Assignment on Practice of NumPy Library

In [2]:
```
import numpy as np
```

1. Create a NumPy array containing the numbers from 1 to 10

In [3]:
```
arr1_to_10 = np.arange(1, 11)
print(arr1_to_10)
```

```
[ 1  2  3  4  5  6  7  8  9 10]
```

2. Convert a given Python list to numpy array

In [4]:
```
python_list = [1, 2, 3, 4, 5]
numpy_array_from_list = np.array(python_list)
print(numpy_array_from_list)
```

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

3. Create 50 evenly spaced numbers between 1 and 10

In [5]:
```
evenly_spaced = np.linspace(1, 10, 50)
print(evenly_spaced)
```

```
[ 1.         1.18367347  1.36734694  1.55102041  1.73469388  1.91836735
  2.10204082  2.28571429  2.46938776  2.65306122  2.83673469  3.02040816
  3.20408163  3.3877551   3.57142857  3.75510204  3.93877551  4.12244898
  4.30612245  4.48979592  4.67346939  4.85714286  5.04081633  5.2244898
  5.40816327  5.59183673  5.7755102   5.95918367  6.14285714  6.32653061
  6.51020408  6.69387755  6.87755102  7.06122449  7.24489796  7.42857143
  7.6122449   7.79591837  7.97959184  8.16326531  8.34693878  8.53061224
  8.71428571  8.89795918  9.08163265  9.26530612  9.44897959  9.63265306
  9.81632653 10.        ]
```

4. Create a 5x5 matrix which contains random samples from standard normal dist

In [6]:
```
normal_matrix = np.random.randn(5, 5)
print(normal_matrix)
```

```
[[ 1.29467982  1.4418879  -1.02720494 -0.34950915 -0.51831172]
 [ 0.06760911  1.79358234 -0.51345068 -0.00686325  1.64610304]
 [ 0.19436168 -0.72723717 -0.25909899 -0.71327823 -0.50771873]
 [-1.07271405  0.50037843 -1.13331928  0.60222127  0.54352901]
 [-0.33518201  0.9620174  -0.6185329   0.76392841  0.55077271]]
```

5. Create 20 random integer numbers between 1 to 100 as a Numpy array

In [7]:
```
random_integers = np.random.randint(1, 101, 20)
print(random_integers)
```

```
[60 78 42 60 77 98 61 40 69 44 22 19 98 17 95  1 89 10 53 75]
```

6. Given the NumPy array ‘arr’, reverse its elements and find its size

In [8]:
```
arr = np.array([1, 2, 3, 4, 5])
reversed_arr = arr[::-1]
print(reversed_arr)
arr_size = arr.size
print(arr_size)
```

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

7. Find the mean, median, and standard deviation of the following NumPy array

In [9]:
```
data = np.array([1, 2, 3, 4, 5, 6])
mean = np.mean(data)
median = np.median(data)
std_dev = np.std(data)
print(mean)
print(median)
print(std_dev)
```

```
3.5
3.5
1.707825127659933
```

8. Create a 3x3 matrix with all values set to 1

In [11]:
```
ones_matrix = np.ones((3, 3))
print(ones_matrix)
```

```
[[1. 1. 1.]
 [1. 1. 1.]
 [1. 1. 1.]]
```

9. Create a 3x3 matrix with all values set to 0

In [12]:
```
zeros_matrix = np.zeros((3, 3))
print(zeros_matrix)
```

```
[[0. 0. 0.]
 [0. 0. 0.]
 [0. 0. 0.]]
```

10. Given two NumPy arrays arr1 and arr2, concatenate them horizontally

In [13]:
```
arr1 = np.array([[1, 2], [3, 4]])
arr2 = np.array([[5, 6], [7, 8]])
concatenated = np.hstack((arr1, arr2))
print(concatenated)
```

```
[[1 2 5 6]
 [3 4 7 8]]
```

11. Create a NumPy array containing all even numbers from 0 to 20

In [14]:
```
evens = np.arange(0, 21, 2)
print(evens)
```

```
[ 0  2  4  6  8 10 12 14 16 18 20]
```

12. Perform element-wise multiplication of two NumPy arrays ‘a’ and ‘b’

In [15]:
```
a = np.array([1, 2, 3])
b = np.array([4, 5, 6])
elementwise_product = a * b
print(elementwise_product)
```

```
[ 4 10 18]
```

13. Reshape the NumPy array into a 2x3 matrix

In [16]:
```
original = np.array([1, 2, 3, 4, 5, 6])
reshaped = original.reshape((2, 3))
print(reshaped)
```

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

14. Find the maximum and minimum values in the NumPy array

In [17]:
```
max_val = original.max()
min_val = original.min()
print("max value:",max_val)
print("min value:",min_val)
```

```
max value: 6
min value: 1
```

15. Calculate the dot product of two NumPy arrays x and y

In [18]:
```
x = np.array([1, 2])
y = np.array([3, 4])
dot_product = np.dot(x, y)
print(dot_product)
```

```
11
```

16. Create a 2D NumPy array with shape (3, 4) filled with random floating-point

In [19]:
```
random_floats = np.random.rand(3, 4)
print(random_floats)
```

```
[[0.7973296  0.50443388 0.6136197  0.78554334]
 [0.20492605 0.57129804 0.71277656 0.44177731]
 [0.13561988 0.04447616 0.66043726 0.46959561]]
```

17. Transpose a 2D NumPy array

In [20]:
```
transposed = random_floats.T
print(transposed)
```

```
[[0.7973296  0.20492605 0.13561988]
 [0.50443388 0.57129804 0.04447616]
 [0.6136197  0.71277656 0.66043726]
 [0.78554334 0.44177731 0.46959561]]
```

18. Create a slice (0 to 5 index) of given array and set the value 100 to the s

In [21]:
```
arr_slice = np.arange(10)
arr_slice[0:6] = 100
print(arr_slice)
```

```
[100 100 100 100 100 100   6   7   8   9]
```

19. Slice the first two rows and the last two columns of a 2D NumPy array

In [23]:
```
matrix = np.arange(1, 17).reshape((4, 4))
top_right = matrix[:2, -2:]
print(top_right)
```

```
[[3 4]
 [7 8]]
```

20. Given a NumPy array of numbers, filter out all values greater than 5

In [24]:
```
arr_example = np.array([1, 6, 3, 8, 2, 7])
filtered = arr_example[arr_example <= 5]
print(filtered)
```

```
[1 3 2]
```

21. Using boolean indexing, extract all even numbers from a NumPy array

In [25]:
```
arr_example2 = np.array([1, 2, 3, 4, 5, 6])
evens_extracted = arr_example2[arr_example2 % 2 == 0]
print(evens_extracted)
```

```
[2 4 6]
```

