MANGALORE UNIVERSITY

MASTER OF SCIENCE

IN

COMPUTER SCIENCE

22CSH202: PRINCIPLES OF DATA SCIENCE

SUBMITTED

BY

II SEMESTER MSC

Computer Science Students

SUBMITTED

TO

Chairperson

The Department of Computer Science

Examiner:

1.

2.

Mangalore University

Dept. of Post-Graduate Studies and Research in Computer Science

Mangalagangothri – 574199

INDEX

1.Write a program to perform central tendency (mean, median, mode) with and without using built-in function on the data

```
# FIRST LET US CREATE AN LIST USING PROMPT

data=[]
n=int(input("enter the number of elements"))
for i in range(0,n):
  ele=int(input("ent5er the elements :"))
  data.append(ele)
print("the created list is",data)
```

```
enter the number of elements8
enter the elements :1
enter the elements :1
enter the elements :9
enter the elements :5
enter the elements :8
enter the elements :7
enter the elements :9
enter the elements :9

the created list is [1, 1, 9, 5, 8, 7, 9, 9]
```

```
#finding mean without using built in function

sum=0
for i in data:
  sum=sum+i
mean=sum/len(data)
print("mean (without using built in function) :",str(mean))

#finding mean using built in function

import numpy as np
print ("mean  (using built in function) : %s "%np.mean(data))
```

```
mean (without using built in function) : 6.125
mean  (using built in function) : 6.125
```

```
#finding median withou using built in function

# let us find  the median
sorted_data=sorted(data)
print("the sorted data is :",sorted_data)
if n%2==0:
  mid_index1=n//2
  mid_index2=mid_index1-1
  median=(sorted_data[mid_index1]+sorted_data[mid_index2])/2
else:
```

```python
  mid_index=n//2
  median=sorted_data[mid_index]
print("the median is :",median)



#finding median using function
import numpy as np
print('median(with using functions): %s'%np.median(data))
```

the sorted data is : [1, 1, 5, 7, 8, 9, 9, 9]
the median is : 7.5
median(with using functions): 7.5

```python
# now let us find the mode
count={}
for element in data:
  if element in count:
    count[element] +=1
  else:
    count[element] =1
max_counts=max(count.values())
mode=[element for element,c in count.items() if c==max_counts]
print("mode without using builtin ",mode)

#finding mode using function
import statistics as stat
print('mode(with using functions): %s'%stat.mode(data))
```

mode without using builtin [9]
mode(with using functions): 9

2. Write a program to perform measure of dispersion (range, variance, standard deviation, IQR) with and without using built-in function on the data

```
# FIRST LET US CREATE AN LIST USING PROMPT

data=[]
n=int(input("enter the number of elements"))
for i in range(0,n):
  ele=int(input("enter t1he elements :"))
  data.append(ele)
print("the created list is",data)
```

```
enter the number of elements5
enter t1he elements :3
enter t1he elements :2
enter t1he elements :8
enter t1he elements :9
enter t1he elements :1
the created list is [3, 2, 8, 9, 1]
```

```
minimum = maximum = data[0]
for num in data:
   if num <= minimum:
      minimum = num
   if num >= maximum:
      maximum = num

range_value = maximum - minimum
print("range (without function):", range_value)

#range with function
import numpy as np
min=np.min(data)
max=np.max(data)
range1=max-min
print('range(with function)',range1)
```

```
range (without function): 8
range(with function) 8
```

```
sum1=0
for i in data:
  sum1=sum1+i
mean=sum1/len(data)

# Calculating the variance
variance = sum((item - mean) ** 2 for item in data) / (len(data))
print("The variance of the set without functions:", variance)
```

```
# Using the numpy function
import numpy as np
print("Variance (with function):", np.var(data))
```

The variance of the set without functions: 10.64
Variance (with function): 10.64

```
# Calculating the standard deviation
std = variance ** 0.5
print("The standard deviation without using function is:", std)

# Using the numpy function
import numpy as np
print("standard deviation  (with function):", np.std(data))
```

The standard deviation without using function is: 3.2619012860600183
standard deviation (with function): 3.2619012860600183

```
import numpy as np
from scipy import stats


# Calculate IQR without using function
q3, q1 = np.percentile(data, [75, 25], interpolation='midpoint')
iqr = q3 - q1
print("IQR without using function:", iqr)

# Calculate IQR using function
IQR = stats.iqr(data, interpolation='midpoint')
print("IQR using function:", IQR)
```

IQR without using function: 6.0
IQR using function: 6.0

3.Implement Naive Bayes Classifier with suitable dataset and Modify the hyper parameter values

```
# Classification template

# Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

# Importing the dataset
dataset = pd.read_csv('/content/drive/MyDrive/DATA SCIENCE LAB/Social_Network_Ads.csv')
dataset
```

	User ID	Gender	Age	EstimatedSalary	Purchased
0	15624510	Male	19	19000	0
398	15755018	Male	36	33000	0
399	15594041	Female	49	36000	1

400 rows × 5 columns

```
X = dataset.iloc[:, [2, 3]].values
y = dataset.iloc[:, 4].values
```

```
# Splitting the dataset into the Training set 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, random_state = 0)
```

```
# Feature Scaling
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

