Assignment -1

WRITE PYTHON PROGRAMS FOR THE FOLLOWING QUESTIONS

1. Show that a List in python can

- **store elements of different types (integer, float, string, etc.)**

Code:

```
list1=[1,2,3,"vivian","serrao",True,9.87]
print("List elements: ",list1)
```

Output:

```
List elements:  [1, 2, 3, 'vivian', 'serrao', True, 9.87]
```

- **store duplicate elements**

Code:

```
list1=[1,2,3,4,1,2,3,5,6,87]
print("Duplicate elements list",list1)
```

Output:

```
Duplicate elements list [1, 2, 3, 4, 1, 2, 3, 5, 6, 87]
```

2. Check if an Element Exists in a List (Note make use of 'in')

Code:

```
list1=[1,2,3,4,5,6,7,8]
a=int(input("Enter the search element: "))
if a in list1:
        print(a, " is present in the list")
else:
        print("Element not found")
```

Output:

```
Enter the search element: 5
5  is present in the list
```

3. Find the length of the list.

Code:

```
list1=[1,2,3,4,5]
print("Length of list:",len(list1))
```

Output:

```
Length of list: 5
```

4. Create a list with value n ** 2 where n is a number from 1 to 5

Code:
```
list1=[]
for i in range(1,6):
        list1.append(i **2)
print(list1)
```

Output:
[1, 4, 9, 16, 25]

5. Python program to print the elements of an array in reverse order

Code:
```
list1=[1,2,3,4,5,6,7,8]
print("Elements in reverse order", list1[::-1])
```

Output:
Elements in reverse order [8, 7, 6, 5, 4, 3, 2, 1]

7. Python program to sort the elements of an array in descending order

Code:
```
list1=[4,6,1,2,3,8,9,0]
list1.sort(reverse=True)
print("Array in descending order: ",list1)
```

Array in descending order: [9, 8, 6, 4, 3, 2, 1, 0]

8. Compare 2 lists in python(use The cmp() function

- **The set() function and == operator**

Code:
```
list1 = [1, 2, 3, 3]
list2 = [3, 2, 1]
list3 = [1, 2, 3, 4]

print(set(list1) == set(list2))
print(set(list1) == set(list3))
```

Output:
True
False

- **The sort() function and == operator)**

Code:
```
list1 = [3, 1, 2]
list2 = [1, 2, 3]
```

list3 = [1, 2, 4]

list1.sort()
list2.sort()
list3.sort()

print(list1 == list2)
print(list1 == list3)

Output:
True
False

9. Remove a specific item from a list by using the three methods remove(), pop(), and clear().

Code:
```
list1=[1,2,3,4,5,6,7,8]
list1.remove(3)
print(list1)

list1.pop()
print(list1)

list1.clear()
print(list1)
```

Output:
```
[1, 2, 4, 5, 6, 7, 8]
[1, 2, 4, 5, 6, 7]
[]
```

10.Write a NumPy program to convert an integer array to a floating type.

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

Output:
```
[1. 2. 3. 4. 5. 6.]
```

11.Write a NumPy program to convert a list and tuple into arrays.

Code:

```
list1=np.array(arr)
print("list array" , list1)

tuple1=tuple(np.array(arr))
print("Tuple array" , tuple1)
```

Output:

list array [1 2 3 4 5 6]
Tuple array (1, 2, 3, 4, 5, 6)

12. Write a NumPy program to convert a list and tuple into arrays.(np.asarray)

Code:

```
list1=np.asarray(arr)
print("list array" , list1)

tuple1=tuple(np.asarray(arr))
print("Tuple array" , tuple1)
```

Output:

list array [1 2 3 4 5 6]
Tuple array (1, 2, 3, 4, 5, 6)

Assignment 2

1.Write Python program to find the LCM

Code:

```
import math

def lcm(a, b):
    return abs(a * b) // math.gcd(a, b)

a = int(input("Enter First Number: "))
b = int(input("Enter Second Number: "))

print("LCM: ", lcm(a, b))
```

Output:

Enter First Number: 10
Enter Second Number: 20
LCM: 20

2. Write a Python Program to Make a Simple Calculator

Code:

```
num1 = int(input("Enter First Number: "))
operation = input("Enter the Operation: ")
num2 = int(input("Enter Second Number: "))
if operation=="+":
    print("Answer is",num1+num2)
elif operation=="-":
    print("Answer is",num1-num2)
elif operation=="*":
    print("Answer is",num1*num2)
elif operation=="/":
    if num2 !=0:
        print("Answer is",num1/num2)
    else:
        print("Cannot devide it by zero")
```

Output:

```
Enter First Number: 10
Enter the Operation: /
Enter Second Number: 0
Cannot devide it by zero
```

3.Program to Merge Two Lists

Code:

```
n1 = int(input("Enter length of First List: "))
n2 = int(input("Enter Length of Second List: "))
list1=[]
list2=[]

print("Enter the First List Element")
for i in range(n1):
    list1.append(input())

print("Enter the Second List Element")
for i in range(n2):
    list2.append(input())

list3=list1+list2
print("Merged List is:",list3)
```

Output:

```
Enter length of First List: 5
```

Enter Length of Second List: 5
Enter the First List Element
1
2
3
4
5
Enter the Second List Element
6
7
8
9
10
Merged List is: ['1', '2', '3', '4', '5', '6', '7', '8', '9', '10']

4. Demonstrate the difference between list and tuple

Code:

```
list1=[1,2,3] #Created using [] bracket
list1.append(1) #can add element
list1.remove(2) #can remove element
list1[1]=2 #can change element
print(list1)

tuple1=(1,2,3) #created using () bracket
tuple1.append(4) #cannot add element
tuple1.remove(1) #cannot remove element
tuple1[0]=2 #cannot modify element
print(tuple1)
```

Output:

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

5. Try the following on tuple

Code:

```
tuple1=(1,2,3,6,5,4)
print(len(tuple1))
print(max(tuple1))
print(min(tuple1))
print(sum(tuple1))
print(sorted(tuple1))
```

Output:

```
6
```

```
6
1
21
[1, 2, 3, 4, 5, 6]
```

6.Demonstrate the following

Code:

```
tuple1=(1,3,5,7,9)
list1=list(tuple1)
print(list1)

tuple2=tuple(list1)
print(tuple2)

str1=str(tuple2)
print(str1)
```

Output:

```
[1, 3, 5, 7, 9]
(1, 3, 5, 7, 9)
(1, 3, 5, 7, 9)
```

7. Demonstrate the following on dictionary

Code:

```
dict1 = dict()
print("Initialized empty dictionary:", dict1)

dict2 = {1: 'vivian', 2: 'serrao', 3: 'mca'}
print("Initialized dictionary with key-value pairs:", dict2)

dict1 = dict2.copy()
print("Copied dict2 into dict1:", dict1)

dict1.clear()
print("Cleared all items from dict1:", dict1)

print("Value for key 1 in dict2:", dict2.get(1))

print("All key-value pairs in dict2:", dict2.items())

print("All keys in dict2:", dict2.keys())

dict2.pop(3)
print("Removed item with key 3 from dict2:", dict2)
```

```
dict2.popitem()
print("Removed and returned an arbitrary (key, value) pair from dict2:", dict2)

dict3 = {4: 'student'}
dict2.update(dict3)
print("Updated dict2 with dict3:", dict2)

print("All values in dict2:", dict2.values())

#fromkeys()
keys = [1, 2, 3]
default_value = 'default'
new_dict = dict.fromkeys(keys, default_value)

print("Dictionary created with fromkeys() and default value 'default':", new_dict)

# setdefault()
dict1 = {1: 'a', 2: 'b'}

value1 = dict1.setdefault(2, 'default_value')
value2 = dict1.setdefault(3, 'default_value')

print("Dictionary after using setdefault():", dict1)
```