In [ ]:
```

```

Assignment on Practice of Pandas Library

In [1]:
```
import numpy as np
import pandas as pd
```

Generate a 5x4 array of random numbers between 0 and 100

In [2]:
```
np.random.seed(42)
random_array = np.random.randint(0, 100, size=(5, 4))
```

Convert to DataFrame with custom row and column names

In [3]:
```
df = pd.DataFrame(random_array, columns=['A', 'B', 'C', 'D'], index=['V', 'W', 'X
```

Display the DataFrame

In [4]:
```
print(df)
```

```
    A   B   C   D
V  51  92  14  71
W  60  20  82  86
X  74  74  87  99
Y  23   2  21  52
Z   1  87  29  37
```

1. Selecting Specific Rows Using Label-Based Indexing (.loc)

In [6]:
```
print(df.loc['W':'Y'])
print(df.iloc[:3])
```

```
    A   B   C   D
W  60  20  82  86
X  74  74  87  99
Y  23   2  21  52
    A   B   C   D
V  51  92  14  71
W  60  20  82  86
X  74  74  87  99
```

2. Selecting Specific Columns

In [7]:
```
print(df[['A', 'C']])
```

```
    A   C
V  51  14
W  60  82
X  74  87
Y  23  21
Z   1  29
```

3. Selecting Specific Rows and Columns

In [8]:
```
print(df.loc['W':'Y', ['B', 'D']])
```

```
    B   D
W  20  86
X  74  99
Y   2  52
```

Select first 3 rows and first 2 columns

In [9]:
```
print(df.iloc[:3, :2])
```

```
    A   B
V  51  92
W  60  20
X  74  74
```

4. Conditional Slicing

Select rows where column 'A' values are greater than 50

In [10]:
```
print(df[df['A'] > 50])
```

```
    A   B   C   D
V  51  92  14  71
W  60  20  82  86
X  74  74  87  99
```

Select rows where column 'C' values are less than 30

In [11]:
```
print(df[df['C'] < 30])
```

```
    A   B   C   D
V  51  92  14  71
Y  23   2  21  52
Z   1  87  29  37
```

5. Dropping a Row

Drop row 'X'

In [12]:
```
df_dropped_row = df.drop(index='X')
```

Display DataFrame after dropping row

In [14]:
```
print(df_dropped_row)
```

```
    A   B   C   D
V  51  92  14  71
W  60  20  82  86
Y  23   2  21  52
Z   1  87  29  37
```

6. Dropping Multiple Rows

Drop rows 'W' and 'Y'

In [15]:
```
df_dropped_rows = df.drop(index=['W', 'Y'])
print(df_dropped_rows)
```

```
    A   B   C   D
V  51  92  14  71
X  74  74  87  99
Z   1  87  29  37
```

7. Dropping a Column

Drop column 'C'

In [16]:
```
df_dropped_col = df.drop(columns='C')
print(df_dropped_col)
```

```
    A   B   D
V  51  92  71
W  60  20  86
X  74  74  99
Y  23   2  52
Z   1  87  37
```

8. Dropping Multiple Columns

In [17]:
```
df_dropped_cols = df.drop(columns=['A', 'D'])
print(df_dropped_cols)
```

```
    B   C
V  92  14
W  20  82
X  74  87
Y   2  21
Z  87  29
```

9. Dropping Rows/Columns In-Place

Drop row 'X' permanentl

In [18]:
```
df.drop(index='X', inplace=True)
```

Drop column 'C' permanently

In [19]:
```
df.drop(columns='C', inplace=True)
print(df)
```

```
    A   B   D
V  51  92  71
W  60  20  86
Y  23   2  52
Z   1  87  37
```

Adding a new column first

In [24]:
```
df['F'] = [5, 10, 15, 20]
```

Adding a new row that includes the new column

In [26]:
```
df.loc['Extra'] = [55, 65, 75, 85]  # Assuming 6 columns now
print(df)
```

```
        A   B   D   F
V      51  92  71   5
W      60  20  86  10
Y      23   2  52  15
Z       1  87  37  20
Extra  55  65  75  85
```

In [ ]:

Assignment on find S Algorithm.

Apply find S on "Enjoy Sport" Data to find Specific hypothesis for it.

In [2]:
```
import pandas as pd
import numpy as np
```

In [3]:
```
# Read the dataset from CSV
df = pd.read_csv("Enjoy_sports.csv")
data = df.to_numpy()
```

In [4]:
```
# Function to implement the Find-S algorithm
def find_s(dataset):
    hypothesis = None

    # Iterate through all examples in the dataset
    for example in dataset:
        if example[-1] == 'Yes':  # Consider only positive examples
            if hypothesis is None:
                hypothesis = example[:-1].copy()  # First positive example
            else:
                for i in range(len(hypothesis)):
                    if hypothesis[i] != example[i]:
                        hypothesis[i] = '?'  # Generalize

    return hypothesis
```

In [5]:
```
# Apply the Find-S algorithm to the dataset
hypothesis = find_s(data)
print("Final hypothesis:", hypothesis)
```

```
Final hypothesis: ['Sunny' 'Warm' '?' 'Strong' '?' '?']
```

4)Assignment on Candidate Elimination Algorithm.

Apply it on "Enjoy Sport" Dataset to find Version Space for it. Sample Dataset: [Sky, Temp, Humidity, Wind, Water, Forecast, PlayTennis]

In [3]:
```
data = [
    ["Sunny", "Warm", "Normal", "Strong", "Warm", "Same", "Yes"],
    ["Sunny", "Warm", "High", "Strong", "Warm", "Same", "Yes"],
    ["Rainy", "Cold", "High", "Strong", "Warm", "Change", "No"],
    ["Sunny", "Warm", "High", "Strong", "Cool", "Change", "Yes"]
]
```

In [4]:
```
def candidate_elimination(data) :
    s=["$"]*(len(data[0])-1)
    G=[["?"]*(len(data[0])-1)]

    for row in data :
        attributes,label=row[:-1],row[-1]
        if label == "Yes":
            for i in range(len(s)):
                if s[i]=="$":
                    s[i]=attributes[i]
                elif s[i]!=attributes[i]:
                    s[i]="?"
            G=[g for g in G if all(g[i]=="?" or g[i]==s[i] for i in range(len(s)))]
        elif label=="No":
            new_G=[]
            for g in G:
                for i in range(len(g)):
                    if g[i]=="?":
                        new_hypothesis=g[:]
                        new_hypothesis[i]=attributes[i]
                        new_G.append(new_hypothesis)
            G=new_G
    return s,G
```

Run candidate elimination

In [6]:
```
S_final,G_final=candidate_elimination(data)

print("Final specific hypothesis : ",S_final)
print("Final General hypothesis : ",G_final)
```

```
Final specific hypothesis :  ['Sunny', 'Warm', '?', 'Strong', '?', '?']
Final General hypothesis :  [['?', '?', '?', 'Strong', '?', '?']]
```

Assignment on Simple Regression.