```
# Fitting Naive Bayes to the Training set
from sklearn.naive_bayes import GaussianNB
classifier = GaussianNB(priors=[0.4,0.6],var_smoothing=1e-9)   #added hyper-parameters
classifier.fit(X_train, y_train)
```

GaussianNB(priors=[0.4, 0.6])

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

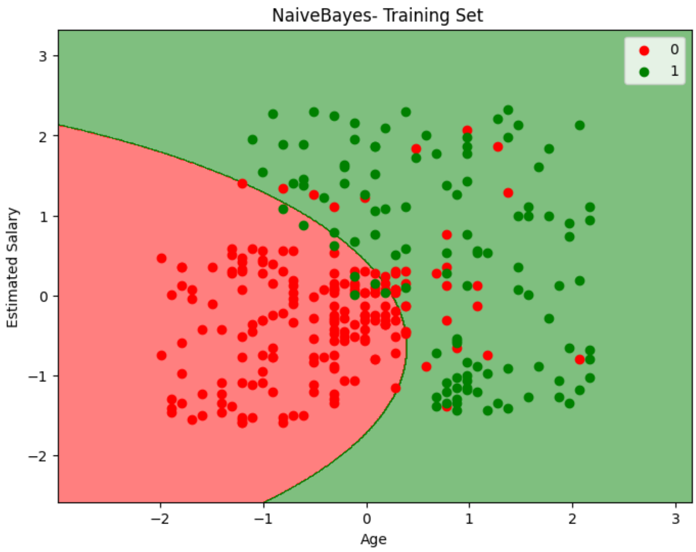
```
array([0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1,
       0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0,
       1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1,
       0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1,
    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

```
# Making the Confusion Matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
cm
```

```
array([[60,  8],
       [ 2, 30]])
```

```
# Visualizing the Training set results
from matplotlib.colors import ListedColormap

# Create a meshgrid to plot the decision boundary
X1, X2 = np.meshgrid(np.arange(start=X_train[:, 0].min() - 1, stop=X_train[:, 0].max() + 1,
step=0.01),
                     np.arange(start=X_train[:, 1].min() - 1, stop=X_train[:, 1].max() + 1, step=0.01))

# Use the classifier to predict the class labels for each point in the meshgrid
Z = classifier.predict(np.array([X1.ravel(), X2.ravel()]).T)
Z = Z.reshape(X1.shape)

# Create a color map for the plot
cmap = ListedColormap(('red', 'green'))

# Plot the training set data points
plt.figure(figsize=(8, 6))
plt.contourf(X1, X2, Z, alpha=0.5, cmap=cmap)
plt.scatter(X_train[y_train == 0, 0], X_train[y_train == 0, 1], color='red', label='0')
plt.scatter(X_train[y_train == 1, 0], X_train[y_train == 1, 1], color='green', label='1')
```

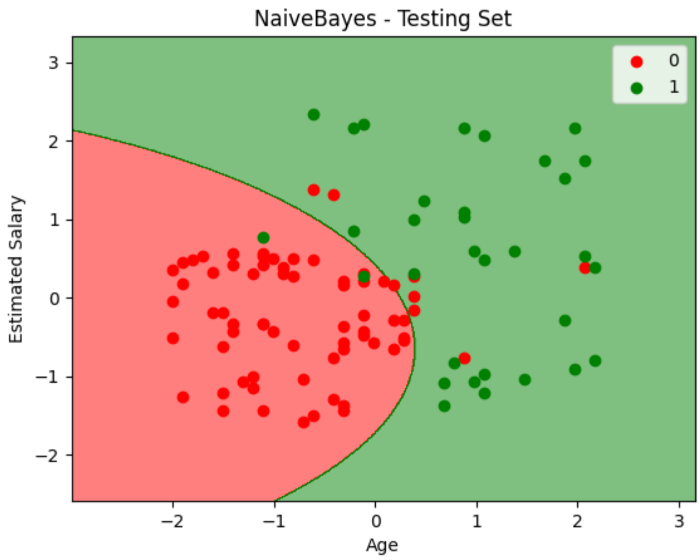
```
plt.title(' NaiveBayes- Training Set')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()
```




```
# Visualizing the Testing set results
from matplotlib.colors import ListedColormap

# Create a meshgrid of feature values
X1, X2 = np.meshgrid(np.arange(start = X_test[:, 0].min() - 1, stop = X_test[:, 0].max() + 1, step =
0.01),
          np.arange(start = X_test[:, 1].min() - 1, stop = X_test[:, 1].max() + 1, step = 0.01))

# Use the trained classifier to make predictions on the meshgrid points
Z = classifier.predict(np.array([X1.ravel(), X2.ravel()]).T)
Z = Z.reshape(X1.shape)

# Create a colormap for the two classes
cmap = ListedColormap(('red', 'green'))

# Plot the contour filled by the predictions
plt.contourf(X1, X2, Z, alpha = 0.5, cmap = cmap)

# Scatter plot the actual data points
plt.scatter(X_test[y_test == 0, 0], X_test[y_test == 0, 1], color = 'red', label = '0')
plt.scatter(X_test[y_test == 1, 0], X_test[y_test == 1, 1], color = 'green', label = '1')

# Add labels and legend
plt.title('NaiveBayes - Testing Set')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
```

plt.legend()

 # Show the plot
plt.show()



from sklearn.metrics import accuracy_score
print("the accuracy score is :",accuracy_score(y_pred,y_test))

the accuracy score is : 0.9

4. Implement Support Vector Machine(SVM) Classifier with

Suitable dataset.

```
# Support Vector Machine (SVM)

# Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

# Importing the dataset
dataset = pd.read_csv('/content/drive/MyDrive/DATA SCIENCE LAB/Social_Network_Ads.csv')
dataset
```

	User ID	Gender	Age	Estimated Salary	Purchased
0	15624510	Male	19	19000	0
1	15810944	Male	35	20000	0
2	15668575	Female	26	43000	0
3	15603246	Female	27	57000	0
4	15804002	Male	19	76000	0
...	...	...	...	...	...
395	15691863	Female	46	41000	1
396	15706071	Male	51	23000	1
397	15654296	Female	50	20000	1
398	15755018	Male	36	33000	0

400 rows × 5 columns

```python
    X = dataset.iloc[:, [2, 3]].values
    y = dataset.iloc[:, 4].values
```

```python
# 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, random_state = 0)
```

```python
# Feature Scaling
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

```python
# Fitting SVM to the Training set
from sklearn.svm import SVC
classifier = SVC(kernel = 'linear', random_state = 0)
classifier.fit(X_train, y_train)
```

```
SVC(kernel='linear', random_state=0)
```

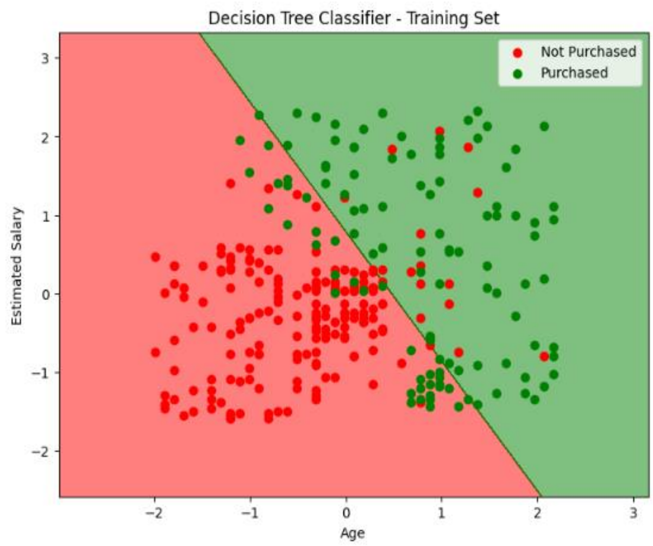
```python
# Predicting the Test set results
y_pred = classifier.predict(X_test)
y_pred
```

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

