Name-Sahil Chhabra

College-Indian Institute of Technology

BHU,Varanasi

Year-2nd

Branch-Metallurgy

Classifying the object as Mine or Rock

Importing necessary Libraries

In [24]:

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

Reading the Data

In [25]:

```
sonar=pd.read_csv('/content/sonar data.csv',header=None)
```

In [26]:

```
sonar.head()
```

Out[26]:

	0	1	2	3	4	5	6	7	8	9	...	51	
0	0.0200	0.0371	0.0428	0.0207	0.0954	0.0986	0.1539	0.1601	0.3109	0.2111	...	0.0027	0.
1	0.0453	0.0523	0.0843	0.0689	0.1183	0.2583	0.2156	0.3481	0.3337	0.2872	...	0.0084	0.
2	0.0262	0.0582	0.1099	0.1083	0.0974	0.2280	0.2431	0.3771	0.5598	0.6194	...	0.0232	0.
3	0.0100	0.0171	0.0623	0.0205	0.0205	0.0368	0.1098	0.1276	0.0598	0.1264	...	0.0121	0.
4	0.0762	0.0666	0.0481	0.0394	0.0590	0.0649	0.1209	0.2467	0.3564	0.4459	...	0.0031	0.

5 rows × 61 columns

In [27]:

```
sonar.shape
```

Out[27]:

```
(208, 61)
```

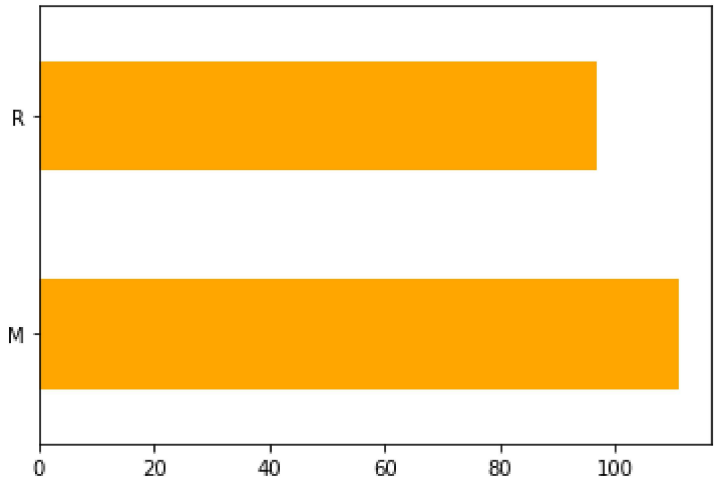
The table has 60 columns and last column has the output whether it is Rock('R') or Mine('M')

In [28]:

```
sonar[60].value_counts().plot(kind='barh',color='orange')
```

Out[28]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f968cdd61d0>
```



In [29]:

```
data=sonar.drop(60,axis=1)
```

In [30]:

```
data.head()
```

Out[30]:

	0	1	2	3	4	5	6	7	8	9	...	50	
0	0.0200	0.0371	0.0428	0.0207	0.0954	0.0986	0.1539	0.1601	0.3109	0.2111	...	0.0232	0.
1	0.0453	0.0523	0.0843	0.0689	0.1183	0.2583	0.2156	0.3481	0.3337	0.2872	...	0.0125	0.
2	0.0262	0.0582	0.1099	0.1083	0.0974	0.2280	0.2431	0.3771	0.5598	0.6194	...	0.0033	0.
3	0.0100	0.0171	0.0623	0.0205	0.0205	0.0368	0.1098	0.1276	0.0598	0.1264	...	0.0241	0.
4	0.0762	0.0666	0.0481	0.0394	0.0590	0.0649	0.1209	0.2467	0.3564	0.4459	...	0.0156	0.

5 rows × 60 columns

In [31]:

```
target=pd.get_dummies(sonar[60])
```

In [32]:

```
target
```

Out[32]:

	M	R
0	0	1
1	0	1
2	0	1
3	0	1
4	0	1
...	...	...
203	1	0
204	1	0
205	1	0
206	1	0
207	1	0

208 rows × 2 columns

In [33]:

```
target_r=target['R']
target_m=target['M']
```

In [34]:

```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(data,target_m,test_size=.30,random_state=43)
```

In [35]:

```
from sklearn.preprocessing import PolynomialFeatures
from sklearn.pipeline import Pipeline
```

In [36]:

```
poly=PolynomialFeatures(2)
```

In [37]:

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier,AdaBoostClassifier
from sklearn.gaussian_process import GaussianProcessClassifier
from sklearn.gaussian_process.kernels import RBF
from sklearn.linear_model import SGDClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.naive_bayes import GaussianNB
```