Build an application where it can predict a salary based on year of experience using single variable Linear Regression (use data set from! Kaggle). Display the co-efficient and intercept. Also Display MSE (Mean Squared Error). Plot of model on Testing data.

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
```

Load dataset

In [3]:
```
file_path =“Salary_dataset.csv"
df = pd.read_csv(file_path)
```

Drop unnecessary column if present

In [4]:
```
if 'Unnamed: 0' in df.columns:
    df = df.drop(columns=['Unnamed: 0'])
```

Display basic information

In [5]:
```
print(df.head())
print(df.info())
```

```
   YearsExperience   Salary
0              1.2  39344.0
1              1.4  46206.0
2              1.6  37732.0
3              2.1  43526.0
4              2.3  39892.0
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 30 entries, 0 to 29
Data columns (total 2 columns):
YearsExperience    30 non-null
float64 Salary             30 non-
null float64 dtypes: float64(2)
memory usage: 560.0 bytes None
```

Define features and target variable

In [6]:
```
X = df[['YearsExperience']]
y = df['Salary']
```

Split dataset into training and testing sets

In [7]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42
```

Train the model

In [8]:
```
model = LinearRegression()
model.fit(X_train, y_train)
```

Out[8]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=False)
```

Make predictions

In [9]:
```
y_pred = model.predict(X_test)
```

Evaluate the model

In [10]:
```
mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)
r2 = r2_score(y_test, y_pred)

print(f"Root Mean Squared Error: {rmse}")
print(f"R-squared Score: {r2}")
```

```
Root Mean Squared Error: 7059.043621901506
R-squared Score: 0.9024461774180498
```

Visualizing the regression line

In [11]:
```
plt.figure(figsize=(10, 6))
plt.scatter(X_test['YearsExperience'], y_test, label='Actual', color='blue', alpha=0.6)
plt.plot(X_test['YearsExperience'], y_pred, label='Predicted', color='red')
plt.xlabel("Years of Experience")
plt.ylabel("Salary")
plt.title("Years of Experience vs. Salary (Regression Line)")
plt.legend()
plt.show()
```

In [ ]:

Assignment on Multi Regression:

Build an application where it can predict price of a house using multiple variable Linear Regression (use USA Housing dataset from Kaggle). Display all co-efficients and MSE.

In [23]:
```
import numpy as np  import
pandas as pd  import
matplotlib.pyplot as plt  import
seaborn as sns
 from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression  from
sklearn.metrics import mean_squared_error,r2_score
```

Load the dataset from a CSV file

In [24]:
```
df=pd.read_csv(USA_Housing.csv')
```

Display the first few rows of the dataset

In [25]:
```
display(df.head())
```

	Avg. Area Income	Avg. Area House Age	Avg. Area Number of Rooms	Avg. Area Number of Bedrooms	Area Population	Price	Address
0	79545.458574	5.682861	7.009188	4.09	23086.800503	1.059034e+06	208 Michael Ferry Apt. 674\nLaurabury, NE 3701...
1	79248.642455	6.002900	6.730821	3.09	40173.072174	1.505891e+06	188 Johnson Views Suite 079\nLake Kathleen, CA...
2	61287.067179	5.865890	8.512727	5.13	36882.159400	1.058988e+06	9127 Elizabeth Stravenue\nDanieltown, WI 06482...
3	63345.240046	7.188236	5.586729	3.26	34310.242831	1.260617e+06	USS Barnett\nFPO AP 44820
4	59982.197226	5.040555	7.839388	4.23	26354.109472	6.309435e+05	USNS Raymond\nFPO AE 09386

Display information about the dataset

In [26]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5000 entries, 0 to 4999
Data columns (total 7 columns):
Avg. Area Income                5000 non-null float64
Avg. Area House Age             5000 non-null float64
Avg. Area Number of Rooms       5000 non-null float64
Avg. Area Number of Bedrooms    5000 non-null float64
Area Population                 5000 non-null float64
Price                           5000 non-null float64
Address                         5000 non-null object
dtypes: float64(6), object(1) memory usage: 273.5+
KB
```

Display summary statistics

In [27]:
```
df.describe()
```

Out[27]:

	Avg. Area Income	Avg. Area House Age	Avg. Area Number of Rooms	Avg. Area Number of Bedrooms	Area Population	Price
count	5000.000000	5000.000000	5000.000000	5000.000000	5000.000000	5.000000e+03
mean	68583.108984	5.977222	6.987792	3.981330	36163.516039	1.232073e+06
std	10657.991214	0.991456	1.005833	1.234137	9925.650114	3.531176e+05
min	17796.631190	2.644304	3.236194	2.000000	172.610686	1.593866e+04
25%	61480.562388	5.322283	6.299250	3.140000	29403.928702	9.975771e+05
50%	68804.286404	5.970429	7.002902	4.050000	36199.406689	1.232669e+06
75%	75783.338666	6.650808	7.665871	4.490000	42861.290769	1.471210e+06
max	107701.748378	9.519088	10.759588	6.500000	69621.713378	2.469066e+06

Split the dataset into input variables (X) and output variable (y)

In [28]:
```
x=df[['Avg. Area Income', 'Avg. Area House Age', 'Avg. Area Number of Rooms','Are
y=df['Price']
```

Split the data into training and testing sets (80% training, 20% testing)

In [29]:
```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_st
```

Initialize and train a linear regression model

In [30]:
```
model=LinearRegression()
model.fit(x_train,y_train)
```

Out[30]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=False)
```

Make predictions on the test set

In [31]:
```
y_pred=model.predict(x_test)
```

Calculate Mean Squared Error (MSE) and R squared score

In [32]:
```
mse=mean_squared_error(y_test,y_pred)
r2=r2_score(y_test,y_pred)
print(f"Mean squared error:{mse}")
print(f"R-squared score: {r2}")
```

