3'],columns=['y1','y2','y3'])
```

	y1	y2	y3
x1	1	2	3
x2	4	5	6
x3	7	8	9

```
a = pd.DataFrame([[1,2,3],[4,5,6],[7,8,9]],['x1','x2','x3'],['y1','y2','y3'])
a
```

	y1	y2	y3
x1	1	2	3
x2	4	5	6
x3	7	8	9

Indexing

```
a['y1']
```

```
x1    1
x2    4
x3    7
Name: y1, dtype: int64
```

```
a.y1
```

```
x1    1
x2    4
x3    7
Name: y1, dtype: int64
```

```
a.loc['x1']
```

```
y1    1
y2    2
y3    3
Name: x1, dtype: int64
```

```
a.iloc[0]
```

```
y1    1
y2    2
y3    3
Name: x1, dtype: int64
```

```
a.loc['x1','y1']
```

```
1
```

```
a.iloc[1,1]
```

```
5
```

```
a.loc[['x1','x3'],['y1','y3']]
```

	y1	y3
x1	1	3
x3	7	9

```
a.iloc[[0,1],[0,1]]
```

	y1	y2
x1	1	2
x2	4	5

```
a
```

	y1	y2	y3
x1	1	2	3
x2	4	5	6
x3	7	8	9

```
# adding new column
a['sum']=a['y1']+a['y2']+a['y3']
a
```

	y1	y2	y3	sum
x1	1	2	3	6
x2	4	5	6	15
x3	7	8	9	24

```
#drop column
a.drop('y3',axis=1)
```

	y1	y2	sum
x1	1	2	6
x2	4	5	15
x3	7	8	24

```
#drop row
a.drop('x2',axis=0)
```

	y1	y2	y3	sum
x1	1	2	3	6
x3	7	8	9	24

```
#selection based on codition
a>5
```

	y1	y2	y3	sum
x1	False	False	False	True
x2	False	False	True	True
x3	True	True	True	True

```
a[a>5]
```

	y1	y2	y3	sum
x1	NaN	NaN	NaN	6
x2	NaN	NaN	6.0	15
x3	7.0	8.0	9.0	24

```
a[a['y2']>5]
```

	y1	y2	y3	sum
x3	7	8	9	24

```
a[(a['y2']>2) & (a['y1']<5) ]
```

	y1	y2	y3	sum
x2	4	5	6	15

```
a
```

	y1	y2	y3	sum
x1	1	2	3	6
x2	4	5	6	15
x3	7	8	9	24

```
a.reset_index()
```

	index	y1	y2	y3	sum
0	x1	1	2	3	6
1	x2	4	5	6	15
2	x3	7	8	9	24

```
a.set_index('sum')
```

sum	y1	y2	y3
6	1	2	3
15	4	5	6
24	7	8	9

```
a
```

	y1	y2	y3	sum
x1	1	2	3	6
x2	4	5	6	15
x3	7	8	9	24

```
b = a.set_index('sum')
```

```
b
```

sum	y1	y2	y3
6	1	2	3
15	4	5	6
24	7	8	9

```
a.set_index('sum',inplace=True)
```

```
a
```

sum	y1	y2	y3
6	1	2	3
15	4	5	6
24	7	8	9

Multi index

```
first = [1,1,1,1,2,2,2,2]
second = [1,1,2,2,1,1,2,2]
third = [1,2,1,2,1,2,1,2]
ind = list(zip(first,second,third))
ind = pd.MultiIndex.from_tuples(ind)
```

```
ind
```

```
MultiIndex([(1, 1, 1),
            (1, 1, 2),
            (1, 2, 1),
            (1, 2, 2),
            (2, 1, 1),
            (2, 1, 2),
            (2, 2, 1),
```

```
                 (2, 2, 2)],
                )

a = pd.DataFrame(np.random.rand(8,3),index=ind,columns=['x1','x2','x3'])
a
```

			x1	x2	x3
1	1	1	0.878336	0.082351	0.487742
		2	0.147995	0.893605	0.772840
	2	1	0.830970	0.926988	0.392117
		2	0.330989	0.414220	0.272215
2	1	1	0.644592	0.741119	0.077986
		2	0.319575	0.808051	0.874261
	2	1	0.504163	0.620544	0.537563
		2	0.917022	0.473722	0.063825

```
a.loc[1]
```

		x1	x2	x3
1	1	0.878336	0.082351	0.487742
	2	0.147995	0.893605	0.772840
2	1	0.830970	0.926988	0.392117
	2	0.330989	0.414220	0.272215