In [38]:

```
classifiers=[(
    'AdaBoost',AdaBoostClassifier()
),('Decision Tree',DecisionTreeClassifier(max_depth=10)),
('Naive Bayes',GaussianNB()),('KNeighbors',KNeighborsClassifier(3)),("SGDClassifier", SGDCl
,("RBF SVM",SVC(gamma=2, C=1,probability=True)),("Random Forest",RandomForestClassifier(n_j
```

In [20]:

```
#clf_n.append('Gaussian Process')
```

In [21]:

```
#clf=Pipeline([('Poly',poly),('CLF', GaussianProcessClassifier(1.0 * RBF(1.0)))])
```

In [22]:

```

```

```
Training Done
```

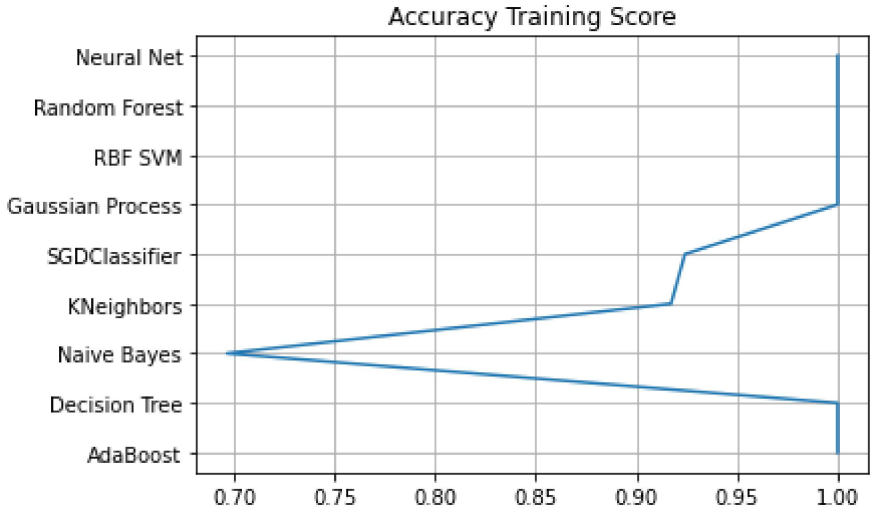
In [39]:

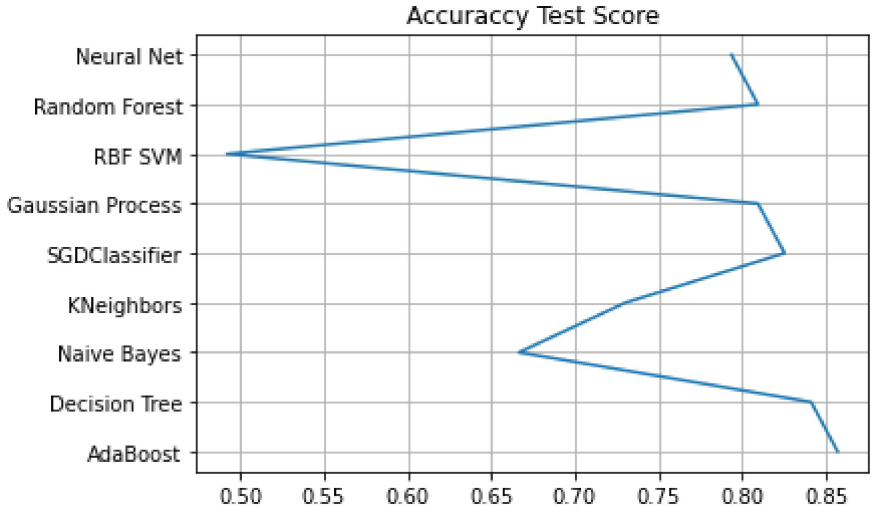
```
clf_n=[]
train_scores=[]
test_scores=[]
for i,clf in classifiers:
  clf_n.append(i)
  clf=Pipeline([('Poly',poly),('CLF',clf)])
  clf.fit(X_train,y_train)
  print(i+' training done')
  train_scores.append(clf.score(X_train,y_train))
  print(i+' training score done')
  test_scores.append(clf.score(X_test,y_test))
  print(i+" testing score done")
```

```
AdaBoost training done
AdaBoost training score done
AdaBoost testing score done
Decision Tree training done
Decision Tree training score done
Decision Tree testing score done
Naive Bayes training done
Naive Bayes training score done
Naive Bayes testing score done
KNeighbors training done
KNeighbors training score done
KNeighbors testing score done
SGDClassifier training done
SGDClassifier training score done
SGDClassifier testing score done
Gaussian Process training done
Gaussian Process training score done
Gaussian Process testing score done
RBF SVM training done
RBF SVM training score done
RBF SVM testing score done
Random Forest training done
Random Forest training score done
Random Forest testing score done
Neural Net training done
Neural Net training score done
Neural Net testing score done

/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multilayer_pe
rceptron.py:696: ConvergenceWarning: Stochastic Optimizer: Maximum iteration
s (200) reached and the optimization hasn't converged yet.
  ConvergenceWarning,
```