```
Mean squared error:10073721633.872011
R-squared score: 0.9181214278738137
```

Visualize the relationship between actual and predicted prices

In [ ]:
```

```

Assignment on Binary classification:

Build application to decide on whether to play tennis using Decision Tree Classifier. Do the required data preprocessing, Display Accuracy score, classific report & confusion Matrix.

Import necessary libraries

In [1]:
```
import numpy as np import
pandas as pd import
matplotlib.pyplot as plt import
seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression from
sklearn.metrics import mean_squared_error,r2_score from
sklearn.tree import DecisionTreeClassifier from
sklearn.model_selection import train_test_split from
sklearn.metrics import accuracy_score
```

Load dataset

In [2]:
```
df=pd.read_csv(PlayTennis.csv')
```

Display dataset

In [3]:
```
display(df)
```

	Outlook	Temperature	Humidity	Wind	Play Tennis
0	Sunny	Hot	High	Weak	No
1	Sunny	Hot	High	Strong	No
2	Overcast	Hot	High	Weak	Yes
3	Rain	Mild	High	Weak	Yes
4	Rain	Cool	Normal	Weak	Yes
5	Rain	Cool	Normal	Strong	No
6	Overcast	Cool	Normal	Strong	Yes
7	Sunny	Mild	High	Weak	No
8	Sunny	Cool	Normal	Weak	Yes
9	Rain	Mild	Normal	Weak	Yes
10	Sunny	Mild	Normal	Strong	Yes
11	Overcast	Mild	High	Strong	Yes
12	Overcast	Hot	Normal	Weak	Yes
13	Rain	Mild	High	Strong	No

In [4]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 14 entries, 0 to 13
Data columns (total 5 columns):
Outlook        14 non-null object
Temperature    14 non-null object
Humidity       14 non-null object
Wind           14 non-null
object Play Tennis    14 non-
null object dtypes: object(5)
memory usage: 640.0+ bytes
```

One-hot encode categorical variables

In [5]:
```
Wed=df['Wind'].str.get_dummies()
display(Wed)
Out=df['Outlook'].str.get_dummies()
display(Out)
Temp=df['Temperature'].str.get_dummies()
display(Temp)
Hum=df['Humidity'].str.get_dummies()
display(Hum)
Tennis=df['Play Tennis'].str.get_dummies()
display(Tennis)
```

	Strong	Weak
0	0	1
1	1	0
2	0	1
3	0	1
4	0	1
5	1	0
6	1	0
7	0	1
8	0	1
9	0	1
10	1	0

Drop original categorical columns

In [6]:
```
df.drop(['Outlook', 'Temperature' ,'Humidity' ,'Wind'],axis=1,inplace=True)
```

Concatenate the dummy variables

In [7]:
```
df=pd.concat([df,Out,Temp,Hum,Wed],axis=1)
```

In [8]:
```
display(df)
```

	Play Tennis	Overcast	Rain	Sunny	Cool	Hot	Mild	High	Normal	Strong	Weak
0	No	0	0	1	0	1	0	1	0	0	1
1	No	0	0	1	0	1	0	1	0	1	0
2	Yes	1	0	0	0	1	0	1	0	0	1
3	Yes	0	1	0	0	0	1	1	0	0	1
4	Yes	0	1	0	1	0	0	0	1	0	1
5	No	0	1	0	1	0	0	0	1	1	0
6	Yes	1	0	0	1	0	0	0	1	1	0
7	No	0	0	1	0	0	1	1	0	0	1
8	Yes	0	0	1	1	0	0	0	1	0	1
9	Yes	0	1	0	0	0	1	0	1	0	1
10	Yes	0	0	1	0	0	1	0	1	1	0
11	Yes	1	0	0	0	0	1	1	0	1	0
12	Yes	1	0	0	0	1	0	0	1	0	1
13	No	0	1	0	0	0	1	1	0	1	0

Map target variable to numeric values (Yes -> 1, No -> 0)

In [9]:
```
df['Play Tennis'] = df['Play Tennis'].map({'Yes': 1, 'No': 0})
```

Prepare features and target variable

In [11]:
```
X = df.drop('Play Tennis', axis=1)
y = df['Play Tennis']
```

Split the dataset into training and testing sets

In [14]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_st
```

Train Decision Tree model

In [15]:
```
dt = DecisionTreeClassifier(criterion='entropy', random_state=0)
dt.fit(X_train, y_train)
```

Out[15]:
```
DecisionTreeClassifier(class_weight=None, criterion='entropy', max_depth=None,
 max_features=None, max_leaf_nodes=None,            min_impurity_decrease=0.0,
 min_impurity_split=None,            min_samples_leaf=1, min_samples_split=2,
           min_weight_fraction_leaf=0.0, presort=False, random_state=0,
 splitter='best')
```

Make predictions

In [16]:
```
y_pred = dt.predict(X_test)
```

Calculate accuracy

In [17]:
```
accuracy = accuracy_score(y_test, y_pred)
print(f"Model Accuracy: {accuracy:.2f}")
```

```
Model Accuracy: 1.00
```

In [ ]:

8)Assignment on Binary classification using Percoptron.

Implement Perception model. Use this model to classify a patient is having cancer or not. (use Breast cancer dataset from sklearn), Display Accuracy score, classification Report and Confusion matrix.

Loading Dataset

In [34]:
```
import sklearn.datasets
import numpy as np
```

In [35]:
```
breast_cancer=sklearn.datasets.load_breast_cancer()
```

In [36]:
```
X=breast_cancer.data
Y=breast_cancer.target
```

In [37]:
```
print(X.shape,Y.shape)
```

```
(569, 30) (569,)
```

In [38]:
```
import pandas as pd
```

In [39]:
```
data=pd.DataFrame(breast_cancer.data,columns=breast_cancer.feature_names)
```

In [40]:
```
data['class']=breast_cancer.target
```

In [41]:
```
data.head()
```

Out[41]:

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	symm
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	0
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	0
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	0
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	0
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	0

5 rows × 31 columns

In [42]:
```
data.describe()
```

Out[42]:

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	m conca
count	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000
mean	14.127292	19.289649	91.969033	654.889104	0.096360	0.104341	0.088
std	3.524049	4.301036	24.298981	351.914129	0.014064	0.052813	0.079
min	6.981000	9.710000	43.790000	143.500000	0.052630	0.019380	0.000
25%	11.700000	16.170000	75.170000	420.300000	0.086370	0.064920	0.029
50%	13.370000	18.840000	86.240000	551.100000	0.095870	0.092630	0.061
75%	15.780000	21.800000	104.100000	782.700000	0.105300	0.130400	0.130
max	28.110000	39.280000	188.500000	2501.000000	0.163400	0.345400	0.426

8 rows × 31 columns

TRAIN TEST SPLIT

In [43]:
```
from sklearn.model_selection import train_test_split
```

In [44]:
```
X=data.drop('class',axis=1)
Y=data['class']
```

In [45]:
```
type(X)
```

Out[45]:
```
pandas.core.frame.DataFrame
```

In [46]:
```
X_train,X_test,Y_train,Y_test=train_test_split(X,Y)
```

In [47]:
```
print(Y.shape,Y_train.shape,Y_test.shape)
```

```
(569,) (426,) (143,)
```

In [48]:
```
X_train,X_test,Y_train,Y_test=train_test_split(X,Y,test_size=0.1)
```

In [49]:
```
print(Y.mean(),Y_train.mean(),Y_test.mean())
```

