In [ ]:
```
#CANDIDATE ELIMINATION

import pandas as pd
import numpy as np
```

In [ ]:
```
data = pd.read_csv("EnjoySport.csv")
data
```

Out[ ]:

	Sky	AirTemp	Humidity	Wind	Water	Forecast	EnjoySport
0	Sunny	Warm	Normal	Strong	Warm	Same	Yes
1	Sunny	Warm	High	Strong	Warm	Same	Yes
2	Rainy	Cold	High	Strong	Warm	Change	No
3	Sunny	Warm	High	Strong	Cool	Change	Yes

In [ ]:
```
def CandidateElimination(data):
    dataset = data.values.tolist()
    print("\nThe dataset is :\n",dataset)

    #initialize the specific hypothesis
    s=dataset[0][0:-1]
    print("The initial value of s is :\n",s)

    #initialize the general hypothesis
    g=[['?' for i in range(len(s))] for j in range(len(s))]
    print("The initial value of g is :\n",g)
    for row in dataset:
        if row[-1]=="Yes":
            for j in range(len(s)):
                if row[j]!=s[j]:
                    s[j]='?'
                    g[j][j]='?'
        elif row[-1]=="No":
            for j in range(len(s)):
                if row[j]!=s[j]:
                    g[j][j]=s[j]
                else:
                    g[j][j]="?"
        print("\nAfter",dataset.index(row)+1,"th insatnce")

        print("Specific boundary is :",s)
        print("General boundary is :",g)
```

In [ ]:
```
CandidateElimination(data)
```

```
The dataset is :
 [['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', 'Coo
l', 'Change', 'Yes']]
The initial value of s is :
 ['Sunny', 'Warm', 'Normal', 'Strong', 'Warm', 'Same']
The initial value of g is :
 [['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?',
'?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?',
'?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

After 1 th insatnce
Specific boundary is : ['Sunny', 'Warm', 'Normal', 'Strong', 'Warm', 'Same
']
General boundary is : [['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?',
'?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'],
['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

After 2 th insatnce
Specific boundary is : ['Sunny', 'Warm', '?', 'Strong', 'Warm', 'Same']
General boundary is : [['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?',
'?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'],
['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

After 3 th insatnce
Specific boundary is : ['Sunny', 'Warm', '?', 'Strong', 'Warm', 'Same']
General boundary is : [['Sunny', '?', '?', '?', '?', '?'], ['?', 'Warm',
'?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?',
'?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', 'Same
']]

After 4 th insatnce
Specific boundary is : ['Sunny', 'Warm', '?', 'Strong', '?', '?']
General boundary is : [['Sunny', '?', '?', '?', '?', '?'], ['?', 'Warm',
'?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?',
'?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]
```

In [ ]:

In [ ]:
```
#DIANA

from sklearn.datasets import load_iris

iris = load_iris()
X = iris.data
X
```

Out[ ]:
```
array([[5.1, 3.5, 1.4, 0.2],
       [4.9, 3. , 1.4, 0.2],
       [4.7, 3.2, 1.3, 0.2],
       [4.6, 3.1, 1.5, 0.2],
       [5. , 3.6, 1.4, 0.2],
       [5.4, 3.9, 1.7, 0.4],
       [4.6, 3.4, 1.4, 0.3],
       [5. , 3.4, 1.5, 0.2],
       [4.4, 2.9, 1.4, 0.2],
       [4.9, 3.1, 1.5, 0.1],
       [5.4, 3.7, 1.5, 0.2],
       [4.8, 3.4, 1.6, 0.2],
       [4.8, 3. , 1.4, 0.1],
       [4.3, 3. , 1.1, 0.1],
       [5.8, 4. , 1.2, 0.2],
       [5.7, 4.4, 1.5, 0.4],
       [5.4, 3.9, 1.3, 0.4],
       [5.1, 3.5, 1.4, 0.3],
       [5.7, 3.8, 1.7, 0.3],
       [5.1, 3.8, 1.5, 0.3],
       [5.4, 3.4, 1.7, 0.2],
       [5.1, 3.7, 1.5, 0.4],
       [4.6, 3.6, 1. , 0.2],
       [5.1, 3.3, 1.7, 0.5],
       [4.8, 3.4, 1.9, 0.2],
       [5. , 3. , 1.6, 0.2],
       [5. , 3.4, 1.6, 0.4],
       [5.2, 3.5, 1.5, 0.2],
       [5.2, 3.4, 1.4, 0.2],
       [4.7, 3.2, 1.6, 0.2],
       [4.8, 3.1, 1.6, 0.2],
       [5.4, 3.4, 1.5, 0.4],
       [5.2, 4.1, 1.5, 0.1],
       [5.5, 4.2, 1.4, 0.2],
       [4.9, 3.1, 1.5, 0.2],
       [5. , 3.2, 1.2, 0.2],
       [5.5, 3.5, 1.3, 0.2],
       [4.9, 3.6, 1.4, 0.1],
       [4.4, 3. , 1.3, 0.2],
       [5.1, 3.4, 1.5, 0.2],
       [5. , 3.5, 1.3, 0.3],
       [4.5, 2.3, 1.3, 0.3],
       [4.4, 3.2, 1.3, 0.2],
       [5. , 3.5, 1.6, 0.6],
       [5.1, 3.8, 1.9, 0.4],
       [4.8, 3. , 1.4, 0.3],
       [5.1, 3.8, 1.6, 0.2],
       [4.6, 3.2, 1.4, 0.2],
       [5.3, 3.7, 1.5, 0.2],
       [5. , 3.3, 1.4, 0.2],
       [7. , 3.2, 4.7, 1.4],
       [6.4, 3.2, 4.5, 1.5],
       [6.9, 3.1, 4.9, 1.5],
       [5.5, 2.3, 4. , 1.3],
       [6.5, 2.8, 4.6, 1.5],
       [5.7, 2.8, 4.5, 1.3],
```

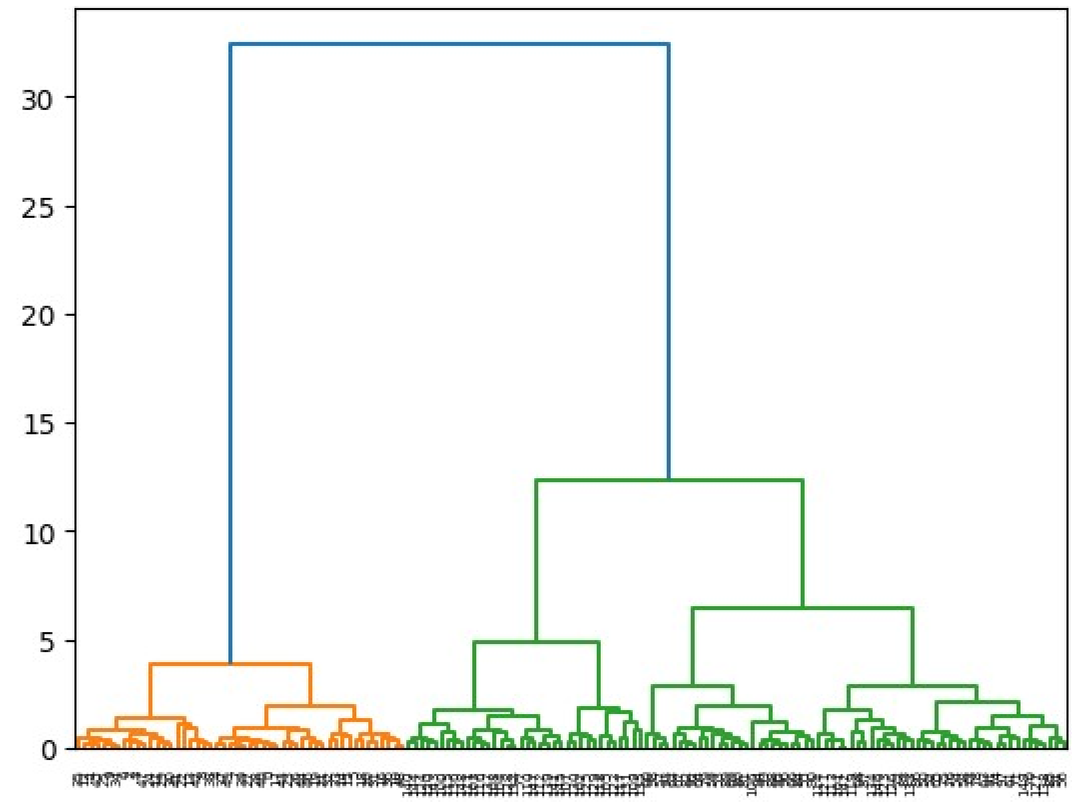
```
[6.3, 3.3, 4.7, 1.6],
[4.9, 2.4, 3.3, 1. ],
[6.6, 2.9, 4.6, 1.3],
[5.2, 2.7, 3.9, 1.4],
[5. , 2. , 3.5, 1. ],
[5.9, 3. , 4.2, 1.5],
[6. , 2.2, 4. , 1. ],
[6.1, 2.9, 4.7, 1.4],
[5.6, 2.9, 3.6, 1.3],
[6.7, 3.1, 4.4, 1.4],
[5.6, 3. , 4.5, 1.5],
[5.8, 2.7, 4.1, 1. ],
[6.2, 2.2, 4.5, 1.5],
[5.6, 2.5, 3.9, 1.1],
[5.9, 3.2, 4.8, 1.8],
[6.1, 2.8, 4. , 1.3],
[6.3, 2.5, 4.9, 1.5],
[6.1, 2.8, 4.7, 1.2],
[6.4, 2.9, 4.3, 1.3],
[6.6, 3. , 4.4, 1.4],
[6.8, 2.8, 4.8, 1.4],
[6.7, 3. , 5. , 1.7],
[6. , 2.9, 4.5, 1.5],
[5.7, 2.6, 3.5, 1. ],
[5.5, 2.4, 3.8, 1.1],
[5.5, 2.4, 3.7, 1. ],
[5.8, 2.7, 3.9, 1.2],
[6. , 2.7, 5.1, 1.6],
[5.4, 3. , 4.5, 1.5],
[6. , 3.4, 4.5, 1.6],
[6.7, 3.1, 4.7, 1.5],
[6.3, 2.3, 4.4, 1.3],
[5.6, 3. , 4.1, 1.3],
[5.5, 2.5, 4. , 1.3],
[5.5, 2.6, 4.4, 1.2],
[6.1, 3. , 4.6, 1.4],
[5.8, 2.6, 4. , 1.2],
[5. , 2.3, 3.3, 1. ],
[5.6, 2.7, 4.2, 1.3],
[5.7, 3. , 4.2, 1.2],
[5.7, 2.9, 4.2, 1.3],
[6.2, 2.9, 4.3, 1.3],
[5.1, 2.5, 3. , 1.1],
[5.7, 2.8, 4.1, 1.3],
[6.3, 3.3, 6. , 2.5],
[5.8, 2.7, 5.1, 1.9],
[7.1, 3. , 5.9, 2.1],
[6.3, 2.9, 5.6, 1.8],
[6.5, 3. , 5.8, 2.2],
[7.6, 3. , 6.6, 2.1],
[4.9, 2.5, 4.5, 1.7],
[7.3, 2.9, 6.3, 1.8],
[6.7, 2.5, 5.8, 1.8],
[7.2, 3.6, 6.1, 2.5],
[6.5, 3.2, 5.1, 2. ],
[6.4, 2.7, 5.3, 1.9],
```

```
       [6.8, 3. , 5.5, 2.1],
       [5.7, 2.5, 5. , 2. ],
       [5.8, 2.8, 5.1, 2.4],
       [6.4, 3.2, 5.3, 2.3],
       [6.5, 3. , 5.5, 1.8],
       [7.7, 3.8, 6.7, 2.2],
       [7.7, 2.6, 6.9, 2.3],
       [6. , 2.2, 5. , 1.5],
       [6.9, 3.2, 5.7, 2.3],
       [5.6, 2.8, 4.9, 2. ],
       [7.7, 2.8, 6.7, 2. ],
       [6.3, 2.7, 4.9, 1.8],
       [6.7, 3.3, 5.7, 2.1],
       [7.2, 3.2, 6. , 1.8],
       [6.2, 2.8, 4.8, 1.8],
       [6.1, 3. , 4.9, 1.8],
       [6.4, 2.8, 5.6, 2.1],
       [7.2, 3. , 5.8, 1.6],
       [7.4, 2.8, 6.1, 1.9],
       [7.9, 3.8, 6.4, 2. ],
       [6.4, 2.8, 5.6, 2.2],
       [6.3, 2.8, 5.1, 1.5],
       [6.1, 2.6, 5.6, 1.4],
       [7.7, 3. , 6.1, 2.3],
       [6.3, 3.4, 5.6, 2.4],
       [6.4, 3.1, 5.5, 1.8],
       [6. , 3. , 4.8, 1.8],
       [6.9, 3.1, 5.4, 2.1],
       [6.7, 3.1, 5.6, 2.4],
       [6.9, 3.1, 5.1, 2.3],
       [5.8, 2.7, 5.1, 1.9],
       [6.8, 3.2, 5.9, 2.3],
       [6.7, 3.3, 5.7, 2.5],
       [6.7, 3. , 5.2, 2.3],
       [6.3, 2.5, 5. , 1.9],
       [6.5, 3. , 5.2, 2. ],
       [6.2, 3.4, 5.4, 2.3],
       [5.9, 3. , 5.1, 1.8]])
```

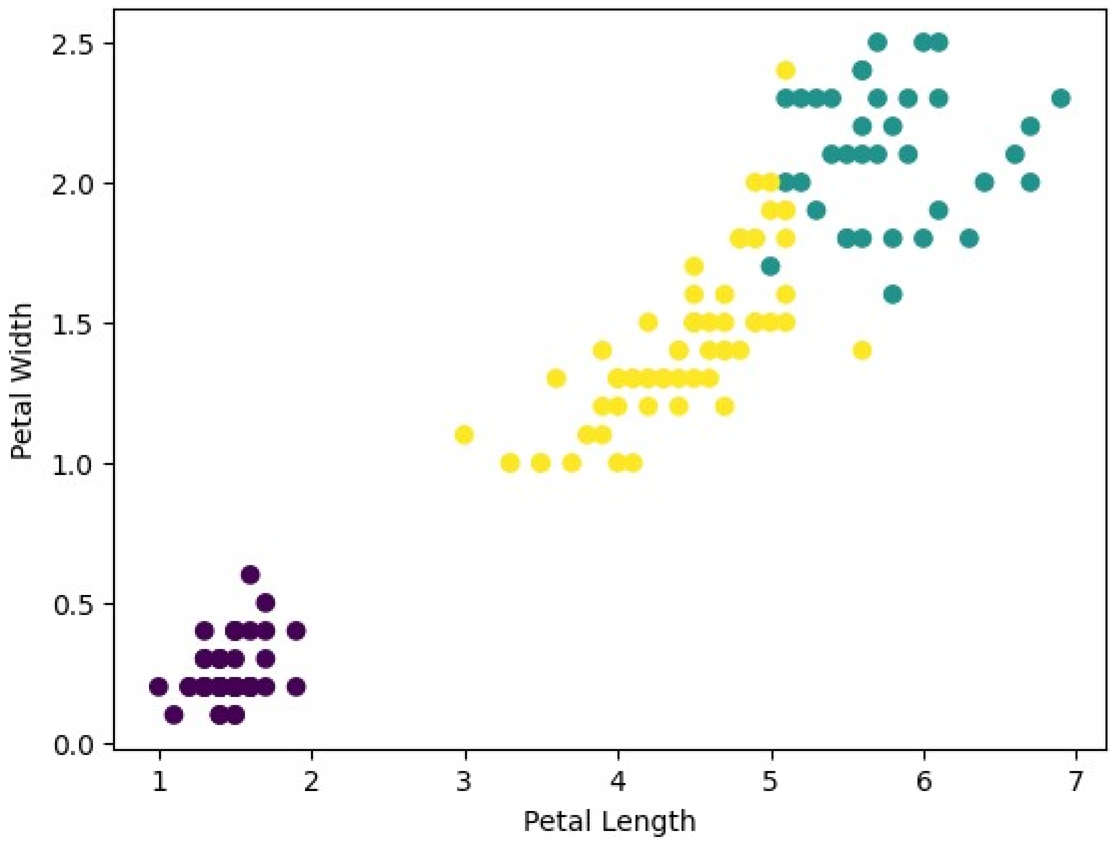
In [ ]:

```python
from scipy.spatial.distance import pdist

dist_matrix = pdist(X)
```

In [ ]:
```
from scipy.cluster.hierarchy import dendrogram, linkage
from scipy.cluster.hierarchy import fcluster

import matplotlib.pyplot as plt
Z = linkage(dist_matrix, method='ward')

clusters = fcluster(Z, t=3, criterion='maxclust')
ltp=plt
dendrogram(Z)
ltp.show()
```



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

df = pd.DataFrame(data=X, columns=['sepal_length', 'sepal_width', 'petal_le
ngth', 'petal_width'])
df['cluster'] = clusters - 1

plt.scatter(df['petal_length'], df['petal_width'], c=df['cluster'])
plt.xlabel('Petal Length')
plt.ylabel('Petal Width')
plt.show()
```

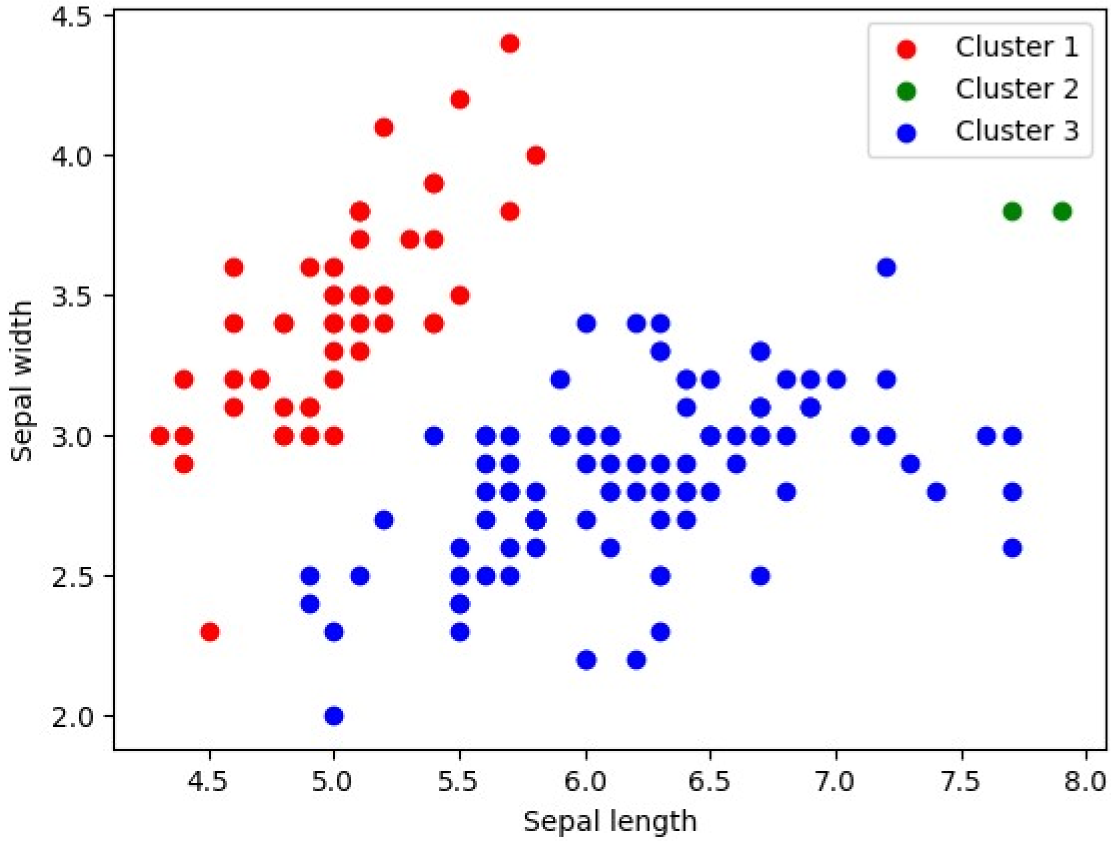


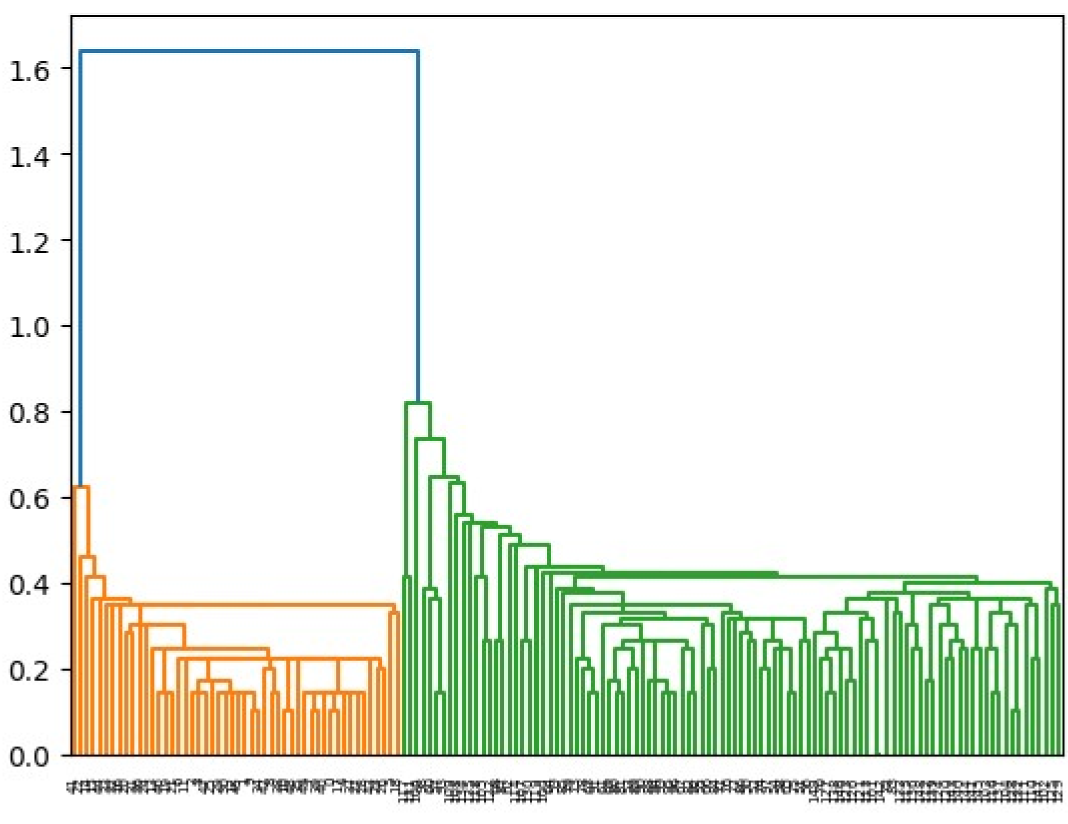
In [ ]:

In [ ]:

In [ ]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from scipy.cluster.hierarchy import dendrogram, fcluster, linkage
iris = load_iris()
X = iris.data
y = iris.target
Z = linkage(X, method='single', metric='euclidean')
clusters = fcluster(Z, t=3, criterion='maxclust')
for i in range(1, 4):
    print(f"Cluster {i} has {np.sum(clusters == i)} points")
colors = ['red', 'green', 'blue']

for i in range(1, 4):
    plt.scatter(X[clusters == i, 0], X[clusters == i, 1], c=colors[i-1], la
bel=f'Cluster {i}')

plt.xlabel('Sepal length')
plt.ylabel('Sepal width')
plt.legend()
plt.show()
ltp=plt
dendrogram(Z)
ltp.show()
```

```
Cluster 1 has 50 points
Cluster 2 has 2 points
Cluster 3 has 98 points
```