```
a.loc[1].loc[2]
```

	x1	x2	x3
1	0.830970	0.926988	0.392117
2	0.330989	0.414220	0.272215

```
a.loc[1].loc[2].loc[1]
```

```
x1    0.830970
x2    0.926988
x3    0.392117
Name: 1, dtype: float64
```

```
a.index.names = ['first','second','third']
```

```
a
```

			x1	x2	x3
first	second	third			
1	1	1	0.878336	0.082351	0.487742
		2	0.147995	0.893605	0.772840
	2	1	0.830970	0.926988	0.392117
		2	0.330989	0.414220	0.272215
2	1	1	0.644592	0.741119	0.077986
		2	0.319575	0.808051	0.874261
	2	1	0.504163	0.620544	0.537563
		2	0.917022	0.473722	0.063825

Cleaning of data

```
a = pd.DataFrame(np.random.randn(5,5),'x1 x2 x3 x4 x5'.split(),'y1 y2 y3 y4 y5'.split())
a
```

	y1	y2	y3	y4	y5
x1	-0.365732	-1.307405	0.251112	1.874596	-0.113036
x2	0.460620	0.129813	-0.438440	-0.594402	0.396822
x3	-0.342980	1.617766	0.002839	1.366161	1.034266
x4	-0.221036	0.574467	0.863552	1.589273	0.245182
x5	-0.402164	-0.604978	1.275178	0.222002	0.012571

```
c = a[a>-1]
c
```

	y1	y2	y3	y4	y5
x1	-0.365732	NaN	0.251112	1.874596	-0.113036
x2	0.460620	0.129813	-0.438440	-0.594402	0.396822
x3	-0.342980	1.617766	0.002839	1.366161	1.034266
x4	-0.221036	0.574467	0.863552	1.589273	0.245182
x5	-0.402164	-0.604978	1.275178	0.222002	0.012571

```
c.dropna()
```

	y1	y2	y3	y4	y5
x2	0.460620	0.129813	-0.438440	-0.594402	0.396822
x3	-0.342980	1.617766	0.002839	1.366161	1.034266
x4	-0.221036	0.574467	0.863552	1.589273	0.245182
x5	-0.402164	-0.604978	1.275178	0.222002	0.012571

```
b = pd.DataFrame(np.random.randn(5,5),'x1 x2 x3 x4 x5'.split(),'y1 y2 y3 y4 y5'.split())
b
```

	y1	y2	y3	y4	y5
x1	0.245216	-0.709425	0.051923	-1.224304	0.848140
x2	-0.392528	0.347231	-0.873556	-0.845489	-2.080556
x3	-0.682184	-0.036344	0.672652	0.420687	0.817973
x4	-0.673642	-2.680293	0.031762	0.788138	-0.310464
x5	0.903321	0.864015	-0.142097	0.533560	1.097787

```
b.iloc[1,4]=0
b.iloc[3,3]=0
b
```

	y1	y2	y3	y4	y5
x1	0.245216	-0.709425	0.051923	-1.224304	0.848140
x2	-0.392528	0.347231	-0.873556	-0.845489	0.000000
x3	-0.682184	-0.036344	0.672652	0.420687	0.817973
x4	-0.673642	-2.680293	0.031762	0.000000	-0.310464
x5	0.903321	0.864015	-0.142097	0.533560	1.097787

```
c = a/b
c
```

	y1	y2	y3	y4	y5
x1	-1.491470	1.842907	4.836213	-1.531152	-0.133276
x2	-1.173468	0.373851	0.501903	0.703028	inf
x3	0.502768	-44.512395	0.004221	3.247453	1.264426
x4	0.328120	-0.214330	27.188111	inf	-0.789728
x5	-0.445206	-0.700194	-8.973992	0.416077	0.011451

```
c.replace([np.inf,-np.inf],np.nan,inplace=True)
c
```

	y1	y2	y3	y4	y5
x1	-1.491470	1.842907	4.836213	-1.531152	-0.133276
x2	-1.173468	0.373851	0.501903	0.703028	NaN
x3	0.502768	-44.512395	0.004221	3.247453	1.264426
x4	0.328120	-0.214330	27.188111	NaN	-0.789728
x5	-0.445206	-0.700194	-8.973992	0.416077	0.011451

```
c.dropna()
```

	y1	y2	y3	y4	y5
x1	-1.491470	1.842907	4.836213	-1.531152	-0.133276
x3	0.502768	-44.512395	0.004221	3.247453	1.264426
x5	-0.445206	-0.700194	-8.973992	0.416077	0.011451