Output:

```
Initialized empty dictionary: {}
Initialized dictionary with key-value pairs: {1: 'vivian', 2: 'serrao', 3: 'mca'}
Copied dict2 into dict1: {1: 'vivian', 2: 'serrao', 3: 'mca'}
Cleared all items from dict1: {}
Value for key 1 in dict2: vivian
All key-value pairs in dict2: dict_items([(1, 'vivian'), (2, 'serrao'), (3, 'mca')])
All keys in dict2: dict_keys([1, 2, 3])
Removed item with key 3 from dict2: {1: 'vivian', 2: 'serrao'}
Removed and returned an arbitrary (key, value) pair from dict2: {1: 'vivian'}
Updated dict2 with dict3: {1: 'vivian', 4: 'student'}
All values in dict2: dict_values(['vivian', 'student'])
Dictionary created with fromkeys() and default value 'default': {1: 'default', 2: 'default', 3: 'default'}
Dictionary after using setdefault(): {1: 'a', 2: 'b', 3: 'default_value'}
```

8. Perform positive indexing, slicing, negative indexing on lists

Code:

```
list1 = ['a', 'b', 'c', 'd', 'e', 'f']

print("Positive Indexing:")
print("Element at index 0:", list1[0])
print("Element at index 5:", list1[5])

print("Negative Indexing:")
print("Last element:", list1[-1])
print("Second to last element:", list1[-2])

print("Slicing:")
print("Elements from index 0 to 2 :", list1[0:3])
print("Elements from index 3 to 5 :", list1[3:6])
print("Elements in reverse order:", list1[::-1])
print("Elements in reverse order and skipping 1 element:", list1[::-2])
```

Output:

```
Positive Indexing:
Element at index 0: a
Element at index 5: f
Negative Indexing:
Last element: f
Second to last element: e
Slicing:
Elements from index 0 to 2 : ['a', 'b', 'c']
Elements from index 3 to 5 : ['d', 'e', 'f']
Elements in reverse order: ['f', 'e', 'd', 'c', 'b', 'a']
Elements in reverse order and skipping 1 element: ['f', 'd', 'b']
```

9.Program to find the second largest and second smallest in one-dimensional array.

Code:

```
arr=[1,2,1,2,3,4,5,6,7,3,2,6,10,8]
arr=list(set(arr))
arr.sort()
print("Second Smallest Element: ",arr[1])
print("Second Largest Element: ",arr[-2])
```

Output:

```
Second Smallest Element:  2
Second Largest Element:  8
```

10.Accept a number and display if it is odd or even. (use only one if block without

else)

Code:

```
def check(num):
    if num %2==0:
        return "even"
    return "odd"
num=int(input("Enter a Number :"))
print("Number is",check(num))
```

Output:

```
Enter a Number :15
Number is odd
```

ASSIGNMENT-3

1. Write a program that asks the user to input his name and print its initials. Assuming that the user always types first name, middle name and last name and does not include any unnecessary spaces.

For example, if the user enters Ajay Kumar Garg the program should display A. K. G.

Code:

```
first_name=input("Enter First Name: ")
middle_name=input("Enter Middle Name: ")
last_name=input("Enter Last Name: ")
initials=first_name[0]+"."+middle_name[0]+"."+last_name[0]+"."
print(initials)
```

Output:

```
Enter First Name: Vivian
Enter Middle Name: Serrao
Enter Last Name: Badyar
V.S.B.
```

2. Write a program in python that accepts a string to setup a passwords. Your entered password must meet the following requirements:

- **The password must be at least eight characters long.**
- **It must contain at least one uppercase letter.**
- **It must contain at least one lowercase letter.**
- **It must contain at least one numeric digit.**

Your program should should perform this validation.

Code:

```
password=input("Enter the Password: ")
upper=False
lower=False
numeric=False
if len(password)>=8:
   for i in password:
      if i.isupper():
         upper=True
      if i.islower():
         lower=True
      if i.isdigit():
         numeric=True
if lower==upper==numeric==True:
     print("Password is valid.")
else:
      print("Password is invalid.")
```

Output:

```
Enter the Password: Vivian123
Password is valid.
```

3. Write a Python program that accepts a string from user. Your program should create and display a new string where the first and last characters have been exchanged.

Code:

```
s1=input("Enter a String: ")
s2=s1[-1]+s1[1:-1]+s1[0]
print(s2)
```

Output:

```
Enter a String: Ilove Python
nlove PythoI
```

4. Write a program that accepts a string from user. Your program should count and display number of vowels in that string.

Code:

```
s1=input("Enter a String: ")
vowels=['a','e','i','o','u']
cnt=0
for i in s1:
   if i.lower() in vowels:
```

```
        cnt+=1
print("Number of vowels are:",cnt)
```

Output:

Enter a String: Artificial Intelligence
Number of vowels are: 10

5. Write a program that reads a string from keyboard and display:
*** The number of uppercase letters in the string**
*** The number of lowercase letters in the string**
*** The number of digits in the string**

Code:

```
s1=input("Enter a String: ")
upper=0
lower=0
digits=0
for i in s1:
    if i.isupper():
        upper+=1
    if i.islower():
        lower+=1
    if i.isdigit():
        digits+=1
print("Number of UpperCase Letters are:",upper)
print("Number of LowerCase Letters are:",lower)
print("Number of Digits are:",digits)
```

Output:

Enter a String: I am Studying In 2 year MCA
Number of UpperCase Letters are: 6
Number of LowerCase Letters are: 14
Number of Digits are: 1

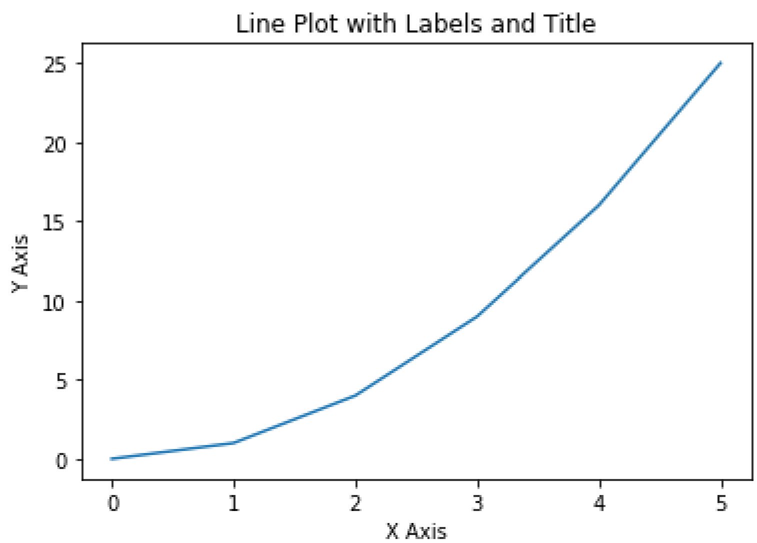
Assignment - 4

1 Write a Python program to draw a line with suitable label in the x axis, y axis and a title.

Code:

```
import matplotlib.pyplot as plt
x = [0, 1, 2, 3, 4, 5]
y = [0, 1, 4, 9, 16, 25]

plt.xlabel('X Axis')
plt.ylabel('Y Axis')
plt.title('Line Plot with Labels and Title')
plt.plot(x, y);
```

Output:



2. Write a Python program to draw a line using given axis values taken from a text file, with suitable label in the x axis, y axis and a title.