In [40]:

```
plt.title('Accuracy Training Score')
plt.grid()
plt.plot(train_scores,clf_n)
plt.show()

plt.title('Accuraccy Test Score')
plt.grid()
plt.plot(test_scores,clf_n)
plt.show()
```





In [57]:

```
rng=np.random.RandomState(1)
clf = GaussianProcessClassifier(1.0 * RBF(1.0))
clf = Pipeline([('POLY', poly),
                ('Gausian Process', clf)])
%time clf.fit(X_train, y_train)
```

```
CPU times: user 512 ms, sys: 445 ms, total: 957 ms
Wall time: 497 ms
```

Out[57]:

```
Pipeline(steps=[('POLY', PolynomialFeatures()),
                ('Gausian Process',
                 GaussianProcessClassifier(kernel=1**2 * RBF(length_scale=
1)))])
```

In [42]:

```
clf.score(X_train, y_train)
```

Out[42]:

```
1.0
```

In [43]:

```
clf.score(X_test, y_test)
```

Out[43]:

```
0.8095238095238095
```

In [44]:

```
pred=clf.predict(X_test)
```

In [45]:

```
clf.predict(X_test).sum()
```

Out[45]:

```
38
```

In [46]:

```
y_test.sum()
```

Out[46]:

```
30
```

In [47]:

```
from sklearn.metrics import accuracy_score
```

In [48]:

```
accuracy=accuracy_score(pred,y_test)
```

In [49]:

```
accuracy
```

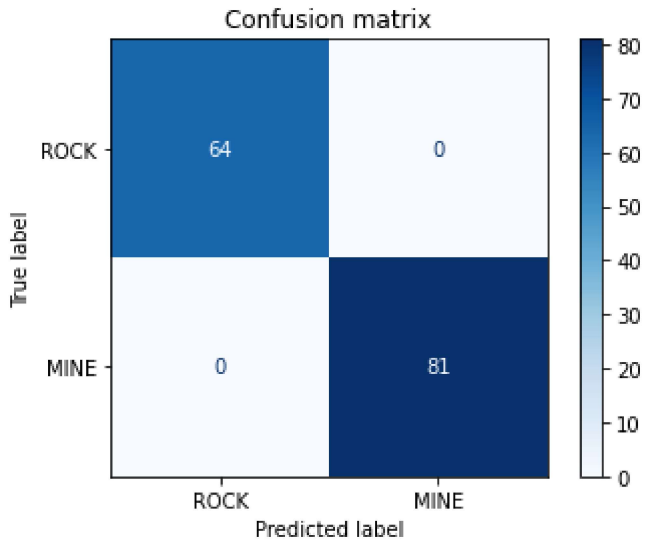
Out[49]:

```
0.8095238095238095
```

In [50]:

```
from sklearn.metrics import plot_confusion_matrix

disp = plot_confusion_matrix(clf, X_train, y_train,
                             display_labels=['ROCK','MINE'],
                             cmap=plt.cm.Blues,
                             normalize=None)
disp.ax_.set_title('Confusion matrix')

print('Train results: confusion matrix')
print(disp.confusion_matrix)
```

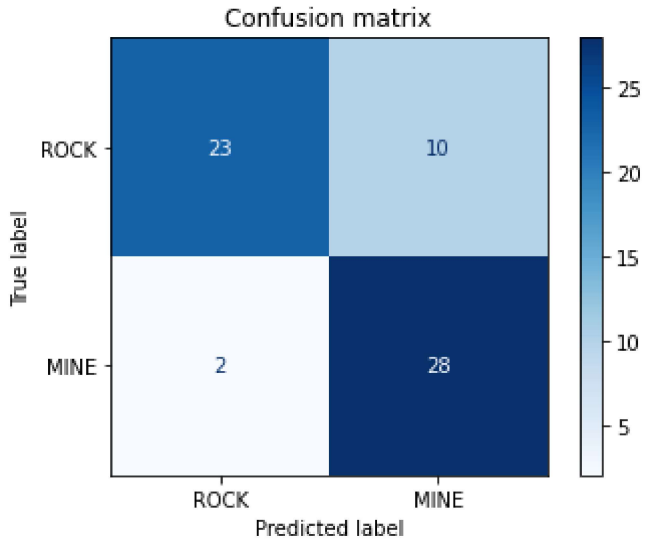
```
/usr/local/lib/python3.7/dist-packages/sklearn/utils/deprecation.py:87: Futu
reWarning: Function plot_confusion_matrix is deprecated; Function `plot_conf
usion_matrix` is deprecated in 1.0 and will be removed in 1.2. Use one of th
e class methods: ConfusionMatrixDisplay.from_predictions or ConfusionMatrixD
isplay.from_estimator.
  warnings.warn(msg, category=FutureWarning)

Train results: confusion matrix
[[64  0]
 [ 0 81]]
```