```
c = a/b
c
```

	y1	y2	y3	y4	y5
x1	-1.491470	1.842907	4.836213	-1.531152	-0.133276
x2	-1.173468	0.373851	0.501903	0.703028	inf
x3	0.502768	-44.512395	0.004221	3.247453	1.264426
x4	0.328120	-0.214330	27.188111	inf	-0.789728
x5	-0.445206	-0.700194	-8.973992	0.416077	0.011451

```
c.replace([np.inf,-np.inf],np.nan).dropna()
```

	y1	y2	y3	y4	y5
x1	-1.491470	1.842907	4.836213	-1.531152	-0.133276
x3	0.502768	-44.512395	0.004221	3.247453	1.264426
x5	-0.445206	-0.700194	-8.973992	0.416077	0.011451

```
c = a/b
c.replace([np.inf,-np.inf],np.nan,inplace=True)
```

```
c.dropna(axis=1)
```

	y1	y2	y3
x1	-1.491470	1.842907	4.836213
x2	-1.173468	0.373851	0.501903
x3	0.502768	-44.512395	0.004221
x4	0.328120	-0.214330	27.188111
x5	-0.445206	-0.700194	-8.973992

```
c.fillna(c.mean())
```

	y1	y2	y3	y4	y5
x1	-1.491470	1.842907	4.836213	-1.531152	-0.133276
x2	-1.173468	0.373851	0.501903	0.703028	0.088218
x3	0.502768	-44.512395	0.004221	3.247453	1.264426
x4	0.328120	-0.214330	27.188111	0.708851	-0.789728
x5	-0.445206	-0.700194	-8.973992	0.416077	0.011451

Data loading and Inspection

```
# Here, student data stored in STUDENT_DATA.csv
a = pd.read_csv("STUDENT_DATA.csv")
```

```
a
```

	student	subject	grade	marks
0	STD1	SUB1	AA	10
1	STD1	SUB2	AB	9
2	STD1	SUB3	BB	8
3	STD2	SUB1	BB	8
4	STD2	SUB2	AA	10
5	STD2	SUB3	AA	10
6	STD3	SUB1	BB	8
7	STD3	SUB2	AB	9
8	STD3	SUB3	AA	10

```
a.to_csv('demo',index=False)
```

```
b = a.groupby('student')
```

```
b
```

```
<pandas.core.groupby.generic.DataFrameGroupBy object at 0x7e98be949db0>
```

```
b.describe()
```

	marks							
	count	mean	std	min	25%	50%	75%	max
student								
STD1	3.0	9.000000	1.000000	8.0	8.5	9.0	9.5	10.0
STD2	3.0	9.333333	1.154701	8.0	9.0	10.0	10.0	10.0
STD3	3.0	9.000000	1.000000	8.0	8.5	9.0	9.5	10.0

```
a.groupby('subject').describe()
```

	marks							
	count	mean	std	min	25%	50%	75%	max
subject								
SUB1	3.0	8.666667	1.154701	8.0	8.0	8.0	9.0	10.0
SUB2	3.0	9.333333	0.577350	9.0	9.0	9.0	9.5	10.0
SUB3	3.0	9.333333	1.154701	8.0	9.0	10.0	10.0	10.0

```
a.groupby('grade').describe()
```

	marks							
	count	mean	std	min	25%	50%	75%	max
grade								
AA	4.0	10.0	0.0	10.0	10.0	10.0	10.0	10.0
AB	2.0	9.0	0.0	9.0	9.0	9.0	9.0	9.0
BB	3.0	8.0	0.0	8.0	8.0	8.0	8.0	8.0

Concat

```
a = pd.DataFrame(np.random.randint(1,4,[3,3]))
a
```

	0	1	2
0	3	3	3
1	1	1	2
2	3	1	2

```
b = pd.DataFrame(np.random.randint(3,6,[3,3]))
b
```

	0	1	2
0	5	4	4
1	5	3	3
2	5	3	4

```
c = pd.DataFrame(np.random.randint(5,8,[3,3]))
c
```

	0	1	2
0	6	6	6
1	7	6	6
2	7	6	5

```
d = pd.concat([a,b,c])
d
```

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

```
e = pd.concat([a,b,c],axis=1)
e
```

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

Merge

```
a = pd.read_csv("STUDENT_DATA.csv")
a
```

	student	subject	grade	marks
0	STD1	SUB1	AA	10
1	STD1	SUB2	AB	9
2	STD1	SUB3	BB	8
3	STD2	SUB1	BB	8
4	STD2	SUB2	AA	10
5	STD2	SUB3	AA	10
6	STD3	SUB1	BB	8
7	STD3	SUB2	AB	9
8	STD3	SUB3	AA	10