Without Lib

In [ ]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
```

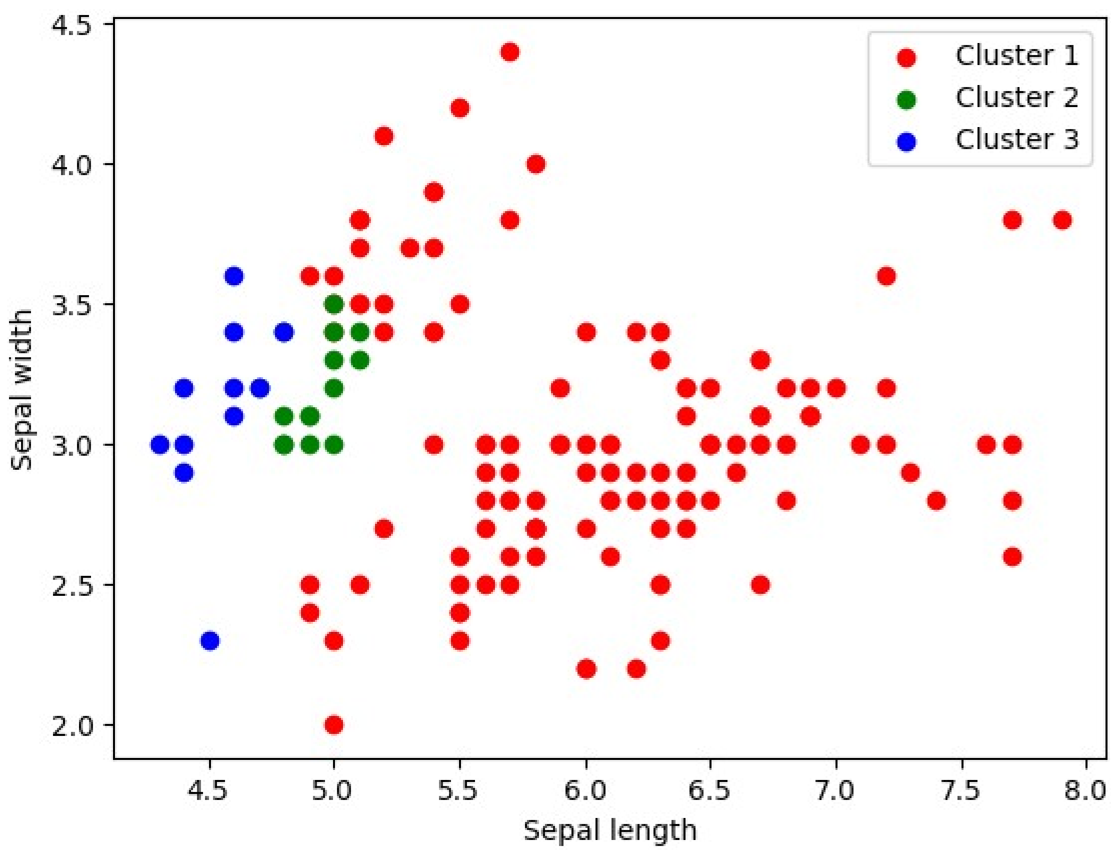
In [ ]:
```
iris = load_iris()
X = iris.data
y = iris.target
```

In [ ]:
```
def euclidean_distance(x1, x2):
    return np.sqrt(np.sum((x1 - x2)**2))

class Cluster:
    def __init__(self, center):
        self.center = center
        self.points = [center]

    def update_center(self):
        self.center = np.mean(self.points, axis=0)

    def distance_to(self, other):
        return euclidean_distance(self.center, other.center)

    def merge(self, other):
        self.points.extend(other.points)
        self.update_center()

def diana_clustering(X, k):
    # Initialize clusters with the first k data points as centers
    clusters = [Cluster(center=X[i]) for i in range(k)]

    # Assign each remaining data point to its closest cluster
    for i in range(k, len(X)):
        distances = [c.distance_to(Cluster(center=X[i])) for c in clusters]
        closest_cluster = clusters[np.argmin(distances)]
        closest_cluster.points.append(X[i])
        closest_cluster.update_center()

    # Iteratively merge clusters until there are only k clusters remaining
    while len(clusters) > k:
        # Compute the distance between each pair of clusters
        distances = np.zeros((len(clusters), len(clusters)))
        for i in range(len(clusters)):
            for j in range(i+1, len(clusters)):
                distances[i,j] = clusters[i].distance_to(clusters[j])
        distances += distances.T

        # Identify the pair of clusters with the minimum distance
        i, j = np.unravel_index(np.argmin(distances), distances.shape)

        # Merge the two clusters
        clusters[i].merge(clusters[j])
        del clusters[j]

    return clusters
```

In [ ]:
```
clusters = diana_clustering(X, k=3)

for i, c in enumerate(clusters):
    print(f"Cluster {i+1} has {len(c.points)} points and center {c.cente
r}")
```

```
Cluster 1 has 124 points and center [6.07096774 3.03467742 4.24354839 1.401
6129 ]
Cluster 2 has 15 points and center [4.92       3.2        1.52666667 0.2533
3333]
Cluster 3 has 11 points and center [4.53636364 3.11818182 1.32727273 0.2090
9091]
```

In [ ]:
```
colors = ['red', 'green', 'blue']
labels = ['Cluster 1', 'Cluster 2', 'Cluster 3']

for i, c in enumerate(clusters):
    plt.scatter([p[0] for p in c.points], [p[1] for p in c.points], c=color
s[i], label=labels[i])


plt.xlabel('Sepal length')
plt.ylabel('Sepal width')
plt.legend()
plt.show()
```



In [ ]:

In [ ]:
```
#FIND-S

import numpy as np
import pandas as pd
```

In [ ]:
```
print("The given data is:")
data = pd.read_csv("EnjoySport.csv")
data
```

```
The given data is:
```

Out[ ]:

	Sky	AirTemp	Humidity	Wind	Water	Forecast	EnjoySport
0	Sunny	Warm	Normal	Strong	Warm	Same	Yes
1	Sunny	Warm	High	Strong	Warm	Same	Yes
2	Rainy	Cold	High	Strong	Warm	Change	No
3	Sunny	Warm	High	Strong	Cool	Change	Yes

In [ ]:
```
def FindS(data):
    dt = np.array(data)
    n = len(dt[0])-1
    target = np.array(data)[:,-1]
    specific_hypothesis=["_"]*n
    print("H0 = ",specific_hypothesis)
    hypothesis = []
    for i, val in enumerate(target):
        if val == 'Yes':
            specific_hypothesis=dt[i][:-1].copy()
            hypothesis.append(specific_hypothesis)
            break
    for i, val in enumerate(dt):

        if target[i] =='Yes':
            for x in range(n):
                if val[x] != specific_hypothesis[x]:
                    specific_hypothesis[x]='?'
                else:
                    pass
        hypothesis.append(specific_hypothesis)
        print("H"+str(i+1)+" = ",specific_hypothesis)

    print("\nThe maximally specific hypothesis is:\n", specific_hypothesis)
    return
```

In [ ]:
```
FindS(data)
```

```
H0 =  ['_', '_', '_', '_', '_', '_']
H1 =  ['Sunny' 'Warm' 'Normal' 'Strong' 'Warm' 'Same']
H2 =  ['Sunny' 'Warm' '?' 'Strong' 'Warm' 'Same']
H3 =  ['Sunny' 'Warm' '?' 'Strong' 'Warm' 'Same']
H4 =  ['Sunny' 'Warm' '?' 'Strong' '?' '?']

The maximally specific hypothesis is:
 ['Sunny' 'Warm' '?' 'Strong' '?' '?']
```

In [ ]:

In [ ]:
```
# ADABOOST


from sklearn.ensemble import AdaBoostClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
X, y = make_classification(n_samples=1000, n_features=10, n_informative=5,
n_redundant=0, random_state=42)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, ra
ndom_state=42)
```

New Section

In [ ]:
```
base_estimator = DecisionTreeClassifier(max_depth=1, random_state=42)
adaboost = AdaBoostClassifier(base_estimator=base_estimator, n_estimators=5
0, random_state=42)
adaboost.fit(X_train, y_train)
y_pred = adaboost.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy: %.2f%%" % (accuracy * 100.0))
```

```
Accuracy: 81.00%

/usr/local/lib/python3.9/dist-packages/sklearn/ensemble/_base.py:166: Futur
eWarning: `base_estimator` was renamed to `estimator` in version 1.2 and wi
ll be removed in 1.4.
  warnings.warn(
```

In [ ]:
```
from sklearn.metrics._plot.confusion_matrix import confusion_matrix
cm=confusion_matrix(y_test,y_pred)
cm
```

Out[ ]:
```
array([[89, 15],
       [23, 73]])
```

In [ ]:

```

```

without libraries

In [ ]:
```
import numpy as np
from typing import List


class DecisionStump:
    def __init__(self):
        self.polarity = 1
        self.feature_index = None
        self.threshold = None
        self.alpha = None


class AdaBoost:
    def __init__(self, num_estimators):
        self.num_estimators = num_estimators
        self.estimators = []

    def fit(self, X, y):
        n_samples, n_features = X.shape

        # Initialize weights to 1/N
        weights = np.full(n_samples, 1 / n_samples)

        for _ in range(self.num_estimators):
            # Train a decision stump on the weighted dataset
            stump = DecisionStump()
            min_error = float('inf')
            for feature_idx in range(n_features):
                feature_values = np.expand_dims(X[:, feature_idx], axis=1)
                unique_values = np.unique(feature_values)
                for threshold in unique_values:
                    # Try all thresholds for this feature
                    p = 1
                    prediction = np.ones_like(y)
                    prediction[X[:, feature_idx] < threshold] = -1
                    error = sum(weights[y != prediction])
                    if error > 0.5:
                        error = 1 - error
                        p = -1

                    # Keep track of the best decision stump so far
                    if error < min_error:
                        stump.polarity = p
                        stump.threshold = threshold
                        stump.feature_index = feature_idx
                        min_error = error

            # Calculate the alpha value for the decision stump
            eps = 1e-10
            stump.alpha = 0.5 * np.log((1.0 - min_error + eps) / (min_error
+ eps))

            # Update the sample weights based on the decision stump
            predictions = np.ones_like(y)
            negative_idx = (stump.polarity * X[:, stump.feature_index] < st
```

```
ump.polarity * stump.threshold)
            predictions[negative_idx] = -1
            weights *= np.exp(-stump.alpha * y * predictions)
            weights /= np.sum(weights)

            # Save the decision stump
            self.estimators.append(stump)

    def predict(self, X):
        n_samples = X.shape[0]
        predictions = np.zeros(n_samples)
        for stump in self.estimators:
            pred = np.ones(n_samples)
            negative_idx = (stump.polarity * X[:, stump.feature_index] < st
ump.polarity * stump.threshold)
            pred[negative_idx] = -1
            predictions += stump.alpha * pred

        return np.sign(predictions)
```

In [ ]:

```
X = np.array([[1, 2], [2, 3], [3, 4], [4, 5], [5, 6], [6, 7]])
y = np.array([1, 1, 1, -1, -1, -1])

adaboost = AdaBoost(num_estimators=3)
adaboost.fit(X, y)

# Predict on new data
X_test = np.array([[0, 1], [7, 8]])
y_pred = adaboost.predict(X_test)
print(y_pred)
```

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

In [ ]:

AGNES(BOTTOM UP)

In [ ]:

```
#AGNES

from sklearn.datasets import load_iris

iris = load_iris()
X = iris.data
```

In [ ]:

```
from sklearn.cluster import AgglomerativeClustering

clustering = AgglomerativeClustering(n_clusters=3, linkage='ward', affinity
='euclidean')
clustering.fit(X)
```

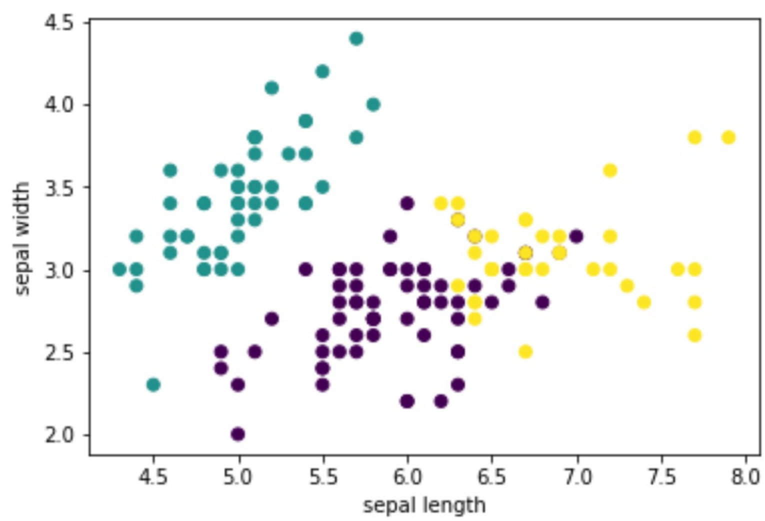
```
/usr/local/lib/python3.9/dist-packages/sklearn/cluster/_agglomerative.py:98
3: FutureWarning: Attribute `affinity` was deprecated in version 1.2 and wi
ll be removed in 1.4. Use `metric` instead
  warnings.warn(
```

Out[ ]:

```
▾ AgglomerativeClustering
AgglomerativeClustering(affinity='euclidean', n_clusters=3)
```

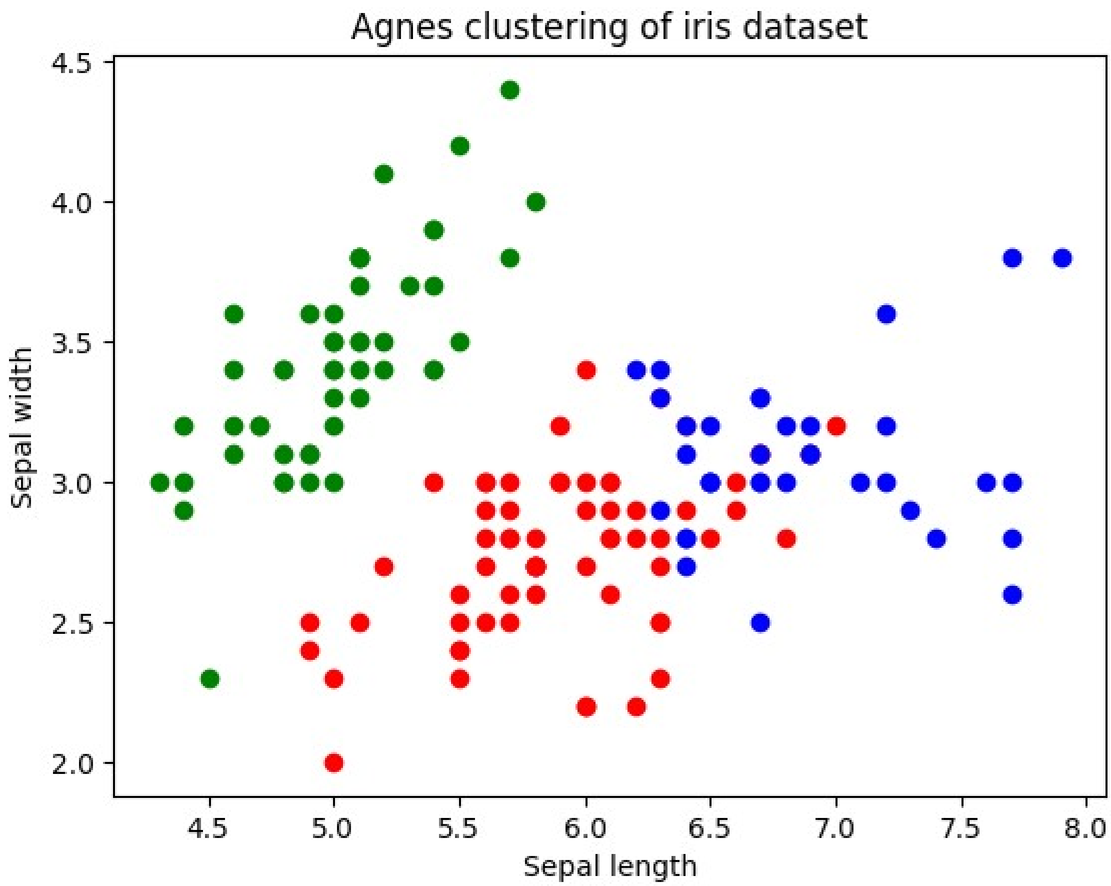
In [ ]:

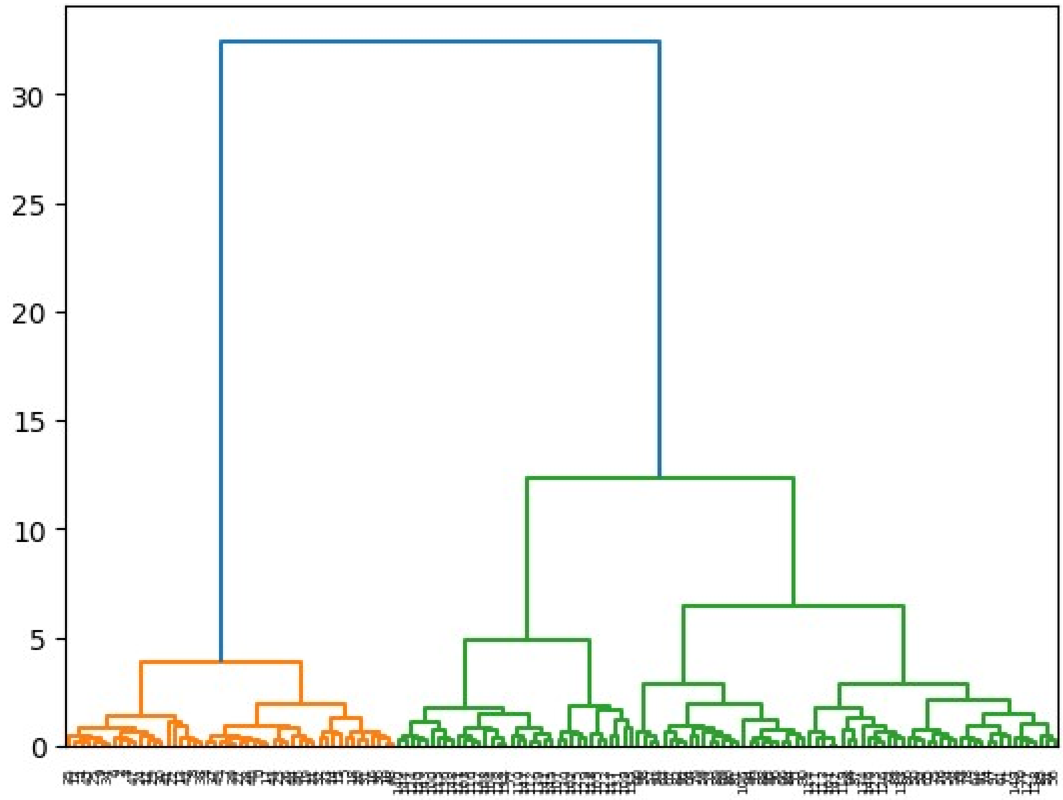
```
import matplotlib.pyplot as plt
import pandas as pd

df = pd.DataFrame(data=X, columns=['sepal_length', 'sepal_width', 'petal_le
ngth', 'petal_width'])
df['cluster'] = clustering.labels_

plt.scatter(df['sepal_length'], df['sepal_width'], c=df['cluster'])
plt.xlabel('sepal length')
plt.ylabel('sepal width')
plt.show()
```



In [ ]:
```
import numpy as np
from typing import List, Tuple
from matplotlib import pyplot as plt
from sklearn.datasets import load_iris
from sklearn.cluster import AgglomerativeClustering
from scipy.cluster.hierarchy import dendrogram, linkage

# Load the iris dataset
iris = load_iris()
X = iris.data
y = iris.target

# Perform Agnes clustering with k=3 (number of classes in the iris dataset)
agnes = AgglomerativeClustering(n_clusters=3)
y_pred = agnes.fit_predict(X)

# Print the clusters
print("Clusters:")
for i in range(3):
    print(f"Cluster {i}: {np.where(y_pred == i)[0]}")

# Visualize the clusters
colors = ['r', 'g', 'b']
for i in range(X.shape[0]):
    plt.scatter(X[i, 0], X[i, 1], color=colors[int(y_pred[i])])
plt.xlabel('Sepal length')
plt.ylabel('Sepal width')
plt.title('Agnes clustering of iris dataset')
plt.show()
ltp=plt
# Create and show the dendrogram
Z = linkage(X, method='ward')
dendrogram(Z)
ltp.show()
```