```
0.6274165202108963 0.619140625 0.7017543859649122
```

In [50]:
```
X_train,X_test,Y_train,Y_test=train_test_split(X,Y,test_size=0.1,stratify=Y)
```

In [51]:
```
print(X_train.mean(),X_test.mean(),X.mean())
```

```
mean radius                14.097201
mean texture               19.339492
mean perimeter             91.803262
mean area                 652.010352
mean smoothness             0.096550
mean compactness            0.105311
mean concavity              0.090223
mean concave points         0.049157
mean symmetry               0.181708
mean fractal dimension      0.063031
radius error                0.405056
texture error               1.216304
perimeter error             2.858814
area error                 40.337998
smoothness error            0.007053
compactness error           0.025656
concavity error             0.032226
concave points error        0.011720
symmetry error              0.020609
fractal dimension error     0.003857
worst radius               16.242553
worst texture              25.738848
worst perimeter           107.107910
worst area                877.892383
worst smoothness            0.132580
worst compactness           0.257271
worst concavity             0.276409
worst concave points        0.114940
worst symmetry              0.291120
worst fractal dimension     0.084519
dtype: float64 mean radius                14.397579
mean texture               18.841930
mean perimeter             93.458070
mean area                 680.747368
mean smoothness             0.094656
mean compactness            0.095631
mean concavity              0.076007
mean concave points         0.046782
mean symmetry               0.176260
mean fractal dimension      0.060701
radius error                0.406216
texture error               1.221791
perimeter error             2.931140
area error                 40.328825
smoothness error            0.006933
compactness error           0.023880
concavity error             0.028913
concave points error        0.012476
symmetry error              0.019941
fractal dimension error     0.003236
worst radius               16.508456
worst texture              25.123684
worst perimeter           108.638246
worst area                904.752632
worst smoothness            0.130469
worst compactness           0.227268
```

```
worst concavity              0.234275
worst concave points         0.111610
worst symmetry               0.280698
worst fractal dimension      0.078797
dtype: float64 mean radius                  14.127292
mean texture                19.289649
mean perimeter              91.969033
mean area                  654.889104
mean smoothness              0.096360
mean compactness             0.104341
mean concavity               0.088799
mean concave points          0.048919
mean symmetry                0.181162
mean fractal dimension       0.062798
radius error                 0.405172
texture error                1.216853
perimeter error              2.866059
area error                  40.337079
smoothness error             0.007041
compactness error            0.025478
concavity error              0.031894
concave points error         0.011796
symmetry error               0.020542
fractal dimension error      0.003795
worst radius                16.269190
worst texture               25.677223
worst perimeter            107.261213
worst area                 880.583128
worst smoothness             0.132369
worst compactness            0.254265
worst concavity              0.272188
worst concave points         0.114606
worst symmetry               0.290076
worst fractal dimension      0.083946
dtype: float64
```

PERCEPTON CLASS

In [52]:
```
X_train=X_train.values
```

In [53]:
```
X_test=X_test.values
```

In [54]:
```
type(X_train)
```

Out[54]: numpy.ndarray

In [55]:
```
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt
```

In [62]:
```
def model(w,b,x):
    return 1 if (np.dot(w,x)>=b) else 0
def predict(w,b,X):
    Y=[]
    for x in X:
        result=model(w,b,x)
```

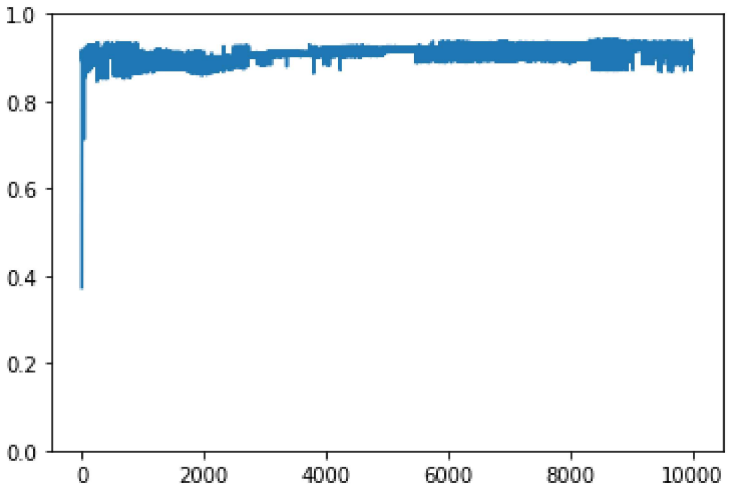
```
        Y.append(result)
    return np.array(Y)
def fit(X,Y,epochs=1,lr=1):
    w=np.ones(X.shape[1])
    b=0
    accuracy={}
    max_accuracy=0
    wt_matrix=[]

    for i in range(epochs):
        for x,y in zip(X,Y):
            Y_pred=model(w,b,x)
            if y==1 and Y_pred==0:
                w=w+lr*x
                b=b-lr*1
            elif y==0 and Y_pred==1:
                w=w-lr*x
                b=b+lr*1
        wt_matrix.append(w)
        accuracy[i]=accuracy_score(predict(w,b,X),Y)
        if(accuracy[i]>max_accuracy):
            max_accuracy=accuracy[i]
            chkptw=w
            chkptb=b
    w=chkptw
    b=chkptb
    print(max_accuracy)
    plt.plot(accuracy.values())
    plt.ylim([0,1])
    plt.show()
    return np.array(wt_matrix),w,b
```

In [63]:
```
"""perceptron=Perceptron()"""
```

Out[63]:
```
'perceptron=Perceptron()'
```

In [68]:
```
wt_matrix,w,b=fit(X_train,Y_train,10000,0.5)
```

```
0.94140625
```



In [59]:
```
print(w)
```

```
[ 50560.04200003  37153.54        68899.96500009  -2414.85000001
  -1660.786995    -7825.57883001 -11139.348847    -4282.231477
  -2205.21585      -572.59932      2003.8744       7065.6618
  -7508.93690002  -9987.1995       -217.121335    -1922.5797015
  -2920.4644425    -621.357875     -640.054529     -122.6664468
  55453.81849999 -44939.30500003 -37504.62500006  -1600.55
  -3223.56573    -25797.14086004 -33948.64083097  -8507.90156999
  -8661.10155     -2439.3208    ]
```

In [60]:
```
Y_pred_test=predict(w,b,X_test)
```

In [61]:
```
print(accuracy_score(Y_pred_test,Y_test))
```