```
b = a.set_index('student')
b
```

student	subject	grade	marks
STD1	SUB1	AA	10
STD1	SUB2	AB	9
STD1	SUB3	BB	8
STD2	SUB1	BB	8
STD2	SUB2	AA	10
STD2	SUB3	AA	10
STD3	SUB1	BB	8
STD3	SUB2	AB	9
STD3	SUB3	AA	10

```
c = b.loc['STD1']
c = c.reset_index()
c
```

	student	subject	grade	marks
0	STD1	SUB1	AA	10
1	STD1	SUB2	AB	9
2	STD1	SUB3	BB	8

```
d = b.loc['STD2']
d = d.reset_index()
d
```

	student	subject	grade	marks
0	STD2	SUB1	BB	8
1	STD2	SUB2	AA	10
2	STD2	SUB3	AA	10

```
e = pd.merge(c,d,on='subject')
e
```

	student_x	subject	grade_x	marks_x	student_y	grade_y	marks_y
0	STD1	SUB1	AA	10	STD2	BB	8
1	STD1	SUB2	AB	9	STD2	AA	10
2	STD1	SUB3	BB	8	STD2	AA	10

join

```
a = pd.DataFrame({'STD1':['AA','AB','BB']},index=['SUB1','SUB2','SUB3'])
a
```

	STD1
SUB1	AA
SUB2	AB
SUB3	BB

```
b = pd.DataFrame({'STD2':['BB','AA','AA']},index=['SUB1','SUB2','SUB3'])
b
```

	STD2
SUB1	BB
SUB2	AA
SUB3	AA

```
a.join(b)
```

	STD1	STD2
SUB1	AA	BB
SUB2	AB	AA
SUB3	BB	AA

Operations

```
a = pd.read_csv("olympic.csv",engine='python')
a
```

```
---------------------------------------------------------------------------
UnicodeDecodeError                        Traceback (most recent call last)
<ipython-input-477-ea35a21d09a4> in <cell line: 1>()
----> 1 a = pd.read_csv("olympic.csv",engine='python')
      2 a

11 frames
/usr/lib/python3.10/codecs.py in decode(self, input, final)
    320         # decode input (taking the buffer into account)
    321         data = self.buffer + input
--> 322         (result, consumed) = self._buffer_decode(data, self.errors, final)
    323         # keep undecoded input until the next call
    324         self.buffer = data[consumed:]

UnicodeDecodeError: 'utf-8' codec can't decode byte 0xf4 in position 1978: invalid continuation byte
```

```
a.head()
```

```
a['rank'].unique() #unique ranks
```

```
a['rank'].nunique() #number of unique ranks
```

```
a['rank'].value_counts()
```

```
a['gold'].mean()
```

```
a[(a['gold']==a['silver']) | (a['gold']==a['bronze'])]
```

```
a['gold'].max()
```

```
a[a['gold']==a['gold'].max()]
```

```
country_10gold_10silver_10bronze = a[(a['gold']>10)&(a['silver']>10)&(a['bronze']>10)]
country_10gold_10silver_10bronze
```

```
country_5gold_5silver_5bronze = a[(a['gold']>5)&(a['silver']>5)&(a['bronze']>5)]
country_5gold_5silver_5bronze
```

```
tot = a[a['country']=='India'].total
```

```
d = a[a['total']<int(tot)]
```

```
d.count()[0]
```

```
a.sort_values(by='gold',ascending=False)
```

```
a = pd.read_csv("STUDENT_DATA.csv")
a.drop('grade',axis=1,inplace=True)
a
```

```
a.pivot_table(values='marks',index='student',columns='subject')
```

Matplotlib

```
!pip install matplotlib


import matplotlib.pyplot as plt
%matplotlib inline


x = np.linspace(0,15,100)
y = np.sin(x)


plt.plot(x,y,'b')
plt.xlabel('time')
plt.ylabel('amplitude')
plt.title('Sine Wave')
plt.show()


z = np.cos(x)


plt.subplot(2,1,1)
plt.plot(x,y,'--b')
plt.ylabel('amplitude')
plt.title('Sine Wave')
plt.subplot(2,1,2)
plt.plot(x,y,'-*r')
plt.xlabel('time')
plt.ylabel('amplitude')
plt.title('Cos Wave')
plt.show()


fig = plt.figure()

ax = fig.add_axes([0,0,1,1])

ax.plot(x, x**2, label="x**2")
ax.plot(x, x**3, label="x**3")
ax.legend()


data = np.random.normal(0, 1, 1000)


plt.hist(data)


x = np.linspace(0,5,100)
fig, axes = plt.subplots(1, 2, figsize=(10,4))

axes[0].plot(x, x**2, x, np.exp(x))
axes[0].set_title("Normal scale")

axes[1].plot(x, x**2, x, np.exp(x))
axes[1].set_yscale("log")
axes[1].set_title("Logarithmic scale (y)");


fig, ax1 = plt.subplots()

ax1.plot(x, x**2, lw=2, color="blue")
ax1.set_ylabel(r"area $(m^2)$", fontsize=18, color="blue")
for label in ax1.get_yticklabels():
    label.set_color("blue")

ax2 = ax1.twinx()
ax2.plot(x, np.exp(x), lw=2, color="red")
ax2.set_ylabel(r"volume $(m^3)$", fontsize=18, color="red")
for label in ax2.get_yticklabels():
    label.set_color("red")
```

```
data = {'x': [1, 2, 3, 4, 5], 'y': [2, 4, 6, 8, 10]}
df = pd.DataFrame(data)

# Plotting a line plot
df.plot(x='x', y='y')
plt.show()
```

seaborn

```
!pip install seaborn
```

```
import seaborn as sns
```

```
tips = sns.load_dataset('tips')
```

```
tips.head()
```

```
sns.distplot(tips['total_bill'])
```

```
sns.scatterplot(data=tips, x="total_bill", y="tip")
```

```
sns.jointplot(x='total_bill',y='tip',data=tips,kind='scatter')
```

```
sns.jointplot(x='total_bill',y='tip',data=tips,kind='hex')
```

```
sns.pairplot(tips)
```

```
a = pd.read_csv("olympic.csv",engine='python')
a.head()
```

```
sns.pairplot(a)
```

```
sns.pairplot(tips,hue='sex',palette='coolwarm')
```

```
sns.barplot(x='day',y='tip',hue='time',data=tips)
```

```
sns.barplot(x='sex',y='tip',hue='day',data=tips)
```

```
sns.boxplot(x="day", y="total_bill",hue='time', data=tips)
```

```
sns.violinplot(x="day", y="total_bill", hue='smoker',data=tips)
```

```
tips.corr()
```

```
sns.heatmap(tips.corr())
```

```
flights = sns.load_dataset('flights')
```

```
flights.head()
```

```
flights.pivot_table(values='passengers',index='month',columns='year')
```

```
pvflights = flights.pivot_table(values='passengers',index='month',columns='year')
sns.heatmap(pvflights)
```

```
sns.clustermap(pvflights)
```

sklearn

```
!pip install sklearn
```

linear regression

```
a = np.random.rand(1000,1)
b = np.random.rand(1000,1)
y = 2*a+5*b+3
d = np.concatenate((a,b,y),axis=1)
d.shape
```

```
data = pd.DataFrame(data=d,columns=['a','b','y'])
```

```
data.head()
```

```
sns.pairplot(data,palette='coolwarm')
```

```
sns.heatmap(data.corr())
```

```
X = data[['a','b']]
y = data['y']
```

```
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.4)
```

```
model = LinearRegression()
```

```
model.fit(X_train,y_train)
```

```
model.intercept_
```

```
model.coef_
```

```
predictions = model.predict(X_test)
```

```
plt.scatter(y_test,predictions)
```

```
X = data[['a','b']]
y = data['y']
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.4)
model = LinearRegression()
model.fit(X_train,y_train)
predictions = model.predict(X_test)
plt.scatter(y_test,predictions)
```

logistic regression

```
iris = sns.load_dataset('iris')
iris.head()
```

```
sns.pairplot(iris,hue='species',palette='coolwarm')
```

```
X = iris[['sepal_length','sepal_width','petal_length','petal_width']]
y = iris['species']
```

```
from sklearn.linear_model import LogisticRegression
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.4)
```

```
model = LogisticRegression()
```

```
model.fit(X_train,y_train)
```

```
prediction = model.predict(X_test)
```

```
from sklearn.metrics import classification_report
print(classification_report(y_test,prediction))
```

```
prediction
```

K- means

```
a = pd.DataFrame([1,2,3,4,5,6,15,16,17,25,26,23])
```

```
from sklearn.cluster import KMeans
kmeans = KMeans(n_clusters=3)
kmeans.fit(a)
labels = kmeans.predict(a)
labels
```

tensorflow

```
#!pip install tensorflow
```

```
import tensorflow as tf
```

```
# Define the inputs
a = tf.constant(5.0)
b = tf.constant(3.0)

# Create the computation graph
c = tf.add(a, b)
```

small neural net

```
print("Result:", c.numpy())
```

```
mnist = tf.keras.datasets.mnist

(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0
```