```
Clusters:
Cluster 0: [ 50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65
66  67
  68  69  70  71  72  73  74  75  76  78  79  80  81  82  83  84  85  86
  87  88  89  90  91  92  93  94  95  96  97  98  99 101 106 113 114 119
 121 123 126 127 133 134 138 142 146 149]
Cluster 1: [ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
21 22 23
 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47
 48 49]
Cluster 2: [ 77 100 102 103 104 105 107 108 109 110 111 112 115 116 117 118
120 122
 124 125 128 129 130 131 132 135 136 137 139 140 141 143 144 145 147 148]
```





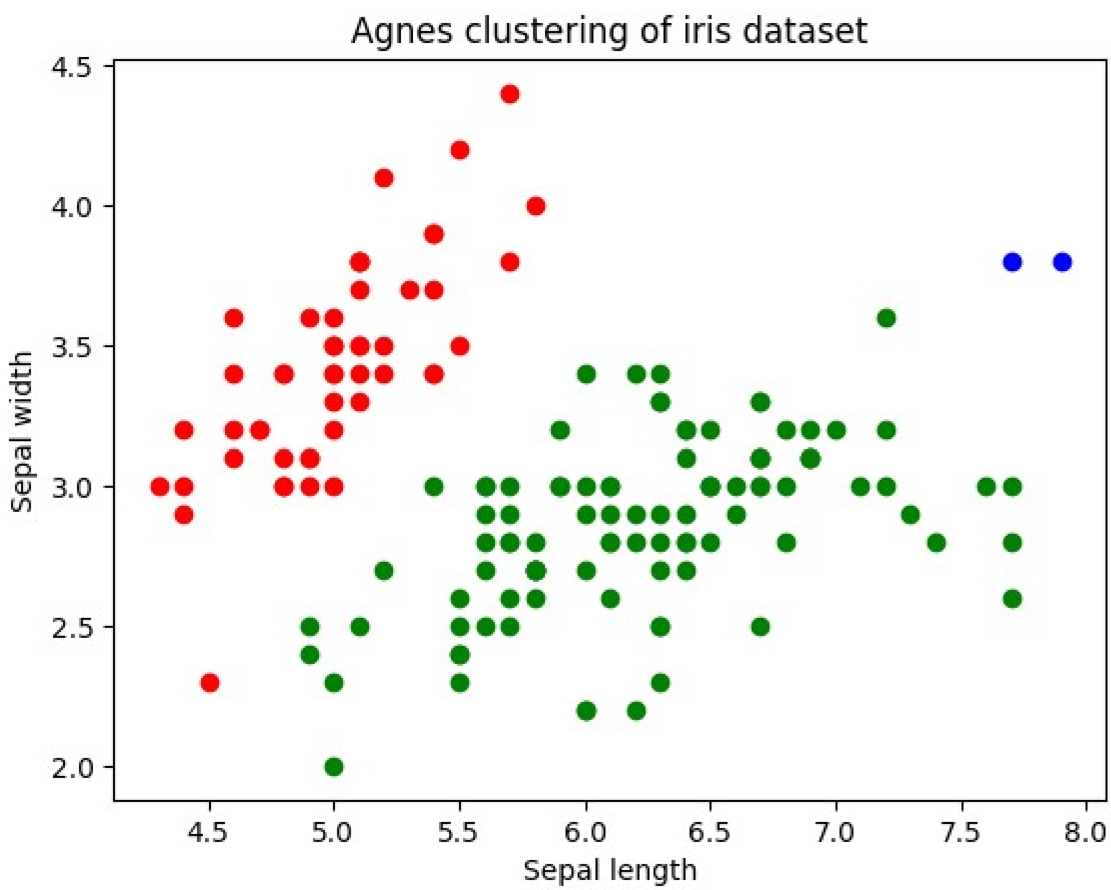
In [ ]:

Without Libraries

In [ ]:
```
import numpy as np
from typing import List, Tuple
from matplotlib import pyplot as plt
from sklearn.datasets import load_iris


class Agnes:
    def __init__(self, k):
        self.k = k
        self.clusters = []

    def fit(self, X):
        n_samples = X.shape[0]
        self.clusters = [[i] for i in range(n_samples)]

        while len(self.clusters) > self.k:
            # Find the closest pair of clusters
            min_dist = float('inf')
            closest_clusters = None
            for i in range(len(self.clusters)):
                for j in range(i + 1, len(self.clusters)):
                    dist = self._single_linkage_dist(X, self.clusters[i], s
elf.clusters[j])
                    if dist < min_dist:
                        min_dist = dist
                        closest_clusters = (i, j)

            # Merge the closest pair of clusters
            self.clusters[closest_clusters[0]] += self.clusters[closest_clu
sters[1]]
            del self.clusters[closest_clusters[1]]

    def predict(self, X):
        y_pred = np.zeros(X.shape[0])
        for i, cluster in enumerate(self.clusters):
            for j in cluster:
                y_pred[j] = i
        return y_pred

    def _single_linkage_dist(self, X, cluster1, cluster2):
        min_dist = float('inf')
        for i in cluster1:
            for j in cluster2:
                dist = np.linalg.norm(X[i] - X[j])
                if dist < min_dist:
                    min_dist = dist
        return min_dist


# Load the iris dataset
iris = load_iris()
X = iris.data
y = iris.target

# Perform Agnes clustering with k=3 (number of classes in the iris dataset)
```

```
agnes = Agnes(k=3)
agnes.fit(X)

# Predict the clusters for the data points
y_pred = agnes.predict(X)

# Print the clusters
print("Clusters:")
for i, cluster in enumerate(agnes.clusters):
    print(f"Cluster {i}: {cluster}")

# Visualize the clusters
colors = ['r', 'g', 'b']
for i in range(X.shape[0]):
    plt.scatter(X[i, 0], X[i, 1], color=colors[int(y_pred[i])])
plt.xlabel('Sepal length')
plt.ylabel('Sepal width')
plt.title('Agnes clustering of iris dataset')
plt.show()
```

```
Clusters:
Cluster 0: [0, 17, 40, 4, 37, 7, 39, 49, 27, 28, 35, 10, 48, 23, 26, 43, 1,
9, 34, 45, 12, 29, 30, 25, 2, 3, 47, 8, 38, 42, 11, 6, 19, 21, 46, 13, 24,
36, 20, 31, 5, 18, 16, 32, 33, 44, 15, 14, 22, 41]
Cluster 1: [50, 52, 86, 51, 56, 54, 58, 65, 75, 74, 97, 77, 76, 71, 53, 89,
69, 80, 81, 67, 82, 92, 88, 94, 95, 96, 99, 90, 61, 55, 66, 84, 63, 91, 78,
73, 79, 85, 59, 70, 127, 138, 123, 126, 146, 149, 101, 142, 113, 121, 72, 8
3, 133, 103, 116, 137, 104, 128, 132, 110, 147, 111, 141, 145, 112, 139, 12
0, 143, 140, 144, 124, 115, 136, 148, 102, 125, 129, 64, 100, 119, 107, 13
0, 114, 62, 68, 87, 105, 122, 118, 135, 134, 108, 109, 57, 93, 60, 98, 106]
Cluster 2: [117, 131]
```



In [ ]:

In [ ]:
```
#CART

import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.preprocessing import LabelEncoder
```

In [ ]:
```
data = pd.read_csv('CART.csv')
print("Sample Dataset - \n",data,"\n")
```

```
Sample Dataset - 
        age    job house     credit loan_approved
0    young  False    No       Fair            No
1    young  False    No       Good            No
2    young   True    No       Good           Yes
3    young   True   Yes       Fair           Yes
4    young  False    No       Fair            No
5   middle  False    No       Fair            No
6   middle  False    No       Good            No
7   middle   True   Yes       Good           Yes
8   middle  False   Yes  Excellent           Yes
9   middle  False   Yes  Excellent           Yes
10     old  False   Yes  Excellent           Yes
11     old  False   Yes       Good           Yes
12     old   True    No       Good           Yes
13     old   True    No  Excellent           Yes
14     old  False    No       Fair            No
```

In [ ]:
```
le_age = LabelEncoder()
data['age_n'] = le_age.fit_transform(data['age'])
le_job = LabelEncoder()
data['job_n'] = le_job.fit_transform(data['job'])
le_house = LabelEncoder()
data['house_n'] = le_house.fit_transform(data['house'])
le_credit = LabelEncoder()
data['credit_n'] = le_credit.fit_transform(data['credit'])
le_loan = LabelEncoder()
data['loan_n'] = le_loan.fit_transform(data['loan_approved'])
print("Given Data after Encoding - \n",data,"\n")
```

```
Given Data after Encoding -
        age    job house     credit loan_approved  age_n  job_n  house_n  \
0    young  False    No       Fair            No      2      0        0
1    young  False    No       Good            No      2      0        0
2    young   True    No       Good           Yes      2      1        0
3    young   True   Yes       Fair           Yes      2      1        1
4    young  False    No       Fair            No      2      0        0
5   middle  False    No       Fair            No      0      0        0
6   middle  False    No       Good            No      0      0        0
7   middle   True   Yes       Good           Yes      0      1        1
8   middle  False   Yes  Excellent           Yes      0      0        1
9   middle  False   Yes  Excellent           Yes      0      0        1
10     old  False   Yes  Excellent           Yes      1      0        1
11     old  False   Yes       Good           Yes      1      0        1
12     old   True    No       Good           Yes      1      1        0
13     old   True    No  Excellent           Yes      1      1        0
14     old  False    No       Fair            No      1      0        0

    credit_n  loan_n
0          1       0
1          2       0
2          2       1
3          1       1
4          1       0
5          1       0
6          2       0
7          2       1
8          0       1
9          0       1
10         0       1
11         2       1
12         2       1
13         0       1
14         1       0
```

In [ ]:
```
X = data[['age_n','job_n','house_n','credit_n']]
print("X - Values\n",X,"\n")
```

```
X - Values
     age_n  job_n  house_n  credit_n
0       2      0        0         1
1       2      0        0         2
2       2      1        0         2
3       2      1        1         1
4       2      0        0         1
5       0      0        0         1
6       0      0        0         2
7       0      1        1         2
8       0      0        1         0
9       0      0        1         0
10      1      0        1         0
11      1      0        1         2
12      1      1        0         2
13      1      1        0         0
14      1      0        0         1
```

In [ ]:
```
y = data['loan_approved']
ly=LabelEncoder()
y=ly.fit_transform(y)
print("Y - Values\n",y,"\n")
```

```
Y - Values
 [0 0 1 1 0 0 0 1 1 1 1 1 1 1 0]
```

In [ ]:
```
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.25)
```

In [ ]:
```
model = DecisionTreeClassifier(criterion='gini')
model.fit(X_train,y_train)
```

In [ ]:
```
print("Pedicted Values - ",model.predict(X_test))
print("Original Values of Predicted Values - ",y_test.values)
print("Predicting for - [young,False,No,Good] - ",model.predict([[2,0,0,
2]]))
print("Accuracy of Model",model.score(X_test,y_test))
```

```
Pedicted Values -  ['Yes' 'Yes' 'No' 'No']
Original Values of Predicted Values -  ['Yes' 'Yes' 'No' 'No']
Predicting for - [young,False,No,Good] -  ['No']
Accuracy of Model 1.0

/usr/local/lib/python3.9/dist-packages/sklearn/base.py:439: UserWarning: X
does not have valid feature names, but DecisionTreeClassifier was fitted wi
th feature names
  warnings.warn(
```

In [ ]:
```
#ID3

import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.preprocessing import LabelEncoder
```

In [ ]:
```
data = pd.read_csv('id3.csv')
print("Sample Dataset - \n",data,"\n")
```

```
Sample Dataset -
       a1    a2      a3 classification
0   True   Hot    High             No
1   True   Hot    High             No
2  False   Hot    High            Yes
3  False  Cool  Normal            Yes
4  False  Cool  Normal            Yes
5   True  Cool    High             No
6   True   Hot    High             No
7   True   Hot  Normal            Yes
8  False  Cool  Normal            Yes
9  False  Cool    High            Yes
```

In [ ]:
```
le_a1 = LabelEncoder()
data['a1_n'] = le_a1.fit_transform(data['a1'])

le_a2 = LabelEncoder()
data['a2_n'] = le_a1.fit_transform(data['a2'])

le_a3 = LabelEncoder()
data['a3_n'] = le_a1.fit_transform(data['a3'])

print("Given Data after Encoding - \n",data,"\n")
```

```
Given Data after Encoding -
       a1    a2      a3 classification  a1_n  a2_n  a3_n
0   True   Hot    High             No     1     1     0
1   True   Hot    High             No     1     1     0
2  False   Hot    High            Yes     0     1     0
3  False  Cool  Normal            Yes     0     0     1
4  False  Cool  Normal            Yes     0     0     1
5   True  Cool    High             No     1     0     0
6   True   Hot    High             No     1     1     0
7   True   Hot  Normal            Yes     1     1     1
8  False  Cool  Normal            Yes     0     0     1
9  False  Cool    High            Yes     0     0     0
```

In [ ]:
```
X = data[['a1_n','a2_n','a3_n']]
print("X - Values\n",X,"\n")

y = data['classification']
print("Y - Values\n",y,"\n")
```

```
X - Values
    a1_n  a2_n  a3_n
0     1     1     0
1     1     1     0
2     0     1     0
3     0     0     1
4     0     0     1
5     1     0     0
6     1     1     0
7     1     1     1
8     0     0     1
9     0     0     0

Y - Values
 0     No
1     No
2    Yes
3    Yes
4    Yes
5     No
6     No
7    Yes
8    Yes
9    Yes
Name: classification, dtype: object
```

In [ ]:
```
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.3)

model = DecisionTreeClassifier(criterion='entropy')
model.fit(X_train,y_train)

print("Values predicted from test dataset - ",model.predict(X_test))
print("Original Values of test dataset - ",y_test.values)
print("Accuracy of Model",model.score(X_test,y_test))
```

```
Values predicted from test dataset -  ['Yes' 'No' 'Yes']
Original Values of test dataset -  ['Yes' 'No' 'Yes']
Accuracy of Model 1.0
```

In [ ]:

In [ ]:

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

def entropy(data, target_attribute):
    # Calculate the entropy of a dataset
    target_labels = data[target_attribute].unique()
    entropy = 0
    for label in target_labels:
        count = len(data[data[target_attribute] == label])
        p = count / len(data)
        entropy -= p * math.log2(p)
    return entropy

def information_gain(data, attribute, target_attribute):
    # Calculate the information gain of an attribute in a dataset
    attribute_values = data[attribute].unique()
    gain = entropy(data, target_attribute)
    for value in attribute_values:
        subset = data[data[attribute] == value]
        p = len(subset) / len(data)
        gain -= p * entropy(subset, target_attribute)
    return gain

def id3(data, attributes, target_attribute):
    # Build a decision tree using the ID3 algorithm
    unique_labels = data[target_attribute].unique()
    if len(unique_labels) == 1:
        # If all examples have the same label, return a leaf node with that
label
        return unique_labels[0]
    if len(attributes) == 0:
        # If there are no more attributes to split on, return a leaf node w
ith the majority label
        label_counts = data[target_attribute].value_counts()
        return label_counts.index[0]
    best_attribute = max(attributes, key=lambda attribute: information_gain
(data, attribute, target_attribute))
    tree = {best_attribute: {}}
    remaining_attributes = [attribute for attribute in attributes if attrib
ute != best_attribute]
    for value in data[best_attribute].unique():
        subset = data[data[best_attribute] == value]
        if len(subset) == 0:
            # If there are no examples with this value, return a leaf node
with the majority label
            label_counts = data[target_attribute].value_counts()
            tree[best_attribute][value] = label_counts.index[0]
        else:
            # Recursively build the subtree using the remaining attributes
            tree[best_attribute][value] = id3(subset, remaining_attributes,
target_attribute)
    return tree

def predict(row, tree):
```

```
    # Traverse the decision tree until a leaf node is reached
    while type(tree) == dict:
        attribute = list(tree.keys())[0]
        value = row[attribute]
        if value not in tree[attribute]:
            # If the value is not in the decision tree, return the majority
class
            label_counts = {}
            for label in tree[attribute].values():
                if label not in label_counts:
                    label_counts[label] = 0
                label_counts[label] += 1
            return max(label_counts, key=label_counts.get)
        tree = tree[attribute][value]
    return tree

# Load the tennis dataset
data = pd.read_csv('tennis.csv')

# Define the target attribute
target_attribute = 'play'

# Define the attributes
attributes = list(data.columns)
attributes.remove(target_attribute)

# Split the data into training and testing sets
split_index = int(0.8 * len(data))
train_data = data.iloc[:split_index]
test_data = data.iloc[split_index:]

# Train the decision tree
tree = id3(train_data, attributes, target_attribute)

# Test the decision tree
correct_predictions = 0
for index, row in test_data.iterrows():
    if predict(row, tree) == row[target_attribute]:
        correct_predictions += 1

accuracy = correct_predictions
accuracy = correct_predictions / len(test_data)
print(f"Accuracy: {accuracy}")
```

```
Accuracy: 0.6666666666666666
```

In [ ]:

CART ALGO without lib

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

# Define the Node class to represent a decision tree node
class Node:
    def __init__(self, feature=None, threshold=None, left=None, right=None,
label=None):
        self.feature = feature  # index of feature to split on
        self.threshold = threshold  # threshold to split on
        self.left = left  # left subtree
        self.right = right  # right subtree
        self.label = label  # label of leaf node

# Define the decision tree function
def decision_tree(X, y):
    n, m = X.shape

    # Base case: all labels are the same
    if len(np.unique(y)) == 1:
        return Node(label=y[0])

    # Base case: no more features to split on
    if m == 0:
        return Node(label=np.bincount(y).argmax())

    # Find the best feature to split on
    best_feature, best_threshold, min_gini = None, None, 1.0
    for i in range(m):
        for threshold in np.unique(X[:, i]):
            left_indices = X[:, i] < threshold
            left_y = y[left_indices]
            right_y = y[~left_indices]
            if len(left_y) > 0 and len(right_y) > 0:
                gini = (len(left_y) / n) * gini_index(left_y) + (len(right_
y) / n) * gini_index(right_y)
                if gini < min_gini:
                    best_feature, best_threshold, min_gini = i, threshold,
gini

    # Create the node and its subtrees
    left_indices = X[:, best_feature] < best_threshold
    left = decision_tree(X[left_indices], y[left_indices])
    right = decision_tree(X[~left_indices], y[~left_indices])
    return Node(feature=best_feature, threshold=best_threshold, left=left,
right=right)

# Define the Gini index function
def gini_index(y):
    _, counts = np.unique(y, return_counts=True)
    probs = counts / len(y)
    return 1 - np.sum(probs ** 2)

# Test the decision tree on the iris dataset
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
```

```
iris = load_iris()
X_train, X_test, y_train, y_test = train_test_split(iris.data, iris.target,
test_size=0.2, random_state=42)

tree = decision_tree(X_train, y_train)

# Define a function to predict the label of a single instance using the dec
ision tree
def predict(instance, tree):
    if tree.label is not None:
        return tree.label
    elif instance[tree.feature] < tree.threshold:
        return predict(instance, tree.left)
    else:
        return predict(instance, tree.right)

# Test the accuracy of the decision tree on the test set
y_pred = np.array([predict(instance, tree) for instance in X_test])
accuracy = np.mean(y_pred == y_test)
print(f"Accuracy: {accuracy}")
```

```
Accuracy: 1.0
```

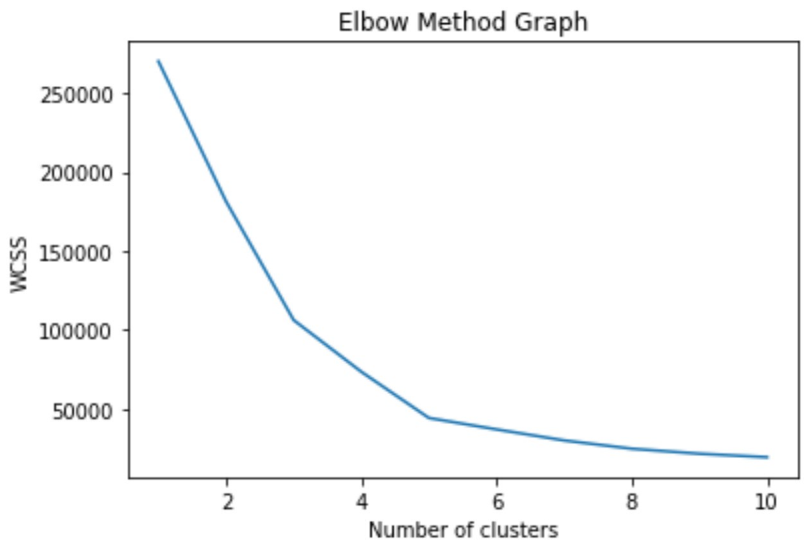
In [ ]:

```
```

In [ ]:

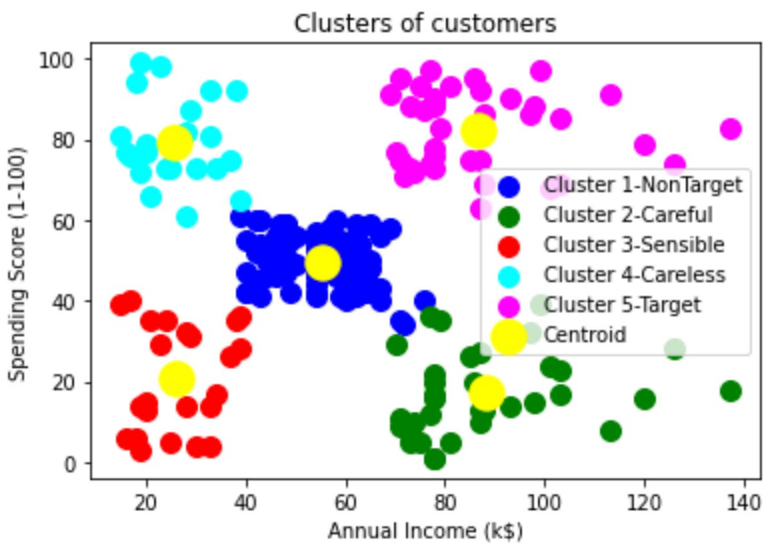
```
#KMeans

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

dataset = pd.read_csv("kmeans.csv")
x = dataset.iloc[:,[3,4]].values
```

In [ ]:
```
from sklearn.cluster import KMeans
import warnings
warnings.filterwarnings("ignore")
wcss_list= []  #Initializing the list for the values of WCSS