In [51]:

```
disp = plot_confusion_matrix(clf, X_test, y_test,
                             display_labels=['ROCK','MINE'],
                             cmap=plt.cm.Blues,
                             normalize=None)
disp.ax_.set_title('Confusion matrix')

print('Test results: confusion matrix')
print(disp.confusion_matrix)
```

```
Test results: confusion matrix
[[23 10]
 [ 2 28]]

/usr/local/lib/python3.7/dist-packages/sklearn/utils/deprecation.py:87: Futu
reWarning: Function plot_confusion_matrix is deprecated; Function `plot_conf
usion_matrix` is deprecated in 1.0 and will be removed in 1.2. Use one of th
e class methods: ConfusionMatrixDisplay.from_predictions or ConfusionMatrixD
isplay.from_estimator.
  warnings.warn(msg, category=FutureWarning)
```



#Using AdaBoostClassifier

In [54]:

```
clf2 = AdaBoostClassifier(n_estimators=1000,random_state=43)
clf2 = Pipeline([('POLY', poly),
                 ('ADABOOST', clf2)])
%time clf2.fit(X_train, y_train)
```

```
CPU times: user 31.7 s, sys: 155 ms, total: 31.8 s
Wall time: 32.6 s
```

Out[54]:

```
Pipeline(steps=[('POLY', PolynomialFeatures()),
                ('ADABOOST',
                 AdaBoostClassifier(n_estimators=1000, random_state=43))])
```

In [55]:

```
clf2.score(X_train, y_train)
```

Out[55]:

```
1.0
```

AdaBoostClassifier is performing better than others

In [56]:

```
clf2.score(X_test,y_test)
```

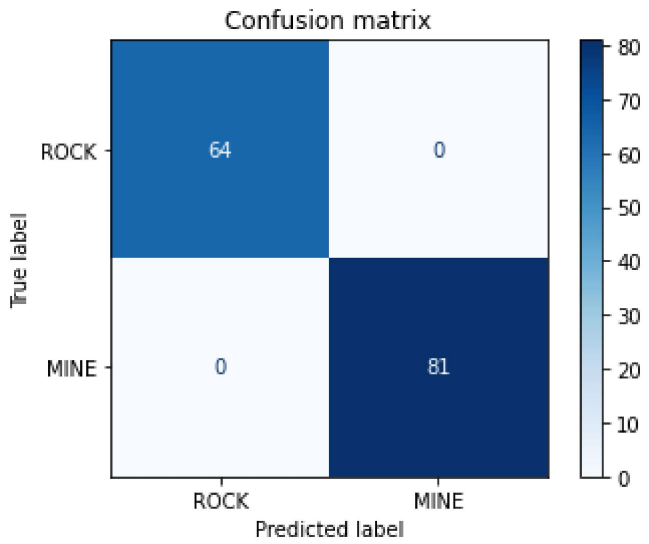
Out[56]:

```
0.8253968253968254
```

In [58]:

```
disp = plot_confusion_matrix(clf2, X_train, y_train,
                             display_labels=['ROCK','MINE'],
                             cmap=plt.cm.Blues,
                             normalize=None)
disp.ax_.set_title('Confusion matrix')

print('Train results: confusion matrix')
print(disp.confusion_matrix)
```

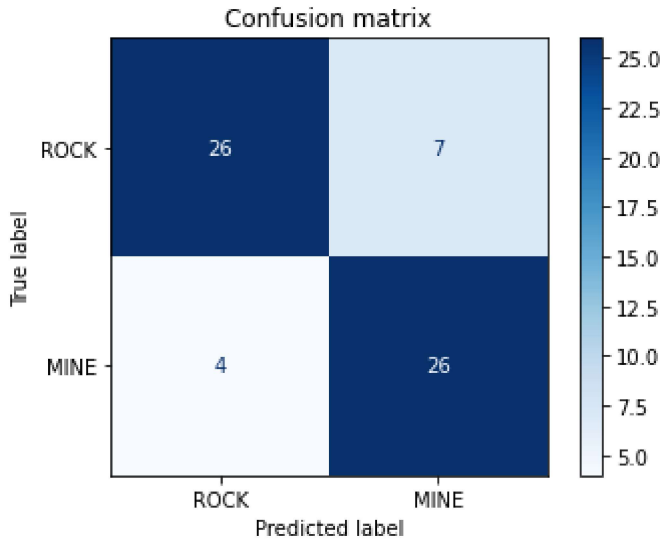
```
/usr/local/lib/python3.7/dist-packages/sklearn/utils/deprecation.py:87: Futu
reWarning: Function plot_confusion_matrix is deprecated; Function `plot_conf
usion_matrix` is deprecated in 1.0 and will be removed in 1.2. Use one of th
e class methods: ConfusionMatrixDisplay.from_predictions or ConfusionMatrixD
isplay.from_estimator.
  warnings.warn(msg, category=FutureWarning)

Train results: confusion matrix
[[64  0]
 [ 0 81]]
```