```python
# Making the Confusion Matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
cm
```

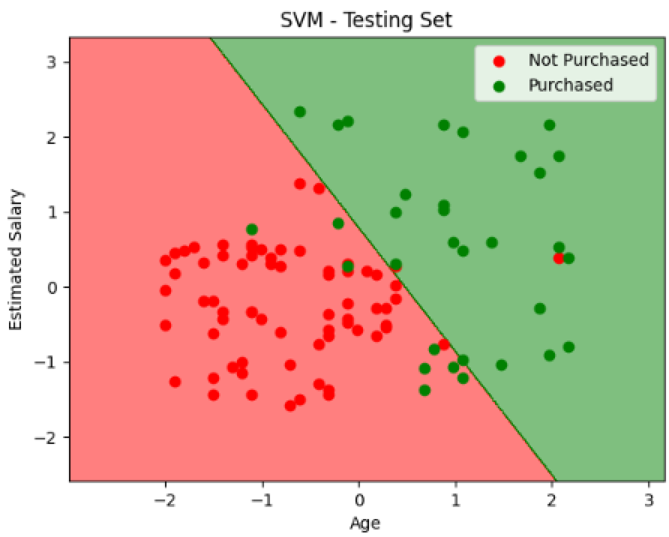
```
array([[66,  2],
       [ 8, 24]])
```

```
# Visualizing the Training set results
from matplotlib.colors import ListedColormap

# Create a meshgrid to plot the decision boundary
X1, X2 = np.meshgrid(np.arange(start=X_train[:, 0].min() - 1, stop=X_train[:, 0].max() + 1,
step=0.01),
                     np.arange(start=X_train[:, 1].min() - 1, stop=X_train[:, 1].max() + 1, step=0.01))

# Use the classifier to predict the class labels for each point in the meshgrid
Z = classifier.predict(np.array([X1.ravel(), X2.ravel()]).T)
Z = Z.reshape(X1.shape)

# Create a color map for the plot
cmap = ListedColormap(('red', 'green'))

# Plot the training set data points
plt.figure(figsize=(8, 6))
plt.contourf(X1, X2, Z, alpha=0.5, cmap=cmap)
plt.scatter(X_train[y_train == 0, 0], X_train[y_train == 0, 1], color='red', label='Not Purchased')
plt.scatter(X_train[y_train == 1, 0], X_train[y_train == 1, 1], color='green', label='Purchased')
plt.title('Decision Tree Classifier - Training Set')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()
```




```
# Visualizing the Testing set results
from matplotlib.colors import ListedColormap

# Create a meshgrid of feature values
```

```
X1, X2 = np.meshgrid(np.arange(start = X_test[:, 0].min() - 1, stop = X_test[:, 0].max() + 1, step = 0.01),
                     np.arange(start = X_test[:, 1].min() - 1, stop = X_test[:, 1].max() + 1, step = 0.01))

# Use the trained classifier to make predictions on the meshgrid points
Z = classifier.predict(np.array([X1.ravel(), X2.ravel()]).T)
Z = Z.reshape(X1.shape)

# Create a colormap for the two classes
cmap = ListedColormap(('red', 'green'))

# Plot the contour filled by the predictions
plt.contourf(X1, X2, Z, alpha = 0.5, cmap = cmap)

# Scatter plot the actual data points
plt.scatter(X_test[y_test == 0, 0], X_test[y_test == 0, 1], color = 'red', label = 'Not Purchased')
plt.scatter(X_test[y_test == 1, 0], X_test[y_test == 1, 1], color = 'green', label = 'Purchased')

# Add labels and legend
plt.title('SVM - Testing Set')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()

# Show the plot
plt.show()
```




```
from sklearn.metrics import accuracy_score
print ('Accuracy : ', accuracy_score(y_test, y_pred))
```

Accuracy : 0.9

5. Implement Decision Tree Clustering.

```
# Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.tree import DecisionTreeClassifier, plot_tree

# Importing the dataset/content/drive/MyDrive/DATA SCIENCE LAB/Social_Network_Ads.csv
dataset = pd.read_csv('/content/drive/MyDrive/DATA SCIENCE LAB/Social_Network_Ads.csv')
dataset
```

	User ID	Gender	Age	EstimatedSalary	Purchased
0	15624510	Male	19	19000	0
1	15810944	Male	35	20000	0
399	15594041	Female	49	36000	1

400 rows × 5 columns

```
X=dataset.iloc[:,[2,3]]
y=dataset.iloc[:,4]

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, random_state = 0)

from sklearn.preprocessing import StandardScaler
sts=StandardScaler()
X_train=sts.fit_transform(X_train)
X_test=sts.transform(X_test)

from sklearn.tree import DecisionTreeClassifier
classifier = DecisionTreeClassifier(criterion = 'entropy', random_state = 0)
classifier.fit(X_train, y_train)

DecisionTreeClassifier(criterion='entropy', random_state=0)

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

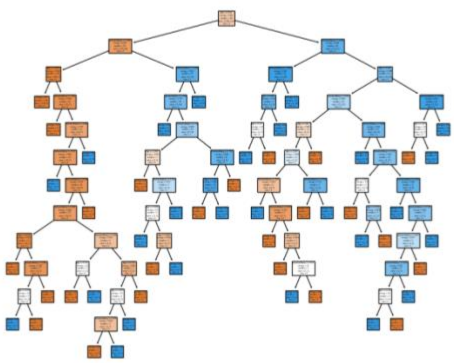
```
y_pred
```

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

```
# Making the Confusion Matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
cm
```