#Using for loop for iterations from 1 to 10.
for i in range(1, 11):
    kmeans = KMeans(n_clusters=i, init='k-means++',random_state= 42)
    kmeans.fit(x)
    wcss_list.append(kmeans.inertia_)
plt.plot(range(1, 11), wcss_list)
plt.title('Elbow Method Graph')
plt.xlabel('Number of clusters')
plt.ylabel('WCSS')
plt.show()
```



In [ ]:
```
kmeans = KMeans(n_clusters = 5, init = 'k-means++', random_state = 42)
y_kmeans = kmeans.fit_predict(x)
```

In [ ]:
```
plt.scatter(x[y_kmeans == 0, 0], x[y_kmeans == 0, 1], s = 100, c = 'blue',
label = 'Cluster 1-NonTarget') #for first cluster
plt.scatter(x[y_kmeans == 1, 0], x[y_kmeans == 1, 1], s = 100, c = 'green',
label = 'Cluster 2-Careful') #for second cluster
plt.scatter(x[y_kmeans == 2, 0], x[y_kmeans == 2, 1], s = 100, c = 'red', l
abel = 'Cluster 3-Sensible') #for third cluster
plt.scatter(x[y_kmeans == 3, 0], x[y_kmeans == 3, 1], s = 100, c = 'cyan',
label = 'Cluster 4-Careless') #for fourth cluster
plt.scatter(x[y_kmeans == 4, 0], x[y_kmeans == 4, 1], s = 100, c = 'magenta
', label = 'Cluster 5-Target') #for fifth cluster
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], s
= 300, c = 'yellow', label = 'Centroid')
plt.title('Clusters of customers')
plt.xlabel('Annual Income (k$)')
plt.ylabel('Spending Score (1-100)')
plt.legend()
plt.show()
```

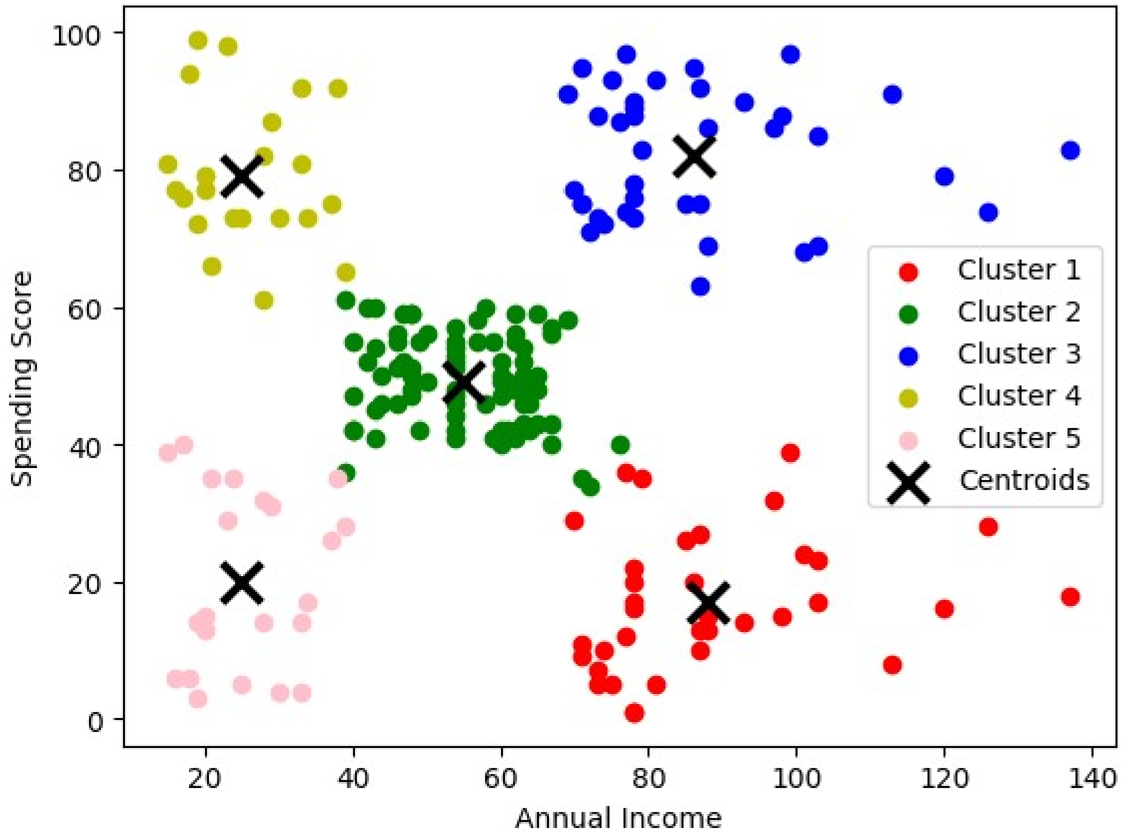


In [ ]:

In [ ]:

without library

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

def k_means(X, K, max_iters=100):
    centroids = X[np.random.choice(X.shape[0], K, replace=False)]

    for i in range(max_iters):
        # Assign points to nearest centroid
        distances = np.sqrt(np.sum((X[:, np.newaxis, :] - centroids) ** 2,
axis=2))
        labels = np.argmin(distances, axis=1)

        # Update centroids
        for k in range(K):
            centroids[k] = np.mean(X[labels == k], axis=0)

    return labels, centroids
import pandas as pd
dataset = pd.read_csv("kmeans.csv")
X = dataset.iloc[:,[3,4]].values
# Apply K-means algorithm
labels, centroids = k_means(X, K=5)

# Plot the clusters and centroids
colors = ['r', 'g', 'b','y','pink']
for i in range(5):
    plt.scatter(X[labels == i, 0], X[labels == i, 1], c=colors[i], label=f'
Cluster {i+1}')
plt.scatter(centroids[:, 0], centroids[:, 1], marker='x', s=200, linewidths
=3, color='k', label='Centroids')
plt.xlabel('Annual Income')
plt.ylabel('Spending Score')
plt.legend()
plt.show()
```


100
80
60
40
20
0
Spending Score
20
40
60
80
100
120
140
Annual Income
Cluster 1
Cluster 2
Cluster 3
Cluster 4
Cluster 5
Centroids

In [ ]:

In [ ]:
```
#KModes

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from kmodes.kmodes import KModes
from sklearn.preprocessing import LabelEncoder
from sklearn.decomposition import PCA
from sklearn.metrics import silhouette_score
df = pd.read_csv("WA_Fn-UseC_-Telco-Customer-Churn.csv")
df = df.drop(['customerID'], axis=1)
le = LabelEncoder()
for column in df.columns:
    if df[column].dtype == np.object:
        df[column] = le.fit_transform(df[column])
```

```
<ipython-input-4-3aef7c2a1293>:12: DeprecationWarning: `np.object` is a dep
recated alias for the builtin `object`. To silence this warning, use `objec
t` by itself. Doing this will not modify any behavior and is safe.
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/
devdocs/release/1.20.0-notes.html#deprecations
  if df[column].dtype == np.object:
<ipython-input-4-3aef7c2a1293>:12: DeprecationWarning: `np.object` is a dep
recated alias for the builtin `object`. To silence this warning, use `objec
t` by itself. Doing this will not modify any behavior and is safe.
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/
devdocs/release/1.20.0-notes.html#deprecations
  if df[column].dtype == np.object:
<ipython-input-4-3aef7c2a1293>:12: DeprecationWarning: `np.object` is a dep
recated alias for the builtin `object`. To silence this warning, use `objec
t` by itself. Doing this will not modify any behavior and is safe.
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/
devdocs/release/1.20.0-notes.html#deprecations
  if df[column].dtype == np.object:
<ipython-input-4-3aef7c2a1293>:12: DeprecationWarning: `np.object` is a dep
recated alias for the builtin `object`. To silence this warning, use `objec
t` by itself. Doing this will not modify any behavior and is safe.
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/
devdocs/release/1.20.0-notes.html#deprecations
  if df[column].dtype == np.object:
<ipython-input-4-3aef7c2a1293>:12: DeprecationWarning: `np.object` is a dep
recated alias for the builtin `object`. To silence this warning, use `objec
t` by itself. Doing this will not modify any behavior and is safe.
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/
devdocs/release/1.20.0-notes.html#deprecations
  if df[column].dtype == np.object:
<ipython-input-4-3aef7c2a1293>:12: DeprecationWarning: `np.object` is a dep
recated alias for the builtin `object`. To silence this warning, use `objec
t` by itself. Doing this will not modify any behavior and is safe.
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/
devdocs/release/1.20.0-notes.html#deprecations
  if df[column].dtype == np.object:
<ipython-input-4-3aef7c2a1293>:12: DeprecationWarning: `np.object` is a dep
recated alias for the builtin `object`. To silence this warning, use `objec
t` by itself. Doing this will not modify any behavior and is safe.
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/
devdocs/release/1.20.0-notes.html#deprecations
  if df[column].dtype == np.object:
<ipython-input-4-3aef7c2a1293>:12: DeprecationWarning: `np.object` is a dep
recated alias for the builtin `object`. To silence this warning, use `objec
t` by itself. Doing this will not modify any behavior and is safe.
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/
devdocs/release/1.20.0-notes.html#deprecations
  if df[column].dtype == np.object:
<ipython-input-4-3aef7c2a1293>:12: DeprecationWarning: `np.object` is a dep
recated alias for the builtin `object`. To silence this warning, use `objec
t` by itself. Doing this will not modify any behavior and is safe.
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/
devdocs/release/1.20.0-notes.html#deprecations
  if df[column].dtype == np.object:
<ipython-input-4-3aef7c2a1293>:12: DeprecationWarning: `np.object` is a dep
recated alias for the builtin `object`. To silence this warning, use `objec
```

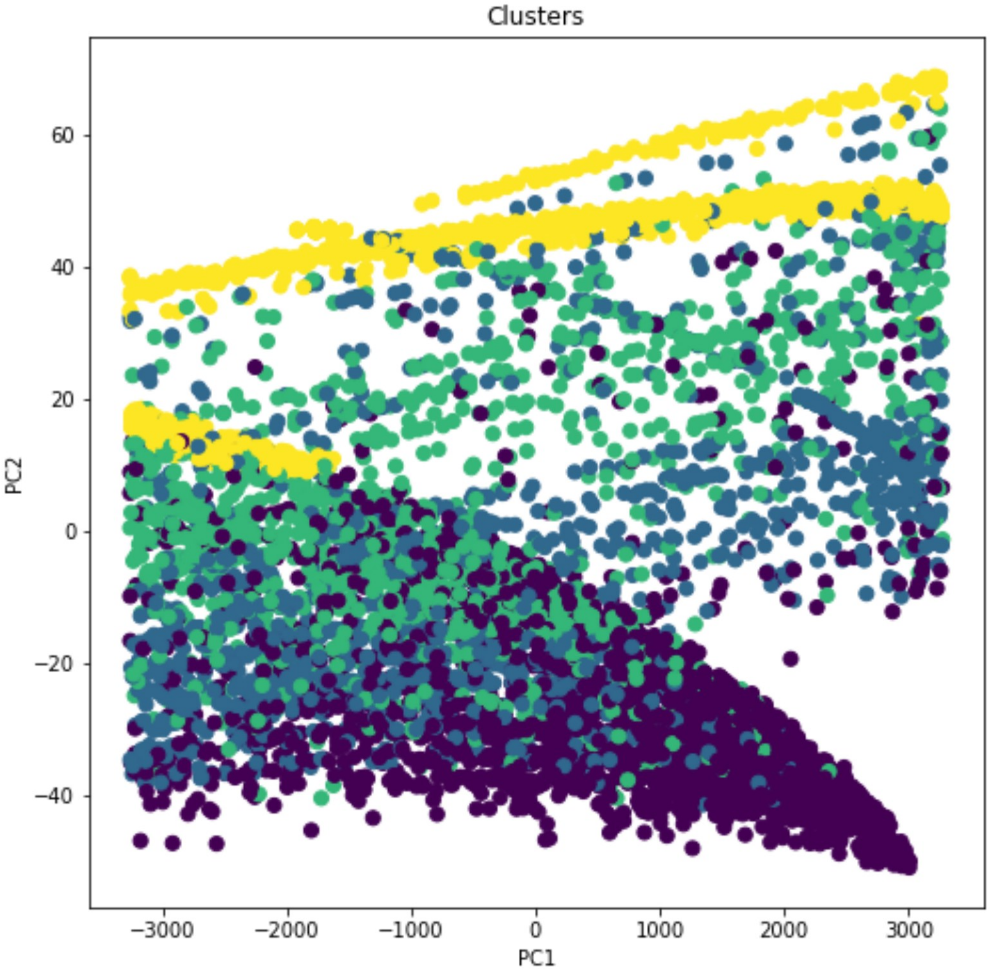
```
t` by itself. Doing this will not modify any behavior and is safe.
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/
devdocs/release/1.20.0-notes.html#deprecations
  if df[column].dtype == np.object:
<ipython-input-4-3aef7c2a1293>:12: DeprecationWarning: `np.object` is a dep
recated alias for the builtin `object`. To silence this warning, use `objec
t` by itself. Doing this will not modify any behavior and is safe.
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/
devdocs/release/1.20.0-notes.html#deprecations
  if df[column].dtype == np.object:
<ipython-input-4-3aef7c2a1293>:12: DeprecationWarning: `np.object` is a dep
recated alias for the builtin `object`. To silence this warning, use `objec
t` by itself. Doing this will not modify any behavior and is safe.
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/
devdocs/release/1.20.0-notes.html#deprecations
  if df[column].dtype == np.object:
<ipython-input-4-3aef7c2a1293>:12: DeprecationWarning: `np.object` is a dep
recated alias for the builtin `object`. To silence this warning, use `objec
t` by itself. Doing this will not modify any behavior and is safe.
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/
devdocs/release/1.20.0-notes.html#deprecations
  if df[column].dtype == np.object:
<ipython-input-4-3aef7c2a1293>:12: DeprecationWarning: `np.object` is a dep
recated alias for the builtin `object`. To silence this warning, use `objec
t` by itself. Doing this will not modify any behavior and is safe.
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/
devdocs/release/1.20.0-notes.html#deprecations
  if df[column].dtype == np.object:
<ipython-input-4-3aef7c2a1293>:12: DeprecationWarning: `np.object` is a dep
recated alias for the builtin `object`. To silence this warning, use `objec
t` by itself. Doing this will not modify any behavior and is safe.
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/
devdocs/release/1.20.0-notes.html#deprecations
  if df[column].dtype == np.object:
<ipython-input-4-3aef7c2a1293>:12: DeprecationWarning: `np.object` is a dep
recated alias for the builtin `object`. To silence this warning, use `objec
t` by itself. Doing this will not modify any behavior and is safe.
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/
devdocs/release/1.20.0-notes.html#deprecations
  if df[column].dtype == np.object:
<ipython-input-4-3aef7c2a1293>:12: DeprecationWarning: `np.object` is a dep
recated alias for the builtin `object`. To silence this warning, use `objec
t` by itself. Doing this will not modify any behavior and is safe.
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/
devdocs/release/1.20.0-notes.html#deprecations
  if df[column].dtype == np.object:
```

In [ ]:
```
pip install kmodes
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/cola
b-wheels/public/simple/
Collecting kmodes
  Downloading kmodes-0.12.2-py2.py3-none-any.whl (20 kB)
Requirement already satisfied: numpy>=1.10.4 in /usr/local/lib/python3.9/di
st-packages (from kmodes) (1.22.4)
Requirement already satisfied: scikit-learn>=0.22.0 in /usr/local/lib/pytho
n3.9/dist-packages (from kmodes) (1.2.2)
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.9/dis
t-packages (from kmodes) (1.1.1)
Requirement already satisfied: scipy>=0.13.3 in /usr/local/lib/python3.9/di
st-packages (from kmodes) (1.10.1)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/pytho
n3.9/dist-packages (from scikit-learn>=0.22.0->kmodes) (3.1.0)
Installing collected packages: kmodes
Successfully installed kmodes-0.12.2
```

In [ ]:
```
kmode = KModes(n_clusters=4, init='Huang', n_init=5, verbose=0)
clusters = kmode.fit_predict(df)
```

In [ ]:
```
pca = PCA(n_components=2)
principal_components = pca.fit_transform(df)
principal_df = pd.DataFrame(data=principal_components, columns=['PC1', 'PC2
'])
principal_df['cluster'] = clusters
plt.figure(figsize=(8, 8))
plt.scatter(principal_df['PC1'], principal_df['PC2'], c=principal_df['clust
er'], s=50)
plt.title('Clusters')
plt.xlabel('PC1')
plt.ylabel('PC2')
plt.show()
```



In [ ]:
```
from sklearn.metrics import silhouette_score

score = silhouette_score(df, clusters, metric='euclidean')
print('Silhouette score:', score)
```

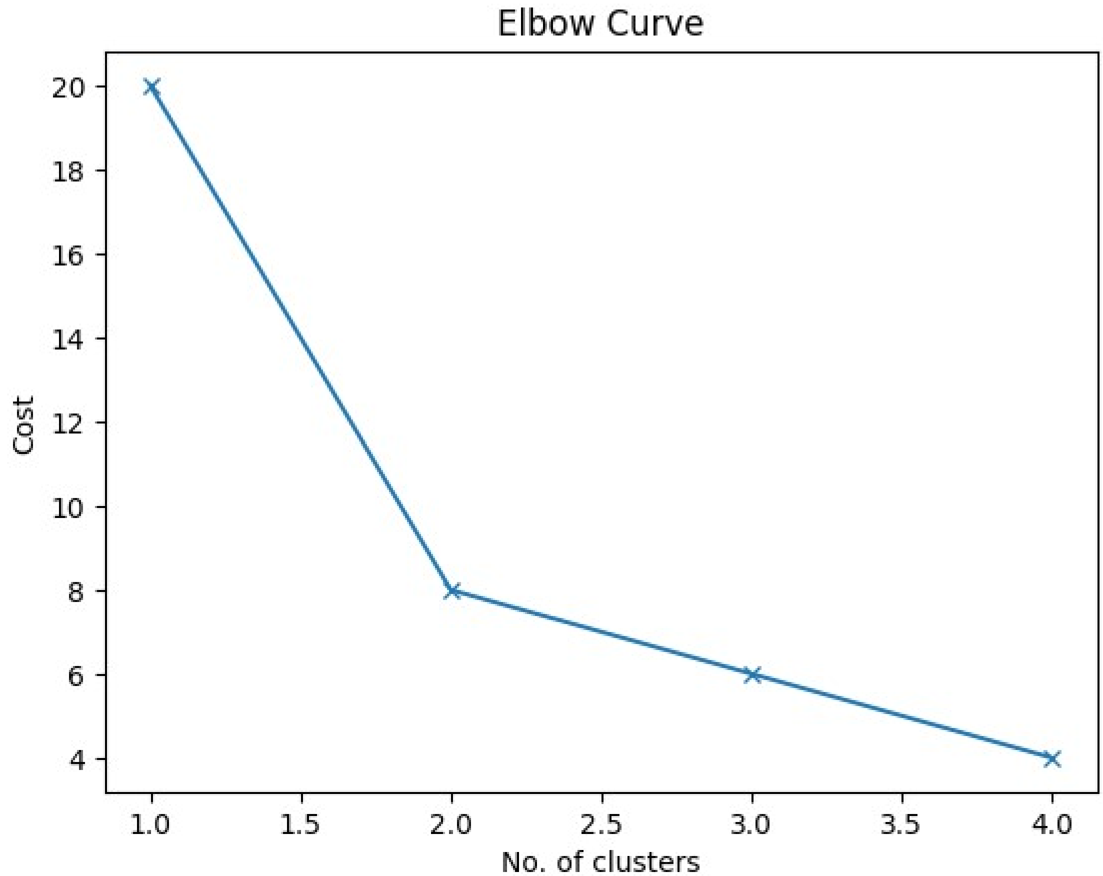
```
Silhouette score: -0.0739289219079896
```

In [ ]:

```
# importing necessary libraries
import pandas as pd
import numpy as np
# !pip install kmodes
from kmodes.kmodes import KModes
import matplotlib.pyplot as plt
%matplotlib inline

# Generate sample data
data = np.array([
    ['A', 'B', 'C', 'D'],
    ['A', 'B', 'E', 'F'],
    ['A', 'B', 'C', 'F'],
    ['A', 'B', 'E', 'D'],
    ['G', 'H', 'I', 'J'],
    ['G', 'H', 'K', 'L'],
    ['G', 'H', 'I', 'L'],
    ['G', 'H', 'K', 'J'],
])

# Elbow curve to find optimal K
cost = []
K = range(1,5)
for k in list(K):
        kmode = KModes(n_clusters=k, init = "random", n_init = 5, verbose=
1)
        kmode.fit_predict(data)
        cost.append(kmode.cost_)

plt.plot(K, cost, 'x-')
plt.xlabel('No. of clusters')
plt.ylabel('Cost')
plt.title('Elbow Curve')
plt.show()
```