test.txt

1 2

2 4

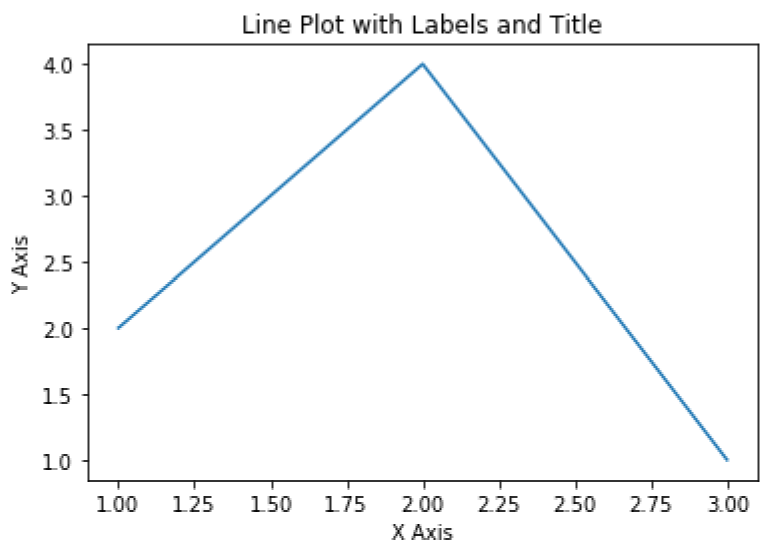
3 1

Code:

```
import matplotlib.pyplot as plt
x=[]
y=[]

with open("Test.txt", "r") as file:
    for line in file:
        part=line.split()
```

```
        if len(part) == 2:
            x.append(int(part[0]))
            y.append(int(part[1]))

plt.xlabel('X Axis')
plt.ylabel('Y Axis')
plt.title('Line Plot with Labels and Title')
plt.plot(x, y);
```

Output:



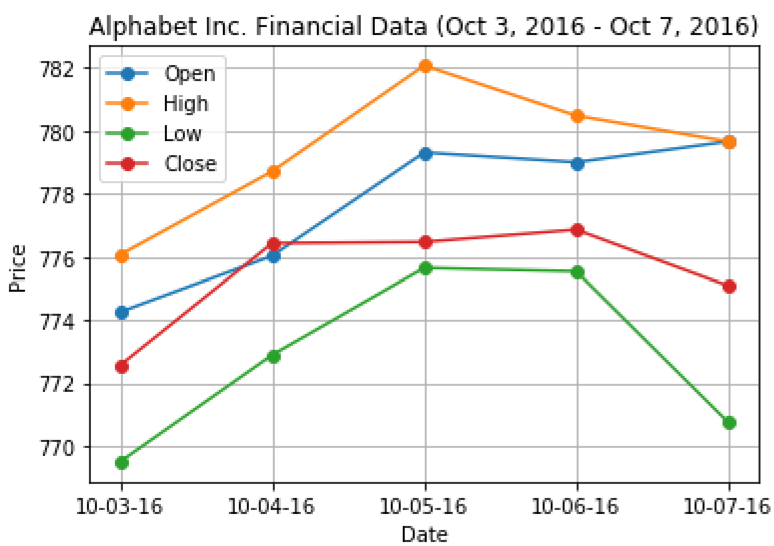
3 Write a Python program to draw line charts of the financial data of Alphabet Inc. between October 3, 2016 to October 7, 2016.
Date,Open,High,Low,Close
10-03-16,774.25,776.065002,769.5,772.559998
10-04-16,776.030029,778.710022,772.890015,776.429993
10-05-16,779.309998,782.070007,775.650024,776.469971
10-06-16,779,780.47998,775.539978,776.859985
10-07-16,779.659973,779.659973,770.75,775.080017

Code:

```
import matplotlib.pyplot as plt

date = ['10-03-16','10-04-16','10-05-16','10-06-16','10-07-16']
open = [774.25, 776.030029, 779.309998, 779, 779.659973]
high = [776.065002, 778.710022, 782.070007, 780.47998, 779.659973]
low = [769.5, 772.890015, 775.650024, 775.539978, 770.75]
close= [772.559998, 776.429993, 776.469971, 776.859985, 775.080017]
```

```
plt.plot(date, open, marker='o', label='Open')
plt.plot(date, high, marker='o', label='High')
plt.plot(date, low, marker='o', label='Low')
plt.plot(date, close, marker='o', label='Close')

plt.xlabel('Date')
plt.ylabel('Price')
plt.title('Alphabet Inc. Financial Data (Oct 3, 2016 - Oct 7, 2016)')
plt.legend()
plt.grid(True)
```

Output:



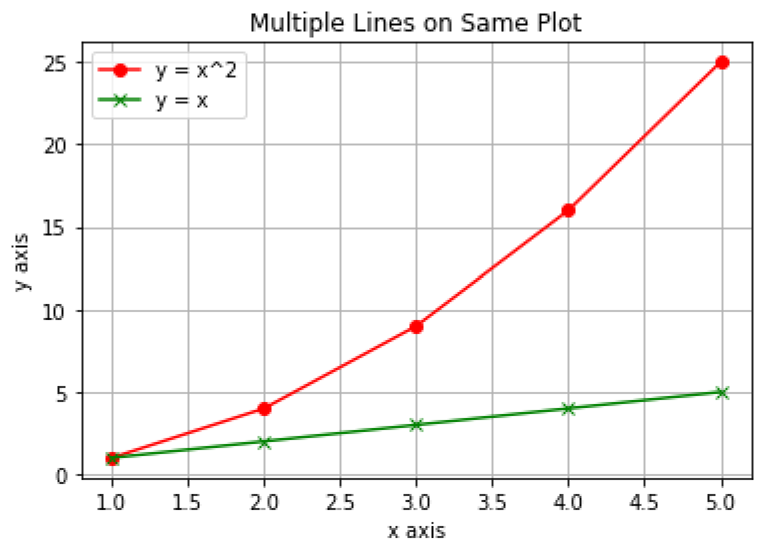
4 Write a Python program to plot two or more lines on same plot with suitable legends of each line.

Code:

```
import matplotlib.pyplot as plt

x=[1,2,3,4,5]
y1=[1,4,9,16,25]
y2=[1,2,3,4,5]

plt.plot(x,y1,marker='o',c='r',label='y = x^2')
plt.plot(x,y2,marker='x',c='g',label='y = x')

plt.xlabel('x axis')
```

```
plt.ylabel('y axis')
plt.title('Multiple Lines on Same Plot')
plt.legend()
plt.grid(True)
```

Output:



5. Write a Pandas program to create and display a one-dimensional array-like object(series) containing an array of data using Pandas module. Accept the values for the series from the keyboard. Display the content.

Code:

```
import pandas as pd
value=input("Enter the values for the series by giving space: ").split()
list1=[]
for i in value:
   list1.append(i)

series=pd.Series(list1)
print("Pandas Series:")
print(series)
```

```
Enter the values for the series by giving space: 77 55 33 22 11 22
Pandas Series:
0    77
1    55
2    33
3    22
4    11
```

5   22
dtype: object

6. Write a Pandas program to convert a Panda module Series to Python list and it's type. Display the list. Hint: dataframe has tolist() function

Code:

```
import pandas as pd
#Create a Pandas Series
data = pd.Series([5,10,15,20,25])

# Convert the Pandas Series to a Python list
data_list=data.tolist()

print("Python list:",data_list)
print("Type of the list:",type(data_list))
```

Output:

```
Python list: [5, 10, 15, 20, 25]
Type of the list: <class 'list'>
```

7. Write a Pandas program to add, subtract, multiple and divide two Pandas Series. Accept data through the keyboard. Display the resultant series. Hint: Use +, -, *, / operator.
Sample Series: [2, 4, 6, 8, 10], [1, 3, 5, 7, 9].