```
0.9649122807017544
```

In [ ]:

9)Assignment on Multiclassification using MLP

Build an application to classify given iris flower Specie using MLP (Use Iris data set from Kaggle/ sklean). Display Accuracy score, classification report and confusion matrix.

In [3]:
```
import pandas as pd
url= "https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data"
names=['sepal-length','sepal-width','petal-length','petal-width','Class']
irisdata=pd.read_csv(url,names=names)
```

In [4]:
```
irisdata.head()
```

Out[4]:

	sepal-length	sepal-width	petal-length	petal-width	Class
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa

In [5]:
```
irisdata.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 5 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   sepal-length  150 non-null    float64
 1   sepal-width   150 non-null    float64
 2   petal-length  150 non-null    float64
 3   petal-width   150 non-null    float64
 4   Class         150 non-null    object 
dtypes: float64(4), object(1)
memory usage: 6.0+ KB
```

In [6]:
```
X=irisdata.iloc[:,0:4]
```

In [7]:
```
y=irisdata.select_dtypes(include=[object])
```

In [8]:
```
y.head()
```

Out[8]:

	Class
0	Iris-setosa
1	Iris-setosa
2	Iris-setosa
3	Iris-setosa
4	Iris-setosa

In [9]:
```
y.Class.unique()
```

Out[9]:
```
array(['Iris-setosa', 'Iris-versicolor', 'Iris-virginica'], dtype=object)
```

In [10]:
```
from sklearn import preprocessing
le=preprocessing.LabelEncoder()
y=y.apply(le.fit_transform)
y
```

Out[10]:

	Class
0	0
1	0
2	0
3	0
4	0
...	...
145	2
146	2
147	2
148	2
149	2

150 rows × 1 columns

In [11]:
```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.30)
print("Dimension of X-train:", X_train.shape,"X-test:",X_test.shape)
y_test
```

```
Dimension of X-train: (105, 4) X-test: (45, 4)
```

Out[11]:

	Class
144	2
130	2
78	1
96	1
127	2
124	2
89	1
139	2
19	0
59	1
43	0
7	0
2	0
72	1
84	1
119	2
112	2
75	1
109	2
28	0
136	2
11	0
31	0
34	0
24	0
40	0
23	0
1	0
108	2
45	0

	Class
141	2
87	1
77	1
33	0
98	1
86	1
117	2
55	1
58	1
135	2
10	0
5	0
90	1
39	0
17	0

In [12]:
```
from sklearn.preprocessing import StandardScaler
scaler=StandardScaler()
scaler.fit(X_train)
X_train=scaler.transform(X_train)
X_test=scaler.transform(X_test)
```

In [13]:
```
from sklearn.neural_network import MLPClassifier
mlp=MLPClassifier(hidden_layer_sizes=(10,10,10),max_iter=1000)
mlp.fit(X_train,y_train.values.ravel())
```

Out[13]:
```
MLPClassifier(hidden_layer_sizes=(10, 10, 10), max_iter=1000)
```

In [14]:
```
predictions=mlp.predict(X_test)
```

In [15]:
```
from sklearn.metrics import classification_report,confusion_matrix
print(confusion_matrix(y_test,predictions))
print(classification_report(y_test,predictions))
```

```
[[18  0  0]
 [ 0 13  1]
 [ 0  0 13]]
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        18
           1       1.00      0.93      0.96        14
           2       0.93      1.00      0.96        13

    accuracy                           0.98        45
   macro avg       0.98      0.98      0.98        45
weighted avg       0.98      0.98      0.98        45
```

In [ ]:

10)Assignment on Regression using KNN.

Build an application where it can predict Salary based on year of experience using KNN (use salary dataset from Kaggle). Display MSE.

In [3]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

In [4]:
```
df=pd.read_csv("salary_dataset.csv")
```

In [5]:
```
df.head()
```

Out[5]:

	Unnamed: 0	YearsExperience	Salary
0	0	1.2	39344.0
1	1	1.4	46206.0
2	2	1.6	37732.0
3	3	2.1	43526.0
4	4	2.3	39892.0

In [6]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 30 entries, 0 to 29
Data columns (total 3 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   Unnamed: 0       30 non-null     int64  
 1   YearsExperience  30 non-null     float64
 2   Salary           30 non-null     float64
dtypes: float64(2), int64(1)
memory usage: 852.0 bytes
```

In [7]:
```
df = df.drop('Unnamed: 0', axis=1)
```

In [8]:
```
df.describe()
```

Out[8]:

	YearsExperience	Salary
count	30.000000	30.000000
mean	5.413333	76004.000000
std	2.837888	27414.429785
min	1.200000	37732.000000
25%	3.300000	56721.750000
50%	4.800000	65238.000000
75%	7.800000	100545.750000
max	10.600000	122392.000000

In [9]:
```
import sklearn
```

In [10]:
```
from sklearn.model_selection import train_test_split
```

In [11]:
```
train,test = train_test_split(df,test_size=0.3)
```

In [12]:
```
x_train=train.drop('Salary',axis=1)
y_train=train['Salary']
```

In [13]:
```
viz_test=plt
viz_test.scatter(x_train,y_train,color='red')
viz_test.title('Salary vs Experience(daa set)')
viz_test.xlabel('Year of Experience')
viz_test.ylabel('Salary')
viz_test.show()
```

In [14]:
```
x_test=test.drop('Salary',axis=1)
y_test=test['Salary']
```

In [15]:
```
x_test
```

Out[15]:

	YearsExperience
15	5.0
7	3.3
24	8.8
20	6.9
2	1.6
1	1.4
25	9.1
4	2.3
19	6.1

In [16]:
```
y_test
```

Out[16]:
```
15     67939.0
7      54446.0
24    109432.0
20     91739.0
2      37732.0
1      46206.0
25    105583.0
4      39892.0
19     93941.0
Name: Salary, dtype: float64
```

In [17]:
```
from sklearn import neighbors
```

In [18]:
```
from sklearn.metrics import mean_squared_error
from math import sqrt
import matplotlib.pyplot as plt
```

In [19]:
```
%matplotlib inline
```

In [20]:
```
model=neighbors.KNeighborsRegressor(n_neighbors=3)
model.fit(x_train,y_train)
pred=model.predict(x_test)
error=sqrt(mean_squared_error(y_test,pred))
```

In [21]:
```
pred
```

Out[21]:
```
array([ 70077.        ,  60413.33333333, 110695.33333333,  93647.        ,
        46504.33333333,  46504.33333333, 114473.        ,  53440.        ,
        76827.66666667])
```

In [22]:
```
error
```

Out[22]:
```
8678.876033788969
```

In [ ]:
```

```

Assignment on Classification using KNN.

Build an application to classify an iris flower into its specie using KNN (Iris dataset from Skleam). Display Accuracy score, classification Report & Confusion Matrix).