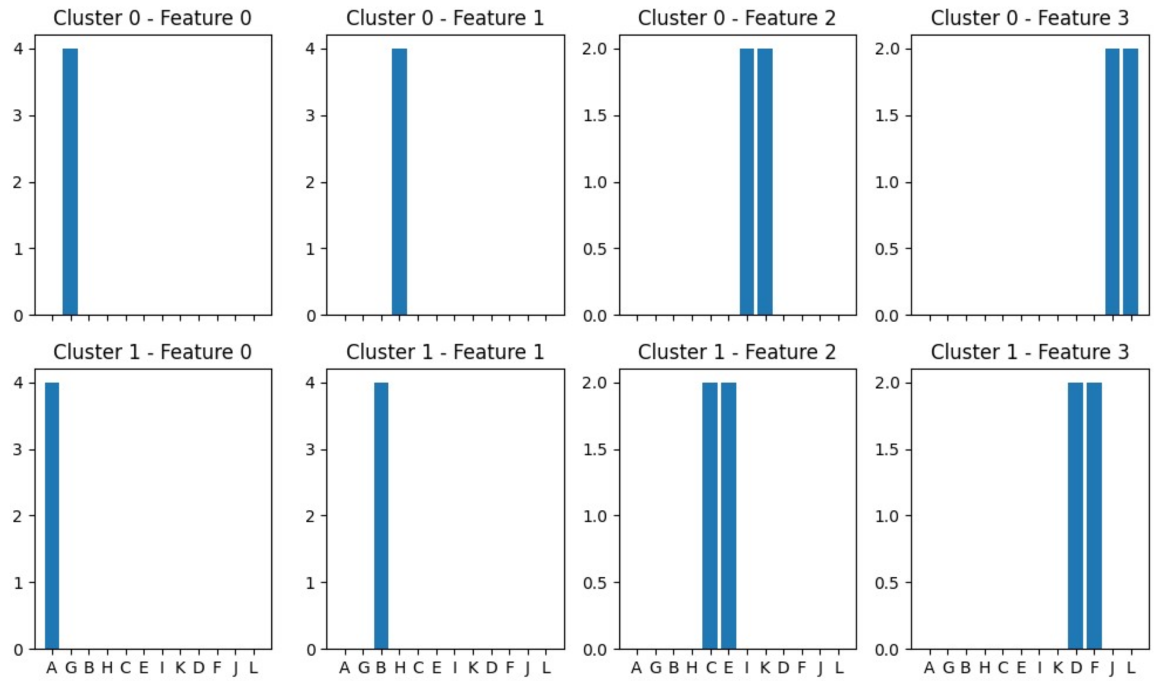
```
Init: initializing centroids
Init: initializing clusters
Starting iterations...
Run 1, iteration: 1/100, moves: 0, cost: 20.0
Init: initializing centroids
Init: initializing clusters
Starting iterations...
Run 2, iteration: 1/100, moves: 0, cost: 20.0
Init: initializing centroids
Init: initializing clusters
Starting iterations...
Run 3, iteration: 1/100, moves: 0, cost: 20.0
Init: initializing centroids
Init: initializing clusters
Starting iterations...
Run 4, iteration: 1/100, moves: 0, cost: 20.0
Init: initializing centroids
Init: initializing clusters
Starting iterations...
Run 5, iteration: 1/100, moves: 0, cost: 20.0
Best run was number 1
Init: initializing centroids
Init: initializing clusters
Starting iterations...
Run 1, iteration: 1/100, moves: 2, cost: 8.0
Run 1, iteration: 2/100, moves: 0, cost: 8.0
Init: initializing centroids
Init: initializing clusters
Starting iterations...
Run 2, iteration: 1/100, moves: 0, cost: 8.0
Init: initializing centroids
Init: initializing clusters
Starting iterations...
Run 3, iteration: 1/100, moves: 0, cost: 8.0
Init: initializing centroids
Init: initializing clusters
Starting iterations...
Run 4, iteration: 1/100, moves: 2, cost: 8.0
Init: initializing centroids
Init: initializing clusters
Starting iterations...
Run 5, iteration: 1/100, moves: 0, cost: 8.0
Best run was number 1
Init: initializing centroids
Init: initializing clusters
Starting iterations...
Run 1, iteration: 1/100, moves: 0, cost: 8.0
Init: initializing centroids
Init: initializing clusters
Starting iterations...
Run 2, iteration: 1/100, moves: 0, cost: 6.0
Init: initializing centroids
Init: initializing clusters
Starting iterations...
Run 3, iteration: 1/100, moves: 2, cost: 6.0
Run 3, iteration: 2/100, moves: 0, cost: 6.0
```

```
Init: initializing centroids
Init: initializing clusters
Starting iterations...
Run 4, iteration: 1/100, moves: 0, cost: 8.0
Init: initializing centroids
Init: initializing clusters
Starting iterations...
Run 5, iteration: 1/100, moves: 0, cost: 6.0
Best run was number 2
Init: initializing centroids
Init: initializing clusters
Starting iterations...
Run 1, iteration: 1/100, moves: 0, cost: 4.0
Init: initializing centroids
Init: initializing clusters
Starting iterations...
Run 2, iteration: 1/100, moves: 0, cost: 5.0
Init: initializing centroids
Init: initializing clusters
Starting iterations...
Run 3, iteration: 1/100, moves: 0, cost: 5.0
Init: initializing centroids
Init: initializing clusters
Starting iterations...
Run 4, iteration: 1/100, moves: 0, cost: 5.0
Init: initializing centroids
Init: initializing clusters
Starting iterations...
Run 5, iteration: 1/100, moves: 0, cost: 4.0
Best run was number 1
```


Elbow Curve
20
18
16
14
12
10
8
6
4
Cost
1.0
1.5
2.0
2.5
3.0
3.5
4.0
No. of clusters

In [ ]:
```
from kmodes.kmodes import KModes
import numpy as np
import matplotlib.pyplot as plt


# Initialize KModes object and fit the data
km = KModes(n_clusters=2, init='Huang', verbose=1)
clusters = km.fit_predict(data)

# Get the frequency of each category within each cluster
cluster_freq = []
for i in range(km.n_clusters):
    freq = {}
    for j in range(data.shape[1]):
        freq[j] = {}
        for k in np.unique(data[:, j]):
            freq[j][k] = np.sum(data[clusters == i, j] == k)
    cluster_freq.append(freq)

# Plot the frequency of each category within each cluster
fig, axs = plt.subplots(km.n_clusters, data.shape[1], figsize=(10, 6), shar
ex=True)
for i in range(km.n_clusters):
    for j in range(data.shape[1]):
        axs[i, j].bar(cluster_freq[i][j].keys(), cluster_freq[i][j].values
())
        axs[i, j].set_title('Cluster {} - Feature {}'.format(i, j))
plt.tight_layout()
plt.show()
```

```
Init: initializing centroids
Init: initializing clusters
Starting iterations...
Run 1, iteration: 1/100, moves: 2, cost: 8.0
Run 1, iteration: 2/100, moves: 0, cost: 8.0
Init: initializing centroids
Init: initializing clusters
Starting iterations...
Run 2, iteration: 1/100, moves: 3, cost: 8.0
Init: initializing centroids
Init: initializing clusters
Starting iterations...
Run 3, iteration: 1/100, moves: 3, cost: 8.0
Run 3, iteration: 2/100, moves: 0, cost: 8.0
Init: initializing centroids
Init: initializing clusters
Starting iterations...
Run 4, iteration: 1/100, moves: 2, cost: 8.0
Init: initializing centroids
Init: initializing clusters
Starting iterations...
Run 5, iteration: 1/100, moves: 2, cost: 8.0
Run 5, iteration: 2/100, moves: 1, cost: 8.0
Init: initializing centroids
Init: initializing clusters
Starting iterations...
Run 6, iteration: 1/100, moves: 2, cost: 8.0
Run 6, iteration: 2/100, moves: 0, cost: 8.0
Init: initializing centroids
Init: initializing clusters
Starting iterations...
Run 7, iteration: 1/100, moves: 2, cost: 8.0
Init: initializing centroids
Init: initializing clusters
Starting iterations...
Run 8, iteration: 1/100, moves: 2, cost: 8.0
Init: initializing centroids
Init: initializing clusters
Starting iterations...
Run 9, iteration: 1/100, moves: 0, cost: 8.0
Init: initializing centroids
Init: initializing clusters
Starting iterations...
Run 10, iteration: 1/100, moves: 0, cost: 8.0
Best run was number 1
```



In [ ]:
```
print(km.cluster_centroids_)
```

```
[['G' 'H' 'I' 'J']
 ['A' 'B' 'C' 'F']]
```

In [ ]:

In [ ]:

Without Library

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

def hamming_distance(x1, x2):
    return np.sum(x1 != x2)

class KModes:
    def __init__(self, n_clusters, max_iter):
        self.n_clusters = n_clusters
        self.max_iter = max_iter

    def fit(self, X):
        # Initialize centroids randomly
        self.centroids = []
        for i in range(self.n_clusters):
            centroid = np.random.choice(X.shape[0])
            self.centroids.append(X[centroid])

        for i in range(self.max_iter):
            # Assign each data point to the nearest centroid
            clusters = [[] for _ in range(self.n_clusters)]
            for j, x in enumerate(X):
                distances = [hamming_distance(x, c) for c in self.centroid
s]
                cluster_idx = np.argmin(distances)
                clusters[cluster_idx].append(j)

            # Update centroids
            for k in range(self.n_clusters):
                if clusters[k]:
                    cluster_data = X[clusters[k]]
                    mode = []
                    for feature in range(cluster_data.shape[1]):
                        feature_counts = np.bincount(cluster_data[:, featur
e])
                        mode.append(np.argmax(feature_counts))
                    self.centroids[k] = mode

        # Return the cluster labels, clusters for each data point, and clus
ter centers
        self.labels_ = np.zeros(X.shape[0])
        self.clusters_ = [[] for _ in range(self.n_clusters)]
        for i, cluster in enumerate(clusters):
            for j in cluster:
                self.labels_[j] = i
                self.clusters_[i].append(X[j])
        self.centroids_ = self.centroids
        return self.labels_, self.clusters_, self.centroids_
```

In [ ]:
```
# Example usage
data = np.array([[1, 2, 3, 4],
                 [1, 2, 3, 5],
                 [2, 3, 4, 5],
                 [2, 3, 5, 6],
                 [7, 8, 9, 10],
                 [7, 8, 9, 11]])
km = KModes(n_clusters=2, max_iter=100)
labels, clusters, centroids = km.fit(data)
print('Cluster centers:')
for centroid in centroids:
    print(centroid)
```

```
Cluster centers:
[2, 3, 9, 5]
[1, 2, 3, 4]
```

In [ ]:
```
for i, cluster in enumerate(clusters):
    print(f'Cluster {i}:')
    for row in cluster:
        print(row)
    print()
```

```
Cluster 0:
[2 3 4 5]
[2 3 5 6]
[ 7  8  9 10]
[ 7  8  9 11]

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

In [ ]:

In [ ]:
```
#KNN

import numpy as np
import pandas as pd
from sklearn.neighbors import KNeighborsClassifier
```

In [ ]:
```
dataset = pd.read_csv('knn.csv')
x=dataset.iloc[:,0:-1].values
y=dataset.iloc[:,-1].values
dataset
```

Out[ ]:

	Height	Weight	Class
0	167	51	Underweight
1	182	62	Normal
2	176	69	Normal
3	173	64	Normal
4	172	65	Normal
5	174	56	Underweight
6	169	58	Normal
7	173	57	Normal
8	170	55	Normal

In [ ]:
```
print("x",x)
print("y",y)
```

```
x [[167  51]
 [182  62]
 [176  69]
 [173  64]
 [172  65]
 [174  56]
 [169  58]
 [173  57]
 [170  55]]
y ['Underweight' 'Normal' 'Normal' 'Normal' 'Normal' 'Underweight' 'Normal'
 'Normal' 'Normal']
```

In [ ]:
```
knn = KNeighborsClassifier(n_neighbors = 4)
knn.fit(x, y)
predictions = knn.predict([[170,57]])
print("Prediction for - [Height=170, Weight=57] for k=3 is ",predictions)
print("Accuracy of Model",knn.score(x,y))
```

```
Prediction for - [Height=170, Weight=57] for k=3 is  ['Normal']
Accuracy of Model 0.7777777777777778
```

In [ ]:

In [ ]:
```
#KNN-without libraries

import numpy as np
import pandas as pd
import scipy.spatial
import math
```

In [ ]:
```
dataset = pd.read_csv('knn.csv')
x=dataset.iloc[:,0:-1].values
y=dataset.iloc[:,-1].values
```

In [ ]:
```
print("x",x)
print("y",y)
```

```
x [[167  51]
 [182  62]
 [176  69]
 [173  64]
 [172  65]
 [174  56]
 [169  58]
 [173  57]
 [170  55]]
y ['Underweight' 'Normal' 'Normal' 'Normal' 'Normal' 'Underweight' 'Normal'
 'Normal' 'Normal']
```

In [ ]:
```
def most_frequent(List):
    counter = 0
    num = List[0]

    for i in List:
        curr_frequency = List.count(i)
        if(curr_frequency > counter):
            counter = curr_frequency
            num = i

    return num
```

In [ ]:
```
def cal_distance(x,y,x_pred,y_pred):
    distance = math.sqrt((x-x_pred)**2+(y-y_pred)**2)

    return distance
```

In [ ]:
```
def knn(x,k):
    x_pred = 170
    y_pred = 57
    dist = []
    res = []
    for i in range(len(x)):
        dist.append(cal_distance(int(x[i][0]),int(x[i][1]),x_pred,y_pred))

    ranks = pd.Series(dist).rank().tolist()

    for i in range(1,k+1):
        res.append(y[ranks.index(i)])

    return most_frequent(res)
```

In [ ]:
```
print("The result for Height = 170 and Weight = 57 is ", knn(x,3))
```

```
The result for Height = 170 and Weight = 57 is  Normal
```

In [ ]:
```
#LinearRegression

import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression

# Importing the dataset
dataset = pd.read_csv('salary_data.csv')
dataset
```

Out[ ]:

	YearsOfExperience	Salary
0	1.2	38976
1	1.3	45897
2	1.5	36987
3	1.4	40587
4	1.3	42984
5	1.7	47986
6	2.0	44578
7	2.2	38789
8	2.4	46986
9	2.6	47986
10	2.9	56642
11	3.0	60150
12	3.2	54445
13	3.3	58763
14	3.5	56498
15	3.9	63218
16	3.2	63987
17	3.6	58736
18	3.9	62948
19	4.0	54874
20	4.0	57983
21	4.2	55876
22	4.0	57643
23	4.1	56983
24	4.5	62000
25	4.9	68943
26	5.1	67938

	YearsOfExperience	Salary
27	5.3	85698
28	5.9	81293
29	5.1	68349
30	5.3	82903
31	5.9	85938
32	6.1	94038
33	6.0	92839
34	6.8	93847
35	6.0	91029
36	6.8	92837
37	7.1	97364
38	7.9	99387
39	7.9	100293
40	7.1	98376
41	7.9	102893
42	8.1	114938
43	8.3	108374
44	8.3	109837
45	8.2	111049
46	8.7	109893
47	9.0	105984
48	9.2	119384
49	9.3	115039

In [ ]:
```
X = dataset.iloc[:, :-1].values #get a copy of dataset exclude last column
y = dataset.iloc[:, 1].values #get array of dataset in column 1st

# Splitting the dataset into the Training set and Test set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=1/3, ra
ndom_state=0)

# Fitting Simple Linear Regression to the Training set

regressor = LinearRegression()
regressor.fit(X_train, y_train)

# Predicting the Test set results
y_pred = regressor.predict(X_test)
y_pred
```

Out[ ]:
```
array([ 74112.71939557,  82662.88348193, 104513.30281375,  81712.86525012,
        54162.33652739,  39912.06305012,  90263.02933648, 108313.37574102,
        93113.08403193,  63662.51884557,  38012.02658648,  53212.31829557,
        76012.75585921, 100713.22988648,  82662.88348193, 102613.26635012,
       113063.46690012,  46562.19067285,  58912.42768648,  83612.90171375])
```

In [ ]:
```
viz_train = plt
viz_train.scatter(X_train, y_train, color='red')
viz_train.plot(X_train, regressor.predict(X_train), color='blue')
viz_train.xlabel('Year of Experience')
viz_train.ylabel('Salary')
viz_train.title('Salary VS Experience (Training set)')
viz_train.show()
```

In [ ]:
```
viz_test = plt
viz_test.scatter(X_test, y_test, color='red')
viz_test.plot(X_train, regressor.predict(X_train), color='blue')
viz_test.title('Salary VS Experience (Test set)')
viz_test.xlabel('Year of Experience')
viz_test.ylabel('Salary')
viz_test.show()
```

In [ ]:
```
print("Equation of the resulting regression line is: y = ", regressor.coef
_,"*x + ",regressor.intercept_)

pd.DataFrame({'x_test':list(X_test), 'y_test':list(y_test), 'y_pred':list(y
_pred)})
```

```
Equation of the resulting regression line is: y =  [9500.18231818] *x +  25
661.789572846363
```

Out[ ]:

	x_test	y_test	y_pred
0	[5.1]	67938	74112.719396
1	[6.0]	91029	82662.883482
2	[8.3]	108374	104513.302814
3	[5.9]	81293	81712.865250
4	[3.0]	60150	54162.336527
5	[1.5]	36987	39912.063050
6	[6.8]	93847	90263.029336
7	[8.7]	109893	108313.375741
8	[7.1]	98376	93113.084032
9	[4.0]	57643	63662.518846
10	[1.3]	42984	38012.026586
11	[2.9]	56642	53212.318296
12	[5.3]	82903	76012.755859
13	[7.9]	102893	100713.229886
14	[6.0]	92839	82662.883482
15	[8.1]	114938	102613.266350
16	[9.2]	119384	113063.466900
17	[2.2]	38789	46562.190673
18	[3.5]	56498	58912.427686
19	[6.1]	94038	83612.901714

In [ ]:

Without Libraries

In [ ]:
```
import pandas as pd
from math import pow


def get_headers(dataframe):
    return dataframe.columns.values


def cal_mean(readings):
    readings_total = sum(readings)
    number_of_readings = len(readings)
    mean = readings_total / float(number_of_readings)
    return mean


def cal_variance(readings):
    readings_mean = cal_mean(readings)
    mean_difference_squared_readings = [pow((reading - readings_mean), 2) f
or reading in readings]
    variance = sum(mean_difference_squared_readings)
    return variance / float(len(readings) - 1)


def cal_covariance(readings_1, readings_2):
    readings_1_mean = cal_mean(readings_1)
    readings_2_mean = cal_mean(readings_2)
    readings_size = len(readings_1)
    covariance = 0.0
    for i in range(0, readings_size):
        covariance += (readings_1[i] - readings_1_mean) * (readings_2[i] -
readings_2_mean)
    return covariance / float(readings_size - 1)


def cal_simple_linear_regression_coefficients(x_readings, y_readings):
    b1 = cal_covariance(x_readings, y_readings) / float(cal_variance(x_read
ings))
    b0 = cal_mean(y_readings) - (b1 * cal_mean(x_readings))
    return b0, b1


def predict_target_value(x, b0, b1):

    return b0 + b1 * x


def cal_rmse(actual_readings, predicted_readings):

    square_error_total = 0.0
    total_readings = len(actual_readings)
    for i in range(0, total_readings):
        error = predicted_readings[i] - actual_readings[i]
        square_error_total += pow(error, 2)
    rmse = square_error_total / float(total_readings)
    return rmse
```

```
def simple_linear_regression(dataset,alpha):

    dataset_headers = get_headers(dataset)
    print ("Dataset Headers :: ", dataset_headers)

    Y_mean = cal_mean(dataset[dataset_headers[0]])
    X_mean = cal_mean(dataset[dataset_headers[1]])
    Y_variance = cal_variance(dataset[dataset_headers[0]])
    X_variance = cal_variance(dataset[dataset_headers[1]])
    covariance_of_X_and_Y = dataset.cov()[dataset_headers[0]][dataset_heade
rs[1]]
    w1 = covariance_of_X_and_Y / float(Y_variance)
    w0 = X_mean - (w1 * Y_mean)
    res=float(w0)+(float(w1)*alpha)
    mse=(pow.sqrt((Y_test-y_pred)**2))/len(Y_test)
    print("Predicted Value for 86 is :",res)
    print("Mean Squared Error is : ",mse)

if __name__ == "__main__":

    input_path = "dataset.csv"
    data = pd.read_csv(input_path)
    alpha=86
    simple_linear_regression(data,alpha)
```

In [ ]:

```
#logisticregression
import pandas as pd
from matplotlib import pyplot as plt
```

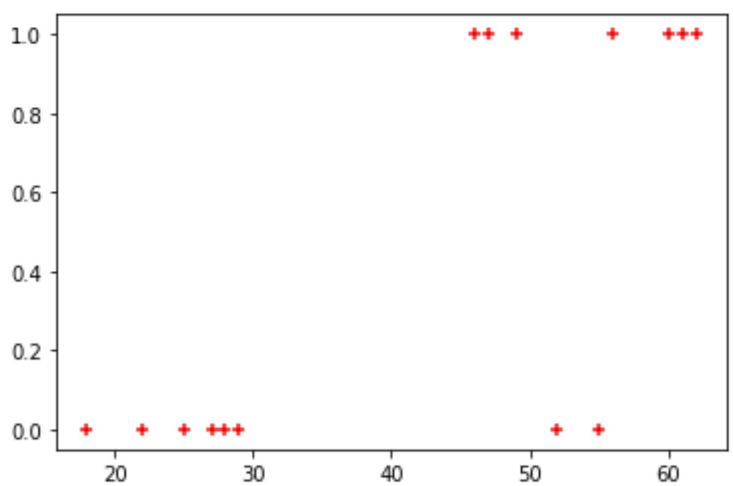
In [ ]:
```
dataset=pd.read_csv("insurance.csv")
dataset
```

Out[ ]:

	age	have_insurance
0	22	0
1	25	0
2	47	1
3	52	0
4	46	1
5	56	1
6	55	0
7	60	1
8	62	1
9	61	1
10	18	0
11	28	0
12	27	0
13	29	0
14	49	1

In [ ]:
```
plt.scatter(dataset.age,dataset.have_insurance,marker='+',color='red')
```

Out[ ]: <matplotlib.collections.PathCollection at 0x7fc472787640>



In [ ]:
```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(dataset[['age']],datase
t.have_insurance,train_size=0.8)
#training data
from sklearn.linear_model import LogisticRegression
model=LogisticRegression()
model.fit(x_train, y_train)
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=Tru
e, intercept_scaling=1, l1_ratio=None, max_iter=100, multi_class='auto', n_
jobs=None, penalty='l2', random_state=None, solver='lbfgs', tol=0.0001, ver
bose=0, warm_start=False)

model.coef_
print("coefficient of x is",model.coef_)


model.intercept_
print("intercept of line is",model.intercept_)
```

```
coefficient of x is [[0.13761379]]
intercept of line is [-6.17996735]
```

In [ ]:
```
import math
def sigmoid(x):
    return 1/(1+math.exp(-x))
def prediction_function(age):
    z= 0.280409*age -7.942535
    #z=mx+c
    z
    y=sigmoid(z)
    return y

#predicting if person with age 33 has insurance or not
age=33
y=prediction_function(age)
print("Probability of person with age 33 having insurance is",y)

print("As 0.787 is greater than 0.5 which means person with age 33 has insu
rance ")
```

```
Probability of person with age 33 having insurance is 0.78767408876316
As 0.787 is greater than 0.5 which means person with age 33 has insurance
```

In [ ]:

Without Lib

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

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

# Extract the feature and target variable
X = data["age"].values.reshape(-1, 1)
y = data["have_insurance"].values.reshape(-1, 1)

# Split the data into training and testing sets
np.random.seed(42)
indices = np.random.permutation(len(X))
split = int(0.8 * len(X))
train_indices, test_indices = indices[:split], indices[split:]
X_train, X_test = X[train_indices], X[test_indices]
y_train, y_test = y[train_indices], y[test_indices]

# Scale the features
mean = np.mean(X_train, axis=0)
std = np.std(X_train, axis=0)
X_train = (X_train - mean) / std
X_test = (X_test - mean) / std

# Define the logistic regression model
def sigmoid(z):
    return 1 / (1 + np.exp(-z))

def predict(X, w):
    z = np.dot(X, w)
    return sigmoid(z)

def loss(X, y, w):
    y_pred = predict(X, w)
    return -np.mean(y * np.log(y_pred) + (1 - y) * np.log(1 - y_pred))

def gradient(X, y, w):
    y_pred = predict(X, w)
    return np.dot(X.T, y_pred - y) / len(y)

def logistic_regression(X, y, num_iterations=1000, learning_rate=0.1):
    # Initialize weights to zero
    w = np.zeros((X.shape[1], 1))

    # Update weights using gradient descent
    for i in range(num_iterations):
        grad = gradient(X, y, w)
        w -= learning_rate * grad

        # Print loss every 100 iterations


    return w

# Train the model
```

```
w = logistic_regression(X_train, y_train)

# Make predictions on the test set
y_pred = predict(X_test, w)

# Convert probabilities to binary predictions
y_pred_binary = np.round(y_pred)

# Calculate RMSE
rmse = np.sqrt(np.mean((y_test - y_pred) ** 2))
print("RMSE:", rmse)

# Predict the result for a particular input
input_data = np.array([33]).reshape(1, -1)
input_data_scaled = (input_data - mean) / std

result = predict(input_data_scaled, w)'''
result =sigmoid(result)
if(result>.5)
 '''
print("Prediction for input:", sigmoid(result))
```

```
RMSE: 0.7948455326138908
Prediction for input: [[0.51363718]]
```

In [ ]:

In [ ]:

```
#multiplereg
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn import linear_model
import math
```

In [ ]:

```
dataset=pd.read_csv("house_price.csv")
dataset
```

Out[ ]:

	area	bedrooms	age	price
0	2600	3	20	550000
1	3000	4	15	565000
2	3200	4	18	610000
3	3600	3	30	595000
4	4000	5	8	760000
5	4100	6	8	810000

In [ ]:

```
X=dataset.iloc[:,:-1]
y=dataset.iloc[:,-1].values.reshape(dataset.shape[0],1)
```

In [ ]:
```
reg = linear_model.LinearRegression()
reg.fit(dataset[['area','bedrooms','age']],dataset.price)
reg.coef_
print("coefficients of x in line are:",reg.coef_)
reg.intercept_
print("intercept of line",reg.intercept_)

#After training, Predict for the new sample.