Code:

```
import pandas as pd

data1=input("Enter the values for First series by giving space:").split()
data2=input("Enter the values for Second series by giving space:").split()

series1=[]
for i in data1:
   series1.append(int(i))
series1=pd.Series(series1)

series2=[]
for i in data2:
   series2.append(int(i))
series2=pd.Series(series2)

addition = series1 + series2
```

```
subtraction = series1 - series2
multiplication = series1 * series2
division = series1 / series2

print("\nAddition of Series:")
print(addition)

print("\nSubtraction of Series:")
print(subtraction)

print("\nMultiplication of Series:")
print(multiplication)

print("\nDivision of Series:")
print(division)
```

Output:

```
Enter the values for First series by giving space:2 4 6 8 10
Enter the values for Second series by giving space:1 3 5 7 9

Addition of Series:
0     3
1     7
2    11
3    15
4    19
dtype: int64

Subtraction of Series:
0    1
1    1
2    1
3    1
4    1
dtype: int64

Multiplication of Series:
0     2
1    12
2    30
3    56
4    90
dtype: int64
```

Division of Series:
0   2.000000
1   1.333333
2   1.200000
3   1.142857
4   1.111111
dtype: float64

8. Write a Pandas program to compare the elements of the two Pandas Series for which data has been accepted through the keyboard. . Hint: Use ==, >, < operators
Sample Series: [2, 4, 6, 8, 10], [1, 3, 5, 7, 10]

Code:
```
import pandas as pd

data1=input("Enter the values for First series by giving space:").split()
data2=input("Enter the values for Second series by giving space:").split()

series1=[]
for i in data1:
   series1.append(int(i))
series1=pd.Series(series1)

series2=[]
for i in data2:
   series2.append(int(i))
series2=pd.Series(series2)

equal = series1 == series2
greater = series1 > series2
less = series1 < series2

print("\nComparison (Equal):")
print(equal)

print("\nComparison (Greater):")
print(greater)

print("\nComparison (Less):")
print(less)
```
Output:
Enter the values for First series by giving space:2 4 6 8 10
Enter the values for Second series by giving space:1 3 5 7 9

Comparison (Equal):
0   False
1   False
2   False
3   False
4   False
dtype: bool

Comparison (Greater):
0   True
1   True
2   True
3   True
4   True
dtype: bool

Comparison (Less):
0   False
1   False
2   False
3   False
4   False
dtype: bool

9. Write a Pandas program to convert a dictionary(with elements empno,ename and basic) for which you accept values through the keyboard, to a Pandas series.
Original dictionary:
eg: {'empno': , 'ename': ‘ann’, 'basic': 3000, }

Code:

```
import pandas as pd

empno = int(input("Enter employee number: "))
ename = input("Enter employee name : ")
basic = float(input("Enter basic salary: "))

data = {
   'empno': empno,
   'ename': ename,
   'basic': basic
}
series=pd.Series(data)
```

```
print("\nPandas Series:")
print(series)
```

Output:

```
Enter employee number: 101
Enter employee name : ann
Enter basic salary: 3000

Pandas Series:
empno     101
ename     ann
basic    3000
dtype: object
```

10. Write a Pandas program to accept 5 elements into a numpy array and convert the NumPy array to a Pandas series.

Code:

```
import pandas as pd
import numpy as np

list1=[]
print("Enter 5 Elements: ")
for i in range(5):
    element=input()
    list1.append(element)

arr=np.array(list1)
series=pd.Series(arr)

print("\nPandas Series:")
print(series)
```

Output:

```
Enter 5 Elements:
2
4
6
8
10

Pandas Series:
0     2
```

```
1    4
2    6
3    8
4   10
dtype: object
```

11. Write Python program to apply simple Linear regression on the following dataset and also estimate the error .

X=1,2,3,4,5,6,7

Y=1.5,3.8,6.7,9.0,11.2,13.6,16

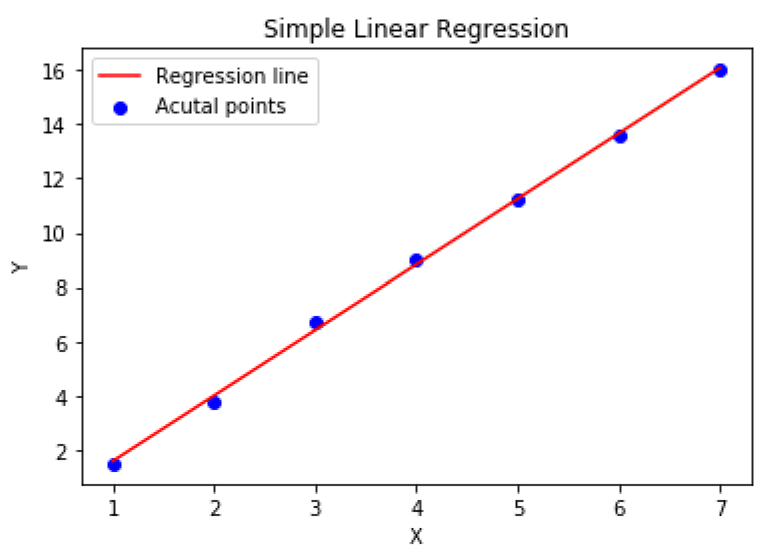
Using Matrix Multiplication Code:

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

data = pd.read_csv("d.csv")

# Extract X and Y values
x = data[["X"]].values
y = data[["Y"]].values

# Reshape X to be a 2D array for matrix operations
x = x.reshape(-1, 1)

# Add a column of ones to X for the intercept term (B0)
X_B = np.c_[np.ones((x.shape[0], 1)), x]

# Calculate B
B = np.linalg.inv(X_B.T @ X_B) @ (X_B.T @ y)

# Make predictions
pred = X_B @ B

# Calculate Mean Squared Error (MSE)
mse = np.mean((y - pred) ** 2)

# Calculate R-squared
r2 = 1 - (np.sum((y - pred) ** 2) / np.sum((y - np.mean(y)) ** 2))

print(f"B0 (Intercept): {B[0][0]}")
print(f"B1 (Slope): {B[1][0]}")
```

```
print(f"MSE: {mse}")
print(f"R-squared: {r2}")

# plot the data and regression line (Optional)
plt.scatter(x, y, color="blue", label="Data")
plt.plot(x, pred, color="red", label="Regression Line")
plt.xlabel('X')
plt.ylabel('Y')
plt.legend()
plt.show()
```

Output:

B0 (Intercept): 1.5
B1 (Slope): 0.9500000000000002
MSE: 2.175
R-squared: 0.674766355140187



ASSIGNMENT- 5

1. Implement Simple Linear Regression using Head Size as the independent variable and Brain Weight as the dependent variable from the `headbrain.csv` file. Also, predict the brain weight for a new head size.

Code:

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

data = pd.read_csv('headbrain.csv')

X = data['Head Size(cm^3)'].values.reshape(-1,1)
y = data['Brain Weight(grams)'].values

X_b = np.c_[np.ones((X.shape[0], 1)), X]
B= np.linalg.inv(X_b.T@(X_b))@(X_b.T)@(y)

y_pred = X_b @ (B)
mse = np.mean((y - y_pred) ** 2)
r2 = 1 - (np.sum((y - y_pred) ** 2) / np.sum((y - np.mean(y)) ** 2))

print(f"Intercept : {B[0]}")
print(f"Slope : {B[1]}")
print(f'Mean Square Error (RMSE): {mse}')
print(f'R-squared value: {r2}')

new_head_size = 4500
new_prediction = np.array([1, new_head_size])@ B  # Include 1 for the intercept
print(f"Predicted brain weight for head size {new_head_size} cm^3: {new_prediction} grams")

plt.scatter(X, y, color='blue', label='Actual Data')
plt.plot(X, y_pred, color='red', linewidth=2, label='Regression Line')
plt.xlabel('Head Size (cm^3)')
plt.ylabel('Brain Weight (grams)')
plt.title('Head Size vs Brain Weight')
plt.legend()
plt.show()
```

2. Implement Simple Linear Regression using the price column as the dependent variable and the column `total_sqft_int` as the independent variable using the file `hprice.csv`. Find the root mean square error and R-squared value.