In [59]:

```
disp = plot_confusion_matrix(clf2, X_test, y_test,
                             display_labels=['ROCK','MINE'],
                             cmap=plt.cm.Blues,
                             normalize=None)
disp.ax_.set_title('Confusion matrix')

print('Train results: confusion matrix')
print(disp.confusion_matrix)
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/utils/deprecation.py:87: Futu
reWarning: Function plot_confusion_matrix is deprecated; Function `plot_conf
usion_matrix` is deprecated in 1.0 and will be removed in 1.2. Use one of th
e class methods: ConfusionMatrixDisplay.from_predictions or ConfusionMatrixD
isplay.from_estimator.
  warnings.warn(msg, category=FutureWarning)


Train results: confusion matrix
[[26  7]
 [ 4 26]]
```



In [ ]:

Customer Segmentation using K-means Clustering

Importing required Libraries

In [1]:

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

Getting the Data

In [2]:

```
customer_data=pd.read_csv('/content/Mall_Customers.csv')
```

In [3]:

```
customer_data.head()
```

Out[3]:

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

In [4]:

```
customer_data.shape
```

Out[4]:

```
(200, 5)
```

In [5]:

```
customer_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200 entries, 0 to 199
Data columns (total 5 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   CustomerID              200 non-null    int64
 1   Gender                  200 non-null    object
 2   Age                     200 non-null    int64
 3   Annual Income (k$)      200 non-null    int64
 4   Spending Score (1-100)  200 non-null    int64
dtypes: int64(4), object(1)
memory usage: 7.9+ KB
```

Storing Annual_Income and Spending_Score in X

In [6]:

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

In [7]:

```
X
```

Out[7]:

```
array([[ 15,  39],
       [ 15,  81],
       [ 16,   6],
       [ 16,  77],
       [ 17,  40],
       [ 17,  76],
       [ 18,   6],
       [ 18,  94],
       [ 19,   3],
       [ 19,  72],
       [ 19,  14],
       [ 19,  99],
       [ 20,  15],
       [ 20,  77],
       [ 20,  13],
       [ 20,  79],
       [ 21,  35],
       [ 21,  66],
```

To choose the number of clusters we use WCSS-With Clusters Sum of Squares

In [8]:

```
from sklearn.cluster import KMeans
```

In [9]:

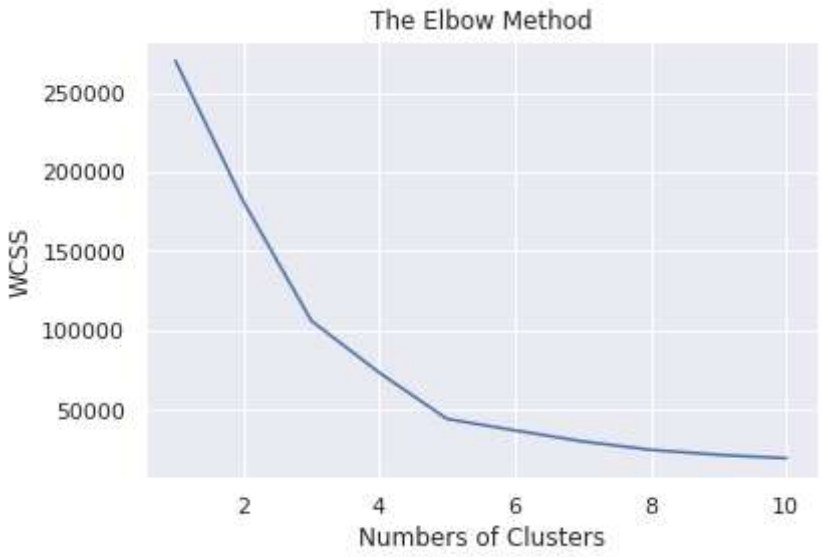
```
wcss=[]
for i in range(1,11):
  kmeans=KMeans(n_clusters=i,init='k-means++',random_state=42)
  kmeans.fit(X)
  wcss.append(kmeans.inertia_)
```

In [10]:

```
sns.set()
plt.plot(range(1,11),wcss)
plt.title('The Elbow Method')
plt.xlabel('Numbers of Clusters')
plt.ylabel('WCSS')
```

Out[10]:

```
Text(0, 0.5, 'WCSS')
```



The Optimum Number of Clusters=5

Training the K-Means Clustering model

In [11]:

```
kmeans=KMeans(n_clusters=5,init='k-means++',random_state=0)
y=kmeans.fit_predict(X)
```