#Find price of home with 3000 sqr ft area, 3 bedrooms, 40 year old house
# price=m1*area+m2*bedrooms+m3*age+c
print("price of home with 3000 sqr ft area, 3 bedrooms, 40 year old house")
print(reg.predict([[3000, 3, 40]]))


#Find price of home with 2500 sqr ft area, 4 bedrooms, 5 year old house
print("price of home with 2500 sqr ft area, 4 bedrooms, 5 year old house")
print(reg.predict([[2500, 4, 5]]))
```

```
coefficients of x in line are: [  112.06244194 23388.88007794 -3231.7179086
3]
intercept of line 221323.00186540396
price of home with 3000 sqr ft area, 3 bedrooms, 40 year old house
[498408.25158031]
price of home with 2500 sqr ft area, 4 bedrooms, 5 year old house
[578876.03748933]

/usr/local/lib/python3.9/dist-packages/sklearn/base.py:439: UserWarning: X
does not have valid feature names, but LinearRegression was fitted with fea
ture names
  warnings.warn(
/usr/local/lib/python3.9/dist-packages/sklearn/base.py:439: UserWarning: X
does not have valid feature names, but LinearRegression was fitted with fea
ture names
  warnings.warn(
```

In [ ]:
```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.2)


X_test=np.array(X_test)
X_test=X_test[:,0:]
X_test
```

Out[ ]:
```
array([[3600,    3,   30],
       [3000,    4,   15]])
```

Without Libraries

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

In [ ]:
```
X = np.vstack((np.ones((X.shape[0], )), X.T)).T
X_test = np.vstack((np.ones((X_test.shape[0], )), X_test.T)).T
```

In [ ]:
```
def model(X, Y, learning_rate, iteration):
    m = Y.size
    theta = np.zeros((X.shape[1], 1))
    cost_list = []
    for i in range(iteration):
      y_pred = np.dot(X, theta)
      cost = (1/(2*m))*np.sum(np.square(y_pred - Y))
      d_theta = (1/m)*np.dot(X.T, y_pred - Y)
      theta = theta - learning_rate*d_theta
      cost_list.append(cost)
      # to print the cost for 10 times
      if(i%(iteration/10) == 0):
        continue
    return theta, cost_list
X.shape
```

Out[ ]: (6, 3)

In [ ]:
```
iteration = 10000
learning_rate = 0.000000005
theta, cost_list = model(X, y, learning_rate = learning_rate, iteration =
iteration)
```

In [ ]:
```
theta.shape
```

Out[ ]: (3, 1)

In [ ]:
```
y_pred = np.dot(X_test, theta)
rmse=np.sqrt(np.mean((y_pred - y_test) ** 2))
error = (1/X_test.shape[0])*np.sum(np.abs(y_pred - y_test))
```

In [ ]:
```
error
```

Out[ ]: 44349.57958707068

In [ ]:
```
rmse
```

Out[ ]: 60884.25284605226

In [ ]:
```
def predict(X, theta):
    # Add constant column to features
    #X = np.c_[np.ones(X.shape[0]), X]
    # Make predictions using the trained model
    #print(X.shape)
    X=np.array(X)
    y_pred = X.dot(theta)
    return y_pred

# Load new data for prediction
new_data = [[3000, 3, 40]]

# Make predictions using trained model
y_pred_new = predict(new_data, theta)
print("Predicted prices: {:.20f}".format(float( y_pred_new)))
```

```
Predicted prices: 567430.84400090889539569616
```

In [ ]:
```
#naive
import numpy as nm
import matplotlib.pyplot as mtp
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
```

In [ ]:
```
dataset = pd.read_csv('naive.csv')
print("Sample Dataset - \n",dataset,"\n")
```

```
Sample Dataset -
      Day   Outlook Temperature Humidity    Wind PlayTennis
0     D1     Sunny         Hot     High    Weak         No
1     D2     Sunny         Hot     High  Strong         No
2     D3  Overcast         Hot     High    Weak        Yes
3     D4      Rain        Mild     High    Weak        Yes
4     D5      Rain        Cool   Normal    Weak        Yes
5     D6      Rain        Cool   Normal  Strong         No
6     D7  Overcast        Cool   Normal  Strong        Yes
7     D8     Sunny        Mild     High    Weak         No
8     D9     Sunny        Cool   Normal    Weak        Yes
9    D10      Rain        Mild   Normal    Weak        Yes
10   D11     Sunny        Mild   Normal  Strong        Yes
11   D12  Overcast        Mild     High  Strong        Yes
12   D13  Overcast         Hot   Normal    Weak        Yes
13   D14      Rain        Mild     High  Strong         No
```

In [ ]:
```
le_outlook = LabelEncoder()
dataset['outlook_n'] = le_outlook.fit_transform(dataset['Outlook'])
le_temperature = LabelEncoder()
dataset['temperature_n'] = le_temperature.fit_transform(dataset['Temperatur
e'])
le_humidity = LabelEncoder()
dataset['humidity_n'] = le_humidity.fit_transform(dataset['Humidity'])
le_wind = LabelEncoder()
dataset['wind_n'] = le_wind.fit_transform(dataset['Wind'])
print("Given Data after Encoding - \n",dataset,"\n")
```

```
Given Data after Encoding -
     Day   Outlook Temperature Humidity    Wind PlayTennis  outlook_n  \
0    D1     Sunny         Hot     High    Weak         No          2
1    D2     Sunny         Hot     High  Strong         No          2
2    D3  Overcast         Hot     High    Weak        Yes          0
3    D4      Rain        Mild     High    Weak        Yes          1
4    D5      Rain        Cool   Normal    Weak        Yes          1
5    D6      Rain        Cool   Normal  Strong         No          1
6    D7  Overcast        Cool   Normal  Strong        Yes          0
7    D8     Sunny        Mild     High    Weak         No          2
8    D9     Sunny        Cool   Normal    Weak        Yes          2
9   D10      Rain        Mild   Normal    Weak        Yes          1
10  D11     Sunny        Mild   Normal  Strong        Yes          2
11  D12  Overcast        Mild     High  Strong        Yes          0
12  D13  Overcast         Hot   Normal    Weak        Yes          0
13  D14      Rain        Mild     High  Strong         No          1

    temperature_n  humidity_n  wind_n
0               1           0       1
1               1           0       0
2               1           0       1
3               2           0       1
4               0           1       1
5               0           1       0
6               0           1       0
7               2           0       1
8               0           1       1
9               2           1       1
10              2           1       0
11              2           0       0
12              1           1       1
13              2           0       0
```

In [ ]:
```
x = dataset[['outlook_n','temperature_n','humidity_n','wind_n']]
print("X - Values\n",x,"\n")
y = dataset['PlayTennis']
print("Y - Values\n",y,"\n")
```

```
X - Values
     outlook_n  temperature_n  humidity_n  wind_n
0           2              1           0       1
1           2              1           0       0
2           0              1           0       1
3           1              2           0       1
4           1              0           1       1
5           1              0           1       0
6           0              0           1       0
7           2              2           0       1
8           2              0           1       1
9           1              2           1       1
10          2              2           1       0
11          0              2           0       0
12          0              1           1       1
13          1              2           0       0

Y - Values
 0      No
1      No
2     Yes
3     Yes
4     Yes
5      No
6     Yes
7      No
8     Yes
9     Yes
10    Yes
11    Yes
12    Yes
13     No
Name: PlayTennis, dtype: object
```

In [ ]:
```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.15,
random_state = 0)
from sklearn.naive_bayes import GaussianNB
gnb = GaussianNB()
gnb.fit(x, y)
y_pred = gnb.predict(x_test)
print("Testing values for play tennis\n",y_test)
print("Predicted values for play tennis",y_pred)
```

```
Testing values for play tennis
 8    Yes
6    Yes
4    Yes
Name: PlayTennis, dtype: object
Predicted values for play tennis ['Yes' 'Yes' 'Yes']
```

In [ ]:
```
from sklearn.metrics import accuracy_score
```

In [ ]:
```
accuracy_score(y_test,y_pred)
```

Out[ ]:
```
1.0
```

In [ ]:
```

```

In [ ]:
```
import pandas as pd
import numpy as np
import math
import random
import warnings
warnings.filterwarnings("ignore")
def load_csv(filename):
    return pd.read_csv(filename)

def str_column_to_int(dataset):
    for column in dataset.columns:
        if dataset[column].dtype == np.object:
            dataset[column] = dataset[column].astype('category').cat.codes
    return dataset

def split_dataset(dataset, split_ratio):
    train_size = int(len(dataset) * split_ratio)
    train_set = dataset.sample(n=train_size)
    test_set = dataset.drop(train_set.index)
    return [train_set, test_set]

def separate_by_class(dataset):
    separated = {}
    for i in range(len(dataset)):
        vector = dataset.iloc[i]
        if (vector.iloc[-1] not in separated):
            separated[vector.iloc[-1]] = []
        separated[vector.iloc[-1]].append(vector)
    return separated

def mean(numbers):
    return sum(numbers)/float(len(numbers))

def stdev(numbers):
    avg = mean(numbers)
    variance = sum([pow(x-avg,2) for x in numbers])/float(len(numbers)-1)
    return math.sqrt(variance)

def summarize(dataset):
    summaries = [(mean(attribute), stdev(attribute)) for attribute in zip(*
dataset)]
    del summaries[-1]
    return summaries

def summarize_by_class(dataset):
    separated = separate_by_class(dataset)
    summaries = {}
    for class_value, instances in separated.items():
        summaries[class_value] = summarize(instances)
    return summaries

def calculate_probability(x, mean, stdev):
    exponent = math.exp(-(math.pow(x-mean,2)/(2*math.pow(stdev,2))))
    return (1 / (math.sqrt(2*math.pi) * stdev)) * exponent
```

```
def calculate_class_probabilities(summaries, input_vector):
    probabilities = {}
    for class_value, class_summaries in summaries.items():
        probabilities[class_value] = 1
        for i in range(len(class_summaries)):
            mean, stdev = class_summaries[i]
            x = input_vector[i]
            probabilities[class_value] *= calculate_probability(x, mean, st
dev)
    return probabilities

def predict(summaries, input_vector):
    probabilities = calculate_class_probabilities(summaries, input_vector)
    best_label, best_prob = None, -1
    for class_value, probability in probabilities.items():
        if best_label is None or probability > best_prob:
            best_prob = probability
            best_label = class_value
    return best_label

def get_predictions(summaries, test_set):
    predictions = []
    for i in range(len(test_set)):
        result = predict(summaries, test_set.iloc[i])
        predictions.append(result)
    return predictions

def get_accuracy(test_set, predictions):
    correct = 0
    for i in range(len(test_set)):
        if test_set.iloc[i,-1] == predictions[i]:
            correct += 1
    return (correct / float(len(test_set))) * 100.0

def main():
    filename = 'naive.csv'
    split_ratio = 0.8
    dataset = load_csv(filename)
    dataset = str_column_to_int(dataset)
    training_set, test_set = split_dataset(dataset, split_ratio)
    print('Split {0} rows into train={1} and test={2} rows'.format(len(data
set), len(training_set), len(test_set)))
    # prepare model
    summaries = summarize_by_class(training_set)
    # test model
    predictions = get_predictions(summaries, test_set)
    accuracy = get_accuracy(test_set, predictions)
    print('Accuracy: {0}%\n\n\n\n'.format(accuracy))

main()
```

```
Split 14 rows into train=11 and test=3 rows
Accuracy: 100.0%
```

In [ ]:

In [ ]:

```
#percept
def predict(r, w):
        activation = w[0]
        for i in range(len(r)-1):
                activation += w[i + 1] * r[i]
        return 1.0 if activation >= 0.0 else 0.0

def trainweights(train, l_rate, n_epoch):
        weights = [0.0 for i in range(len(train[0]))]
        for epoch in range(n_epoch):
                sum_error = 0.0
                for row in train:
                        prediction = predict(row, weights)
                        error = row[-1] - prediction
                        sum_error += error**2
                        weights[0] = weights[0] + l_rate * error
                        for i in range(len(row)-1):
                                weights[i + 1] = weights[i + 1] + l_rate *
error * row[i]
                print('>epoch=%d, lrate=%.3f, error=%.3f' % (epoch, l_rate,
sum_error))
        return weights

# Calculate weights
dataset = [[0,0,0],
        [0,1,1],
        [1,0,1],
        [1,1,1]]
l_rate = 0.5
n_epoch = 5
weights = trainweights(dataset, l_rate, n_epoch)
print('New values of w1=', weights[0],' w2=',weights[1])
```

In [ ]:
```
import numpy as np
import tensorflow as tf
from tensorflow import keras
import matplotlib.pyplot as plt
import pandas as pd

data = pd.read_csv("/home/matlab/data.csv")

print(data)

def activation_func(value):
    return ((np.exp(value)-np.exp(-value))/(np.exp(value)+np.exp(-value)))

def perceptron_train(in_data,labels,alpha):
    X=np.array(in_data)
    y=np.array(labels)
    weights=np.random.random(X.shape[1])
    original=weights
    bias=np.random.random_sample()
    for key in range(X.shape[0]):
        a=activation_func(np.matmul(np.transpose(weights),X[key]))
        yn=0
        if a>=0.7:
            yn=1
        elif a<=(-0.7):
            yn=-1
        weights=weights+alpha*(yn-y[key])*X[key]
        print('Iteration '+str(key)+': '+str(weights))
    print('Difference: '+str(weights-original))
    return weights


def perceptron_test(in_data,label_shape,weights):
    X=np.array(in_data)
    y=np.zeros(label_shape)
    for key in range(X.shape[1]):
        a=activation_func((weights*X[key]).sum())
        y[key]=0
        if a>=0.7:
            y[key]=1
        elif a<=(-0.7):
            y[key]=-1
    return y


def score(result,labels):
    difference=result-np.array(labels)
    correct_ctr=0
    for elem in range(difference.shape[0]):
        if difference[elem]==0:
            correct_ctr+=1
    score=correct_ctr*100/difference.size
    print('Score='+str(score))
```

```
divider = np.random.rand(len(data)) < 0.70
d_train=data[divider]
d_test=data[~divider]



d_train_y=d_train['Y']
d_train_X=d_train.drop(['Y'],axis=1)

d_test_y=d_test['Y']
d_test_X=d_test.drop(['Y'],axis=1)

alpha = 0.05

weights = perceptron_train(d_train_X, d_train_y, alpha)

result_test=perceptron_test(d_test_X,d_test_y.shape,weights)

print("w1=",weights[0],"w2=",weights[1])
score(result_test,d_test_y)
```

In [ ]:

```
#singleperceptnolib
def predict(r, w):
        activation = w[0]
        for i in range(len(r)-1):
                activation += w[i + 1] * r[i]
        return 1.0 if activation >= 0.0 else 0.0

def trainweights(train, l_rate, n_epoch):
        weights = [0.0 for i in range(len(train[0]))]
        for epoch in range(n_epoch):
                sum_error = 0.0
                for row in train:
                        prediction = predict(row, weights)
                        error = row[-1] - prediction
                        sum_error += error**2
                        weights[0] = weights[0] + l_rate * error
                        for i in range(len(row)-1):
                                weights[i + 1] = weights[i + 1] + l_rate * 
error * row[i]
                print('>epoch=%d, lrate=%.3f, error=%.3f' % (epoch, l_rate, 
sum_error))
        return weights

# Calculate weights
```

In [ ]:
```
dataset = [[0,0,0],
        [0,1,1],
        [1,0,1],
        [1,1,1]]
l_rate = 0.5
n_epoch = 5
weights = trainweights(dataset, l_rate, n_epoch)
print('New values of w1=', weights[0],' w2=',weights[1])
```

```
>epoch=0, lrate=0.500, error=2.000
>epoch=1, lrate=0.500, error=2.000
>epoch=2, lrate=0.500, error=1.000
>epoch=3, lrate=0.500, error=0.000
>epoch=4, lrate=0.500, error=0.000
New values of w1= -0.5  w2= 0.5
```

In [ ]:

In [ ]:
```
#singleperceptron
from sklearn.datasets import load_iris
from sklearn.linear_model import Perceptron
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
import numpy as np
iris = load_iris()
X = iris.data
y = iris.target
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2, rand
om_state=42)
model = Perceptron(random_state=42)
model.fit(X_train, y_train)
y_pred = model.predict(X_val)
accuracy = accuracy_score(y_val, y_pred)
print("Accuracy:", accuracy)
```

```
Accuracy: 0.6333333333333333
```

In [ ]:
```
iris
```

```
Out[ ]: {'data': array([[5.1, 3.5, 1.4, 0.2],
               [4.9, 3. , 1.4, 0.2],
               [4.7, 3.2, 1.3, 0.2],
               [4.6, 3.1, 1.5, 0.2],
               [5. , 3.6, 1.4, 0.2],
               [5.4, 3.9, 1.7, 0.4],
               [4.6, 3.4, 1.4, 0.3],
               [5. , 3.4, 1.5, 0.2],
               [4.4, 2.9, 1.4, 0.2],
               [4.9, 3.1, 1.5, 0.1],
               [5.4, 3.7, 1.5, 0.2],
               [4.8, 3.4, 1.6, 0.2],
               [4.8, 3. , 1.4, 0.1],
               [4.3, 3. , 1.1, 0.1],
               [5.8, 4. , 1.2, 0.2],
               [5.7, 4.4, 1.5, 0.4],
               [5.4, 3.9, 1.3, 0.4],
               [5.1, 3.5, 1.4, 0.3],
               [5.7, 3.8, 1.7, 0.3],
               [5.1, 3.8, 1.5, 0.3],
               [5.4, 3.4, 1.7, 0.2],
               [5.1, 3.7, 1.5, 0.4],
               [4.6, 3.6, 1. , 0.2],
               [5.1, 3.3, 1.7, 0.5],
               [4.8, 3.4, 1.9, 0.2],
               [5. , 3. , 1.6, 0.2],
               [5. , 3.4, 1.6, 0.4],
               [5.2, 3.5, 1.5, 0.2],
               [5.2, 3.4, 1.4, 0.2],
               [4.7, 3.2, 1.6, 0.2],
               [4.8, 3.1, 1.6, 0.2],
               [5.4, 3.4, 1.5, 0.4],
               [5.2, 4.1, 1.5, 0.1],
               [5.5, 4.2, 1.4, 0.2],
               [4.9, 3.1, 1.5, 0.2],
               [5. , 3.2, 1.2, 0.2],
               [5.5, 3.5, 1.3, 0.2],
               [4.9, 3.6, 1.4, 0.1],
               [4.4, 3. , 1.3, 0.2],
               [5.1, 3.4, 1.5, 0.2],
               [5. , 3.5, 1.3, 0.3],
               [4.5, 2.3, 1.3, 0.3],
               [4.4, 3.2, 1.3, 0.2],
               [5. , 3.5, 1.6, 0.6],
               [5.1, 3.8, 1.9, 0.4],
               [4.8, 3. , 1.4, 0.3],
               [5.1, 3.8, 1.6, 0.2],
               [4.6, 3.2, 1.4, 0.2],
               [5.3, 3.7, 1.5, 0.2],
               [5. , 3.3, 1.4, 0.2],
               [7. , 3.2, 4.7, 1.4],
               [6.4, 3.2, 4.5, 1.5],
               [6.9, 3.1, 4.9, 1.5],
               [5.5, 2.3, 4. , 1.3],
               [6.5, 2.8, 4.6, 1.5],
               [5.7, 2.8, 4.5, 1.3],
```