In [1]:
```
import pandas as pd
```

Location of dataset

In [2]:
```
url = "https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data"
```

Assign colum names to the dataset

In [3]:
```
names = ['sepal-length', 'sepal-width', 'petal-length', 'petal-width', 'Class']
```

Read dataset to pandas dataframe

In [4]:
```
irisdata = pd.read_csv(url, names=names)
```

In [5]:
```
irisdata.head()
```

Out[5]:

	sepal-length	sepal-width	petal-length	petal-width	Class
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa

In [6]:
```
irisdata.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 5 columns):
sepal-length    150 non-null float64
sepal-width     150 non-null float64
petal-length    150 non-null float64
petal-width     150 non-null float64
Class           150 non-null object
dtypes: float64(4), object(1)
memory usage: 5.9+ KB
```

In [7]:
```
x=irisdata.iloc[:,0:4]
```

Assign data from first fifth columns to y variable

In [8]:
```
y=irisdata.select_dtypes(include=[object])
```

In [9]:
```
y.head()
```

Out[9]:

	Class
0	Iris-setosa
1	Iris-setosa
2	Iris-setosa
3	Iris-setosa
4	Iris-setosa

In [10]:
```
y.Class.unique()
```

Out[10]:
```
array(['Iris-setosa', 'Iris-versicolor', 'Iris-virginica'], dtype=object)
```

In [11]:
```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.30)
print("dimension of x-train:",x_train.shape, "x_test:",x_test.shape)
y_test
```

```
dimension of x-train: (105, 4) x_test: (45, 4)
```

Out[11]:

	Class
147	Iris-virginica
127	Iris-virginica
86	Iris-versicolor
78	Iris-versicolor
82	Iris-versicolor
60	Iris-versicolor
133	Iris-virginica
1	Iris-setosa
53	Iris-versicolor
61	Iris-versicolor
130	Iris-virginica
25	Iris-setosa
107	Iris-virginica
9	Iris-setosa
74	Iris-versicolor
104	Iris-virginica
115	Iris-virginica
108	Iris-virginica
145	Iris-virginica
68	Iris-versicolor
118	Iris-virginica
46	Iris-setosa
31	Iris-setosa
8	Iris-setosa
90	Iris-versicolor
146	Iris-virginica
114	Iris-virginica
77	Iris-versicolor
81	Iris-versicolor
11	Iris-setosa
76	Iris-versicolor
122	Iris-virginica
84	Iris-versicolor
75	Iris-versicolor
40	Iris-setosa
134	Iris-virginica

	Class
144	Iris-virginica
41	Iris-setosa
43	Iris-setosa
89	Iris-versicolor
23	Iris-setosa
54	Iris-versicolor
3	Iris-setosa
65	Iris-versicolor
48	Iris-setosa

In [15]:
```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(x_train)

x_train = scaler.transform(x_train)
x_test = scaler.transform(x_test)
```

In [16]:
```
from sklearn.neighbors import KNeighborsClassifier
```

Initialize the KNN classifier with 3 neighbors

In [17]:
```
knn = KNeighborsClassifier(n_neighbors=3)
```

Fit the model to the training data

In [18]:
```
knn.fit(x_train, y_train.values.ravel())
```

Out[18]:
```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=1, n_neighbors=3, p=2,
           weights='uniform')
```

In [19]:
```
predictions = knn.predict(x_test)
```

In [20]:
```
from sklearn.metrics import classification_report, confusion_matrix
print(confusion_matrix(y_test,predictions))
print(classification_report(y_test,predictions))
```

```
[[12  1  0]
 [ 0 16  1]
 [ 0  2 13]]
                 precision    recall  f1-score   support

    Iris-setosa       1.00      0.92      0.96        13
Iris-versicolor       0.84      0.94      0.89        17
 Iris-virginica       0.93      0.87      0.90        15

    avg / total       0.92      0.91      0.91        45
```

In [ ]:
```

```

12)Assignment on K-mean clustering.

Apply K-mean clustering on Income data set to form 3 Clusters and display there clasters using scatter grap

[3]:
```
from sklearn.cluster import KMeans
import pandas as pd
from sklearn.preprocessing import MinMaxScaler
from matplotlib import pyplot as plt
%matplotlib inline
```

[4]:
```
df=pd.read_csv("income.csv")
df.rename(columns={'Income':'Income($)'},inplace=True)
df.rename(columns={'Age':'age'},inplace=True)
```

[5]:
```
df.head()
```

[5]:

	Name	age	Income($)
0	Rob	27	70000
1	Michael	29	90000
2	Mohan	29	61000
3	Ismail	28	60000
4	Kory	42	150000

13)Assignment on Hierarchical clustering.

Apply it on Mall customers to form 5 clusters and display these clusters using scattergraph and also display dendrogram.

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

In [2]:
```
ourData=pd.read_csv('Mall_Customers.csv')
ourData.head()
```

Out[2]:

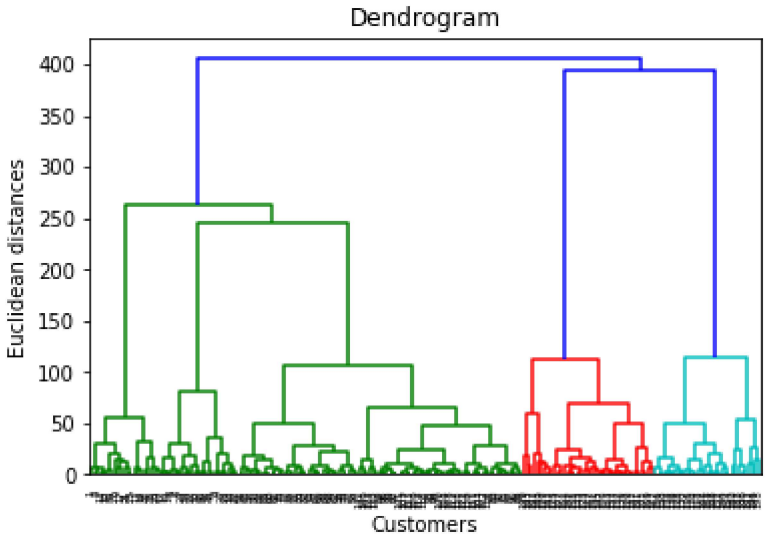
	CustomerID	Gender	Age	Annual Income (k$)	Spending Score (1-100)
0	1	Male	19	15	39
1	2	Male	21	15	81
2	3	Female	20	16	6
3	4	Female	23	16	77
4	5	Female	31	17	40

extract the two features from our dataset

In [4]:
```
newData=ourData.iloc[:,[3,4]].values
```

importing scipy.cluster.hierarchy for dendrogram

In [6]:
```
import scipy.cluster.hierarchy as sch
dendrogram=sch.dendrogram(sch.linkage(newData,method='ward'))
plt.title('Dendrogram')
plt.xlabel('Customers')
plt.ylabel('Euclidean distances')
plt.show()
```



In [7]:
```
from sklearn.cluster import AgglomerativeClustering
Agg_hc=AgglomerativeClustering(n_clusters=5,affinity='euclidean',linkage='ward')
y_hc=Agg_hc.fit_predict(newData)
```

plotting cluster 1

In [15]:
```
plt.scatter(newData[y_hc==0,0],newData[y_hc==0,1],s=50,c='red',label='cluster1')
plt.scatter(newData[y_hc==1,0],newData[y_hc==1,1],s=50,c='blue',label='cluster2')
plt.scatter(newData[y_hc==2,0],newData[y_hc==2,1],s=50,c='green',label='cluster3')
plt.scatter(newData[y_hc==3,0],newData[y_hc==3,1],s=50,c='cyan',label='cluster4')
plt.scatter(newData[y_hc==4,0],newData[y_hc==4,1],s=50,c='magenta',label='cluster5'
plt.title('Clusters of customers')
plt.xlabel('Annual Income (k$)')
plt.ylabel('Spending Score(1-100)')
plt.legend()
plt.show()
```

14)Assignment on Dimensionality Reduction

Apply Principal component Analyses(PCA) on Iris dataset to reduce its dimensionality into 3 principal componct Display data before and after reduction Scatter graph.