Code:

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

data = pd.read_csv('hprice.csv')

#  Define independent (X) and dependent (y) variables
X = data['total_sqft_int'].values.reshape(-1, 1)
y = data['price'].values

#  Add a bias term (intercept) to the feature matrix
X_b = np.c_[np.ones((X.shape[0], 1)), X]

# Calculate the coefficients using the Normal Equation
B = np.linalg.inv(X_b.T @ X_b)@(X_b.T) @ y

# Make predictions on the dataset using the calculated coefficients
y_pred = X_b @ B

# Calculate Root Mean Square Error (RMSE) and R-squared (R²)
rmse = np.sqrt(np.mean((y - y_pred) ** 2))
r2 = 1 - (np.sum((y - y_pred) ** 2) / np.sum((y - np.mean(y)) ** 2))

print(f"Intercept (B0): {B[0]}")
print(f"Slope (B1): {B[1]}")
print(f"Root Mean Square Error (RMSE): {rmse}")
print(f"R-squared value: {r2}")

plt.scatter(X, y, color='blue', label='Actual Data')
plt.plot(X, y_pred, color='red', linewidth=2, label='Regression Line')
plt.xlabel('Total Square Feet (int)')
plt.ylabel('Price')
plt.title('Linear Regression - Price vs. Total Square Feet')
plt.legend()
plt.show()
```

3. Predict the price for one new price and then for three new prices.

Code:

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

data = pd.read_csv('hprice.csv')
```

```python
# Define independent (X) and dependent (y) variables
X = data['total_sqft_int'].values.reshape(-1, 1)  # Independent variable: total square footage
y = data['price'].values  # Dependent variable: house price

# Add a bias term (intercept) to the feature matrix
X_b = np.c_[np.ones((X.shape[0], 1)), X]  # Adding a column of ones for the intercept

# Calculate the coefficients using the Normal Equation
B = np.linalg.inv(X_b.T @ X_b) @ (X_b.T) @ y

# Make predictions on the dataset using the calculated coefficients
y_pred = X_b @ B

# Calculate Root Mean Square Error (RMSE) and R-squared (R²)
rmse = np.sqrt(np.mean((y - y_pred) ** 2))  # RMSE
r2 = 1 - (np.sum((y - y_pred) ** 2) / np.sum((y - np.mean(y)) ** 2))  # R-squared

# Print the coefficients and evaluation metrics
print(f"Intercept (B0): {B[0]}")
print(f"Slope (B1): {B[1]}")
print(f"Root Mean Square Error (RMSE): {rmse}")
print(f"R-squared value: {r2}")

# Visualize the data and the regression line
plt.scatter(X, y, color='blue', label='Actual Data')
plt.plot(X, y_pred, color='red', linewidth=2, label='Regression Line')
plt.xlabel('Total Square Feet (int)')
plt.ylabel('Price')
plt.title('Linear Regression - Price vs. Total Square Feet')
plt.legend()
plt.show()

# Predict the price for new total_sqft_int values
new_total_sqft_int = np.array([1200, 1500, 1800, 2000])

# Add a bias term (intercept) for the new input values
new_X_b = np.c_[np.ones((new_total_sqft_int.shape[0], 1)), new_total_sqft_int]

# Predict the prices for the new values using matrix multiplication
new_prices = new_X_b @ B

# Print the predicted prices
for i, sqft in enumerate(new_total_sqft_int):
```

```
    print(f"Predicted price for {sqft} sq. ft.: ${new_prices[i]:.2f}")
```

4. Implement the third question using the Sklearn API. Use only 75% of the data for training and the rest for testing. (Plot the graph)

Code:

```
# Import necessary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split

data = pd.read_csv('hprice.csv')

# Define independent (X) and dependent (y) variables
X = data['total_sqft_int'].values.reshape(-1, 1)  # Independent variable: total square footage
y = data['price'].values  # Dependent variable: house price

# Split the data into training (75%) and testing (25%)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=42)

# Add a bias term (intercept) to the feature matrix
X_b_train = np.c_[np.ones((X_train.shape[0], 1)), X_train]  # Adding a column of ones for the intercept
X_b_test = np.c_[np.ones((X_test.shape[0], 1)), X_test]  # Adding a column of ones for the intercept

# Calculate the coefficients using the Normal Equation
theta_best = np.linalg.inv(X_b_train.T.dot(X_b_train)).dot(X_b_train.T).dot(y_train)

# Predict prices for the test set using matrix multiplication
y_pred_test = X_b_test.dot(theta_best)

# Create an array of new total_sqft_int values to predict prices for
new_total_sqft_int = np.array([1200, 1500, 1800, 2000]).reshape(-1, 1)

# Add a bias term for the new input values
new_X_b = np.c_[np.ones((new_total_sqft_int.shape[0], 1)), new_total_sqft_int]

# Predict the prices for the new values using matrix multiplication
new_prices = new_X_b.dot(theta_best)

# Plot the graph
plt.scatter(X_test, y_test, color='blue', label='Actual Data')  # Scatter plot of actual data points
```

```
plt.plot(X_test, y_pred_test, color='red', linewidth=2, label='Regression Line')  # Regression line
for test data
plt.scatter(new_total_sqft_int, new_prices, color='green', marker='o', label='Predicted Prices')  #
Predicted prices for new values
plt.xlabel('Total Square Feet (int)')
plt.ylabel('Price')
plt.title('Linear Regression - Price vs. Total Square Feet')
plt.legend()
plt.show()  # Display the plot

# Print the predicted prices for new values
for i, sqft in enumerate(new_total_sqft_int):
    print(f"Predicted price for {sqft[0]} sq. ft.: ${new_prices[i]:.2f}")
```

ASSIGNMENT- 6

1. Implement Multilinear Regression on Data1.csv. Display the coefficients. (that is)
b.Predict Y values for

Y	X1	X2	X3
?	50	70	80
?	30	40	50

Code:

```
import pandas as pd
import numpy as np

# Load the data from Data1.csv
data = pd.read_csv("Data1.csv")

# Separate the features (X) and target variable (Y)
X = data[['X1', 'X2', 'X3']]
Y = data['Y']

# Add a column of ones for the intercept term (optional in scikit-learn)
X = np.c_[np.ones(len(X)), X]

# Convert to NumPy arrays for efficient matrix operations
X = np.asarray(X)
Y = np.asarray(Y)
```

```
# Calculate the coefficients (weights) using matrix multiplication and least squares
B = np.linalg.inv((X.T @ X)) @ (X.T @ Y)

# Print the coefficients
print("Coefficients:")
print(f"B0 (intercept): {B[0]}")
print(f"B1: {B[1]}")
print(f"B2: {B[2]}")
print(f"B3: {B[3]}")  # Assuming 3 features (X1, X2, X3)

# Predict Y values for new data points
new_data = np.array([[1, 50, 70, 80], [1, 30, 40, 50]])  # Include intercept term
predicted_y = new_data @ B
print("Predicted Y values:")
print(predicted_y)
```

2. Implement Multiple Linear Regression to predict the price given the data set below. Do data preprocessing to fill the null value. Display the coefficients. (that is)

Area	Bedrooms	Age	Price
2600	3	20	550000
3000	4	15	565000
3200		18	610000
3600	3	30	595000
4000	5	8	760000
Predict the price for the below			
3000	3	40	?
2500	4	5	?