```
[6.3, 3.3, 4.7, 1.6],
[4.9, 2.4, 3.3, 1. ],
[6.6, 2.9, 4.6, 1.3],
[5.2, 2.7, 3.9, 1.4],
[5. , 2. , 3.5, 1. ],
[5.9, 3. , 4.2, 1.5],
[6. , 2.2, 4. , 1. ],
[6.1, 2.9, 4.7, 1.4],
[5.6, 2.9, 3.6, 1.3],
[6.7, 3.1, 4.4, 1.4],
[5.6, 3. , 4.5, 1.5],
[5.8, 2.7, 4.1, 1. ],
[6.2, 2.2, 4.5, 1.5],
[5.6, 2.5, 3.9, 1.1],
[5.9, 3.2, 4.8, 1.8],
[6.1, 2.8, 4. , 1.3],
[6.3, 2.5, 4.9, 1.5],
[6.1, 2.8, 4.7, 1.2],
[6.4, 2.9, 4.3, 1.3],
[6.6, 3. , 4.4, 1.4],
[6.8, 2.8, 4.8, 1.4],
[6.7, 3. , 5. , 1.7],
[6. , 2.9, 4.5, 1.5],
[5.7, 2.6, 3.5, 1. ],
[5.5, 2.4, 3.8, 1.1],
[5.5, 2.4, 3.7, 1. ],
[5.8, 2.7, 3.9, 1.2],
[6. , 2.7, 5.1, 1.6],
[5.4, 3. , 4.5, 1.5],
[6. , 3.4, 4.5, 1.6],
[6.7, 3.1, 4.7, 1.5],
[6.3, 2.3, 4.4, 1.3],
[5.6, 3. , 4.1, 1.3],
[5.5, 2.5, 4. , 1.3],
[5.5, 2.6, 4.4, 1.2],
[6.1, 3. , 4.6, 1.4],
[5.8, 2.6, 4. , 1.2],
[5. , 2.3, 3.3, 1. ],
[5.6, 2.7, 4.2, 1.3],
[5.7, 3. , 4.2, 1.2],
[5.7, 2.9, 4.2, 1.3],
[6.2, 2.9, 4.3, 1.3],
[5.1, 2.5, 3. , 1.1],
[5.7, 2.8, 4.1, 1.3],
[6.3, 3.3, 6. , 2.5],
[5.8, 2.7, 5.1, 1.9],
[7.1, 3. , 5.9, 2.1],
[6.3, 2.9, 5.6, 1.8],
[6.5, 3. , 5.8, 2.2],
[7.6, 3. , 6.6, 2.1],
[4.9, 2.5, 4.5, 1.7],
[7.3, 2.9, 6.3, 1.8],
[6.7, 2.5, 5.8, 1.8],
[7.2, 3.6, 6.1, 2.5],
[6.5, 3.2, 5.1, 2. ],
[6.4, 2.7, 5.3, 1.9],
```

```
       [6.8, 3. , 5.5, 2.1],
       [5.7, 2.5, 5. , 2. ],
       [5.8, 2.8, 5.1, 2.4],
       [6.4, 3.2, 5.3, 2.3],
       [6.5, 3. , 5.5, 1.8],
       [7.7, 3.8, 6.7, 2.2],
       [7.7, 2.6, 6.9, 2.3],
       [6. , 2.2, 5. , 1.5],
       [6.9, 3.2, 5.7, 2.3],
       [5.6, 2.8, 4.9, 2. ],
       [7.7, 2.8, 6.7, 2. ],
       [6.3, 2.7, 4.9, 1.8],
       [6.7, 3.3, 5.7, 2.1],
       [7.2, 3.2, 6. , 1.8],
       [6.2, 2.8, 4.8, 1.8],
       [6.1, 3. , 4.9, 1.8],
       [6.4, 2.8, 5.6, 2.1],
       [7.2, 3. , 5.8, 1.6],
       [7.4, 2.8, 6.1, 1.9],
       [7.9, 3.8, 6.4, 2. ],
       [6.4, 2.8, 5.6, 2.2],
       [6.3, 2.8, 5.1, 1.5],
       [6.1, 2.6, 5.6, 1.4],
       [7.7, 3. , 6.1, 2.3],
       [6.3, 3.4, 5.6, 2.4],
       [6.4, 3.1, 5.5, 1.8],
       [6. , 3. , 4.8, 1.8],
       [6.9, 3.1, 5.4, 2.1],
       [6.7, 3.1, 5.6, 2.4],
       [6.9, 3.1, 5.1, 2.3],
       [5.8, 2.7, 5.1, 1.9],
       [6.8, 3.2, 5.9, 2.3],
       [6.7, 3.3, 5.7, 2.5],
       [6.7, 3. , 5.2, 2.3],
       [6.3, 2.5, 5. , 1.9],
       [6.5, 3. , 5.2, 2. ],
       [6.2, 3.4, 5.4, 2.3],
       [5.9, 3. , 5.1, 1.8]]),
 'target': array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2]),
 'frame': None,
 'target_names': array(['setosa', 'versicolor', 'virginica'], dtype='<U10
'),
 'DESCR': '.. _iris_dataset:\n\nIris plants dataset\n--------------------
\n\n**Data Set Characteristics:**\n\n    :Number of Instances: 150 (50 in e
ach of three classes)\n    :Number of Attributes: 4 numeric, predictive att
ributes and the class\n    :Attribute Information:\n        - sepal length
in cm\n        - sepal width in cm\n        - petal length in cm\n        -
petal width in cm\n        - class:\n                - Iris-Setosa\n
- Iris-Versicolour\n                - Iris-Virginica\n                \n
```

```
:Summary Statistics:\n\n    ============== ==== ==== ======= ===== ========
============\n                    Min  Max   Mean    SD   Class Correlatio
n\n    ============== ==== ==== ======= ===== ====================\n    sep
al length:   4.3  7.9   5.84   0.83    0.7826\n    sepal width:    2.0  4.4
3.05   0.43   -0.4194\n    petal length:   1.0  6.9   3.76   1.76    0.9490
(high!)\n    petal width:    0.1  2.5   1.20   0.76    0.9565  (high!)\n
============== ==== ==== ======= ===== ====================\n\n    :Missing
Attribute Values: None\n    :Class Distribution: 33.3% for each of 3 classe
s.\n    :Creator: R.A. Fisher\n    :Donor: Michael Marshall (MARSHALL%PLU@i
o.arc.nasa.gov)\n    :Date: July, 1988\n\nThe famous Iris database, first u
sed by Sir R.A. Fisher. The dataset is taken\nfrom Fisher\'s paper. Note th
at it\'s the same as in R, but not as in the UCI\nMachine Learning Reposito
ry, which has two wrong data points.\n\nThis is perhaps the best known data
base to be found in the\npattern recognition literature.  Fisher\'s paper i
s a classic in the field and\nis referenced frequently to this day.  (See D
uda & Hart, for example.)  The\ndata set contains 3 classes of 50 instances
each, where each class refers to a\ntype of iris plant.  One class is linea
rly separable from the other 2; the\nlatter are NOT linearly separable from
each other.\n\n.. topic:: References\n\n   - Fisher, R.A. "The use of multi
ple measurements in taxonomic problems"\n     Annual Eugenics, 7, Part II,
179-188 (1936); also in "Contributions to\n     Mathematical Statistics" (J
ohn Wiley, NY, 1950).\n   - Duda, R.O., & Hart, P.E. (1973) Pattern Classif
ication and Scene Analysis.\n     (Q327.D83) John Wiley & Sons.  ISBN 0-471
-22361-1.  See page 218.\n   - Dasarathy, B.V. (1980) "Nosing Around the Ne
ighborhood: A New System\n     Structure and Classification Rule for Recogn
ition in Partially Exposed\n     Environments".  IEEE Transactions on Patte
rn Analysis and Machine\n     Intelligence, Vol. PAMI-2, No. 1, 67-71.\n
- Gates, G.W. (1972) "The Reduced Nearest Neighbor Rule".  IEEE Transaction
s\n     on Information Theory, May 1972, 431-433.\n   - See also: 1988 MLC
Proceedings, 54-64.  Cheeseman et al"s AUTOCLASS II\n     conceptual cluste
ring system finds 3 classes in the data.\n   - Many, many more ...',
 'feature_names': ['sepal length (cm)',
  'sepal width (cm)',
  'petal length (cm)',
  'petal width (cm)'],
 'filename': 'iris.csv',
```

In [ ]:

```
```

In [ ]:

```
#svm
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
dataset=pd.read_csv('SVM.csv')
x=dataset.iloc[:, 2:-1].values
y=dataset.iloc[:, -1].values
```

In [ ]:
```
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.27,random_st
ate=0)
sc=StandardScaler()
x_train=sc.fit_transform(x_train)
x_test=sc.transform(x_test)
X=sc.fit_transform(x)
```

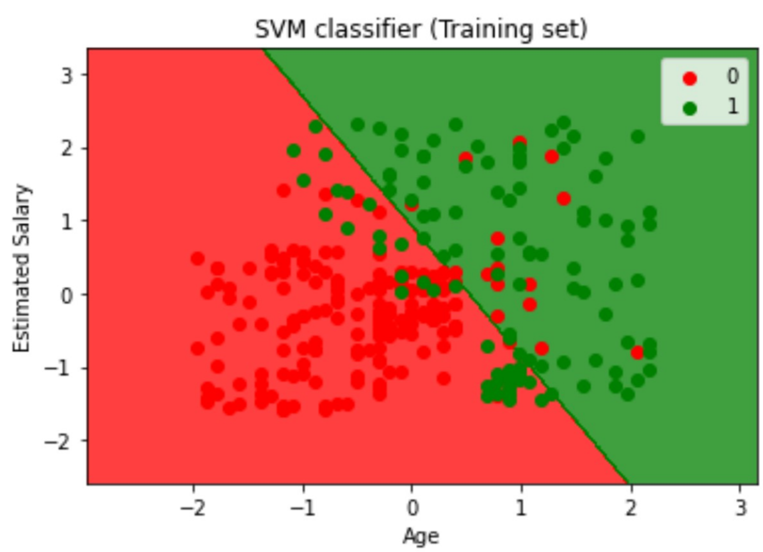
In [ ]:
```
from sklearn.svm import SVC
classifier=SVC(kernel='linear',random_state=0)
classifier.fit(x_train,y_train)
classifier.predict(sc.transform([[30,87000]]))
y_pred=classifier.predict(x_test)
```

In [ ]:
```
from sklearn.metrics import confusion_matrix,accuracy_score
cm=confusion_matrix(y_test,y_pred)
print(cm)
print('Accuracy Score: ',accuracy_score(y_test,y_pred))
```

```
[[70  2]
 [10 26]]
Accuracy Score:  0.8888888888888888
```

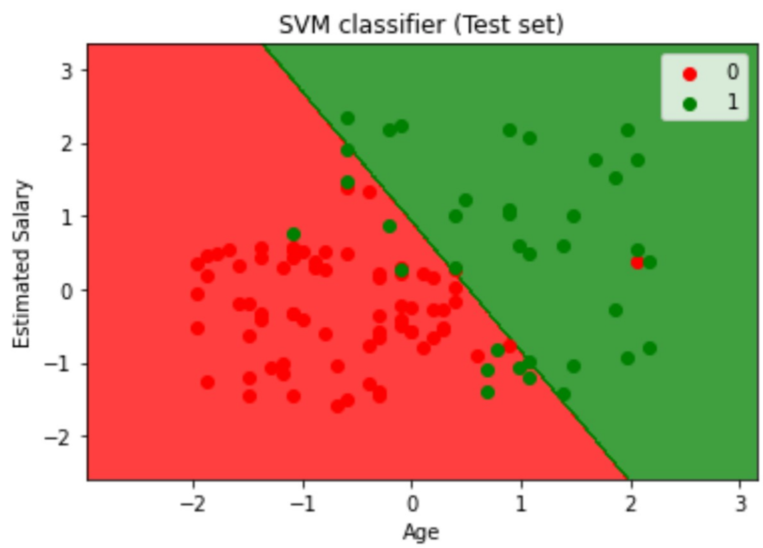
In [ ]:
```
from matplotlib.colors import ListedColormap
x_set, y_set = x_train, y_train
x1, x2 = np.meshgrid(np.arange(start = x_set[:, 0].min() - 1, stop = x_set
[:, 0].max() + 1, step  =0.01),
np.arange(start = x_set[:, 1].min() - 1, stop = x_set[:, 1].max() + 1, step
= 0.01))
plt.contourf(x1, x2, classifier.predict(np.array([x1.ravel(), x2.ravel()]).
T).reshape(x1.shape),
alpha = 0.75, cmap = ListedColormap(('red', 'green')))
plt.xlim(x1.min(), x1.max())
plt.ylim(x2.min(), x2.max())
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1],
        c = ListedColormap(('red', 'green'))(i), label = j)
plt.title('SVM classifier (Training set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()
```

```
<ipython-input-5-505925e35550>:10: UserWarning: *c* argument looks like a s
ingle numeric RGB or RGBA sequence, which should be avoided as value-mappin
g will have precedence in case its length matches with *x* & *y*.  Please u
se the *color* keyword-argument or provide a 2D array with a single row if
you intend to specify the same RGB or RGBA value for all points.
  plt.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1],
```



In [ ]:
```
#Visulaizing the test set result
from matplotlib.colors import ListedColormap
x_set, y_set = x_test, y_test
x1, x2 = np.meshgrid(np.arange(start = x_set[:, 0].min() - 1, stop = x_set
[:, 0].max() + 1, step  =0.01),
np.arange(start = x_set[:, 1].min() - 1, stop = x_set[:, 1].max() + 1, step
= 0.01))
plt.contourf(x1, x2, classifier.predict(np.array([x1.ravel(), x2.ravel()]).
T).reshape(x1.shape),
alpha = 0.75, cmap = ListedColormap(('red','green' )))
plt.xlim(x1.min(), x1.max())
plt.ylim(x2.min(), x2.max())
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1],
        c = ListedColormap(('red', 'green'))(i), label = j)
plt.title('SVM classifier (Test set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()
```

```
<ipython-input-6-ca44ee60ddc3>:11: UserWarning: *c* argument looks like a s
ingle numeric RGB or RGBA sequence, which should be avoided as value-mappin
g will have precedence in case its length matches with *x* & *y*.  Please u
se the *color* keyword-argument or provide a 2D array with a single row if
you intend to specify the same RGB or RGBA value for all points.
  plt.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1],
```



In [ ]:

In [ ]:

In [ ]:

without libraries

In [ ]:
```
# importing numpy library
import numpy as np
class SVM_classifier():


  # initiating the hyperparameters
  def __init__(self, learning_rate, no_of_iterations, lambda_parameter):

    self.learning_rate = learning_rate
    self.no_of_iterations = no_of_iterations
    self.lambda_parameter = lambda_parameter



  # fitting the dataset to SVM Classifier
  def fit(self, X, Y):

    # m  --> number of Data points --> number of rows
    # n  --> number of input features --> number of columns
    self.m, self.n = X.shape

    # initiating the weight value and bias value

    self.w = np.zeros(self.n)

    self.b = 0

    self.X = X

    self.Y = Y

    # implementing Gradient Descent algorithm for Optimization

    for i in range(self.no_of_iterations):
      self.update_weights()



  # function for updating the weight and bias value
  def update_weights(self):

    # label encoding
    y_label = np.where(self.Y <= 0, -1, 1)



    # gradients ( dw, db)
    for index, x_i in enumerate(self.X):

      condition = y_label[index] * (np.dot(x_i, self.w) - self.b) >= 1

      if (condition == True):

        dw = 2 * self.lambda_parameter * self.w
        db = 0
```

```
      else:

        dw = 2 * self.lambda_parameter * self.w - np.dot(x_i, y_label[inde
x])
        db = y_label[index]


      self.w = self.w - self.learning_rate * dw

      self.b = self.b - self.learning_rate * db



  # predict the label for a given input value
  def predict(self, X):

    output = np.dot(X, self.w) - self.b

    predicted_labels = np.sign(output)

    y_hat = np.where(predicted_labels <= -1, 0, 1)

    return y_hat
```

In [ ]:

```
classifier = SVM_classifier(learning_rate=0.001, no_of_iterations=1000, lam
bda_parameter=0.01)
```

In [ ]:

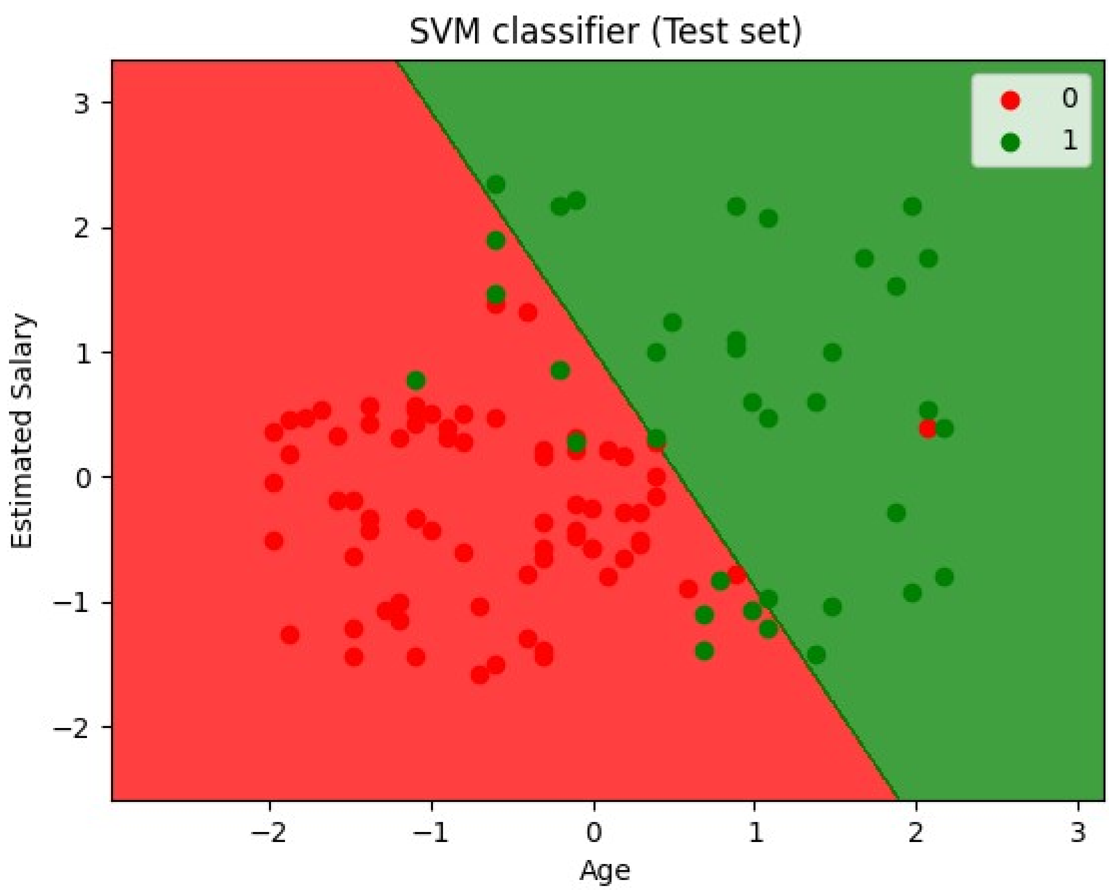
```
# training the SVM classifier with training data
classifier.fit(x_train, y_train)
```

In [ ]:
```
from matplotlib.colors import ListedColormap
x_set, y_set = x_train, y_train
x1, x2 = np.meshgrid(np.arange(start = x_set[:, 0].min() - 1, stop = x_set
[:, 0].max() + 1, step  =0.01),
np.arange(start = x_set[:, 1].min() - 1, stop = x_set[:, 1].max() + 1, step
= 0.01))
plt.contourf(x1, x2, classifier.predict(np.array([x1.ravel(), x2.ravel()]).
T).reshape(x1.shape),
alpha = 0.75, cmap = ListedColormap(('red', 'green')))
plt.xlim(x1.min(), x1.max())
plt.ylim(x2.min(), x2.max())
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1],
        c = ListedColormap(('red', 'green'))(i), label = j)
plt.title('SVM classifier (Training set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()
```

```
<ipython-input-8-505925e35550>:10: UserWarning: *c* argument looks like a s
ingle numeric RGB or RGBA sequence, which should be avoided as value-mappin
g will have precedence in case its length matches with *x* & *y*.  Please u
se the *color* keyword-argument or provide a 2D array with a single row if
you intend to specify the same RGB or RGBA value for all points.
  plt.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1],
```

In [ ]:
```
y_pred = classifier.predict(x_test)
#Visulaizing the test set result
from matplotlib.colors import ListedColormap
x_set, y_set = x_test, y_test
x1, x2 = np.meshgrid(np.arange(start = x_set[:, 0].min() - 1, stop = x_set
[:, 0].max() + 1, step  =0.01),
np.arange(start = x_set[:, 1].min() - 1, stop = x_set[:, 1].max() + 1, step
= 0.01))
plt.contourf(x1, x2, classifier.predict(np.array([x1.ravel(), x2.ravel()]).
T).reshape(x1.shape),
alpha = 0.75, cmap = ListedColormap(('red','green' )))
plt.xlim(x1.min(), x1.max())
plt.ylim(x2.min(), x2.max())
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1],
        c = ListedColormap(('red', 'green'))(i), label = j)
plt.title('SVM classifier (Test set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()
```

```
<ipython-input-11-8591ce39a30b>:12: UserWarning: *c* argument looks like a
single numeric RGB or RGBA sequence, which should be avoided as value-mappi
ng will have precedence in case its length matches with *x* & *y*.  Please
use the *color* keyword-argument or provide a 2D array with a single row if
you intend to specify the same RGB or RGBA value for all points.
  plt.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1],
```



In [ ]:
```
def accuracy(y_true, y_pred):
    # count the number of correctly predicted samples
    correct = np.sum(y_true == y_pred)
    # return the accuracy as a fraction of the total number of samples
    return correct / len(y_true)

print(accuracy(y_test,y_pred))
```

```
0.8981481481481481
```

In [ ]:
```
input_data = (5,166,72,19,175,25.8,0.587,51)