In [3]:
```
import pandas as pd
df=pd.read_csv('iris.csv')
from sklearn.preprocessing import StandardScaler
features=['sepal_length','sepal_width','petal_length','petal_width']
```

Separating out the features and Separating out the target

In [5]:
```
x=df.loc[:,features].values
y=df.loc[:,['species']].values
x=StandardScaler().fit_transform(x)
```

Standardizing the features

In [7]:
```
from sklearn.decomposition import PCA
pca=PCA(n_components=3)
principalComponents=pca.fit_transform(x)
principaldf=pd.DataFrame(data=principalComponents, columns=['principal component 1'
finalDf=pd.concat([principaldf,df[['species']]],axis=1)
finalDf.head()
```

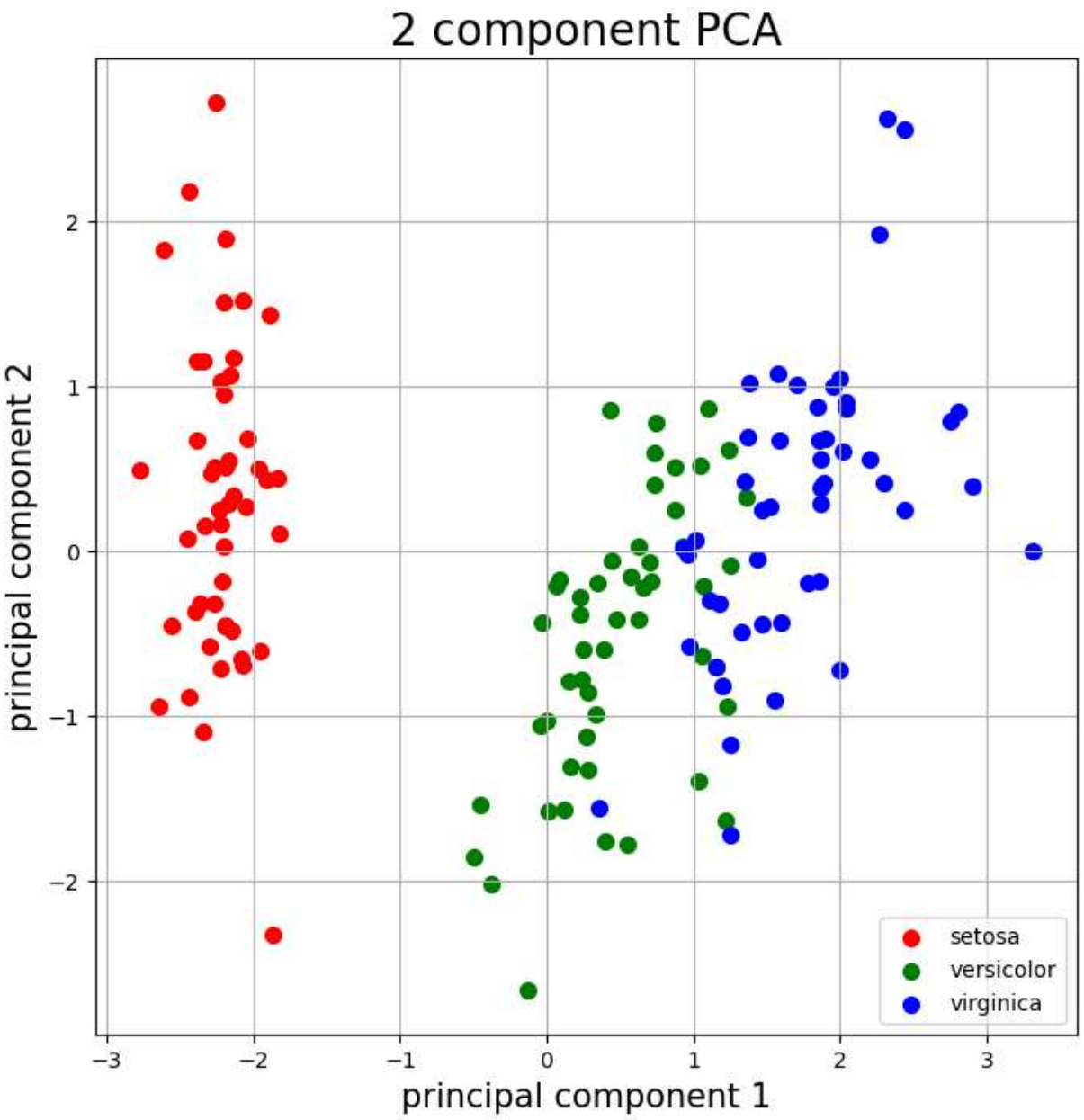
Out[7]:

	principal component 1	principal component 2	principal component 3	species
0	-2.264542	0.505704	-0.121943	setosa
1	-2.086426	-0.655405	-0.227251	setosa
2	-2.367950	-0.318477	0.051480	setosa
3	-2.304197	-0.575368	0.098860	setosa
4	-2.388777	0.674767	0.021428	setosa

In [8]:
```
import matplotlib.pyplot as plt
```

In [9]:
```
fig=plt.figure(figsize=(8,8))
ax=fig.add_subplot(1,1,1)
ax.set_xlabel('principal component 1', fontsize=15)
ax.set_ylabel('principal component 2', fontsize=15)
ax.set_title('2 component PCA',fontsize=20)
targets=['setosa','versicolor','virginica']
colors=['r','g','b']
for target, color in zip(targets,colors):
    indicesToKeep= finalDf['species']==target
```

```
    ax.scatter(finalDf.loc[indicesToKeep,'principal component 1'],finalDf.loc[indic
ax.legend(targets)
ax.grid()
```



In [10]:
```
pca.explained_variance_ratio_
```

Out[10]:
```
array([0.72770452, 0.23030523, 0.03683832])
```