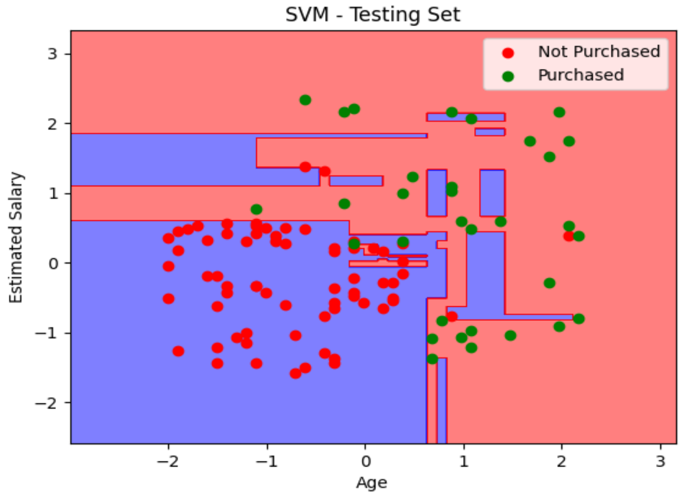
```
array([[62,  6],
       [ 3, 29]])
```

```
# Plotting the Decision Tree graph
plt.figure(figsize=(10, 8))
plot_tree(classifier, feature_names=['Age', 'EstimatedSalary'], class_names=['0', '1'], filled=True)
plt.show()
```



```
# Visualizing the Testing set results
from matplotlib.colors import ListedColormap

# Create a meshgrid of feature values
X1, X2 = np.meshgrid(np.arange(start = X_test[:, 0].min() - 1, stop = X_test[:, 0].max() + 1, step =
0.01),
            np.arange(start = X_test[:, 1].min() - 1, stop = X_test[:, 1].max() + 1, step = 0.01))

# Use the trained classifier to make predictions on the meshgrid points
Z = classifier.predict(np.array([X1.ravel(), X2.ravel()]).T)
Z = Z.reshape(X1.shape)

# Create a colormap for the two classes
cmap = ListedColormap(('blue',"red"))

# Plot the contour filled by the predictions
plt.contourf(X1, X2, Z, alpha = 0.5, cmap = cmap)
```

```
# Scatter plot the actual data points
plt.scatter(X_test[y_test == 0, 0], X_test[y_test == 0, 1], color = 'red', label = 'Not Purchased')
plt.scatter(X_test[y_test == 1, 0], X_test[y_test == 1, 1], color = 'green', label = 'Purchased')

# Add labels and legend
plt.title('SVM - Testing Set')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()

# Show the plot
plt.show()
```




```
from sklearn.metrics import accuracy_score
print("the accuracy score is :",accuracy_score(y_pred,y_test))
```

the accuracy score is : 0.91

6.Implement K-means Clustering.

```
# Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

# Importing the dataset
dataset = pd.read_csv('/content/drive/MyDrive/DATA SCIENCE LAB/Mall_Customers.csv')
dataset
```

	CustomerID	Genre	Age	Annual Income (k$)	Spending Score (1-100)
0	1	Male	19	15	39
198	199	Male	32	137	18
199	200	Male	30	137	83

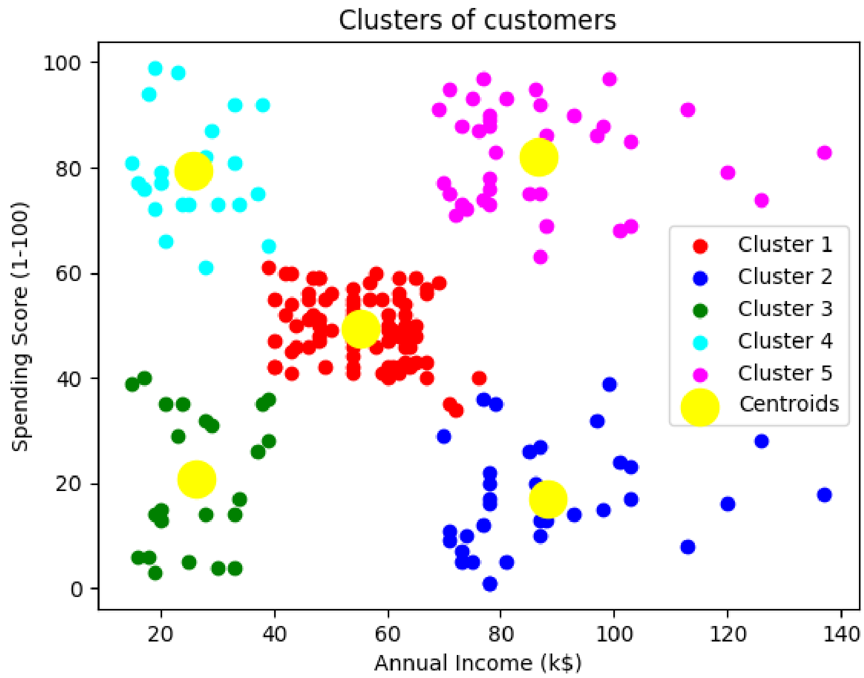
200 rows × 5 columns

```
import matplotlib.pyplot as plt

# Example data points
spending_scores =dataset.iloc[:,3].values
annual_incomes = dataset.iloc[:, 4].values

# Plotting the scatter plot
plt.scatter(spending_scores,annual_incomes)
plt.title('Spending Score vs. Annual Income')
plt.xlabel('Annual Income')
plt.ylabel('Spending Score')
plt.show()
```


```
X = dataset.iloc[:, [3, 4]].values
```

```
# Fitting K-Means to the dataset
from sklearn.cluster import KMeans
kmeans = KMeans(n_clusters = 5, init = 'k-means++', random_state = 42)
y_kmeans = kmeans.fit_predict(X)
```

```
# Visualising the clusters
plt.scatter(X[y_kmeans == 0, 0], X[y_kmeans == 0, 1], c = 'red', label = 'Cluster 1')
plt.scatter(X[y_kmeans == 1, 0], X[y_kmeans == 1, 1], c = 'blue', label = 'Cluster 2')
plt.scatter(X[y_kmeans == 2, 0], X[y_kmeans == 2, 1], c = 'green', label = 'Cluster 3')
plt.scatter(X[y_kmeans == 3, 0], X[y_kmeans == 3, 1], c = 'cyan', label = 'Cluster 4')
plt.scatter(X[y_kmeans == 4, 0], X[y_kmeans == 4, 1], c = 'magenta', label = 'Cluster 5')
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], s = 300, c = 'yellow', label
= 'Centroids')
plt.title('Clusters of customers')
plt.xlabel('Annual Income (k$)')
plt.ylabel('Spending Score (1-100)')
plt.legend()
plt.show()
```


Clusters of customers
Spending Score (1-100)
Annual Income (k$)
0
20
40
60
80
100
20
40
60
80
100
120
140
Cluster 1
Cluster 2
Cluster 3
Cluster 4
Cluster 5
Centroids

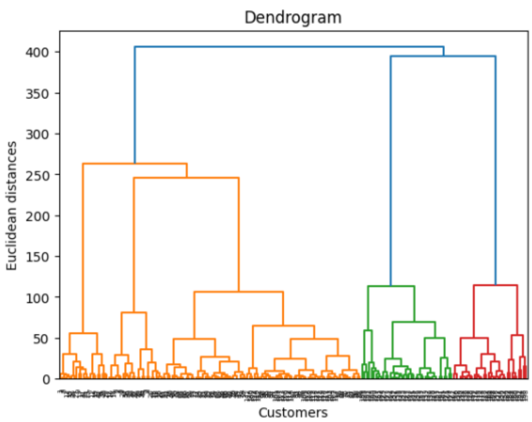
7.Implement Hierarchical clustering with suitable dataset.