Code:

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

# Step 1: Load the dataset
data = pd.read_csv("home_price.csv")

# Step 2: Check for missing values
# This prints the count of missing values for each column
```

```
print(data.isnull().sum())

# Step 3: Handle missing values
# Fill missing values in 'Bedrooms' column with the mean value of that column
data['Bedrooms'] = data['Bedrooms'].fillna(data['Bedrooms'].mean())

# Step 4: Define independent and dependent variables
x = data[["Area", "Bedrooms", "Age"]]  # Independent features (predictors)
y = data[["Price"]]  # Dependent feature (target)

# Step 5: Add intercept term (bias) to the independent variables
# This column of 1's allows us to compute the intercept (B0) of the regression model
data['intercept'] = 1
x = data[["intercept", "Area", "Bedrooms", "Age"]]

# Step 6: Compute the coefficients (B) using the Normal Equation
X_T = x.T
B = np.linalg.inv(X_T @ x) @ X_T @ y

# Set the column names of the coefficient vector to match the variable names
B.index = x.columns

# Step 7: Display the coefficients (B)
print("\nCoefficients:")
print(B)

# Step 8: Make predictions using the calculated coefficients (B)
predictions = x @ B

# Step 9: Calculate SSR (Sum of Squared Residuals)
SSR = ((y - predictions) ** 2).sum()
print("\nSum of Squared Residuals (SSR):", SSR)

# Step 10: Calculate TSS (Total Sum of Squares)
TSS = ((y - y.mean()) ** 2).sum()
print("\nTotal Sum of Squares (TSS):", TSS)

# Step 11: Calculate R-squared (R²)
R2 = 1 - (SSR / TSS)
print("\nR-squared (R²):", R2)

# Step 12: Predict the price for new data points
new_data = np.array([[1, 50, 70, 80], [1, 30, 40, 50]])
```

# Step 13: Make predictions for new data
predictions = new_data @ B
print("\nPredicted Prices:")
print(predictions)

Output :

```
Area        0
Bedrooms    1
Age         0
Price       0
dtype: int64

Coefficients:
                  Price
intercept  827087.520308
Area          230.598261
Bedrooms  -167936.097238
Age        -18554.365485

Sum of Squared Residuals (SSR): Price    9.911021e+07
dtype: float64

Total Sum of Squares (TSS): Price    2.817000e+10
dtype: float64

R-squared (R²): Price    0.996482
dtype: float64

Predicted Prices:
          Price
0 -1.240126e+07
1 -6.811157e+06
```

1. **Implement Polinomial regression on Data1.csv. Display the coefficients.**

```
import pandas as pd

# Polynomial Regression Function
def polynomial_regression(x, y, degree):
    n = len(x)
    X = [[xi ** d for d in range(degree + 1)] for xi in x]  #
Generate X matrix
    XT = list(zip(*X))  # Transpose of X
    XTX = [[sum(XT[i][j] * X[j][k] for j in range(n)) for k in
range(degree + 1)] for i in range(degree + 1)]  # X^T * X
    XTY = [sum(XT[i][j] * y[j] for j in range(n)) for i in
range(degree + 1)]  # X^T * Y

    # Solve for coefficients using Gaussian elimination
    coeff = [XTY[i] / XTX[i][i] for i in range(degree + 1)]  #
Assuming diagonal dominance
    return coeff

# Load data from CSV file
data = pd.read_csv('/content/Data1.csv')

# Extract input (X1) and target (y) variables
x = data['X1'].values  # Input variable
y = data['Y'].values   # Target variable

# Degree of polynomial
degree = 2  # Adjust as needed

# Perform polynomial regression
coefficients = polynomial_regression(x, y, degree)
```

```
# Display results
print("Coefficients:", coefficients)
```

2. Apply Logistic Regression on Pima Indian Diabetes dataset to predict the output.

```
import numpy as np
import pandas as pd

# Sigmoid function
def sigmoid(z):
    return 1 / (1 + np.exp(-z))

# Logistic Regression
def logistic_regression(X, y, learning_rate, iterations):
    n, m = X.shape
    w = np.zeros(m)  # Initialize weights
    b = 0            # Initialize bias

    for _ in range(iterations):
        # Compute linear combination and predictions
        linear_model = np.dot(X, w) + b
        predictions = sigmoid(linear_model)

        # Compute gradients
        dw = (1 / n) * np.dot(X.T, (predictions - y))
        db = (1 / n) * np.sum(predictions - y)

        # Update weights and bias
        w -= learning_rate * dw
        b -= learning_rate * db
```

```
    return w, b

# Load Pima Indian Diabetes Dataset
data = pd.read_csv('/content/diabetes.csv')

# Feature selection and preprocessing
X = data.iloc[:, :-1].values  # All columns except the last one
y = data.iloc[:, -1].values   # Target column

# Normalize features
X = (X - X.mean(axis=0)) / X.std(axis=0)

# Train logistic regression
learning_rate = 0.01
iterations = 1000
weights, bias = logistic_regression(X, y, learning_rate,
iterations)

print("Weights:", weights)
print("Bias:", bias)
```

3.Predict the Digits in Images Using a Logistic Regression Classifier in Python.

```
import numpy as np

# Sigmoid function
def sigmoid(z):
    return 1 / (1 + np.exp(-z))

# Logistic Regression Model
def logistic_regression(X, y, lr=0.01, epochs=1000):
    m, n = X.shape
```

```
    weights = np.zeros(n)  # Initialize weights
    bias = 0

    for _ in range(epochs):
        # Linear model
        z = np.dot(X, weights) + bias
        # Prediction
        y_pred = sigmoid(z)
        # Compute gradients
        dw = (1/m) * np.dot(X.T, (y_pred - y))
        db = (1/m) * np.sum(y_pred - y)
        # Update weights and bias
        weights -= lr * dw
        bias -= lr * db

    return weights, bias

# Prediction function
def predict(X, weights, bias):
    z = np.dot(X, weights) + bias
    y_pred = sigmoid(z)
    return (y_pred > 0.5).astype(int)

# Example Dataset (MNIST digits simplified for binary
classification: digit 0 vs 1)
# Here we use small dummy data for simplicity
X_train = np.array([[1, 2], [2, 1], [2, 3], [3, 4], [4, 3]])  #
Features
y_train = np.array([0, 0, 1, 1, 1])  # Labels (binary
classification)

# Train the model
weights, bias = logistic_regression(X_train, y_train)
```

```
# Test the model
X_test = np.array([[1, 1], [4, 4]])
predictions = predict(X_test, weights, bias)
print("Predictions:", predictions)
```

4.Apply K-means clustering on the following data.
.x = [4, 5, 10, 4, 3, 11, 14 , 6, 10, 12]
y = [21, 19, 24, 17, 16, 25, 24, 22, 21, 21]

```
def k_means(x, y, k, iterations):
    # Initialize centroids as the first `k` points
    centroids = [(x[i], y[i]) for i in range(k)]

    for _ in range(iterations):
        clusters = [[] for _ in range(k)]

        # Assign points to nearest centroid
        for i in range(len(x)):
            distances = [((x[i] - cx) ** 2 + (y[i] - cy) ** 2) for cx,
cy in centroids]
            cluster_index = distances.index(min(distances))
            clusters[cluster_index].append((x[i], y[i]))

        # Update centroids
        centroids = [(sum([p[0] for p in cluster]) / len(cluster),
                      sum([p[1] for p in cluster]) / len(cluster))
                     if cluster else centroids[i]
                     for i, cluster in enumerate(clusters)]

    return centroids, clusters

# Example Usage
```

```
x = [4, 5, 10, 4, 3, 11, 14 , 6, 10, 12]
y = [21, 19, 24, 17, 16, 25, 24, 22, 21, 21]
k = 2
iterations = 10
centroids, clusters = k_means(x, y, k, iterations)
print("Centroids:", centroids)
print("Clusters:", clusters)
```

5. Perform Hierarchical Clusteringon Mall_Customers_data.csv. draw the dendogram.

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

# Load Mall_Customers_data.csv
data = pd.read_csv('/content/Mall_Customers.csv')

# Extract relevant columns (e.g., Annual Income and Spending
Score)
# Adjust column names based on your dataset
features = data[['Annual Income (k$)', 'Spending Score
(1-100)']].values

# Compute Euclidean distance between two points
def euclidean_distance(p1, p2):
    return np.sqrt(np.sum((p1 - p2) ** 2))

# Perform hierarchical clustering
def hierarchical_clustering(data):
    n = len(data)
    clusters = {i: [i] for i in range(n)}  # Each point is its own
cluster
```

```
    distances = [[euclidean_distance(data[i], data[j]) for j in
range(n)] for i in range(n)]
    merges = []

    while len(clusters) > 1:
        # Find closest clusters
        min_dist = float('inf')
        to_merge = None
        for i in clusters:
            for j in clusters:
                if i < j:
                    dist = min(distances[p1][p2] for p1 in clusters[i]
for p2 in clusters[j])
                    if dist < min_dist:
                        min_dist = dist
                        to_merge = (i, j)

        # Merge clusters
        i, j = to_merge
        merges.append((i, j, min_dist))
        clusters[i].extend(clusters[j])
        del clusters[j]

    return merges

# Plot dendrogram
def plot_dendrogram(merges):
    plt.figure(figsize=(8, 5))
    current_positions = {i: i for i in range(len(merges) + 1)}
    for i, (a, b, height) in enumerate(merges):
        x1, x2 = current_positions[a], current_positions[b]
        x_mid = (x1 + x2) / 2
        plt.plot([x1, x1, x2, x2], [0, height, height, 0], 'b')
```

```
        current_positions[a] = x_mid  # Update cluster position
        del current_positions[b]
    plt.xlabel("Data Points")
    plt.ylabel("Distance")
    plt.title("Dendrogram")
    plt.show()

# Run and plot
merges = hierarchical_clustering(features)
plot_dendrogram(merges)
```

7.Find the optimal hyperplane for SVM use the following data set
positive class:(3,1),(3,-1),(6,1),(6,-1)
Negative Class:(1,0),(0,1),(0,-1),(-1,0)

```
import numpy as np

# Dataset: Positive and Negative classes
positive_class = np.array([[3, 1], [3, -1], [6, 1], [6, -1]])
negative_class = np.array([[1, 0], [0, 1], [0, -1], [-1, 0]])

# Combine data and labels
X = np.vstack((positive_class, negative_class))
y = np.hstack((np.ones(len(positive_class)),
-np.ones(len(negative_class))))

# Helper functions for SVM
def compute_svm(X, y):
    n_samples, n_features = X.shape
```

```
    # Initialize weights and bias
    w = np.zeros(n_features)
    b = 0
    lr = 0.01  # Learning rate
    epochs = 1000

    # Gradient Descent for optimization
    for _ in range(epochs):
        for i in range(n_samples):
            if y[i] * (np.dot(w, X[i]) + b) < 1:  # Misclassified
                w += lr * (y[i] * X[i] - 2 * (1 / epochs) * w)
                b += lr * y[i]
            else:  # Correct classification
                w -= lr * (2 * (1 / epochs) * w)

    return w, b

# Solve for the optimal hyperplane
w, b = compute_svm(X, y)

# Decision boundary equation
print(f"Optimal weight vector: {w}")
print(f"Optimal bias: {b}")
print(f"Decision boundary: {w[0]} * x1 + {w[1]} * x2 + {b} =
0")

# Plot the data and decision boundary
import matplotlib.pyplot as plt

# Plot data points
plt.scatter(positive_class[:, 0], positive_class[:, 1],
color='blue', label='Positive Class')
```

```
plt.scatter(negative_class[:, 0], negative_class[:, 1],
color='red', label='Negative Class')

# Plot decision boundary
x1 = np.linspace(-2, 7, 100)
x2 = -(w[0] * x1 + b) / w[1]
plt.plot(x1, x2, color='green', label='Decision Boundary')

plt.xlabel('x1')
plt.ylabel('x2')
plt.title('SVM Decision Boundary')
plt.legend()
plt.grid()
plt.show()
```

Write a program to implement the naïve Bayesian classifier for a sample training data set stored as a .CSV file. Compute the accuracy of the classifier, considering few test data sets. Tennisdata.csv

```
import csv
from collections import defaultdict

def load_data(filename):
    with open(filename, 'r') as f:
        return list(csv.DictReader(f))

def calc_probabilities(data):
    total = len(data)
    class_prob = defaultdict(int)
    cond_prob = defaultdict(lambda: defaultdict(int))

    # Calculate class probabilities and conditional probabilities
    for row in data:
        class_prob[row['PlayTennis']] += 1
        for col, val in row.items():
            if col != 'PlayTennis':
                cond_prob[(col, val)][row['PlayTennis']] += 1

    class_prob = {k: v / total for k, v in class_prob.items()}
    cond_prob = {k: {label: v / sum(d.values()) for label, v in d.items()} for k, d in cond_prob.items()}

    return class_prob, cond_prob

def predict(instance, class_prob, cond_prob):
    probs = {}
    for cls, p_cls in class_prob.items():
        prob = p_cls
        for col, val in instance.items():
            if (col, val) in cond_prob:
                prob *= cond_prob[(col, val)].get(cls, 0)
        probs[cls] = prob
    return max(probs, key=probs.get)

def accuracy(data, class_prob, cond_prob):
    correct = sum(predict({k: v for k, v in row.items() if k != 'PlayTennis'}, class_prob, cond_prob) ==
row['PlayTennis'] for row in data)
    return correct / len(data)

def naive_bayes(filename):
    data = load_data(filename)
    class_prob, cond_prob = calc_probabilities(data)
    print(f'Accuracy: {accuracy(data, class_prob, cond_prob) * 100:.2f}%')

naive_bayes('Tennisdata.csv')
```

2. You are provided with a dataset containing information about various plants with two features: Height (cm) and Width (cm). Each plant is labeled as either "Flower" or "Shrub." You need to use the K-Nearest Neighbors (K-NN) algorithm to classify a new, unlabeled plant based on its height and width.

```
import numpy as np

# Example dataset (with n rows and m columns)
X = np.array([[5, 2], [6, 3], [7, 2], [8, 3], [4, 1]])  # n=5, m=2
y = np.array(["flower", "flower", "shrub", "shrub", "flower"])

def knn_predict(X_train, y_train, test_point, k=3):
    # Vectorized computation of Euclidean distances between test_point and all training points
    distances = np.linalg.norm(X_train - test_point, axis=1)

    # Get the indices of the k smallest distances
    sorted_indices = distances.argsort()[:k]

    # Get the labels of the nearest neighbors
    nearest_labels = y_train[sorted_indices]

    # Predict the most common class among the k neighbors
    prediction = max(set(nearest_labels), key=list(nearest_labels).count)

    return prediction

# Take input for the test point
try:
    test_height = float(input("Enter the height of the plant: "))
    test_width = float(input("Enter the width of the plant: "))
    test_point = np.array([test_height, test_width])

    # Predict the class of the test point
    predicted_class = knn_predict(X, y, test_point)

    print(f"Predicted Class for the plant (Height: {test_height}, Width: {test_width}):
{predicted_class}")

except ValueError:
    print("Invalid input. Please enter numeric values for height and width.")
```