In [12]:

```
y
```

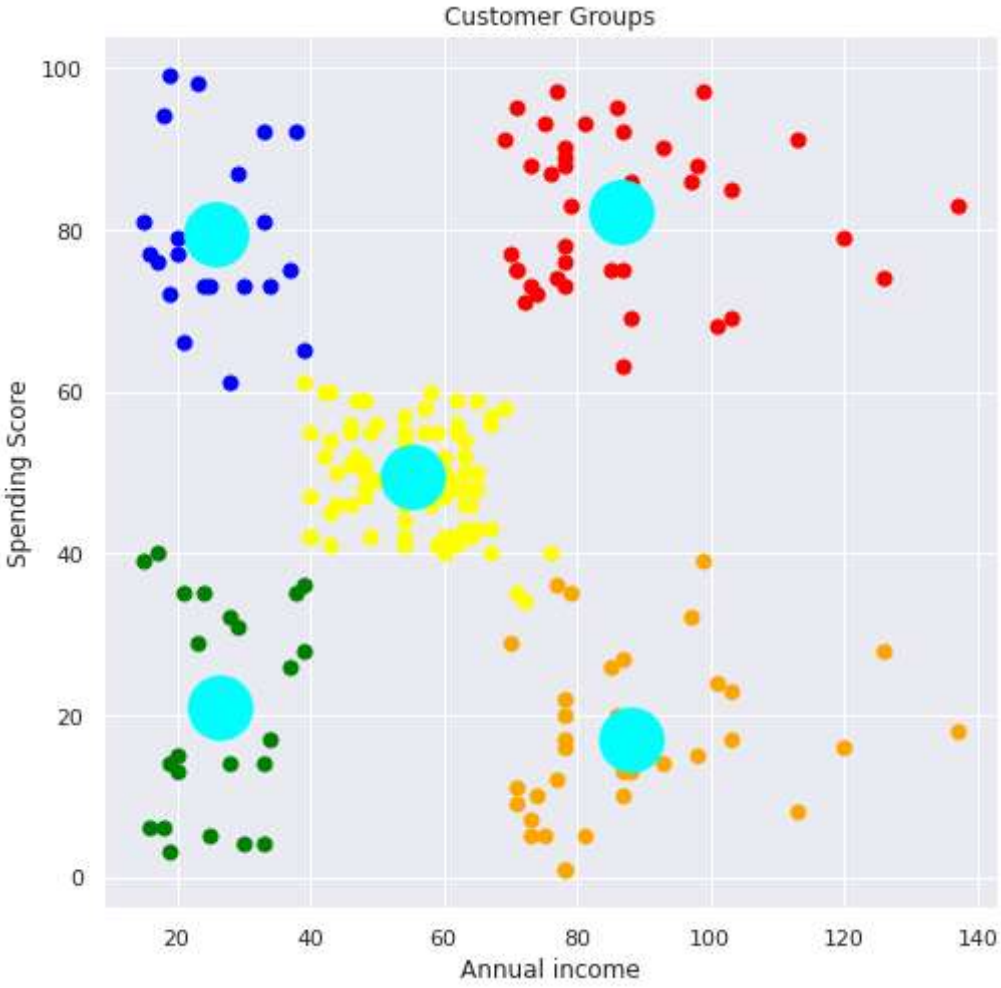
Out[12]:

```
array([4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3,
       4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 1,
       4, 3, 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, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 0, 2, 1, 2, 0, 2, 0, 2,
       1, 2, 0, 2, 0, 2, 0, 2, 0, 2, 1, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2,
       0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2,
       0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2,
       0, 2], dtype=int32)
```

Visualizing the Clusters

In [13]:

```
plt.figure(figsize=(8,8))
plt.scatter(X[y==0,0],X[y==0,1],s=50,c='Orange',label='Cluster1')
plt.scatter(X[y==1,0],X[y==1,1],s=50,c='Yellow',label='Cluster2')
plt.scatter(X[y==2,0],X[y==2,1],s=50,c='Red',label='Cluster3')
plt.scatter(X[y==3,0],X[y==3,1],s=50,c='Blue',label='Cluster4')
plt.scatter(X[y==4,0],X[y==4,1],s=50,c='Green',label='Cluster5')
plt.scatter(kmeans.cluster_centers_[:,0],kmeans.cluster_centers_[:,1],s=1000,c='cyan',label
plt.title('Customer Groups')
plt.xlabel('Annual income')
plt.ylabel('Spending Score')
plt.show()
```



#Conclusions- Customers with annual income less than 40K and greater than 70K are either spending with a score of 60+ or with 40- Hence the Caretakers of the Mall should sort this issue by providing some membership card to keep the customers engaged with the mall. Customers with annual income between 40K and 70K have a score between 40 and 60 which is good.