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

# Importing the dataset
dataset = pd.read_csv('/content/drive/MyDrive/DATA SCIENCE LAB/Mall_Customers.csv')
X = dataset.iloc[:, [3, 4]].values
dataset
```

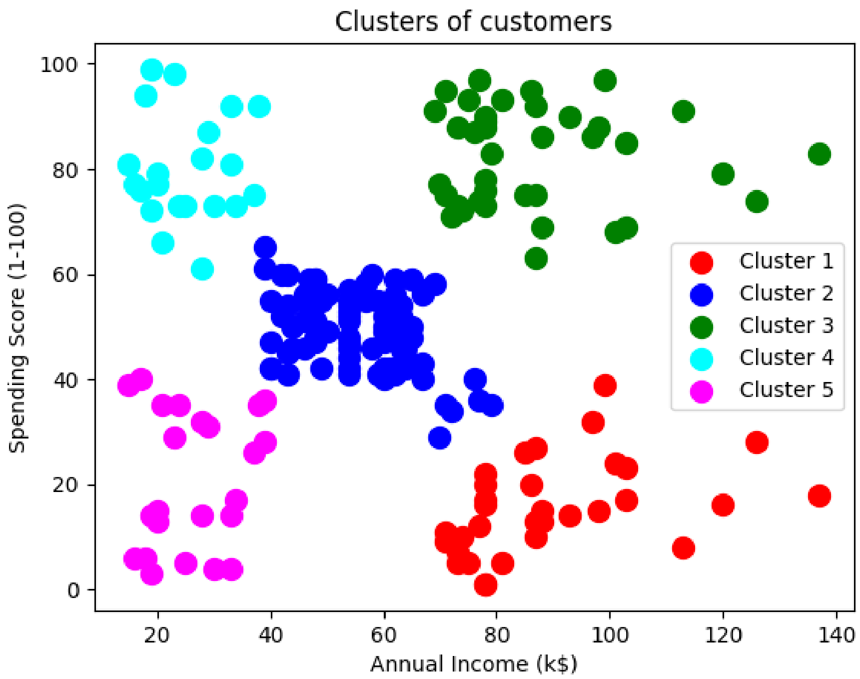
	CustomerID	Genre	Age	Annual Income (k$)	Spending Score (1-100)
0	1	Male	19	15	39
1	2	Male	21	15	81
199	200	Male	30	137	83

200 rows × 5 columns

```
# Using the dendrogram to find the optimal number of clusters
import scipy.cluster.hierarchy as sch
dendrogram = sch.dendrogram(sch.linkage(X, method = 'ward'))
plt.title('Dendrogram')
plt.xlabel('Customers')
plt.ylabel('Euclidean distances')
plt.show()
```



```
# Fitting Hierarchical Clustering to the dataset
from sklearn.cluster import AgglomerativeClustering
hc = AgglomerativeClustering(n_clusters = 5, affinity = 'euclidean', linkage = 'ward')
y_hc = hc.fit_predict(X)


# Visualising the clusters
plt.scatter(X[y_hc == 0, 0], X[y_hc == 0, 1], s = 100, c = 'red', label = 'Cluster 1')
plt.scatter(X[y_hc == 1, 0], X[y_hc == 1, 1], s = 100, c = 'blue', label = 'Cluster 2')
plt.scatter(X[y_hc == 2, 0], X[y_hc == 2, 1], s = 100, c = 'green', label = 'Cluster 3')
plt.scatter(X[y_hc == 3, 0], X[y_hc == 3, 1], s = 100, c = 'cyan', label = 'Cluster 4')
plt.scatter(X[y_hc == 4, 0], X[y_hc == 4, 1], s = 100, c = 'magenta', label = 'Cluster 5')
plt.title('Clusters of customers')
plt.xlabel('Annual Income (k$)')
plt.ylabel('Spending Score (1-100)')
plt.legend()
plt.show()
```



8.Implement Density Based Clustering.

```
import pandas as pd
from sklearn.cluster import DBSCAN
import matplotlib.pyplot as plt
from sklearn.datasets import make_blobs

# Read data from CSV file
data = pd.read_csv('/content/drive/MyDrive/DATA SCIENCE LAB/blobs.csv')
data
# Generate synthetic data
# data, _ = make_blobs(n_samples=300, centers=4, cluster_std=.60, random_state=0)
```

	0	1
0	8.622185	1.935796
1	-4.736710	-7.970958
999	6.120559	0.968963

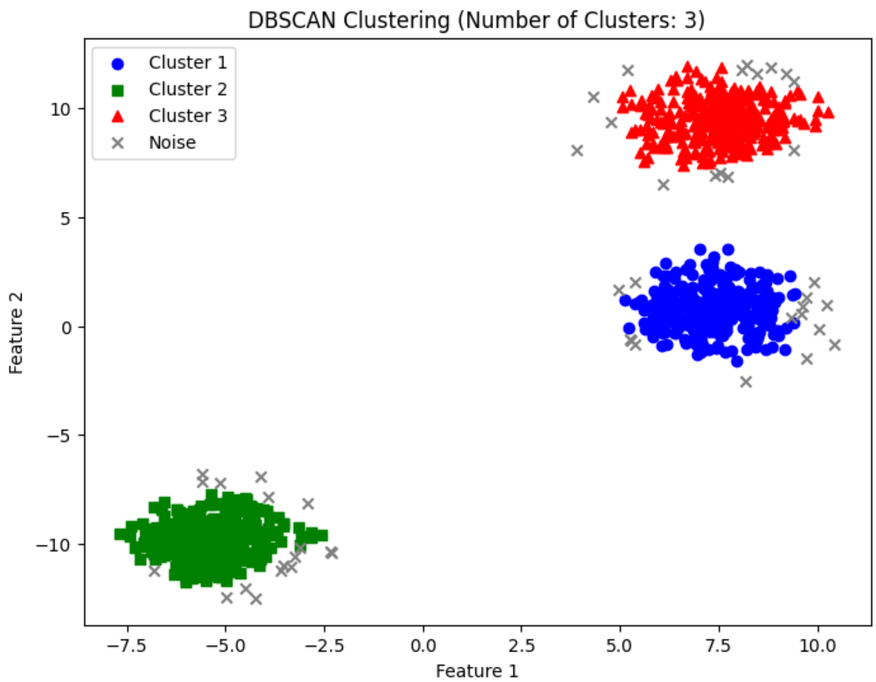
1000 rows × 2 columns

```
# Extract the features (assuming your CSV file has columns 'Feature1' and 'Feature2')
X = data.iloc[:,[0,1]].values
X
```

```
array([[ 8.62218539,   1.93579579]
      [ -4.73670958,  -7.97095765],
      [  9.62122205,   0.92542315],
      ...,
      [ -6.2522678 ,  -8.412482  ],
      [ -5.479154  , -10.53695547],
      [  6.12055883,   0.96896287]])
```

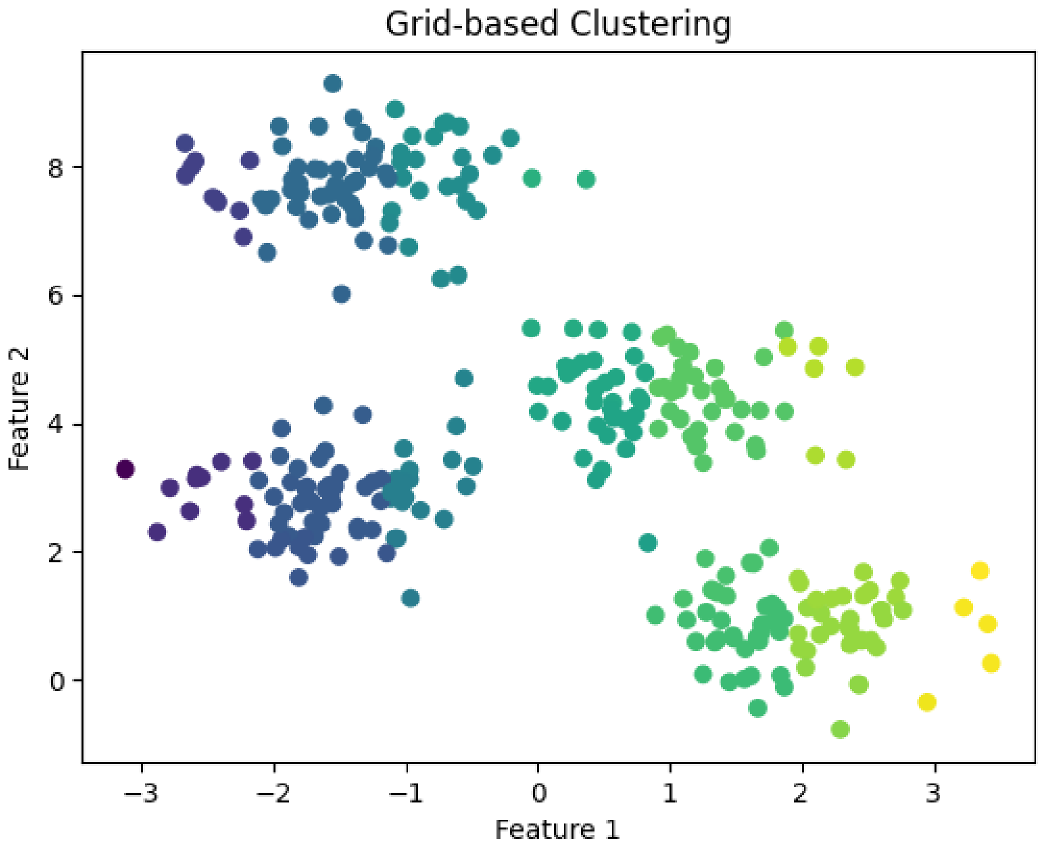
```
# DBSCAN clustering
db = DBSCAN(eps=0.5, min_samples=5)
y_db = db.fit_predict(X)

# Number of clusters in labels, ignoring noise if present (-1)
n_clusters_ = len(set(y_db)) - (1 if -1 in y_db else 0)
```

```
# Plot the clusters
plt.figure(figsize=(8, 6))
plt.scatter(X[y_db == 0][:, 0], X[y_db == 0][:, 1], c='blue', marker='o', label='Cluster 1')
plt.scatter(X[y_db == 1][:, 0], X[y_db == 1][:, 1], c='green', marker='s', label='Cluster 2')
plt.scatter(X[y_db == 2][:, 0], X[y_db == 2][:, 1], c='red', marker='^', label='Cluster 3')
plt.scatter(X[y_db == -1][:, 0], X[y_db == -1][:, 1], c='gray', marker='x', label='Noise')
plt.legend(loc='best')
plt.title(f"DBSCAN Clustering (Number of Clusters: {n_clusters_})")
plt.xlabel("Feature 1")
plt.ylabel("Feature 2")
plt.show()
```



9. Implement Grid Based Clustering.

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_blobs

# Generate synthetic data
data, _ = make_blobs(n_samples=300, centers=4, cluster_std=.60, random_state=0)


# Set the grid size (you can adjust this based on your data distribution)
grid_size = 1.0

# Get the minimum and maximum values for x and y coordinates
x_min, x_max = data[:, 0].min(), data[:, 0].max()
y_min, y_max = data[:, 1].min(), data[:, 1].max()


# Create a grid by defining intervals using the minimum and maximum values
x_grid = np.arange(x_min, x_max + grid_size, grid_size)
y_grid = np.arange(y_min, y_max + grid_size, grid_size)


# Initialize labels array with zeros
labels = np.zeros(data.shape[0], dtype=int)


# Assign each data point to a grid cell based on its coordinates
for i, point in enumerate(data):
   x, y = point
   x_label = np.searchsorted(x_grid, x) - 1
   y_label = np.searchsorted(y_grid, y) - 1
   labels[i] = x_label * len(y_grid) + y_label


# Visualize the clusters
plt.scatter(data[:, 0], data[:, 1], c=labels, cmap='viridis')
plt.title('Grid-based Clustering')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.show()
```



Grid-based Clustering
Feature 2
Feature 1
8
6
4
2
0
−3
−2
−1
0
1
2
3

10. Implement K-Nearest Neighbour.

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

```
data_set= pd.read_csv('/content/drive/MyDrive/DATA SCIENCE LAB/Social_Network_Ads.csv')
data_set
```

	User ID	Gender	Age	EstimatedSalary	Purchased
0	15624510	Male	19	19000	0
1	15810944	Male	35	20000	0
399	15594041	Female	49	36000	1

400 rows × 5 columns

```
#Extracting Independent and dependent Variable
x= data_set.iloc[:, [2,3]].values
y= data_set.iloc[:, 4].values
```

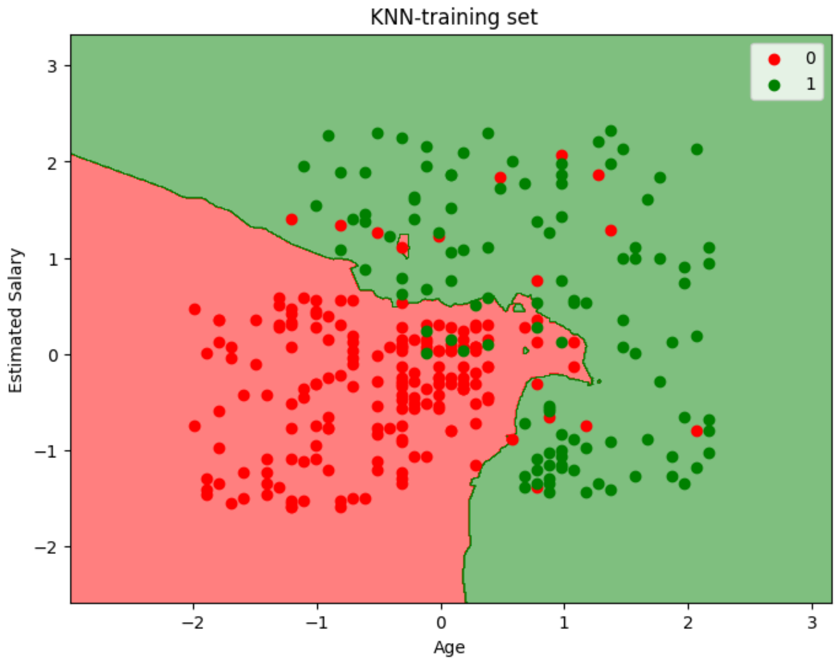
```
# Splitting the dataset into training and test set.
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test= train_test_split(x, y, test_size= 0.25, random_state=0)
```

```
#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)
```

```
#Fitting K-NN classifier to the training set
from sklearn.neighbors import KNeighborsClassifier
```

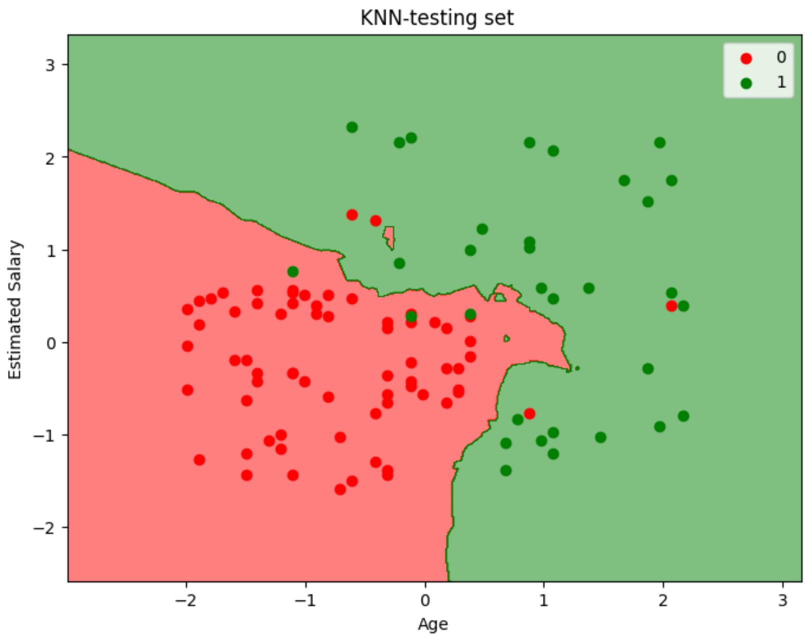
```
classifier= KNeighborsClassifier(n_neighbors=5, metric='minkowski', p=2 )
classifier.fit(X_train, y_train)


KNeighborsClassifier()

#Predicting the test set result
y_pred= classifier.predict(X_test)
y_pred


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


#Creating the Confusion matrix
from sklearn.metrics import confusion_matrix
cm= confusion_matrix(y_test, y_pred)
print('Confusion Matrix : \n',cm)

Confusion Matrix :
 [[64  4]
 [ 3 29]]


# Visualizing the Training set results
import numpy as np
from matplotlib.colors import ListedColormap

# Create a meshgrid to plot the decision boundary
X1, X2 = np.meshgrid(np.arange(start=X_train[:, 0].min() - 1, stop=X_train[:, 0].max() + 1,
step=0.01),
                np.arange(start=X_train[:, 1].min() - 1, stop=X_train[:, 1].max() + 1, step=0.01))

# Use the classifier to predict the class labels for each point in the meshgrid
Z = classifier.predict(np.array([X1.ravel(), X2.ravel()]).T)
Z = Z.reshape(X1.shape)

# Create a color map for the plot
cmap = ListedColormap(('red', 'green'))

# Plot the training set data points
plt.figure(figsize=(8, 6))
plt.contourf(X1, X2, Z, alpha=0.5, cmap=cmap)
plt.scatter(X_train[y_train == 0, 0], X_train[y_train == 0, 1], color='red', label='0')
plt.scatter(X_train[y_train == 1, 0], X_train[y_train == 1, 1], color='green', label='1')
plt.title(' KNN-training set')
plt.xlabel('Age')
```

```
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()
```




```
# Visualizing the Training set results
import numpy as np
from matplotlib.colors import ListedColormap

# Create a meshgrid to plot the decision boundary
X1, X2 = np.meshgrid(np.arange(start=X_test[:, 0].min() - 1, stop=X_test[:, 0].max() + 1, step=0.01),
                     np.arange(start=X_test[:, 1].min() - 1, stop=X_test[:, 1].max() + 1, step=0.01))

# Use the classifier to predict the class labels for each point in the meshgrid
Z = classifier.predict(np.array([X1.ravel(), X2.ravel()]).T)
Z = Z.reshape(X1.shape)

# Create a color map for the plot
cmap = ListedColormap(('red', 'green'))

# Plot the training set data points
plt.figure(figsize=(8, 6))
plt.contourf(X1, X2, Z, alpha=0.5, cmap=cmap)
plt.scatter(X_test[y_test == 0, 0], X_test[y_test== 0, 1], color='red', label='0')
plt.scatter(X_test[y_test == 1, 0], X_test[y_test == 1, 1], color='green', label='1')
plt.title(' KNN-testing set')
```

```
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()
```



11. Write a python program to perform chi square test

```
import pandas as pd
# Load your data into a pandas DataFrame
data = pd.read_csv('/content/drive/MyDrive/DATA SCIENCE LAB/fruit_data_with_colours.csv')
data.head(5)
```

	fruit_label	fruit_name	fruit_subtype	mass	width	height	color_score
0	1	apple	granny_smith	192	8.4	7.3	0.55
1	1	apple	granny_smith	180	8.0	6.8	0.59
2	1	apple	granny_smith	176	7.4	7.2	0.60
3	2	mandarin	mandarin	86	6.2	4.7	0.80
4	2	mandarin	mandarin	84	6.0	4.6	0.79

```
# Separate the target variable from features
dff = data.drop('fruit_label', axis=1)
df = dff.drop('fruit_subtype', axis=1)
X = df.drop('fruit_name', axis=1)
y = data['fruit_label']
```

```
from sklearn.feature_selection import SelectKBest, chi2
# Step 1: Feature selection using chi-square test

k_selected_features = 4  # Adjust this value based on how many top features you want to select
chi2_selector = SelectKBest(chi2, k=k_selected_features)
X_selected = chi2_selector.fit_transform(X, y)
```

```
# Step 2: Split the data into training and testing sets

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X_selected, y, test_size=0.2, random_state=42)
```

```python
# Step 3: Train the SVM classifier
from sklearn.svm import SVC
svm_classifier = SVC(kernel='linear')
svm_classifier.fit(X_train, y_train)
```

```
SVC(kernel='linear')
```

```python
# Step 4: Evaluate the SVM classifier
from sklearn.metrics import accuracy_score, classification_report
y_pred = svm_classifier.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
report = classification_report(y_test, y_pred)
# Print results
print("Accuracy:", accuracy)
print("Classification Report:")
print(report)
```

```
Accuracy: 0.75
Classification Report:
              precision    recall  f1-score   support

           1       0.67      0.67      0.67         3
           2       1.00      1.00      1.00         2
           3       0.33      0.50      0.40         2
           4       1.00      0.80      0.89         5

    accuracy                           0.75        12
   macro avg       0.75      0.74      0.74        12
weighted avg       0.81      0.75      0.77        12
```

12. Write a python program to perform T-test

```
import pandas as pd
# Load your data into a pandas DataFrame
data = pd.read_csv('/content/drive/MyDrive/DATA SCIENCE LAB/fruit_data_with_colours.csv')
```

```
# Separate the target variable from features
dff = data.drop('fruit_label', axis=1)
df = dff.drop('fruit_subtype', axis=1)
X = df.drop('fruit_name', axis=1)
y = data['fruit_label']
```

```
from sklearn.feature_selection import SelectKBest, f_classif

# Step 1: Feature selection using t-test (f_classif)
k_selected_features = 3  # Adjust this value based on how many top features you want to select
f_classif_selector = SelectKBest(f_classif, k=k_selected_features)
X_selected = f_classif_selector.fit_transform(X, y)
```

```
# Step 2: Split the data into training and testing sets
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X_selected, y, test_size=0.2, random_state=42)
```

```
# Step 3: Train the SVM classifier
from sklearn.svm import SVC
svm_classifier = SVC(kernel='linear')
svm_classifier.fit(X_train, y_train)
```

```
SVC(kernel='linear')
```

```
# Step 4: Evaluate the SVM classifier
from sklearn.metrics import accuracy_score, classification_report
y_pred = svm_classifier.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
report = classification_report(y_test, y_pred)
# Print results
print("Accuracy:", accuracy)
print("Classification Report:")
```

print(report)

Accuracy: 0.75
Classification Report:

```
              precision    recall  f1-score   support

           1       0.67      0.67      0.67         3
           2       1.00      1.00      1.00         2
           3       0.33      0.50      0.40         2
           4       1.00      0.80      0.89         5

    accuracy                           0.75        12
   macro avg       0.75      0.74      0.74        12
weighted avg       0.81      0.75      0.77        12
```

13. Write a python program to perform Anova test

import pandas as pd
 # Load your data into a pandas DataFrame
data = pd.read_csv('/content/drive/MyDrive/DATA SCIENCE LAB/fruit_data_with_colours.csv')
data.head(5)

	fruit_label	fruit_name	fruit_subtype	mass	width	height	color_score
0	1	apple	granny_smith	192	8.4	7.3	0.55
1	1	apple	granny_smith	180	8.0	6.8	0.59
2	1	apple	granny_smith	176	7.4	7.2	0.60
3	2	mandarin	mandarin	86	6.2	4.7	0.80
4	2	mandarin	mandarin	84	6.0	4.6	0.79

```
# Separate the target variable from features
dff = data.drop('fruit_label', axis=1)
df = dff.drop('fruit_subtype', axis=1)
X = df.drop('fruit_name', axis=1)
y = data['fruit_label']
```

```
from sklearn.feature_selection import SelectKBest,f_regression
# Step 1: Feature selection using chi-square test
k_selected_features = 4  # Adjust this value based on how many top features you want to select
anova_selector = SelectKBest(f_regression, k=k_selected_features)
X_selected = anova_selector.fit_transform(X, y)
```

```
# Step 2: Split the data into training and testing sets
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X_selected, y, test_size=0.2, random_state=42)
```

```
# Step 3: Train the SVM classifier
```

```
from sklearn.svm import SVC
svm_classifier = SVC(kernel='linear')
svm_classifier.fit(X_train, y_train)
```

```
SVC(kernel='linear')
```

```
# Step 4: Evaluate the SVM classifier
from sklearn.metrics import accuracy_score, classification_report
y_pred = svm_classifier.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
report = classification_report(y_test, y_pred)
# Print results
print("Accuracy:", accuracy)
print("Classification Report:")
print(report)
```

```
Accuracy: 0.75
Classification Report:
              precision    recall  f1-score   support

           1       0.67      0.67      0.67         3
           2       1.00      1.00      1.00         2
           3       0.33      0.50      0.40         2
           4       1.00      0.80      0.89         5

    accuracy                           0.75        12
   macro avg       0.75      0.74      0.74        12
weighted avg       0.81      0.75      0.77        12
```