6. Construct decision tree also display the information gain for sunny, overcast and rain.

Outlook	Temperature	Humidity	Wind	Played football(yes/no)
Sunny	Hot	High	Weak	No
Sunny	Hot	High	Strong	No
Overcast	Hot	High	Weak	Yes
Rain	Mild	High	Weak	Yes
Rain	Cool	Normal	Weak	Yes
Rain	Cool	Normal	Strong	No
Overcast	Cool	Normal	Strong	Yes
Sunny	Mild	High	Weak	No
Sunny	Cool	Normal	Weak	Yes
Rain	Mild	Normal	Weak	Yes
Sunny	Mild	Normal	Strong	Yes
Overcast	Mild	High	Strong	Yes
Overcast	Hot	Normal	Weak	Yes

```
import math

# Dataset
# Format: [Outlook, Temperature, Humidity, Windy, PlayTennis]
data = [
    ["sunny", "hot", "high", False, "no"],
    ["sunny", "hot", "high", True, "no"],
    ["overcast", "hot", "high", False, "yes"],
    ["rain", "mild", "high", False, "yes"],
    ["rain", "cool", "normal", False, "yes"],
    ["rain", "cool", "normal", True, "no"],
    ["overcast", "cool", "normal", True, "yes"],
    ["sunny", "mild", "high", False, "no"],
    ["sunny", "cool", "normal", False, "yes"],
    ["rain", "mild", "normal", False, "yes"],
    ["sunny", "mild", "normal", True, "yes"],
    ["overcast", "mild", "high", True, "yes"],
    ["overcast", "hot", "normal", False, "yes"],
    ["rain", "mild", "high", True, "no"]
]

# Calculate entropy
def entropy(labels):
    total = len(labels)
    counts = {label: labels.count(label) for label in set(labels)}
    return -sum((count / total) * math.log2(count / total) for count in counts.values())

# Information gain calculation
def information_gain(data, attribute_index, target_index):
    total_entropy = entropy([row[target_index] for row in data])
    values = set(row[attribute_index] for row in data)
    weighted_entropy = 0
    for value in values:
```

```
        subset = [row for row in data if row[attribute_index] == value]
        subset_labels = [row[target_index] for row in subset]
        subset_entropy = entropy(subset_labels)
        weighted_entropy += (len(subset) / len(data)) * subset_entropy
    return total_entropy - weighted_entropy

# Display information gain for 'Outlook' (attribute index 0)
attributes = ["Outlook", "Temperature", "Humidity", "Windy"]
target_index = -1  # 'PlayTennis' is the target column

print("Information Gain for attributes:")
for i, attribute in enumerate(attributes):
    gain = information_gain(data, i, target_index)
    print(f"{attribute}: {gain:.4f}")
```

3 . Apply PCA to reduce the dimensionality to 1 component, and visualise the result in a 2D scatter plot.

Sample	Feature_1	Feature_2
1	5.702	4.386
2	9.884	1.020
3	2.089	1.613
4	6.531	2.533
5	4.663	2.444
6	1.590	1.104
7	6.563	1.382
8	1.966	3.687
9	8.210	0.971
10	8.379	0.961

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

# Sample data
data = np.array([
    [5.702, 4.386],
    [9.884, 1.020],
    [2.089, 1.613],
    [6.531, 2.533],
    [4.663, 2.444],
    [1.590, 1.104],
    [6.563, 1.382],
    [1.966, 3.687],
    [8.210, 0.971],
    [8.379, 0.961],
])

# PCA function
def pca_manual(data, n_components=1):
    # Step 1: Center the data
    mean_vec = np.mean(data, axis=0)
    centered_data = data - mean_vec

    # Step 2: Calculate the covariance matrix
```

```
    cov_matrix = np.cov(centered_data.T)

    # Step 3: Calculate eigenvalues and eigenvectors
    eig_values, eig_vectors = np.linalg.eig(cov_matrix)

    # Step 4: Sort eigenvectors by eigenvalues in descending order
    sorted_indices = np.argsort(eig_values)[::-1]
    eig_vectors = eig_vectors[:, sorted_indices]
    eig_values = eig_values[sorted_indices]

    # Step 5: Project the data onto the top n_components eigenvectors
    reduced_data = centered_data @ eig_vectors[:, :n_components]
    return reduced_data, eig_vectors[:, :n_components]

# Reduce to 1 dimension
reduced_data, top_components = pca_manual(data, n_components=1)

# Plot the reduced data
plt.scatter(data[:, 0], data[:, 1], color='blue', label='Original Data')
plt.scatter(reduced_data, np.zeros(len(reduced_data)), color='red', label='PCA Reduced Data')
plt.axhline(0, color='black', linewidth=0.5)
plt.title('PCA: Original Data vs Reduced Data')
plt.legend()
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.show()
```

4. Classify the retinal diseases using CNN. USE dataset from this :
https://www.kaggle.com/code/muhammadfaizan65/retinal-disease-classification

```
import tensorflow as tf
from tensorflow.keras import layers, models
import numpy as np
import matplotlib.pyplot as plt
import os
from tensorflow.keras.preprocessing.image import ImageDataGenerator

# Set up ImageDataGenerator for data augmentation
train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=30,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest'
)

test_datagen = ImageDataGenerator(rescale=1./255)

# Set up directories for training and testing
```

```
train_dir = '/path_to_train_data'
test_dir = '/path_to_test_data'

# Prepare data generators
train_generator = train_datagen.flow_from_directory(
    train_dir,
    target_size=(224, 224),
    batch_size=32,
    class_mode='categorical'
)

test_generator = test_datagen.flow_from_directory(
    test_dir,
    target_size=(224, 224),
    batch_size=32,
    class_mode='categorical'
)
test_loss, test_acc = model.evaluate(test_generator, verbose=2)
print(f"Test accuracy: {test_acc:.2f}")
plt.plot(history.history['accuracy'], label='accuracy')
plt.plot(history.history['val_accuracy'], label = 'val_accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.ylim([0, 1])
plt.legend(loc='lower right')
plt.show()

plt.plot(history.history['loss'], label='loss')
plt.plot(history.history['val_loss'], label = 'val_loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.ylim([0, 1])
plt.legend(loc='upper right')
plt.show()

model.save('retinal_disease_classifier.h5')

model = tf.keras.models.load_model('retinal_disease_classifier.h5')
```

5. Apply gradient on a simple linear regression (single variable). It takes a set of 15 data points and iteratively updates the parameters to minimise the mean squared error

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

# Sample data
X = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15])
y = np.array([3, 4, 2, 5, 6, 7, 8, 6, 10, 9, 11, 14, 13, 16, 15])
```

```python
# Initialize parameters
m = 0  # Slope
b = 0  # Intercept
learning_rate = 0.01
epochs = 1000

# Gradient descent
for epoch in range(epochs):
    y_pred = m * X + b
    error = y - y_pred
    m_gradient = -(2 / len(X)) * np.sum(X * error)
    b_gradient = -(2 / len(X)) * np.sum(error)
    m -= learning_rate * m_gradient
    b -= learning_rate * b_gradient

# Final parameters
print(f"Final slope (m): {m}, intercept (b): {b}")

# Plot the results
plt.scatter(X, y, color="blue", label="Original Data")
plt.plot(X, m * X + b, color="red", label="Linear Regression Fit")
plt.xlabel("X")
plt.ylabel("y")
plt.title("Gradient Descent for Linear Regression")
plt.legend()
plt.show()
```