Clustering the Data based on Age and Spend-Score

In [14]:

```
Y=customer_data.iloc[:,[2,4]].values
```

In [15]:

```
Y
```

Out[15]:

```
array([[19, 39],
       [21, 81],
       [20,  6],
       [23, 77],
       [31, 40],
       [22, 76],
       [35,  6],
       [23, 94],
       [64,  3],
       [30, 72],
       [67, 14],
       [35, 99],
       [58, 15],
       [24, 77],
       [37, 13],
       [22, 79],
       [35, 35],
       [20, 66],
```

In [17]:

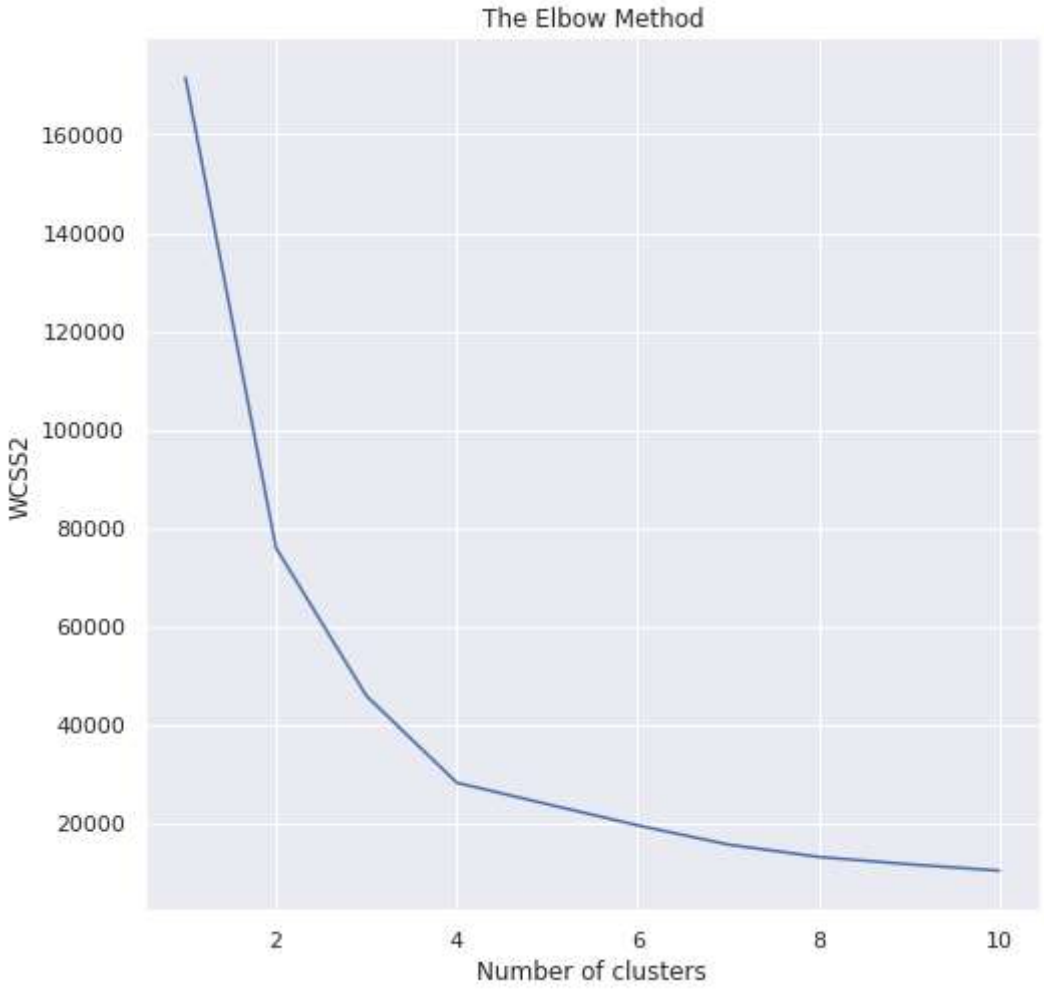
```
wcss2=[]
for i in range(1,11):
  kmeans=KMeans(n_clusters=i,init='k-means++',random_state=0)
  kmeans.fit(Y)
  wcss2.append(kmeans.inertia_)
```

In [18]:

```
plt.figure(figsize=(8,8))
plt.plot(range(1,11),wcss2)
plt.title('The Elbow Method')
plt.xlabel('Number of clusters')
plt.ylabel('WCSS2')
```

Out[18]:

```
Text(0, 0.5, 'WCSS2')
```

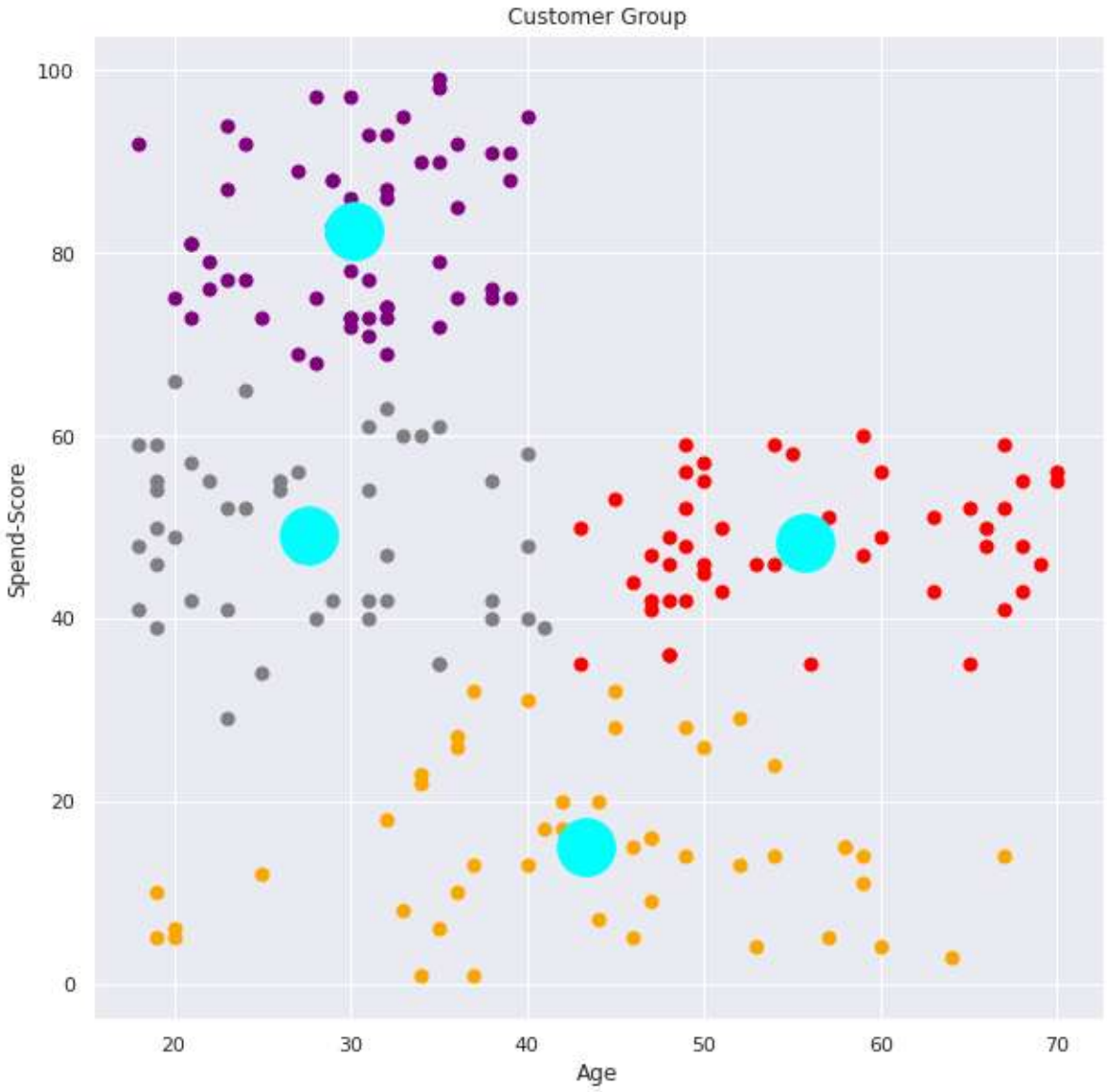


In [19]:

```
kmeans=KMeans(n_clusters=4,init='k-means++',random_state=0)
x=kmeans.fit_predict(Y)
```

In [23]:

```
plt.figure(figsize=(10,10))
plt.scatter(Y[x==0,0],Y[x==0,1],s=50,c='orange',label='Cluster1')
plt.scatter(Y[x==1,0],Y[x==1,1],s=50,c='purple',label='Cluster2')
plt.scatter(Y[x==2,0],Y[x==2,1],s=50,c='grey',label='Cluster3')
plt.scatter(Y[x==3,0],Y[x==3,1],s=50,c='red',label='Cluster4')
plt.scatter(kmeans.cluster_centers_[:,0],kmeans.cluster_centers_[:,1],s=1000,c='cyan',label
plt.title('Customer Group')
plt.xlabel('Age')
plt.ylabel('Spend-Score')
plt.show()
```



Customers with age greater than 60 have spending score between 40-60 hence aged people are more loyal towards our mall. Less than 40 years age of people are also spending at a score greater than 40. Only middle aged people have to targeted to increase their spend-score

In [ ]:

Github Account link:

https://github.com/ChhaSahil/Rinex_MP