# change the input data to numpy array
input_data_as_numpy_array = np.asarray(input_data)

# reshape the array
input_data_reshaped = input_data_as_numpy_array.reshape(1,-1)

# standardizing the input data
std_data = sc.transform(input_data_reshaped)
print(std_data)

prediction = classifier.predict(std_data)
print(prediction)

if (prediction[0] == 0):
  print('The person is not diabetic')

else:
  print('The Person is diabetic')
```

In [ ]:
```
#svmnonlinear
import numpy as nm
import matplotlib.pyplot as mtp
import pandas as pd
```

In [ ]:
```
data_set= pd.read_csv('SVM.csv')

#Extracting Independent and dependent Variable
X= data_set.iloc[:, [2,3]].values
y= data_set.iloc[:, 4].values
#X = np.random.randn(200, 2)
#y = np.sign(X[:, 0]**2 + X[:, 1]**2 - 0.5)
# Splitting the dataset into training and test set.
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test= train_test_split(X, y, test_size= 0.25, r
andom_state=0)
#feature Scaling
from sklearn.preprocessing import StandardScaler
st_x= StandardScaler()
x_train= st_x.fit_transform(x_train)
x_test= st_x.transform(x_test)
```

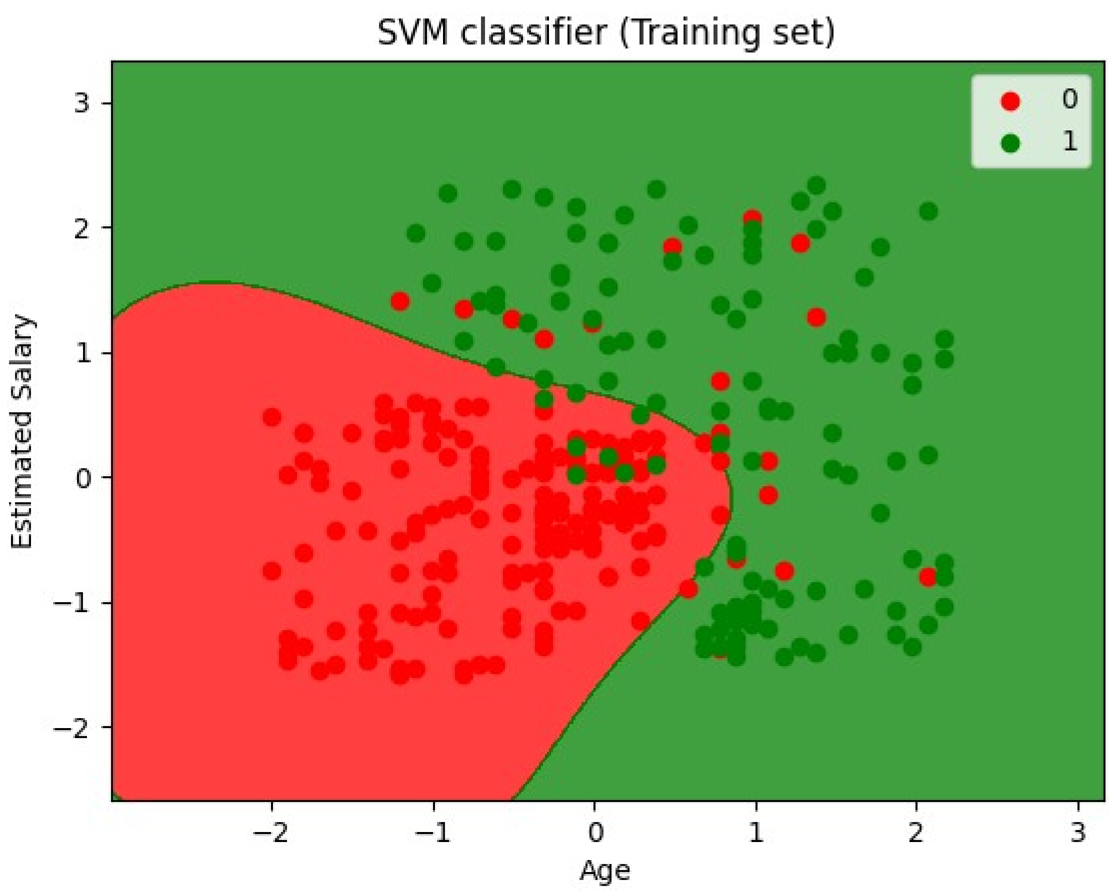
In [ ]:
```
from sklearn.svm import SVC
classifier = SVC(kernel='rbf', random_state=0)
classifier.fit(x_train, y_train)
y_pred= classifier.predict(x_test)
```

In [ ]:
```
from sklearn.metrics import confusion_matrix
cm= confusion_matrix(y_test, y_pred)
cm
```

Out[ ]:
```
array([[64,  4],
       [ 3, 29]])
```

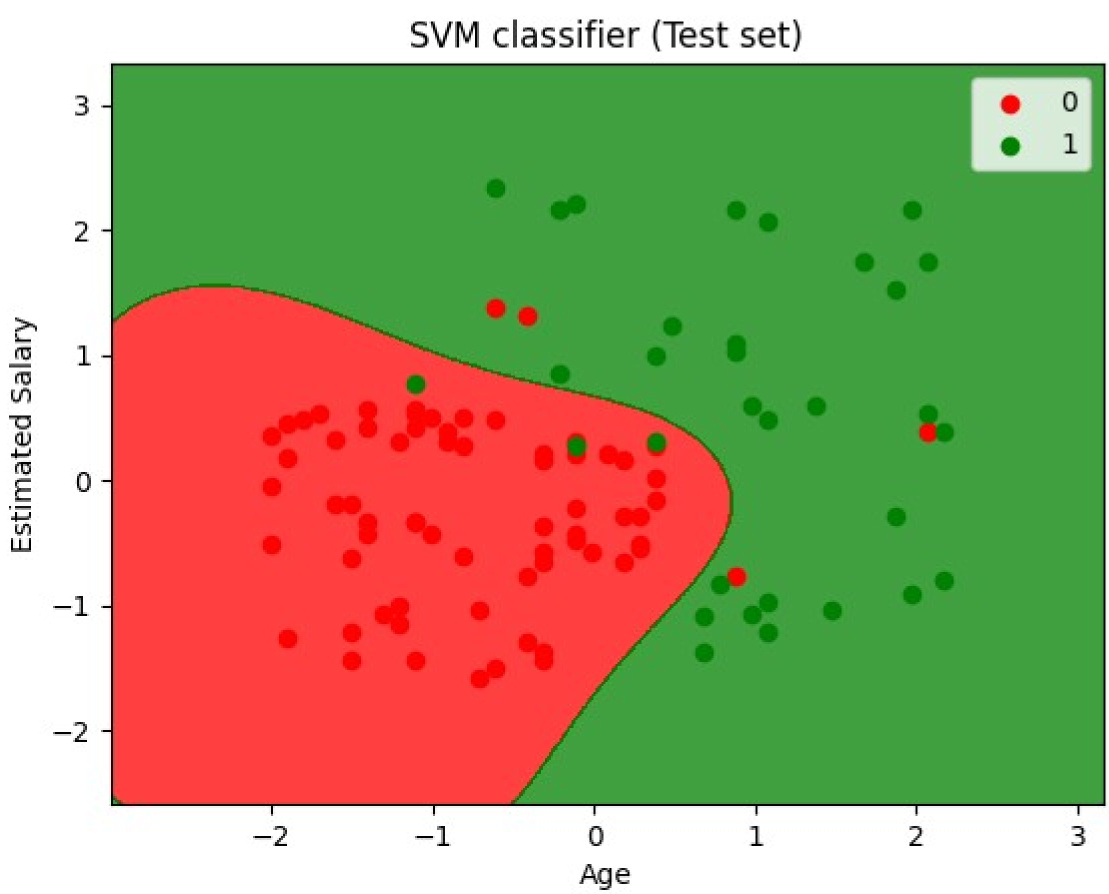
In [ ]:
```
from matplotlib.colors import ListedColormap
x_set, y_set = x_train, y_train
x1, x2 = nm.meshgrid(nm.arange(start = x_set[:, 0].min() - 1, stop = x_set
[:, 0].max() + 1, step  =0.01),
nm.arange(start = x_set[:, 1].min() - 1, stop = x_set[:, 1].max() + 1, step
= 0.01))
mtp.contourf(x1, x2, classifier.predict(nm.array([x1.ravel(), x2.ravel()]).
T).reshape(x1.shape),
alpha = 0.75, cmap = ListedColormap(('red', 'green')))
mtp.xlim(x1.min(), x1.max())
mtp.ylim(x2.min(), x2.max())
for i, j in enumerate(nm.unique(y_set)):
    mtp.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1],
        c = ListedColormap(('red', 'green'))(i), label = j)
mtp.title('SVM classifier (Training set)')
mtp.xlabel('Age')
mtp.ylabel('Estimated Salary')
mtp.legend()
mtp.show()
```

```
<ipython-input-25-d6fcd6d8ecf7>:10: UserWarning: *c* argument looks like a
single numeric RGB or RGBA sequence, which should be avoided as value-mappi
ng will have precedence in case its length matches with *x* & *y*.  Please
use the *color* keyword-argument or provide a 2D array with a single row if
you intend to specify the same RGB or RGBA value for all points.
  mtp.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1],
```



In [ ]:
```
#Visulaizing the test set result
from matplotlib.colors import ListedColormap
x_set, y_set = x_test, y_test
x1, x2 = nm.meshgrid(nm.arange(start = x_set[:, 0].min() - 1, stop = x_set
[:, 0].max() + 1, step  =0.01),
nm.arange(start = x_set[:, 1].min() - 1, stop = x_set[:, 1].max() + 1, step
= 0.01))
mtp.contourf(x1, x2, classifier.predict(nm.array([x1.ravel(), x2.ravel()]).
T).reshape(x1.shape),
alpha = 0.75, cmap = ListedColormap(('red','green' )))
mtp.xlim(x1.min(), x1.max())
mtp.ylim(x2.min(), x2.max())
for i, j in enumerate(nm.unique(y_set)):
    mtp.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1],
        c = ListedColormap(('red', 'green'))(i), label = j)
mtp.title('SVM classifier (Test set)')
mtp.xlabel('Age')
mtp.ylabel('Estimated Salary')
mtp.legend()
mtp.show()
```

```
<ipython-input-26-ddf28ec3e788>:11: UserWarning: *c* argument looks like a
single numeric RGB or RGBA sequence, which should be avoided as value-mappi
ng will have precedence in case its length matches with *x* & *y*.  Please
use the *color* keyword-argument or provide a 2D array with a single row if
you intend to specify the same RGB or RGBA value for all points.
  mtp.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1],
```



In [ ]:

In [ ]:

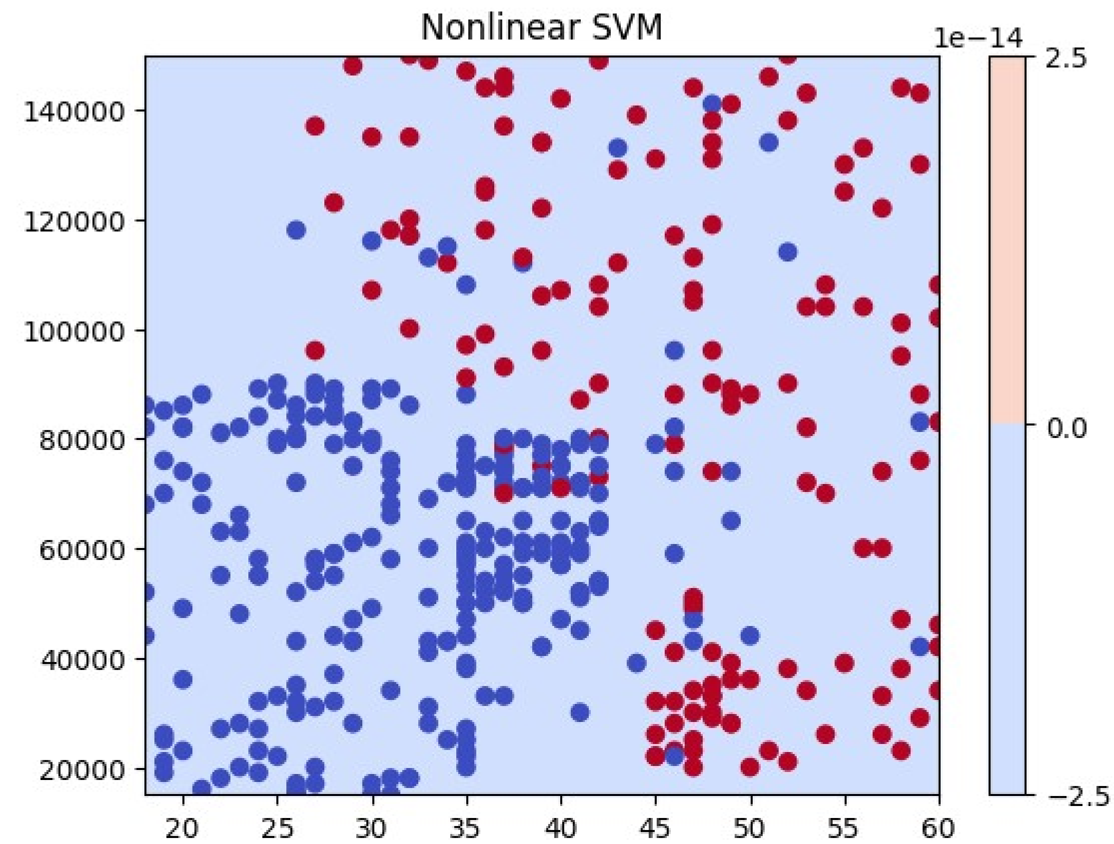
In [ ]:
```
import numpy as np
import matplotlib.pyplot as plt

class NonlinearSVM:
    def __init__(self, kernel='rbf', gamma=1.0, C=1.0, tol=1e-3, max_passes
=5):
        self.kernel = kernel
        self.gamma = gamma
        self.C = C
        self.tol = tol
        self.max_passes = max_passes

    def _kernel(self, X1, X2):
        if self.kernel == 'rbf':
            return np.exp(-self.gamma*np.linalg.norm(X1 - X2)**2)
        elif self.kernel == 'poly':
            return (1 + np.dot(X1, X2))**self.gamma

    def fit(self, X, y):
        self.X = X
        self.y = y
        self.alphas = np.zeros(len(X))
        self.b = 0.0
        self.K = np.zeros((len(X), len(X)))
        for i in range(len(X)):
            for j in range(len(X)):
                self.K[i, j] = self._kernel(X[i], X[j])
        self._smo(X, y)

    def _predict_one(self, x):
        return np.sign(np.sum(self.alphas*self.y*self._kernel(self.X, x)) +
self.b)

    def predict(self, X):
        return np.array([self._predict_one(x) for x in X])

    def _smo(self, X, y):
        passes = 0
        num_changed_alphas = 0
        while passes < self.max_passes and num_changed_alphas > 0:
            num_changed_alphas = 0
            for i in range(len(X)):
                Ei = self.predict(X[i]) - y[i]
                if (y[i]*Ei < -self.tol and self.alphas[i] < self.C) or (y
[i]*Ei > self.tol and self.alphas[i] > 0):
                    j = np.random.choice(list(range(i)) + list(range(i+1, l
en(X))))
                    Ej = self.predict(X[j]) - y[j]
                    alpha_i_old = self.alphas[i]
                    alpha_j_old = self.alphas[j]
                    if y[i] != y[j]:
                        L = max(0, self.alphas[j] - self.alphas[i])
                        H = min(self.C, self.C + self.alphas[j] - self.alph
as[i])
                    else:
```

```
                        L = max(0, self.alphas[i] + self.alphas[j] - self.
C)
                        H = min(self.C, self.alphas[i] + self.alphas[j])
                    if L == H:
                        continue
                    eta = 2*self.K[i, j] - self.K[i, i] - self.K[j, j]
                    if eta >= 0:
                        continue
                    self.alphas[j] -= y[j]*(Ei - Ej)/eta
                    self.alphas[j] = max(self.alphas[j], L)
                    self.alphas[j] = min(self.alphas[j], H)
                    if abs(self.alphas[j] - alpha_j_old) < 1e-5:
                        continue
                    self.alphas[i] += y[i]*y[j]*(alpha_j_old - self.alphas
[j])
                    b1 = self.b - Ei - y[i]*(self.alphas[i] - alpha_i_old)*
self.K[i, i] - y[j]*(self.alphas[j] - alpha_j_old)*self.K[i, j]

                    b2 = self.b - Ej - y[i]*(self.alphas[i] - alpha_i_old)*
self.K[i, j] - y[j]*(self.alphas[j] - alpha_j_old)*self.K[j, j]
                    if 0 < self.alphas[i] < self.C:
                        self.b = b1
                    elif 0 < self.alphas[j] < self.C:
                        self.b = b2
                    else:
                        self.b = (b1 + b2)/2.0
                    num_changed_alphas += 1
            if num_changed_alphas == 0:
                passes += 1

    def plot(self, X, y):
    # create a meshgrid over the feature space
      x1, x2 = np.meshgrid(np.linspace(np.min(X[:, 0]), np.max(X[:, 0]), 10
0),
                          np.linspace(np.min(X[:, 1]), np.max(X[:, 1]), 10
0))
      # compute predicted values for each point in the meshgrid
      Z = np.zeros(x1.shape)
      for i in range(x1.shape[0]):
          for j in range(x1.shape[1]):
              Z[i, j] = self._predict_one(np.array([x1[i, j], x2[i, j]]))

      # plot the contour plot of predicted values
      plt.contourf(x1, x2, Z, alpha=0.4, cmap=plt.cm.coolwarm)
      plt.colorbar()
      # plot the training data points
      plt.scatter(X[:, 0], X[:, 1], c=y, cmap=plt.cm.coolwarm)

      # plot the hyperplane
      w = np.dot(self.alphas * y, self.X)
      b = self.b
      xp = np.linspace(np.min(X[:, 0]), np.max(X[:, 0]), 100)
      yp = - (w[0] * xp + b) / w[1]
      plt.plot(xp, yp, '-k')

      plt.title('Nonlinear SVM')
```

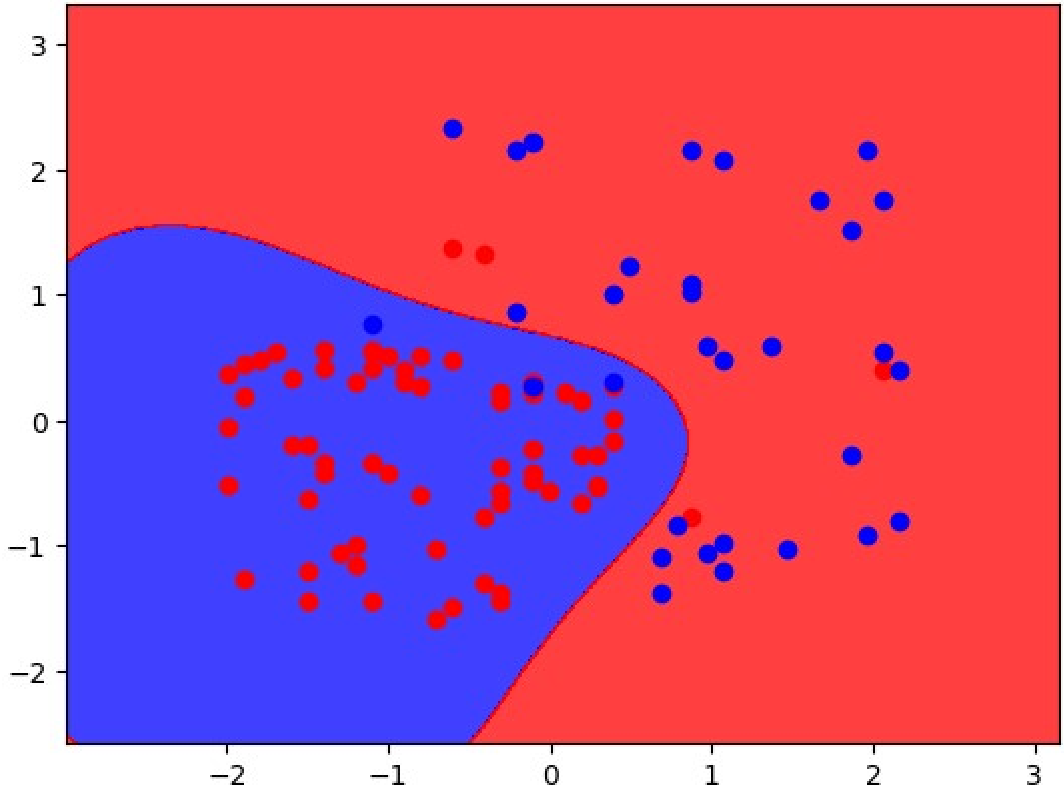
```
plt.show()
```

In [ ]:
```
# generate sample data
#X = np.random.randn(200, 2)
#y = np.sign(X[:, 0]**2 + X[:, 1]**2 - 0.5)

# train SVM
svm = NonlinearSVM(kernel='rbf', gamma=10.0, C=10.0, tol=1e-3, max_passes=
5)
svm.fit(X, y)

# plot decision boundary

svm.plot(X, y)
x_set, y_set = x_test, y_test
x1, x2 = nm.meshgrid(nm.arange(start = x_set[:, 0].min() - 1, stop = x_set
[:, 0].max() + 1, step  =0.01),
nm.arange(start = x_set[:, 1].min() - 1, stop = x_set[:, 1].max() + 1, step
= 0.01))
mtp.contourf(x1, x2, classifier.predict(nm.array([x1.ravel(), x2.ravel()]).
T).reshape(x1.shape),  alpha = 0.75, cmap = ListedColormap(('blue','red'
)))

mtp.xlim(x1.min(), x1.max())
mtp.ylim(x2.min(), x2.max())
for i, j in enumerate(nm.unique(y_set)):
    mtp.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1],
        c = ListedColormap(('red', 'blue'))(i), label = j)
```

```
<ipython-input-38-7cf8fd4f54cc>:97: RuntimeWarning: invalid value encounter
ed in true_divide
  yp = - (w[0] * xp + b) / w[1]
```




```
<ipython-input-42-d8fe7532bf95>:20: UserWarning: *c* argument looks like a
single numeric RGB or RGBA sequence, which should be avoided as value-mappi
ng will have precedence in case its length matches with *x* & *y*.  Please
use the *color* keyword-argument or provide a 2D array with a single row if
you intend to specify the same RGB or RGBA value for all points.
  mtp.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1],
```



In [ ]: