In [1]:
```python
import pandas as pd
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt
from sklearn.metrics import classification_report
from sklearn.model_selection import train_test_split,StratifiedKFold,cross_val_scor
from sklearn.tree import plot_tree
import graphviz
from sklearn import tree
```

In [2]:
```python
#importing data
df = pd.read_csv('Admission_Predict.csv',sep=',')
df.columns
```

Out[2]:
```
Index(['Serial No.', 'GRE Score', 'TOEFL Score', 'University Rating', 'SOP',
       'LOR ', 'CGPA', 'Research', 'Chance of Admit '],
      dtype='object')
```

In [3]:
```python
df.columns = df.columns.str.rstrip()
df.columns
```

Out[3]:
```
Index(['Serial No.', 'GRE Score', 'TOEFL Score', 'University Rating', 'SOP',
       'LOR', 'CGPA', 'Research', 'Chance of Admit'],
      dtype='object')
```

In [4]:
```python
# replace values in in Chance of Admit column by 0 or 1. Set criteria to 80%
df.loc[df['Chance of Admit'] >=0.80,'Chance of Admit']=1
df.loc[df['Chance of Admit'] < 0.80,'Chance of Admit']=0
df
```

Out[4]:

	Serial No.	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research	Chance of Admit
0	1	337	118	4	4.5	4.5	9.65	1	1.0
1	2	324	107	4	4.0	4.5	8.87	1	0.0
2	3	316	104	3	3.0	3.5	8.00	1	0.0
3	4	322	110	3	3.5	2.5	8.67	1	1.0
4	5	314	103	2	2.0	3.0	8.21	0	0.0
...	...	...	...	...	...	...	...	...	...
395	396	324	110	3	3.5	3.5	9.04	1	1.0
396	397	325	107	3	3.0	3.5	9.11	1	1.0
397	398	330	116	4	5.0	4.5	9.45	1	1.0
398	399	312	103	3	3.5	4.0	8.78	0	0.0
399	400	333	117	4	5.0	4.0	9.66	1	1.0

400 rows × 9 columns

In [5]:
```python
df=df.drop('Serial No.',axis=1)
df
```

Out[5]:

	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research	Chance of Admit
0	337	118	4	4.5	4.5	9.65	1	1.0
1	324	107	4	4.0	4.5	8.87	1	0.0
2	316	104	3	3.0	3.5	8.00	1	0.0
3	322	110	3	3.5	2.5	8.67	1	1.0
4	314	103	2	2.0	3.0	8.21	0	0.0
...	...	...	...	...	...	...	...	...
395	324	110	3	3.5	3.5	9.04	1	1.0
396	325	107	3	3.0	3.5	9.11	1	1.0
397	330	116	4	5.0	4.5	9.45	1	1.0
398	312	103	3	3.5	4.0	8.78	0	0.0
399	333	117	4	5.0	4.0	9.66	1	1.0

400 rows × 8 columns

In [6]:
```
X = df.iloc[:,0:7].values
y = df.iloc[:,7].values
y
```

Out[6]:
```
array([1., 0., 0., 1., 0., 1., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 1., 1., 1., 1., 0., 0., 0., 0., 0., 0., 1., 1.,
       1., 1., 0., 0., 0., 0., 0., 0., 0., 1., 1., 1., 1., 1., 1., 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., 0., 0., 0., 0., 0., 0., 0., 1., 1., 1., 1.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 1., 0., 0., 0.,
       0., 0., 0., 0., 1., 1., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 1., 1., 0., 0., 0., 0., 1., 0., 1., 1., 1., 0., 0., 0., 1., 1.,
       0., 0., 1., 0., 1., 1., 1., 1., 1., 1., 0., 1., 1., 0., 1., 1., 1.,
       0., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0.,
       0., 1., 1., 1., 1., 1., 1., 1., 0., 0., 0., 0., 0., 0., 0., 1., 1.,
       1., 1., 1., 1., 1., 1., 1., 0., 0., 0., 0., 0., 0., 0., 0., 1., 1.,
       0., 0., 0., 0., 0., 0., 0., 1., 1., 1., 1., 1., 1., 1., 1., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 1., 1., 1., 1.,
       0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 1., 0., 0., 0., 0., 1., 1.,
       0., 0., 0., 0., 1., 1., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0.,
       0., 0., 0., 0., 1., 0., 0., 0., 0., 1., 1., 1., 1., 1., 1., 1., 1.,
       0., 0., 0., 0., 0., 0., 0., 0., 1., 1., 0., 0., 0., 0., 0., 0., 0.,
       0., 1., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0.,
       0., 0., 1., 0., 0., 1., 0., 1., 0., 0., 0., 0., 1., 0., 1., 1., 1.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 1., 1., 1., 1., 0., 0., 1., 0., 0., 0., 0., 0., 1., 1., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 1., 1., 0., 0., 0., 0., 0.,
       0., 1., 0., 1., 1., 1., 1., 0., 1.])
```

In [7]:
```
X_train,X_test,y_train,y_test= train_test_split(X,y,test_size=0.25,random_state=0)
```

In [8]:
```
model = DecisionTreeClassifier(criterion='entropy',max_depth=2)
model.fit(X_train,y_train)
y_pred=model.predict(X_test)
```

In [9]:
```
matrix=confusion_matrix(y_test,y_pred,labels=[0.0,1.0])
matrix
```

Out[9]:
```
array([[65,  6],
       [ 2, 27]], dtype=int64)
```

In [10]:
```
acc = accuracy_score(y_test,y_pred)
print('Accuracy of Decision Tree model:  ',acc)
```

```
Accuracy of Decision Tree model:   0.92
```

In [11]:
```
print(classification_report(y_test, y_pred))
```

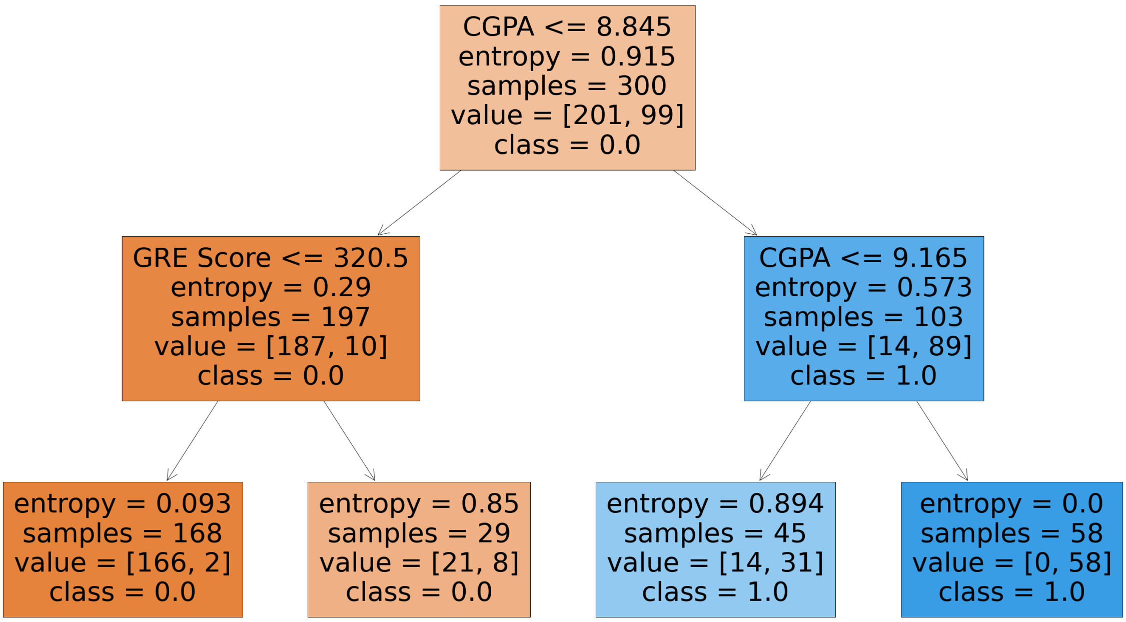
```
              precision    recall  f1-score   support

         0.0       0.97      0.92      0.94        71
         1.0       0.82      0.93      0.87        29

    accuracy                           0.92       100
   macro avg       0.89      0.92      0.91       100
weighted avg       0.93      0.92      0.92       100
```

In [12]:
```
feature_names=df.columns[0:7]
print(feature_names,end=' ')
class_names=[str(x) for x in model.classes_]
class_names
```

```
Index(['GRE Score', 'TOEFL Score', 'University Rating', 'SOP', 'LOR', 'CGPA',
       'Research'],
      dtype='object')
```

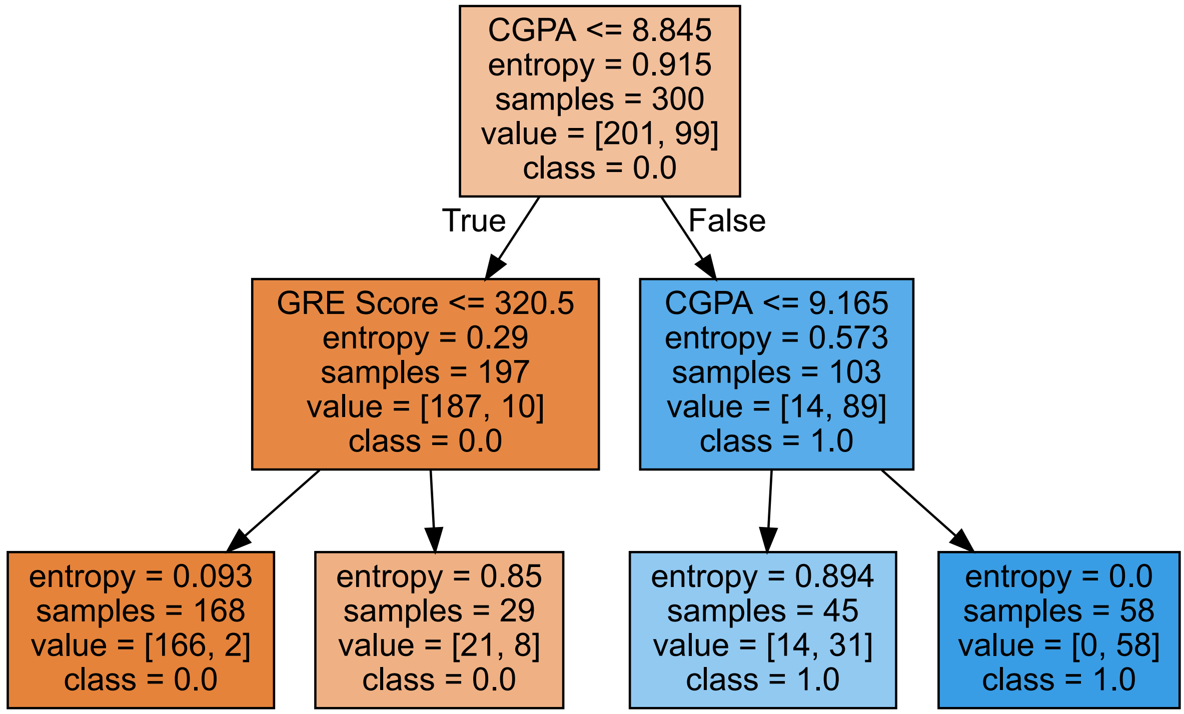
Out[12]:
```
['0.0', '1.0']
```

In [13]:
```
fig=plt.figure(figsize=(50,30))
plot_tree(model,feature_names=feature_names,class_names=class_names,filled=True)
plt.savefig('tree_visualization.png')
```



In [14]:
```
dot_data = tree.export_graphviz(model,out_file=None, feature_names=feature_names,
graph=graphviz.Source(dot_data,format="png")
graph
```

Out[14]:



In [15]:

```
sf = StratifiedKFold(n_splits=5,shuffle=True,random_state=0)
depth=[1,2,3,4,5,6,7,8,9,10]

for d in depth :
    score = cross_val_score(tree.DecisionTreeClassifier(criterion='entropy',max_de
    print("Average score for depth {}: {}".format(d,score.mean()))
```

```
Average score for depth 1: 0.9199999999999999
Average score for depth 2: 0.9199999999999999
Average score for depth 3: 0.9233333333333332
Average score for depth 4: 0.9033333333333333
Average score for depth 5: 0.8833333333333334
Average score for depth 6: 0.9
Average score for depth 7: 0.89
Average score for depth 8: 0.8866666666666667
Average score for depth 9: 0.9
Average score for depth 10: 0.9033333333333333
```

In [16]:

```
print("Average cross validation score:",score.mean())
```

```
Average cross validation score: 0.9033333333333333
```

In [ ]:

In [1]:

```python
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
import matplotlib.cm as cm
from sklearn.metrics import silhouette_samples, silhouette_score
import numpy as np
import scipy.cluster.hierarchy as sch
from sklearn.cluster import AgglomerativeClustering
```

In [2]:

```python
df = pd.read_csv('mall.csv')
```

In [3]:

```python
df.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   Genre                   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
```

In [4]:

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

```
Out[4]: 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],
       [ 23,  29],
       [ 23,  98],
       [ 24,  35],
       [ 24,  73],
       [ 25,   5],
       [ 25,  73],
       [ 28,  14],
       [ 28,  82],
       [ 28,  32],
       [ 28,  61],
       [ 29,  31],
       [ 29,  87],
       [ 30,   4],
       [ 30,  73],
       [ 33,   4],
       [ 33,  92],
       [ 33,  14],
       [ 33,  81],
       [ 34,  17],
       [ 34,  73],
       [ 37,  26],
       [ 37,  75],
       [ 38,  35],
       [ 38,  92],
       [ 39,  36],
       [ 39,  61],
       [ 39,  28],
       [ 39,  65],
       [ 40,  55],
       [ 40,  47],
       [ 40,  42],
       [ 40,  42],
       [ 42,  52],
       [ 42,  60],
       [ 43,  54],
       [ 43,  60],
       [ 43,  45],
       [ 43,  41],
       [ 44,  50],
       [ 44,  46],
       [ 46,  51],
       [ 46,  46],
       [ 46,  56],
       [ 46,  55],
       [ 47,  52],
       [ 47,  59],
```

```
       [113,   8],
       [113,  91],
       [120,  16],
       [120,  79],
       [126,  28],
       [126,  74],
       [137,  18],
       [137,  83]], dtype=int64)
```

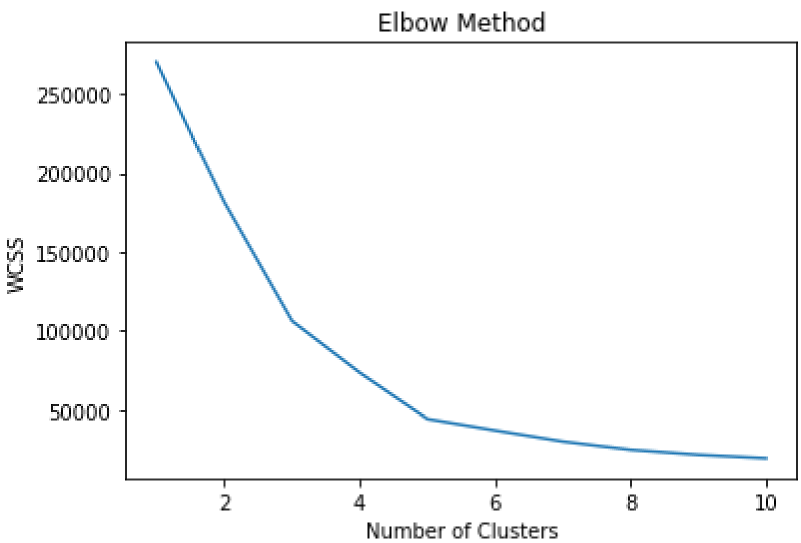
In [5]:

```
wcss=[]

for i in range(1,11):
    kmeans=KMeans(n_clusters=i, init='k-means++',max_iter=300,random_state=42)
    kmeans.fit(X)
    wcss.append(kmeans.inertia_)

plt.plot(range(1,11),wcss)
plt.title("Elbow Method")
plt.xlabel('Number of Clusters')
plt.ylabel('WCSS')
```

```
C:\Users\DELL\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1036: UserWar
ning: KMeans is known to have a memory leak on Windows with MKL, when there are le
ss chunks than available threads. You can avoid it by setting the environment vari
able OMP_NUM_THREADS=1.
  warnings.warn(
```

Out[5]:

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



In [6]:

```
kmeans=KMeans(n_clusters=5,init='k-means++',max_iter=300,random_state=42)
y_kmeans=kmeans.fit_predict(X)
y_kmeans
```

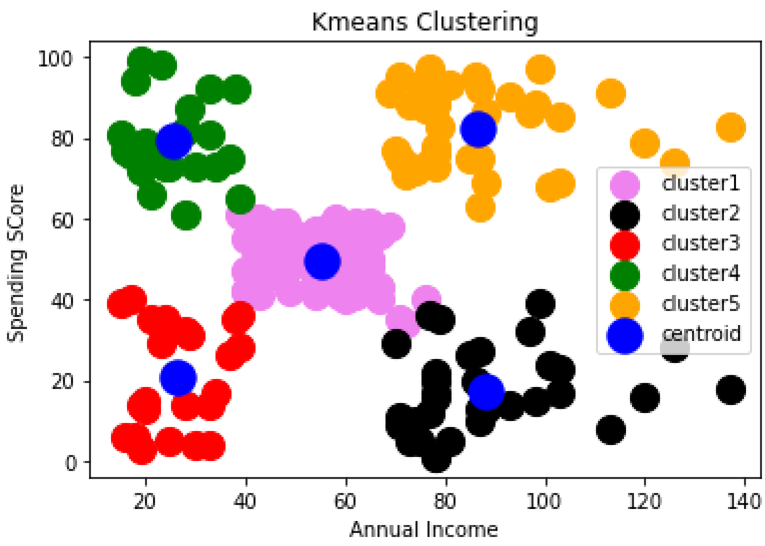
Out[6]:

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

In [13]:

```
plt.scatter(X[y_kmeans==0,0],X[y_kmeans==0,1],s=200,c='violet',label='cluster1')
```

```
plt.scatter(X[y_kmeans==1,0],X[y_kmeans==1,1],s=200,c='black',label='cluster2')
plt.scatter(X[y_kmeans==2,0],X[y_kmeans==2,1],s=200,c='red',label='cluster3')
plt.scatter(X[y_kmeans==3,0],X[y_kmeans==3,1],s=200,c='green',label='cluster4')
plt.scatter(X[y_kmeans==4,0],X[y_kmeans==4,1],s=200,c='orange',label='cluster5')
plt.scatter(kmeans.cluster_centers_[:,0],kmeans.cluster_centers_[:,1],s=300,c='blu
plt.title("Kmeans Clustering")
plt.xlabel('Annual Income')
plt.ylabel('Spending SCore')
plt.legend()
plt.show()
```



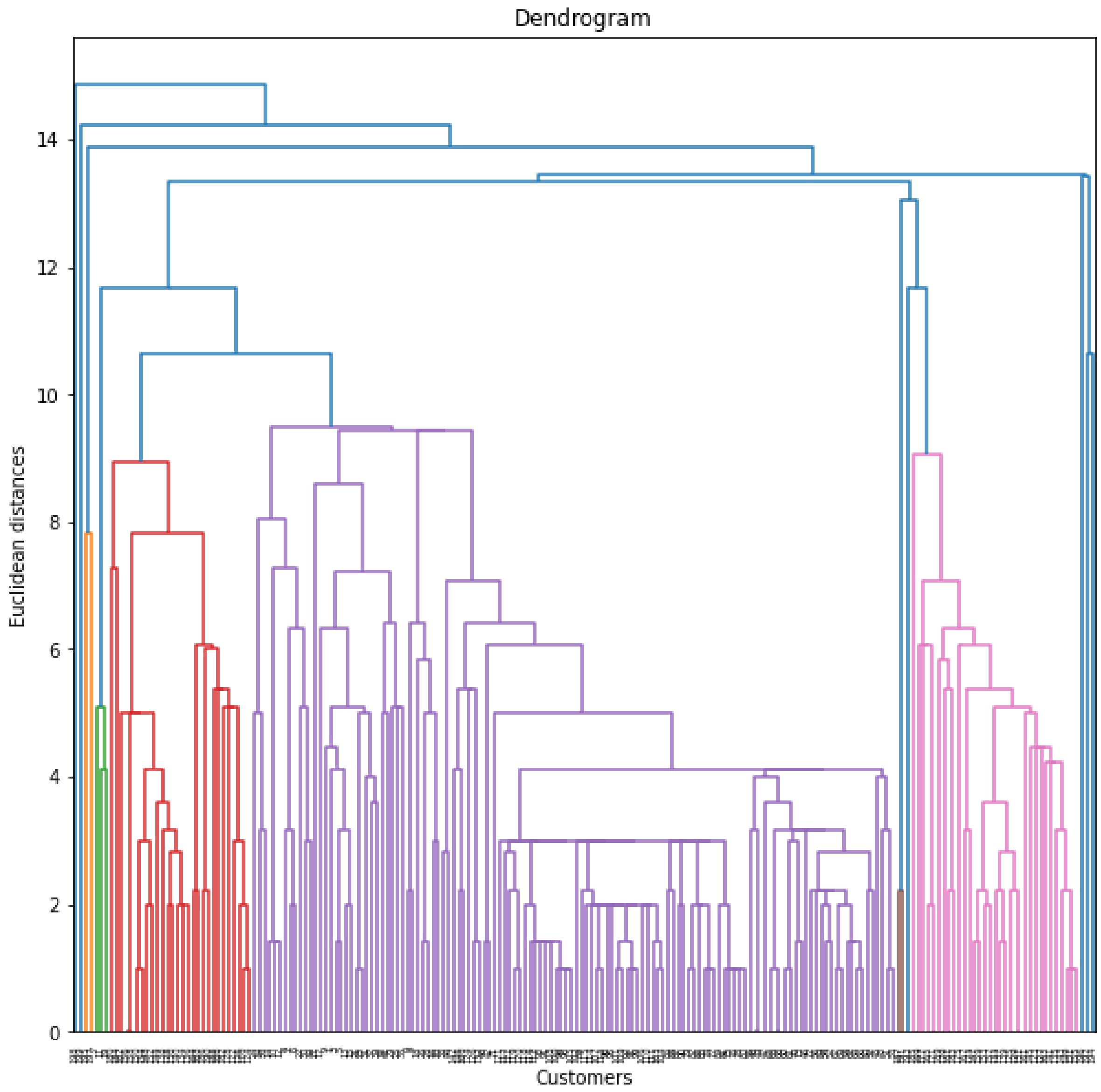
In [8]:

```
range_n_clusters = [2, 3, 4, 5, 6]

for n_clusters in range_n_clusters:
    clusterer = KMeans(n_clusters=n_clusters, random_state=10)
    cluster_labels = clusterer.fit_predict(X)

    silhouette_avg = silhouette_score(X, cluster_labels)
    print("For n_clusters =",n_clusters,"The average silhouette_score is :",silhou
```

```
For n_clusters = 2 The average silhouette_score is : 0.2968969162503008
For n_clusters = 3 The average silhouette_score is : 0.46761358158775435
For n_clusters = 4 The average silhouette_score is : 0.4931963109249047
For n_clusters = 5 The average silhouette_score is : 0.553931997444648
For n_clusters = 6 The average silhouette_score is : 0.5376203956398481
```

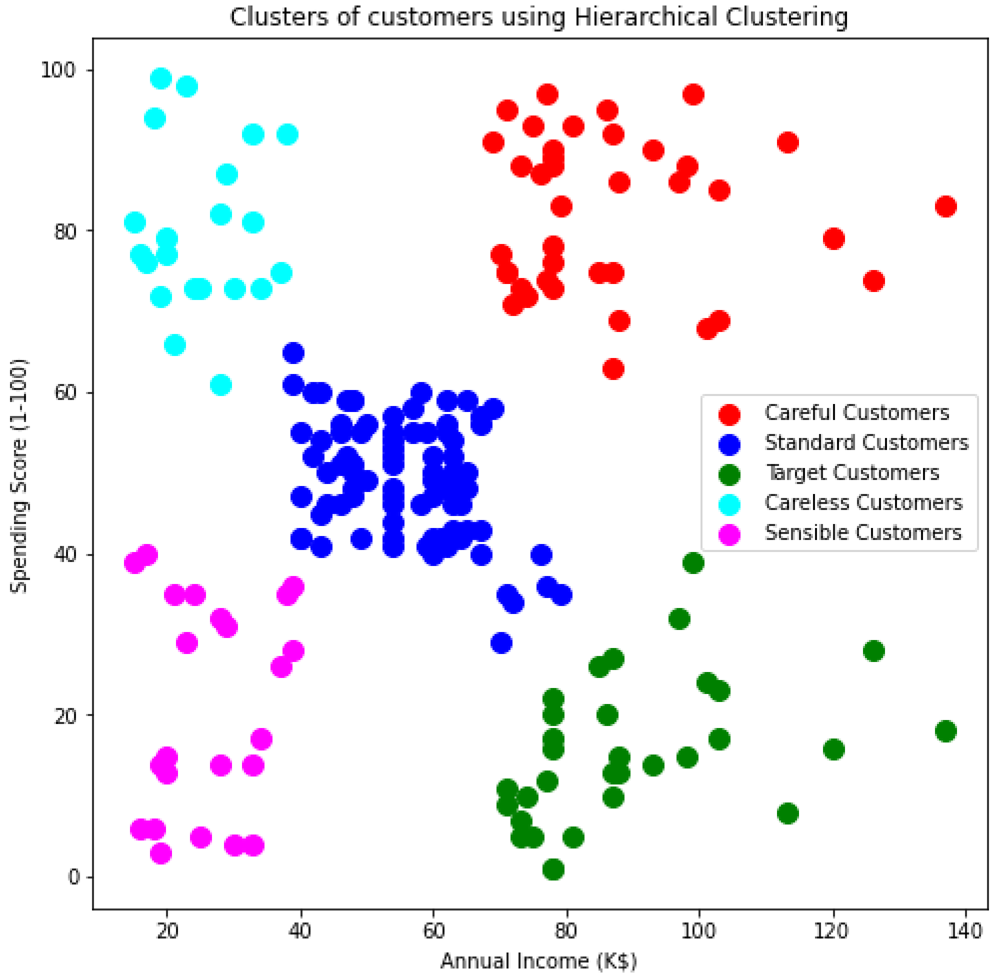
In [14]:

```
plt.figure(figsize=(10,10))
dendrogram = sch.dendrogram(sch.linkage(X, method = 'single'))
plt.title('Dendrogram')
plt.xlabel('Customers')
plt.ylabel('Euclidean distances')
plt.show()
```



In [10]:
```
hc = AgglomerativeClustering(n_clusters = 5, affinity = 'euclidean', linkage = 'co
y_hc = hc.fit_predict(X)
```

In [11]:
```
plt.figure(figsize=(8,8))
plt.scatter(X[y_hc == 0, 0], X[y_hc == 0, 1], s = 100, c = 'red', label = 'Careful
plt.scatter(X[y_hc == 1, 0], X[y_hc == 1, 1], s = 100, c = 'blue', label = 'Standar
plt.scatter(X[y_hc == 2, 0], X[y_hc == 2, 1], s = 100, c = 'green', label = 'Target
plt.scatter(X[y_hc == 3, 0], X[y_hc == 3, 1], s = 100, c = 'cyan', label = 'Careles
plt.scatter(X[y_hc == 4, 0], X[y_hc == 4, 1], s = 100, c = 'magenta', label = 'Sens
plt.title('Clusters of customers using Hierarchical Clustering')
plt.xlabel('Annual Income (K$)')
plt.ylabel('Spending Score (1-100)')
plt.legend()
plt.show()
```


Clusters of customers using Hierarchical Clustering
Spending Score (1-100)
Annual Income (K$)
0
20
40
60
80
100
20
40
60
80
100
120
140
Careful Customers
Standard Customers
Target Customers
Careless Customers
Sensible Customers

In [ ]: