Assignment 4 (ML-II)

Mall Customers Dataset (Example 1)

Wali Ullah (09745)

In [1]:
```
import warnings
warnings.filterwarnings('ignore')
warnings.simplefilter('ignore')
```

In [2]:
```
import pandas as pd
import matplotlib.pyplot as plt
from matplotlib.patches import Rectangle
import numpy as np
from pprint import pprint as pp
import csv
from pathlib import Path
import seaborn as sns
from itertools import product
import string
from sklearn.cluster import KMeans
from sklearn.cluster import OPTICS
import scipy.cluster.hierarchy as sch
from matplotlib import pyplot
import nltk
from nltk.corpus import stopwords
from nltk.stem.wordnet import WordNetLemmatizer

from imblearn.over_sampling import SMOTE
from imblearn.over_sampling import BorderlineSMOTE
from imblearn.pipeline import Pipeline

from sklearn.linear_model import LinearRegression, LogisticRegression
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import r2_score, classification_report, confusion_matrix, accuracy
from sklearn.metrics import homogeneity_score, silhouette_score
from sklearn.ensemble import RandomForestClassifier, VotingClassifier
from sklearn.preprocessing import MinMaxScaler
from sklearn.cluster import MiniBatchKMeans, DBSCAN

import gensim
from gensim import corpora
```

In [3]:
```
# Load Data
def load_data(file_name):
    def readcsv(file_name):
        return pd.read_csv(file_name)
    def readexcel(file_name):
        return pd.read_excel(file_name)
    func_map = {
        "csv": readcsv,
        "xlsx": readexcel,
    }
```

```
    # default reader = readcsv
    reader = func_map.get("csv")

    for k,v in func_map.items():
        if file_name.endswith(k):
            reader = v
            break
    return reader(file_name)
```

Data Set

In [4]:
```
FILE_NAME = "Mall_Customers.csv"
#FILE_NAME = "banksim_adj.csv"
#LABEL_COL = "fraud"
df = load_data(FILE_NAME)
display(df.head())
print(df.shape)
print(df.dtypes)
```

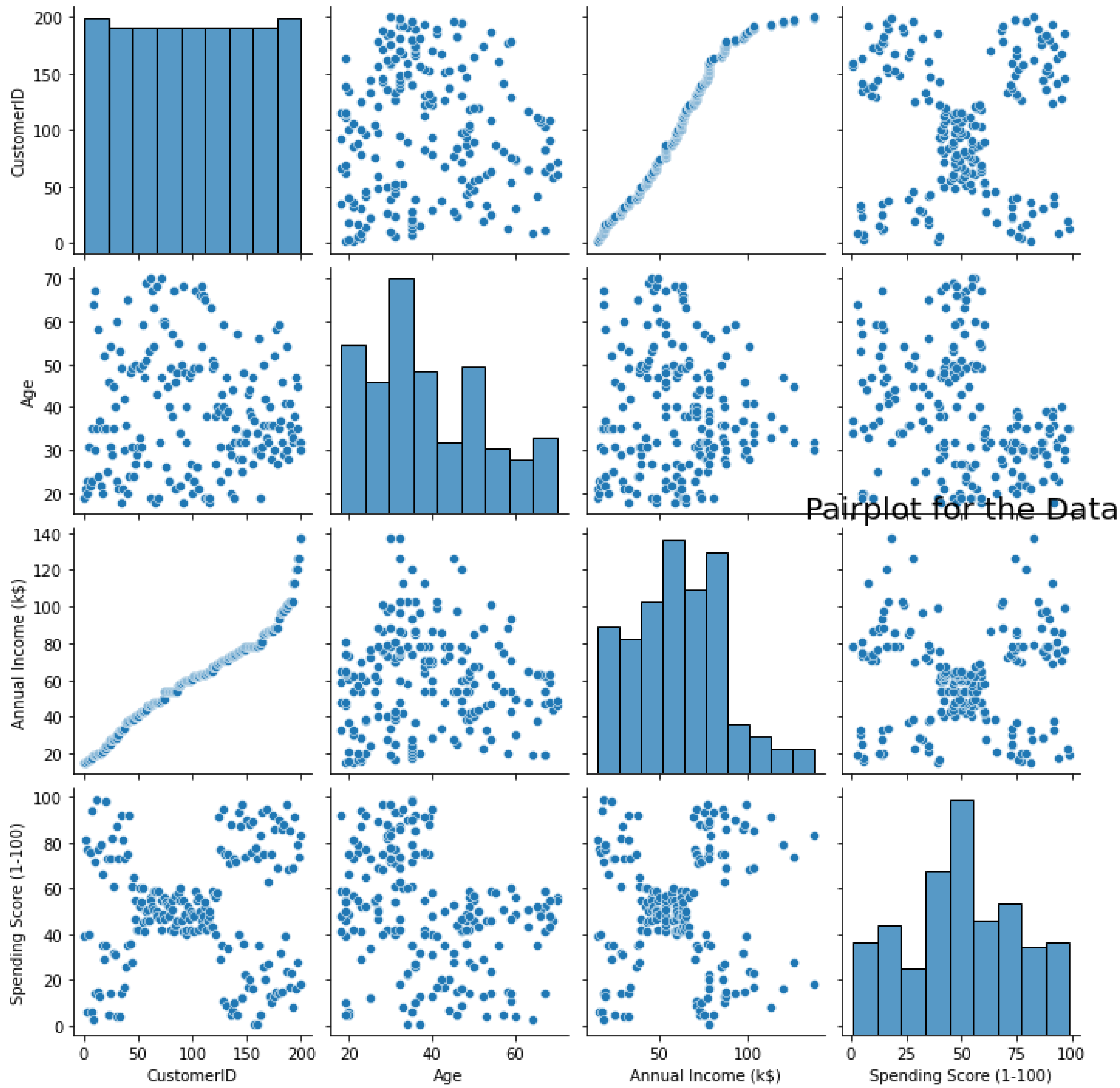
	CustomerID	Gender	Age	Annual Income (k$)	Spending Score (1-100)
0	1	Male	19	15	39
1	2	Male	21	15	81
2	3	Female	20	16	6
3	4	Female	23	16	77
4	5	Female	31	17	40

```
(200, 5)
CustomerID                 int64
Gender                    object
Age                        int64
Annual Income (k$)         int64
Spending Score (1-100)     int64
dtype: object
```

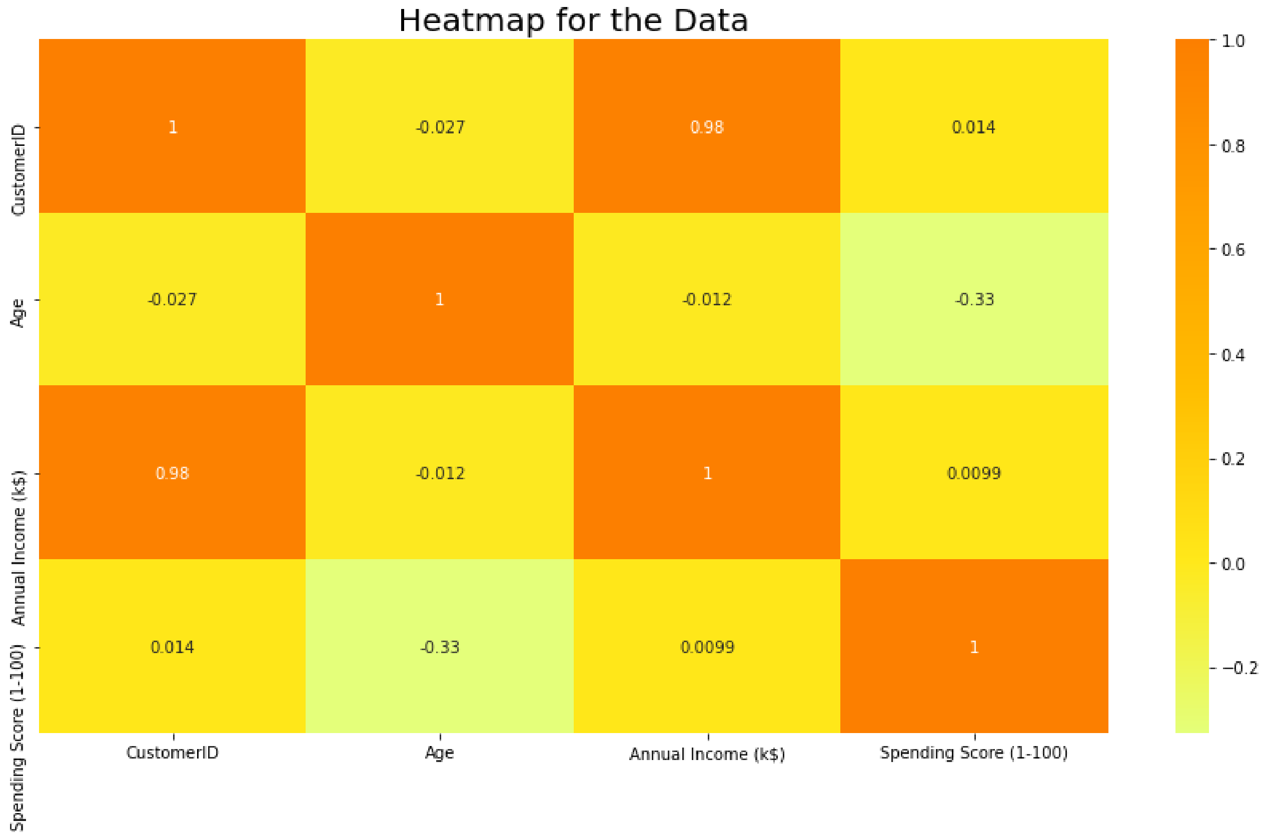
In [5]:
```
df.isnull().any().any()
```

Out[5]:
```
False
```

In [6]:
```
sns.pairplot(df)
plt.title('Pairplot for the Data', fontsize = 20)
plt.show()
```



In [7]:
```
plt.rcParams['figure.figsize'] = (15, 8)
sns.heatmap(df.corr(), cmap = 'Wistia', annot = True)
plt.title('Heatmap for the Data', fontsize = 20)
plt.show()
```



The Above Graph for Showing the correlation between the different attributes of the Mall Customer Segementation Dataset, This Heat map reflects the most correlated features with Orange Color and least correlated features with yellow color.

We can clearly see that these attributes do not have good correlation among them, that's why we will proceed with all of the features.

In [8]:
```
df.Gender[df.Gender == 'Male'] = 1
df.Gender[df.Gender == 'Female'] = 0
df
```

Out[8]:

	CustomerID	Gender	Age	Annual Income (k$)	Spending Score (1-100)
0	1	1	19	15	39
1	2	1	21	15	81
2	3	0	20	16	6
3	4	0	23	16	77
4	5	0	31	17	40
...	...	...	...	...	...
195	196	0	35	120	79
196	197	0	45	126	28
197	198	1	32	126	74
198	199	1	32	137	18
199	200	1	30	137	83

200 rows × 5 columns

Clustering Analysis

In [9]:
```
X = df.iloc[:, [1,2,3, 4]].values
# Let's check the shape of x
print(X.shape)
X
```

```
(200, 4)
```

Out[9]:
```
array([[1, 19, 15, 39],
       [1, 21, 15, 81],
       [0, 20, 16, 6],
       [0, 23, 16, 77],
       [0, 31, 17, 40],
       [0, 22, 17, 76],
       [0, 35, 18, 6],
       [0, 23, 18, 94],
       [1, 64, 19, 3],
       [0, 30, 19, 72],
       [1, 67, 19, 14],
       [0, 35, 19, 99],
       [0, 58, 20, 15],
       [0, 24, 20, 77],
       [1, 37, 20, 13],
       [1, 22, 20, 79],
       [0, 35, 21, 35],
       [1, 20, 21, 66],
       [1, 52, 23, 29],
       [0, 35, 23, 98],
       [1, 35, 24, 35],
       [1, 25, 24, 73],
       [0, 46, 25, 5],
       [1, 31, 25, 73],
       [0, 54, 28, 14],
       [1, 29, 28, 82],
       [0, 45, 28, 32],
       [1, 35, 28, 61],
       [0, 40, 29, 31],
       [0, 23, 29, 87],
       [1, 60, 30, 4],
       [0, 21, 30, 73],
       [1, 53, 33, 4],
       [1, 18, 33, 92],
       [0, 49, 33, 14],
       [0, 21, 33, 81],
       [0, 42, 34, 17],
       [0, 30, 34, 73],
       [0, 36, 37, 26],
       [0, 20, 37, 75],
       [0, 65, 38, 35],
       [1, 24, 38, 92],
       [1, 48, 39, 36],
       [0, 31, 39, 61],
       [0, 49, 39, 28],
       [0, 24, 39, 65],
       [0, 50, 40, 55],
       [0, 27, 40, 47],
       [0, 29, 40, 42],
       [0, 31, 40, 42],
       [0, 49, 42, 52],
```

```
       [1, 33, 42, 60],
       [0, 31, 43, 54],
       [1, 59, 43, 60],
       [0, 50, 43, 45],
       [1, 47, 43, 41],
       [0, 51, 44, 50],
       [1, 69, 44, 46],
       [0, 27, 46, 51],
       [1, 53, 46, 46],
       [1, 70, 46, 56],
       [1, 19, 46, 55],
       [0, 67, 47, 52],
       [0, 54, 47, 59],
       [1, 63, 48, 51],
       [1, 18, 48, 59],
       [0, 43, 48, 50],
       [0, 68, 48, 48],
       [1, 19, 48, 59],
       [0, 32, 48, 47],
       [1, 70, 49, 55],
       [0, 47, 49, 42],
       [0, 60, 50, 49],
       [0, 60, 50, 56],
       [1, 59, 54, 47],
       [1, 26, 54, 54],
       [0, 45, 54, 53],
       [1, 40, 54, 48],
       [0, 23, 54, 52],
       [0, 49, 54, 42],
       [1, 57, 54, 51],
       [1, 38, 54, 55],
       [1, 67, 54, 41],
       [0, 46, 54, 44],
       [0, 21, 54, 57],
       [1, 48, 54, 46],
       [0, 55, 57, 58],
       [0, 22, 57, 55],
       [0, 34, 58, 60],
       [0, 50, 58, 46],
       [0, 68, 59, 55],
       [1, 18, 59, 41],
       [1, 48, 60, 49],
       [0, 40, 60, 40],
       [0, 32, 60, 42],
       [1, 24, 60, 52],
       [0, 47, 60, 47],
       [0, 27, 60, 50],
       [1, 48, 61, 42],
       [1, 20, 61, 49],
       [0, 23, 62, 41],
       [0, 49, 62, 48],
       [1, 67, 62, 59],
       [1, 26, 62, 55],
       [1, 49, 62, 56],
       [0, 21, 62, 42],
       [0, 66, 63, 50],
       [1, 54, 63, 46],
       [1, 68, 63, 43],
       [1, 66, 63, 48],
       [1, 65, 63, 52],
       [0, 19, 63, 54],
       [0, 38, 64, 42],
       [1, 19, 64, 46],
       [0, 18, 65, 48],
       [0, 19, 65, 50],
```

```
       [0, 63, 65, 43],
       [0, 49, 65, 59],
       [0, 51, 67, 43],
       [0, 50, 67, 57],
       [1, 27, 67, 56],
       [0, 38, 67, 40],
       [0, 40, 69, 58],
       [1, 39, 69, 91],
       [0, 23, 70, 29],
       [0, 31, 70, 77],
       [1, 43, 71, 35],
       [1, 40, 71, 95],
       [1, 59, 71, 11],
       [1, 38, 71, 75],
       [1, 47, 71, 9],
       [1, 39, 71, 75],
       [0, 25, 72, 34],
       [0, 31, 72, 71],
       [1, 20, 73, 5],
       [0, 29, 73, 88],
       [0, 44, 73, 7],
       [1, 32, 73, 73],
       [1, 19, 74, 10],
       [0, 35, 74, 72],
       [0, 57, 75, 5],
       [1, 32, 75, 93],
       [0, 28, 76, 40],
       [0, 32, 76, 87],
       [1, 25, 77, 12],
       [1, 28, 77, 97],
       [1, 48, 77, 36],
       [0, 32, 77, 74],
       [0, 34, 78, 22],
       [1, 34, 78, 90],
       [1, 43, 78, 17],
       [1, 39, 78, 88],
       [0, 44, 78, 20],
       [0, 38, 78, 76],
       [0, 47, 78, 16],
       [0, 27, 78, 89],
       [1, 37, 78, 1],
       [0, 30, 78, 78],
       [1, 34, 78, 1],
       [0, 30, 78, 73],
       [0, 56, 79, 35],
       [0, 29, 79, 83],
       [1, 19, 81, 5],
       [0, 31, 81, 93],
       [1, 50, 85, 26],
       [0, 36, 85, 75],
       [1, 42, 86, 20],
       [0, 33, 86, 95],
       [0, 36, 87, 27],
       [1, 32, 87, 63],
       [1, 40, 87, 13],
       [1, 28, 87, 75],
       [1, 36, 87, 10],
       [1, 36, 87, 92],
       [0, 52, 88, 13],
       [0, 30, 88, 86],
       [1, 58, 88, 15],
       [1, 27, 88, 69],
       [1, 59, 93, 14],
       [1, 35, 93, 90],
       [0, 37, 97, 32],
```

```
       [0, 32, 97, 86],
       [1, 46, 98, 15],
       [0, 29, 98, 88],
       [0, 41, 99, 39],
       [1, 30, 99, 97],
       [0, 54, 101, 24],
       [1, 28, 101, 68],
       [0, 41, 103, 17],
       [0, 36, 103, 85],
       [0, 34, 103, 23],
       [0, 32, 103, 69],
       [1, 33, 113, 8],
       [0, 38, 113, 91],
       [0, 47, 120, 16],
       [0, 35, 120, 79],
       [0, 45, 126, 28],
       [1, 32, 126, 74],
       [1, 32, 137, 18],
       [1, 30, 137, 83]], dtype=object)
```

In [10]:

```
# Define the scaler and apply to the data
scaler = MinMaxScaler()
X_scaled = scaler.fit_transform(X)
X=X_scaled
X
```

Out[10]:

```
array([[1.        , 0.01923077, 0.        , 0.3877551 ],
       [1.        , 0.05769231, 0.        , 0.81632653],
       [0.        , 0.03846154, 0.00819672, 0.05102041],
       [0.        , 0.09615385, 0.00819672, 0.7755102 ],
       [0.        , 0.25      , 0.01639344, 0.39795918],
       [0.        , 0.07692308, 0.01639344, 0.76530612],
       [0.        , 0.32692308, 0.02459016, 0.05102041],
       [0.        , 0.09615385, 0.02459016, 0.94897959],
       [1.        , 0.88461538, 0.03278689, 0.02040816],
       [0.        , 0.23076923, 0.03278689, 0.7244898 ],
       [1.        , 0.94230769, 0.03278689, 0.13265306],
       [0.        , 0.32692308, 0.03278689, 1.        ],
       [0.        , 0.76923077, 0.04098361, 0.14285714],
       [0.        , 0.11538462, 0.04098361, 0.7755102 ],
       [1.        , 0.36538462, 0.04098361, 0.12244898],
       [1.        , 0.07692308, 0.04098361, 0.79591837],
       [0.        , 0.32692308, 0.04918033, 0.34693878],
       [1.        , 0.03846154, 0.04918033, 0.66326531],
       [1.        , 0.65384615, 0.06557377, 0.28571429],
       [0.        , 0.32692308, 0.06557377, 0.98979592],
       [1.        , 0.32692308, 0.07377049, 0.34693878],
       [1.        , 0.13461538, 0.07377049, 0.73469388],
       [0.        , 0.53846154, 0.08196721, 0.04081633],
       [1.        , 0.25      , 0.08196721, 0.73469388],
       [0.        , 0.69230769, 0.10655738, 0.13265306],
       [1.        , 0.21153846, 0.10655738, 0.82653061],
       [0.        , 0.51923077, 0.10655738, 0.31632653],
       [1.        , 0.32692308, 0.10655738, 0.6122449 ],
       [0.        , 0.42307692, 0.1147541 , 0.30612245],
       [0.        , 0.09615385, 0.1147541 , 0.87755102],
       [1.        , 0.80769231, 0.12295082, 0.03061224],
       [0.        , 0.05769231, 0.12295082, 0.73469388],
       [1.        , 0.67307692, 0.14754098, 0.03061224],
       [1.        , 0.        , 0.14754098, 0.92857143],
       [0.        , 0.59615385, 0.14754098, 0.13265306],
       [0.        , 0.05769231, 0.14754098, 0.81632653],
       [0.        , 0.46153846, 0.1557377 , 0.16326531],
       [0.        , 0.23076923, 0.1557377 , 0.73469388],
```

```
[0.        , 0.34615385, 0.18032787, 0.25510204],
[0.        , 0.03846154, 0.18032787, 0.75510204],
[0.        , 0.90384615, 0.18852459, 0.34693878],
[1.        , 0.11538462, 0.18852459, 0.92857143],
[1.        , 0.57692308, 0.19672131, 0.35714286],
[0.        , 0.25      , 0.19672131, 0.6122449 ],
[0.        , 0.59615385, 0.19672131, 0.2755102 ],
[0.        , 0.11538462, 0.19672131, 0.65306122],
[0.        , 0.61538462, 0.20491803, 0.55102041],
[0.        , 0.17307692, 0.20491803, 0.46938776],
[0.        , 0.21153846, 0.20491803, 0.41836735],
[0.        , 0.25      , 0.20491803, 0.41836735],
[0.        , 0.59615385, 0.22131148, 0.52040816],
[1.        , 0.28846154, 0.22131148, 0.60204082],
[0.        , 0.25      , 0.2295082 , 0.54081633],
[1.        , 0.78846154, 0.2295082 , 0.60204082],
[0.        , 0.61538462, 0.2295082 , 0.44897959],
[1.        , 0.55769231, 0.2295082 , 0.40816327],
[0.        , 0.63461538, 0.23770492, 0.5       ],
[1.        , 0.98076923, 0.23770492, 0.45918367],
[0.        , 0.17307692, 0.25409836, 0.51020408],
[1.        , 0.67307692, 0.25409836, 0.45918367],
[1.        , 1.        , 0.25409836, 0.56122449],
[1.        , 0.01923077, 0.25409836, 0.55102041],
[0.        , 0.94230769, 0.26229508, 0.52040816],
[0.        , 0.69230769, 0.26229508, 0.59183673],
[1.        , 0.86538462, 0.2704918 , 0.51020408],
[1.        , 0.        , 0.2704918 , 0.59183673],
[0.        , 0.48076923, 0.2704918 , 0.5       ],
[0.        , 0.96153846, 0.2704918 , 0.47959184],
[1.        , 0.01923077, 0.2704918 , 0.59183673],
[0.        , 0.26923077, 0.2704918 , 0.46938776],
[1.        , 1.        , 0.27868852, 0.55102041],
[0.        , 0.55769231, 0.27868852, 0.41836735],
[0.        , 0.80769231, 0.28688525, 0.48979592],
[0.        , 0.80769231, 0.28688525, 0.56122449],
[1.        , 0.78846154, 0.31967213, 0.46938776],
[1.        , 0.15384615, 0.31967213, 0.54081633],
[0.        , 0.51923077, 0.31967213, 0.53061224],
[1.        , 0.42307692, 0.31967213, 0.47959184],
[0.        , 0.09615385, 0.31967213, 0.52040816],
[0.        , 0.59615385, 0.31967213, 0.41836735],
[1.        , 0.75      , 0.31967213, 0.51020408],
[1.        , 0.38461538, 0.31967213, 0.55102041],
[1.        , 0.94230769, 0.31967213, 0.40816327],
[0.        , 0.53846154, 0.31967213, 0.43877551],
[0.        , 0.05769231, 0.31967213, 0.57142857],
[1.        , 0.57692308, 0.31967213, 0.45918367],
[0.        , 0.71153846, 0.3442623 , 0.58163265],
[0.        , 0.07692308, 0.3442623 , 0.55102041],
[0.        , 0.30769231, 0.35245902, 0.60204082],
[0.        , 0.61538462, 0.35245902, 0.45918367],
[0.        , 0.96153846, 0.36065574, 0.55102041],
[1.        , 0.        , 0.36065574, 0.40816327],
[1.        , 0.57692308, 0.36885246, 0.48979592],
[0.        , 0.42307692, 0.36885246, 0.39795918],
[0.        , 0.26923077, 0.36885246, 0.41836735],
[1.        , 0.11538462, 0.36885246, 0.52040816],
[0.        , 0.55769231, 0.36885246, 0.46938776],
[0.        , 0.17307692, 0.36885246, 0.5       ],
[1.        , 0.57692308, 0.37704918, 0.41836735],
[1.        , 0.03846154, 0.37704918, 0.48979592],
[0.        , 0.09615385, 0.3852459 , 0.40816327],
[0.        , 0.59615385, 0.3852459 , 0.47959184],
[1.        , 0.94230769, 0.3852459 , 0.59183673],
```

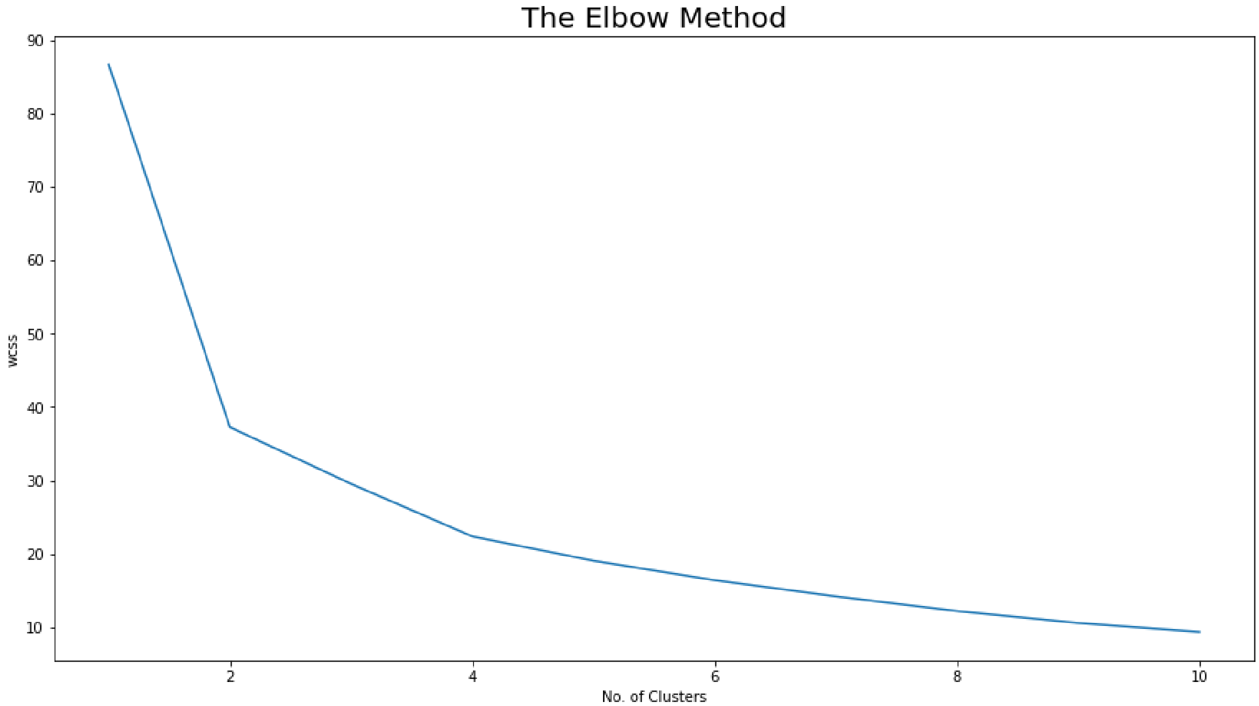
```
       [1.        , 0.15384615, 0.3852459 , 0.55102041],
       [1.        , 0.59615385, 0.3852459 , 0.56122449],
       [0.        , 0.05769231, 0.3852459 , 0.41836735],
       [0.        , 0.92307692, 0.39344262, 0.5       ],
       [1.        , 0.69230769, 0.39344262, 0.45918367],
       [1.        , 0.96153846, 0.39344262, 0.42857143],
       [1.        , 0.92307692, 0.39344262, 0.47959184],
       [1.        , 0.90384615, 0.39344262, 0.52040816],
       [0.        , 0.01923077, 0.39344262, 0.54081633],
       [0.        , 0.38461538, 0.40163934, 0.41836735],
       [1.        , 0.01923077, 0.40163934, 0.45918367],
       [0.        , 0.        , 0.40983607, 0.47959184],
       [0.        , 0.01923077, 0.40983607, 0.5       ],
       [0.        , 0.86538462, 0.40983607, 0.42857143],
       [0.        , 0.59615385, 0.40983607, 0.59183673],
       [0.        , 0.63461538, 0.42622951, 0.42857143],
       [0.        , 0.61538462, 0.42622951, 0.57142857],
       [1.        , 0.17307692, 0.42622951, 0.56122449],
       [0.        , 0.38461538, 0.42622951, 0.39795918],
       [0.        , 0.42307692, 0.44262295, 0.58163265],
       [1.        , 0.40384615, 0.44262295, 0.91836735],
       [0.        , 0.09615385, 0.45081967, 0.28571429],
       [0.        , 0.25      , 0.45081967, 0.7755102 ],
       [1.        , 0.48076923, 0.45901639, 0.34693878],
       [1.        , 0.42307692, 0.45901639, 0.95918367],
       [1.        , 0.78846154, 0.45901639, 0.10204082],
       [1.        , 0.38461538, 0.45901639, 0.75510204],
       [1.        , 0.55769231, 0.45901639, 0.08163265],
       [1.        , 0.40384615, 0.45901639, 0.75510204],
       [0.        , 0.13461538, 0.46721311, 0.33673469],
       [0.        , 0.25      , 0.46721311, 0.71428571],
       [1.        , 0.03846154, 0.47540984, 0.04081633],
       [0.        , 0.21153846, 0.47540984, 0.8877551 ],
       [0.        , 0.5       , 0.47540984, 0.06122449],
       [1.        , 0.26923077, 0.47540984, 0.73469388],
       [1.        , 0.01923077, 0.48360656, 0.09183673],
       [0.        , 0.32692308, 0.48360656, 0.7244898 ],
       [0.        , 0.75      , 0.49180328, 0.04081633],
       [1.        , 0.26923077, 0.49180328, 0.93877551],
       [0.        , 0.19230769, 0.5       , 0.39795918],
       [0.        , 0.26923077, 0.5       , 0.87755102],
       [1.        , 0.13461538, 0.50819672, 0.1122449 ],
       [1.        , 0.19230769, 0.50819672, 0.97959184],
       [1.        , 0.57692308, 0.50819672, 0.35714286],
       [0.        , 0.26923077, 0.50819672, 0.74489796],
       [0.        , 0.30769231, 0.51639344, 0.21428571],
       [1.        , 0.30769231, 0.51639344, 0.90816327],
       [1.        , 0.48076923, 0.51639344, 0.16326531],
       [1.        , 0.40384615, 0.51639344, 0.8877551 ],
       [0.        , 0.5       , 0.51639344, 0.19387755],
       [0.        , 0.38461538, 0.51639344, 0.76530612],
       [0.        , 0.55769231, 0.51639344, 0.15306122],
       [0.        , 0.17307692, 0.51639344, 0.89795918],
       [1.        , 0.36538462, 0.51639344, 0.        ],
       [0.        , 0.23076923, 0.51639344, 0.78571429],
       [1.        , 0.30769231, 0.51639344, 0.        ],
       [0.        , 0.23076923, 0.51639344, 0.73469388],
       [0.        , 0.73076923, 0.52459016, 0.34693878],
       [0.        , 0.21153846, 0.52459016, 0.83673469],
       [1.        , 0.01923077, 0.54098361, 0.04081633],
       [0.        , 0.25      , 0.54098361, 0.93877551],
       [1.        , 0.61538462, 0.57377049, 0.25510204],
       [0.        , 0.34615385, 0.57377049, 0.75510204],
       [1.        , 0.46153846, 0.58196721, 0.19387755],
       [0.        , 0.28846154, 0.58196721, 0.95918367],
```

```
       [0.        , 0.34615385, 0.59016393, 0.26530612],
       [1.        , 0.26923077, 0.59016393, 0.63265306],
       [1.        , 0.42307692, 0.59016393, 0.12244898],
       [1.        , 0.19230769, 0.59016393, 0.75510204],
       [1.        , 0.34615385, 0.59016393, 0.09183673],
       [1.        , 0.34615385, 0.59016393, 0.92857143],
       [0.        , 0.65384615, 0.59836066, 0.12244898],
       [0.        , 0.23076923, 0.59836066, 0.86734694],
       [1.        , 0.76923077, 0.59836066, 0.14285714],
       [1.        , 0.17307692, 0.59836066, 0.69387755],
       [1.        , 0.78846154, 0.63934426, 0.13265306],
       [1.        , 0.32692308, 0.63934426, 0.90816327],
       [0.        , 0.36538462, 0.67213115, 0.31632653],
       [0.        , 0.26923077, 0.67213115, 0.86734694],
       [1.        , 0.53846154, 0.68032787, 0.14285714],
       [0.        , 0.21153846, 0.68032787, 0.8877551 ],
       [0.        , 0.44230769, 0.68852459, 0.3877551 ],
       [1.        , 0.23076923, 0.68852459, 0.97959184],
       [0.        , 0.69230769, 0.70491803, 0.23469388],
       [1.        , 0.19230769, 0.70491803, 0.68367347],
       [0.        , 0.44230769, 0.72131148, 0.16326531],
       [0.        , 0.34615385, 0.72131148, 0.85714286],
       [0.        , 0.30769231, 0.72131148, 0.2244898 ],
       [0.        , 0.26923077, 0.72131148, 0.69387755],
       [1.        , 0.28846154, 0.80327869, 0.07142857],
       [0.        , 0.38461538, 0.80327869, 0.91836735],
       [0.        , 0.55769231, 0.86065574, 0.15306122],
       [0.        , 0.32692308, 0.86065574, 0.79591837],
       [0.        , 0.51923077, 0.90983607, 0.2755102 ],
       [1.        , 0.26923077, 0.90983607, 0.74489796],
       [1.        , 0.26923077, 1.        , 0.17346939],
       [1.        , 0.23076923, 1.        , 0.83673469]])
```

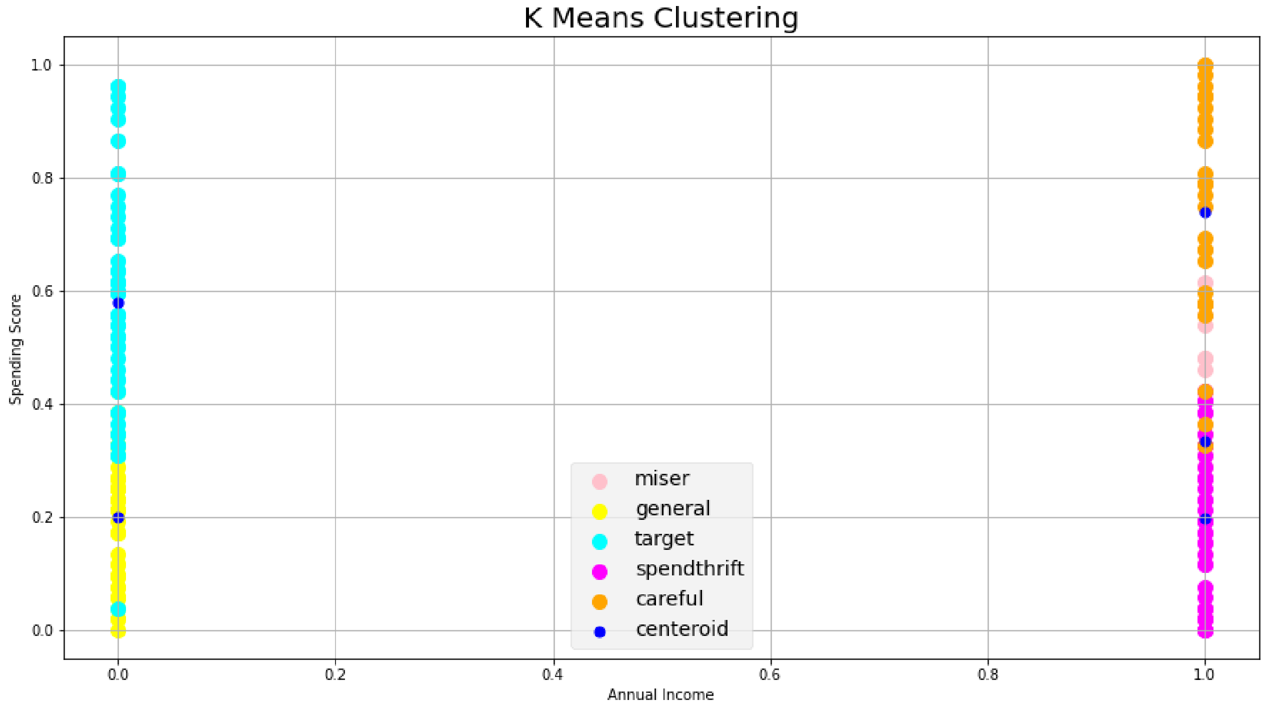
K-Mean

In [11]:

```python
from sklearn.cluster import KMeans
wcss = []
for i in range(1, 11):
    km = KMeans(n_clusters = i, init = 'k-means++', max_iter = 300, n_init = 10, random
    km.fit(X)
    wcss.append(km.inertia_)
plt.plot(range(1, 11), wcss)
plt.title('The Elbow Method', fontsize = 20)
plt.xlabel('No. of Clusters')
plt.ylabel('wcss')
plt.show()
```



In [12]:
```
km = KMeans(n_clusters = 5, init = 'k-means++', max_iter = 300, n_init = 10, random_sta
y_means = km.fit_predict(X)
plt.scatter(X[y_means == 0, 0], X[y_means == 0, 1], s = 100, c = 'pink', label = 'miser
plt.scatter(X[y_means == 1, 0], X[y_means == 1, 1], s = 100, c = 'yellow', label = 'gen
plt.scatter(X[y_means == 2, 0], X[y_means == 2, 1], s = 100, c = 'cyan', label = 'targe
plt.scatter(X[y_means == 3, 0], X[y_means == 3, 1], s = 100, c = 'magenta', label = 'sp
plt.scatter(X[y_means == 4, 0], X[y_means == 4, 1], s = 100, c = 'orange', label = 'car
plt.scatter(km.cluster_centers_[:,0], km.cluster_centers_[:, 1], s = 50, c = 'blue' , l
plt.style.use('fivethirtyeight')
plt.title('K Means Clustering', fontsize = 20)
plt.xlabel('Annual Income')
plt.ylabel('Spending Score')
plt.legend()
plt.grid()
plt.show()
```

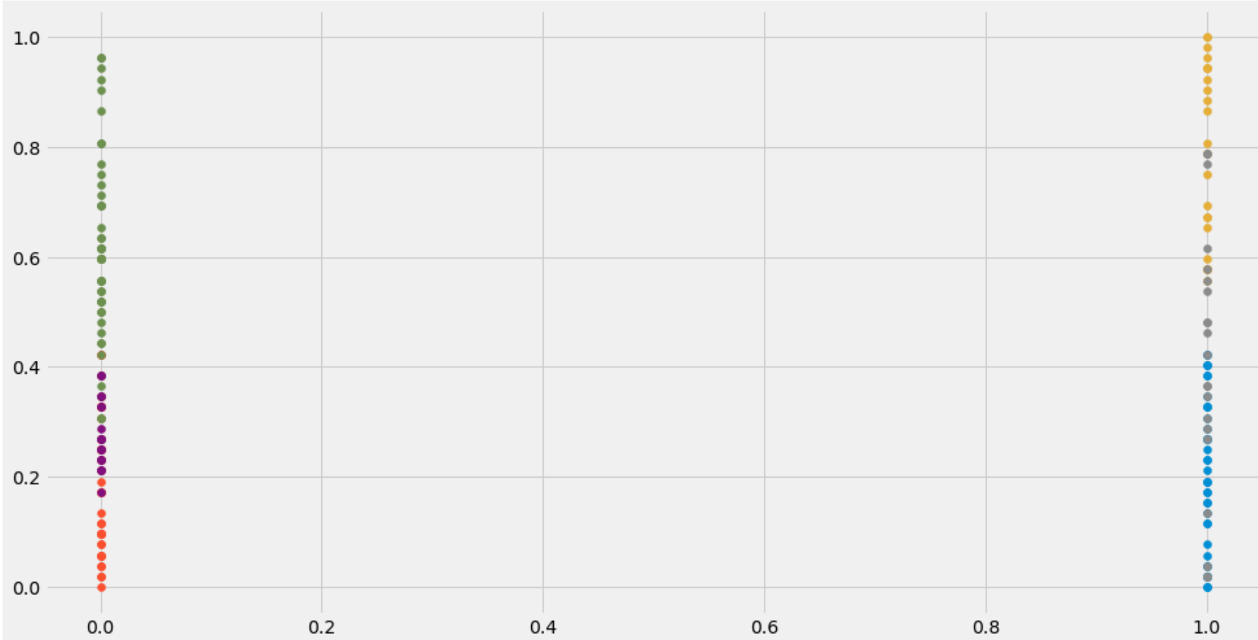


This Clustering Analysis gives us a very clear insight about the different segments of the customers in the Mall. There are clearly Five segments of Customers namely Miser, General, Target, Spendthrift, Careful based on their Annual Income and Spending Score which are reportedly the best factors/attributes to determine the segments of a customer in a Mall.

Mini-Batch K-Means

In [13]:

```
# mini-batch k-means clustering
from numpy import unique
from numpy import where
from sklearn.datasets import make_classification
from sklearn.cluster import MiniBatchKMeans
from matplotlib import pyplot
# define dataset
#X, _ = make_classification(n_samples=1000, n_features=2, n_informative=2, n_redundant=
# define the model
model = MiniBatchKMeans(n_clusters=6)
# fit the model
model.fit(X)
# assign a cluster to each example
yhat = model.predict(X)
# retrieve unique clusters
clusters = unique(yhat)
# create scatter plot for samples from each cluster
for cluster in clusters:
        # get row indexes for samples with this cluster
        row_ix = where(yhat == cluster)
        # create scatter of these samples
        pyplot.scatter(X[row_ix, 0], X[row_ix, 1])
# show the plot
pyplot.show()
```

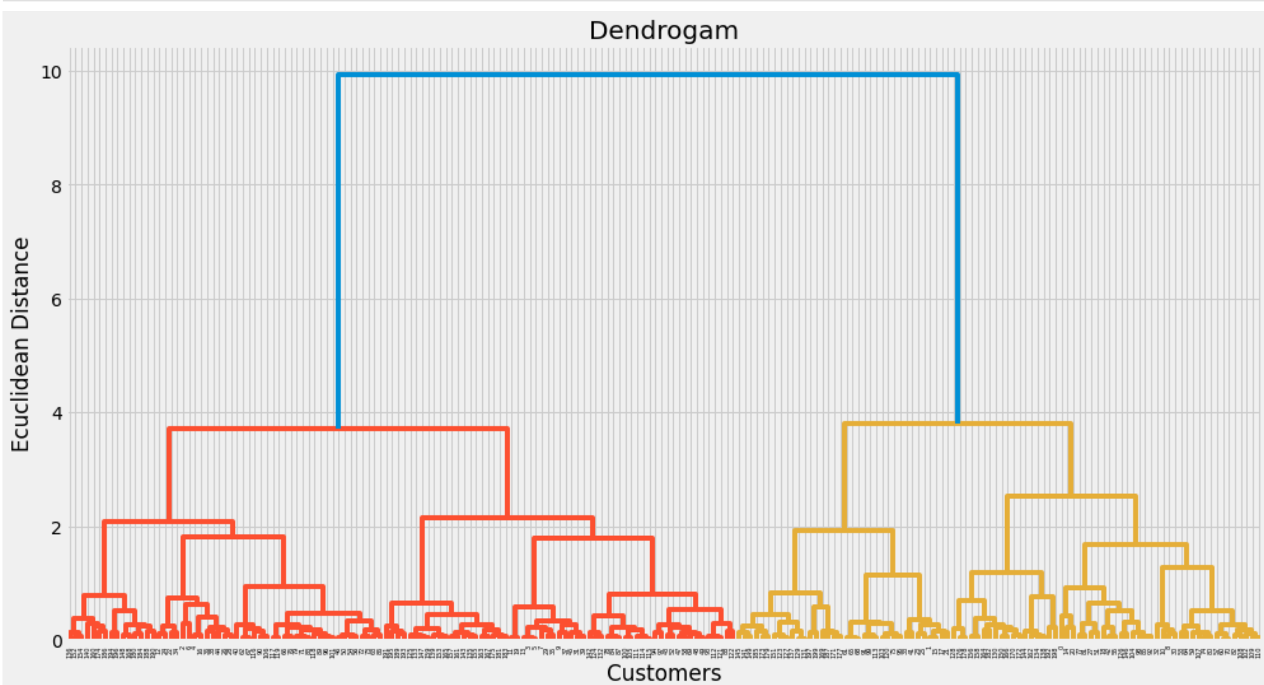


The Mini-Batch K-Means results are similar almost with the K-Mean clustering.

Hierarchial Clustering

Using Dendrograms to find the no. of Optimal Clusters

In [15]:
```
import scipy.cluster.hierarchy as sch
dendrogram = sch.dendrogram(sch.linkage(X, method = 'ward'))
plt.title('Dendrogam', fontsize = 20)
plt.xlabel('Customers')
plt.ylabel('Ecuclidean Distance')
plt.show()
```

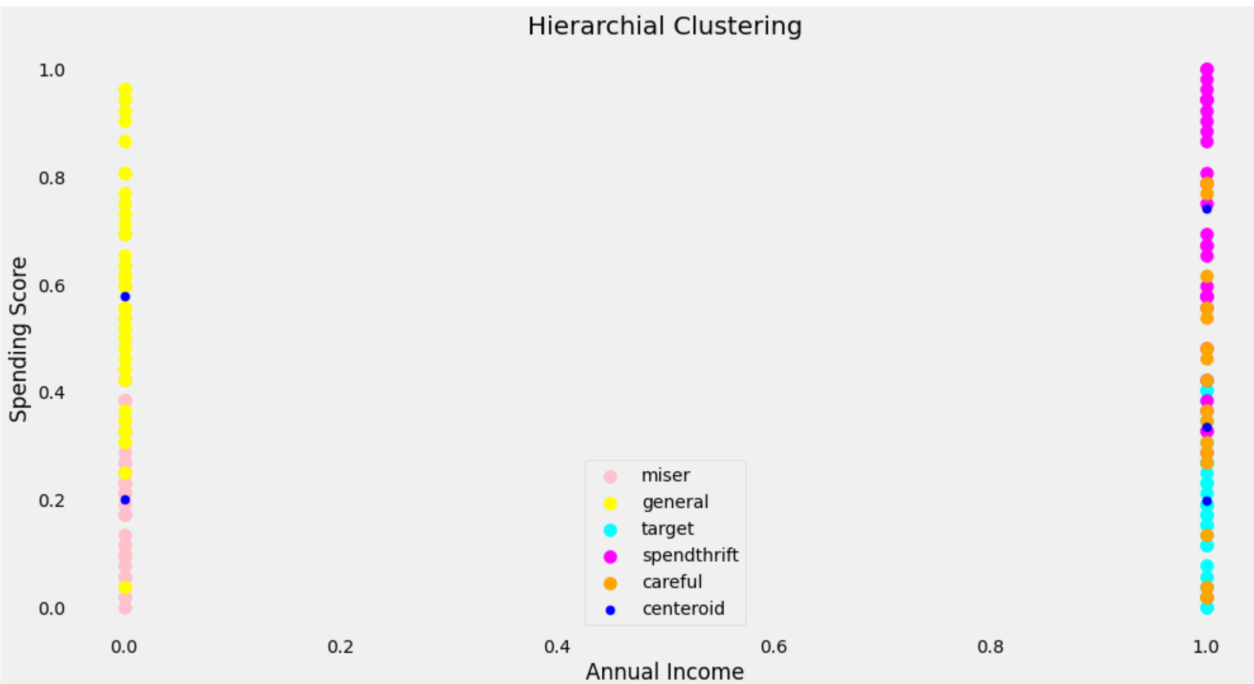


Visualizing the Clusters of Hierarchial

Clustering

In [16]:
```
km = KMeans(n_clusters = 5, init = 'k-means++', max_iter = 300, n_init = 10, random_sta
y_means = km.fit_predict(X)
```

In [17]:
```
from sklearn.cluster import AgglomerativeClustering
hc = AgglomerativeClustering(n_clusters = 5, affinity = 'euclidean', linkage = 'ward')
y_hc = hc.fit_predict(X)
plt.scatter(X[y_hc == 0, 0], X[y_hc == 0, 1], s = 100, c = 'pink', label = 'miser')
plt.scatter(X[y_hc == 1, 0], X[y_hc == 1, 1], s = 100, c = 'yellow', label = 'general')
plt.scatter(X[y_hc == 2, 0], X[y_hc == 2, 1], s = 100, c = 'cyan', label = 'target')
plt.scatter(X[y_hc == 3, 0], X[y_hc == 3, 1], s = 100, c = 'magenta', label = 'spendthr
plt.scatter(X[y_hc == 4, 0], X[y_hc == 4, 1], s = 100, c = 'orange', label = 'careful')
plt.scatter(km.cluster_centers_[:,0], km.cluster_centers_[:, 1], s = 50, c = 'blue' , l
plt.style.use('fivethirtyeight')
plt.title('Hierarchial Clustering', fontsize = 20)
plt.xlabel('Annual Income')
plt.ylabel('Spending Score')
plt.legend()
plt.grid()
plt.show()
```

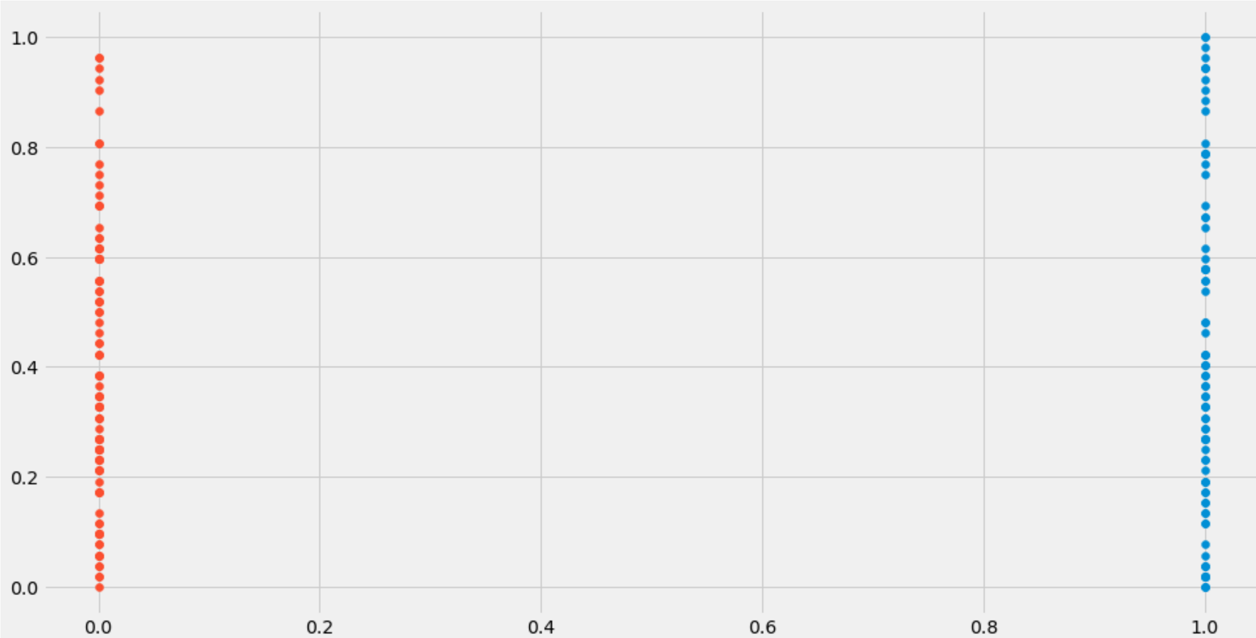


DBSCAN Clustering

In [18]:
```
from sklearn.cluster import DBSCAN
db = DBSCAN(eps=0.9, min_samples=10, n_jobs=-1).fit(X)
```

In [19]:
```
# define dataset
#X, _ = make_classification(n_samples=1000, n_features=2, n_informative=2, n_redundant=
# define the model
model = DBSCAN(eps=0.5, min_samples=10, n_jobs=-1).fit(X)
# fit model and predict clusters
yhat = model.fit_predict(X)
# retrieve unique clusters
clusters = unique(yhat)
```

```
# create scatter plot for samples from each cluster
for cluster in clusters:
        # get row indexes for samples with this cluster
        row_ix = where(yhat == cluster)
        # create scatter of these samples
        pyplot.scatter(X[row_ix, 0], X[row_ix, 1])
# show the plot
pyplot.show()
```



In [20]:

```
# Get the cluster labels (aka numbers)
pred_labels = db.labels_
# Count the total number of clusters
n_clusters_ = len(set(pred_labels)) - (1 if -1 in pred_labels else 0)
# Print model results
print(f'Estimated number of clusters: {n_clusters_}')
```

```
Estimated number of clusters: 2
```

In [21]:

```
import sklearn.metrics as metrics
```

In [22]:

```
# Print model results
print(f'Silhouette Coefficient: {metrics.silhouette_score(X, pred_labels):0.3f}')
```

```
Silhouette Coefficient: 0.519
```

In [23]:

```
# Get sample counts in each cluster
counts = np.bincount(pred_labels[pred_labels>=0])
print(counts)
```

```
[ 88 112]
```

In [25]:

```
# Initialize and fit the DBscan model
db = DBSCAN(eps=0.9, min_samples=10, n_jobs=-1).fit(X)
# Obtain the predicted labels and calculate number of clusters
pred_labels = db.labels_
n_clusters = len(set(pred_labels)) - (1 if -1 in pred_labels else 0)
```

In [26]:

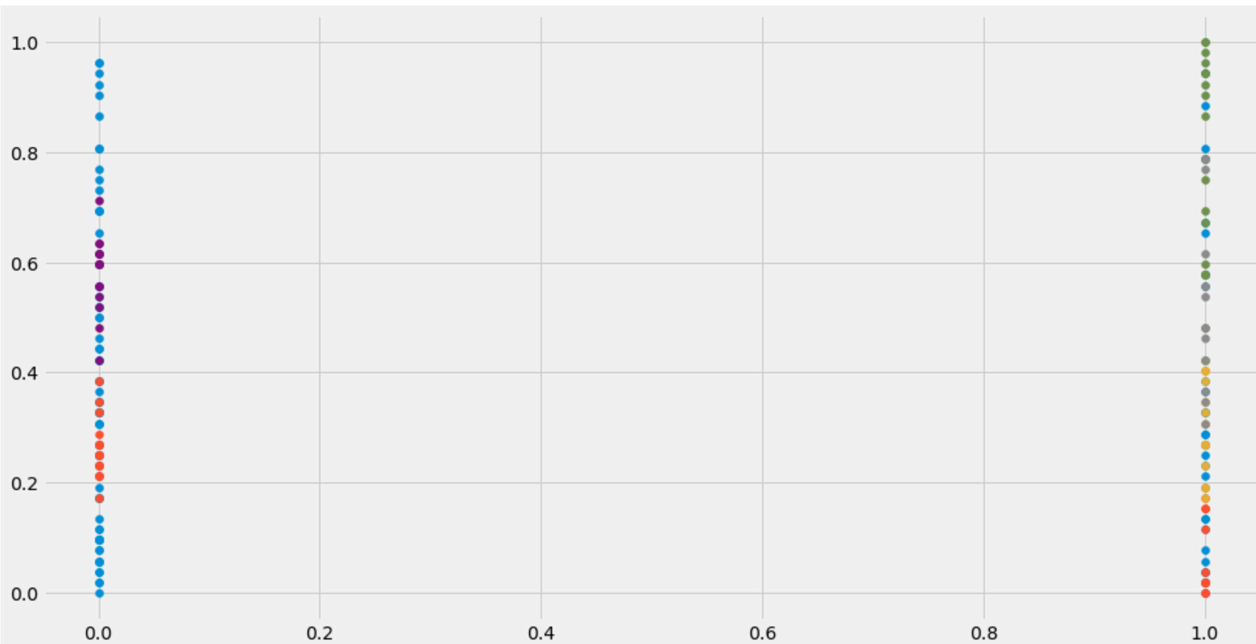
```
# Print performance metrics for DBscan
print(f'Estimated number of clusters: {n_clusters}')
```

```
Estimated number of clusters: 2
```

It devides the entire sample in two clusters

Optics Clustering

In [27]:
```
# optics clustering
from numpy import unique
from numpy import where
from sklearn.cluster import OPTICS
from matplotlib import pyplot
# define dataset
#X, _ = make_classification(n_samples=1000, n_features=2, n_informative=2, n_redundant=
# define the model
model = OPTICS(eps=0.8, min_samples=10)
# fit model and predict clusters
yhat = model.fit_predict(X)
# retrieve unique clusters
clusters = unique(yhat)
# create scatter plot for samples from each cluster
for cluster in clusters:
        # get row indexes for samples with this cluster
        row_ix = where(yhat == cluster)
        # create scatter of these samples
        pyplot.scatter(X[row_ix, 0], X[row_ix, 1])
# show the plot
pyplot.show()
```



In [28]:
```
# Get the cluster labels (aka numbers)
pred_labels = model.labels_
# Count the total number of clusters
n_clusters_ = len(set(pred_labels)) - (1 if -1 in pred_labels else 0)
# Print model results
print(f'Estimated number of clusters: {n_clusters_}')
```

```
Estimated number of clusters: 7
```

In [29]:
```
# Get sample counts in each cluster
counts = np.bincount(pred_labels[pred_labels>=0])
```

```
print(counts)
```

```
[10 15 20 11 16 19 17]
```

In [30]:

```
# Initialize and fit the DBscan model
model = OPTICS(eps=0.8, min_samples=10, n_jobs=-1).fit(X)
# Obtain the predicted labels and calculate number of clusters
pred_labels = model.labels_
n_clusters = len(set(pred_labels)) - (1 if -1 in pred_labels else 0)
```

In [31]:

```
# Print performance metrics for DBscan
print(f'Estimated number of clusters: {n_clusters}')
```

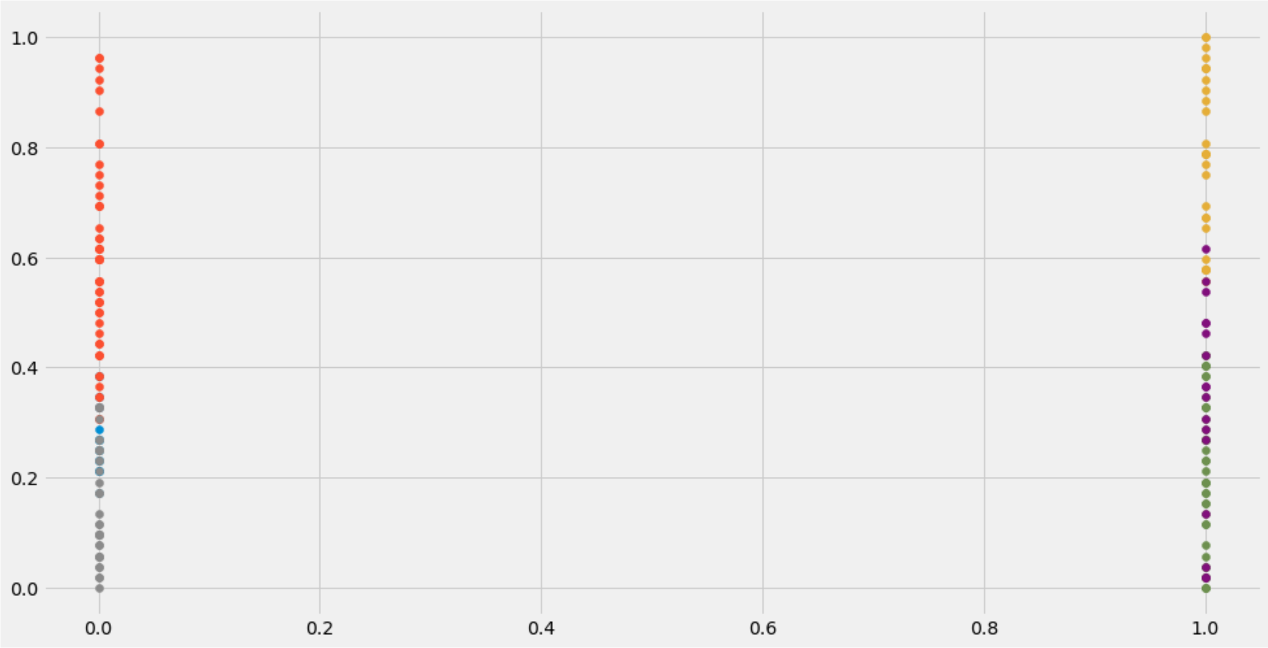
```
Estimated number of clusters: 7
```

The OPTICS devides the sample into seven clusters

Spectral Clustering

In [32]:

```
# spectral clustering
from numpy import unique
from numpy import where
from sklearn.datasets import make_classification
from sklearn.cluster import SpectralClustering
from matplotlib import pyplot
# define dataset
#X, _ = make_classification(n_samples=1000, n_features=2, n_informative=2, n_redundant=
# define the model
model = SpectralClustering(n_clusters=6)
# fit model and predict clusters
yhat = model.fit_predict(X)
# retrieve unique clusters
clusters = unique(yhat)
# create scatter plot for samples from each cluster
for cluster in clusters:
        # get row indexes for samples with this cluster
        row_ix = where(yhat == cluster)
        # create scatter of these samples
        pyplot.scatter(X[row_ix, 0], X[row_ix, 1])
# show the plot
pyplot.show()
```



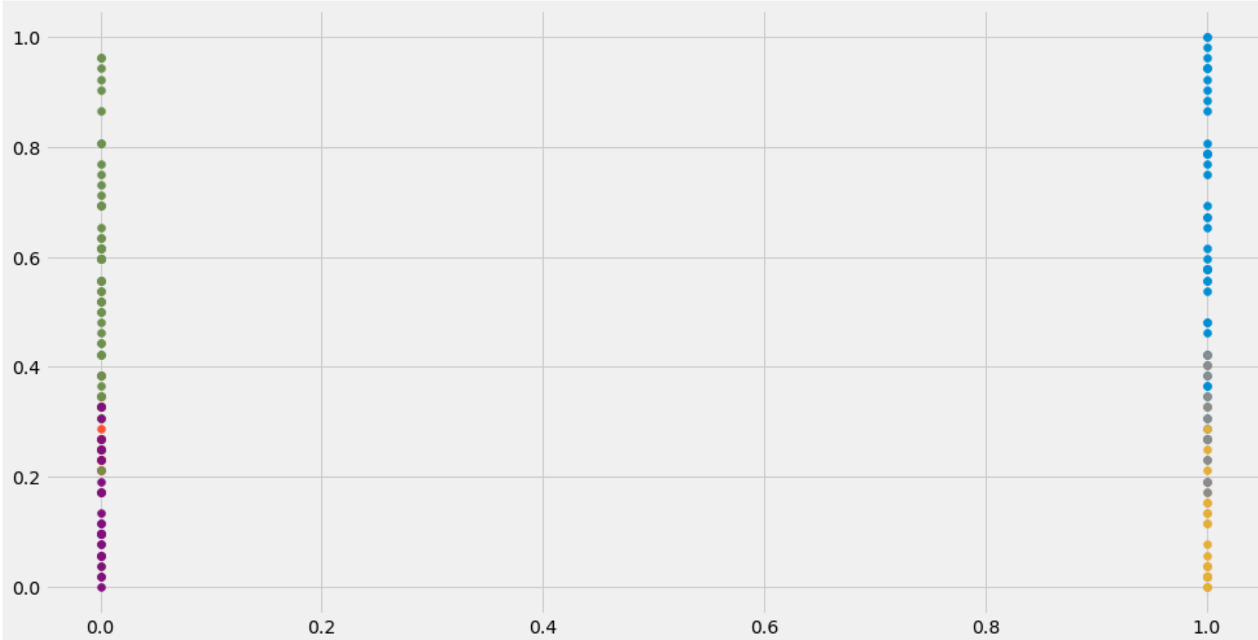
In [33]:
```
clusters = unique(yhat)
clusters
```

Out[33]: `array([0, 1, 2, 3, 4, 5])`

It devides the sample into six clusters, which seems reasonable.

Gaussian Mixture Clustering Model

In [34]:
```
# gaussian mixture clustering
from numpy import unique
from numpy import where
from sklearn.datasets import make_classification
from sklearn.mixture import GaussianMixture
from matplotlib import pyplot
# define the model
model = GaussianMixture(n_components=6)
# fit the model
model.fit(X)
# assign a cluster to each example
yhat = model.predict(X)
# retrieve unique clusters
clusters = unique(yhat)
# create scatter plot for samples from each cluster
for cluster in clusters:
        # get row indexes for samples with this cluster
        row_ix = where(yhat == cluster)
        # create scatter of these samples
        pyplot.scatter(X[row_ix, 0], X[row_ix, 1])
# show the plot
pyplot.show()
```



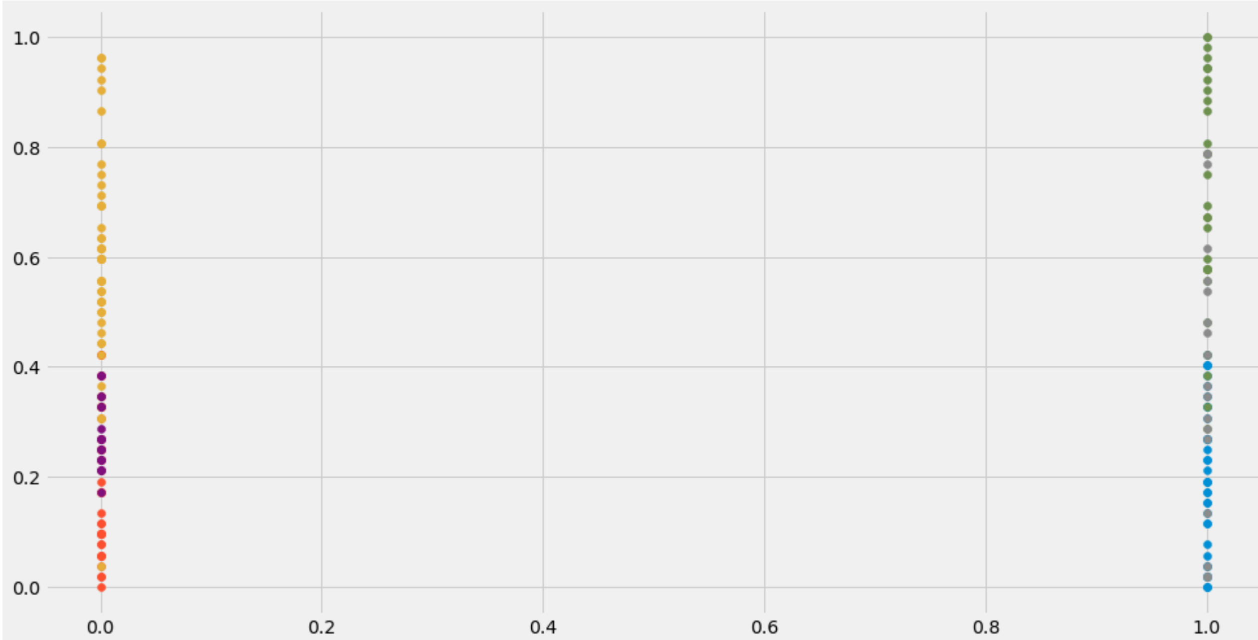
In [35]:
```
clusters = unique(yhat)
clusters
```

Out[35]:
```
array([0, 1, 2, 3, 4, 5], dtype=int64)
```

It devides the sample into six clusters, which seems reasonable.

BIRCH Clustering

In [36]:
```
# birch clustering
from numpy import unique
from numpy import where
from sklearn.datasets import make_classification
from sklearn.cluster import Birch
from matplotlib import pyplot
# define dataset
#X, _ = make_classification(n_samples=1000, n_features=2, n_informative=2, n_redundant=
# define the model
model = Birch(threshold=0.01, n_clusters=6)
# fit the model
model.fit(X)
# assign a cluster to each example
yhat = model.predict(X)
# retrieve unique clusters
clusters = unique(yhat)
# create scatter plot for samples from each cluster
for cluster in clusters:
        # get row indexes for samples with this cluster
        row_ix = where(yhat == cluster)
        # create scatter of these samples
        pyplot.scatter(X[row_ix, 0], X[row_ix, 1])
# show the plot
pyplot.show()
```

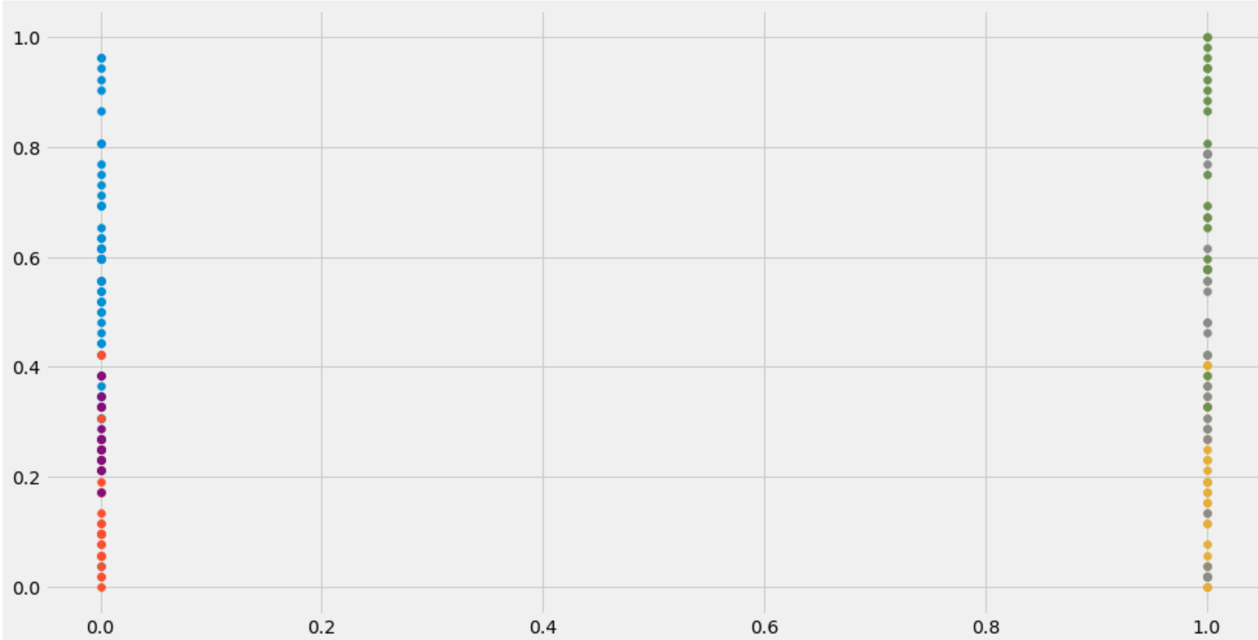


In [37]:
```
clusters = unique(yhat)
clusters
```

Out[37]:
```
array([0, 1, 2, 3, 4, 5], dtype=int64)
```

Agglomerative Clustering

In [38]:
```
# agglomerative clustering
from numpy import unique
from numpy import where
from sklearn.datasets import make_classification
from sklearn.cluster import AgglomerativeClustering
from matplotlib import pyplot
# define dataset
#X, _ = make_classification(n_samples=1000, n_features=2, n_informative=2, n_redundant=
# define the model
model = AgglomerativeClustering(n_clusters=6)
# fit model and predict clusters
yhat = model.fit_predict(X)
# retrieve unique clusters
clusters = unique(yhat)
# create scatter plot for samples from each cluster
for cluster in clusters:
        # get row indexes for samples with this cluster
        row_ix = where(yhat == cluster)
        # create scatter of these samples
        pyplot.scatter(X[row_ix, 0], X[row_ix, 1])
# show the plot
pyplot.show()
```

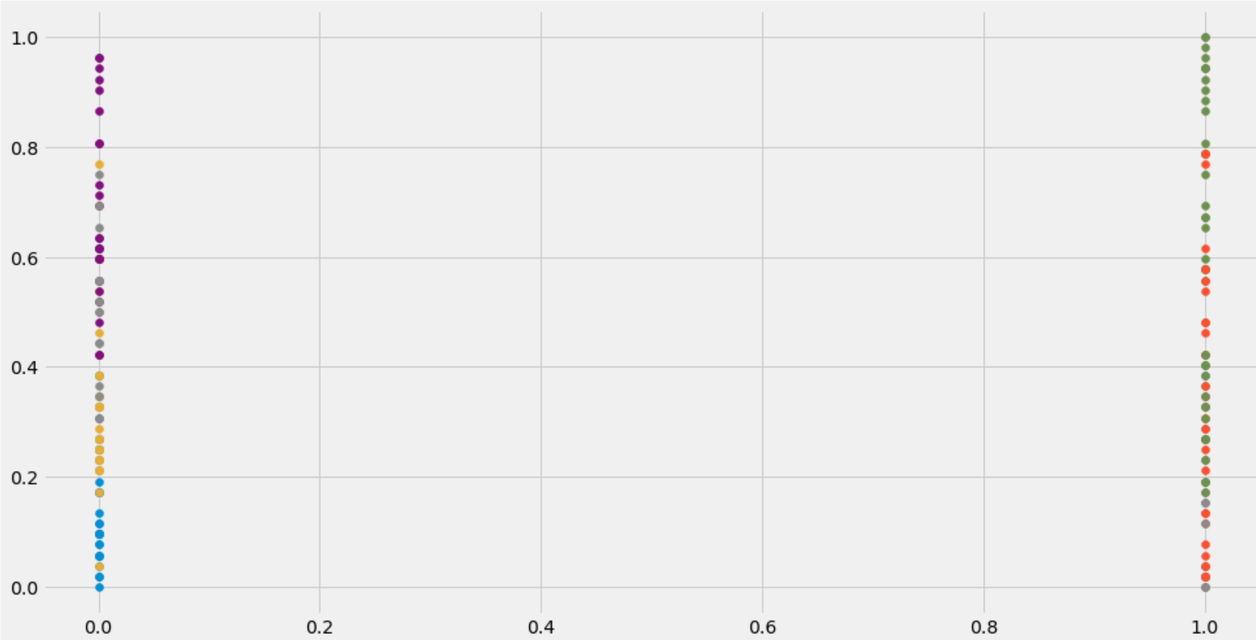


In [39]:
```
clusters = unique(yhat)
clusters
```

Out[39]:
```
array([0, 1, 2, 3, 4, 5], dtype=int64)
```

Affinity Propagation Clustering

In [40]:
```
# affinity propagation clustering
from numpy import unique
from numpy import where
from sklearn.datasets import make_classification
from sklearn.cluster import AffinityPropagation
from matplotlib import pyplot
# define dataset
#X, _ = make_classification(n_samples=1000, n_features=2, n_informative=2, n_redundant=
# define the model
model = AffinityPropagation(damping=0.9)
# fit the model
model.fit(X)
# assign a cluster to each example
yhat = model.predict(X)
# retrieve unique clusters
clusters = unique(yhat)
# create scatter plot for samples from each cluster
for cluster in clusters:
        # get row indexes for samples with this cluster
        row_ix = where(yhat == cluster)
        # create scatter of these samples
        pyplot.scatter(X[row_ix, 0], X[row_ix, 1])
# show the plot
pyplot.show()
```



In [41]:
```
clusters = unique(yhat)
clusters
```

Out[41]:
```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10], dtype=int64)
```

It puts the entire sample in one clusters.

Clustering Alg. Comparison

In [42]:
```
from sklearn.cluster import KMeans, AgglomerativeClustering, AffinityPropagation, Spect

def clustering_alg(X,y):
    algorithms = []
    algorithms.append(KMeans(n_clusters=6, init = 'k-means++', max_iter = 300, n_init =
    algorithms.append(AgglomerativeClustering(n_clusters=6))
    algorithms.append(AffinityPropagation(damping=0.9))
    algorithms.append(SpectralClustering(n_clusters=6, random_state=1,
                                         affinity='nearest_neighbors'))
    algorithms.append(DBSCAN(eps=0.9, min_samples=2, n_jobs=1).fit(X))
    algorithms.append(OPTICS(eps=0.8, min_samples=10))
    algorithms.append(GaussianMixture(n_components=6))
    algorithms.append(MiniBatchKMeans(n_clusters=6))
    algorithms.append(Birch(threshold=0.01, n_clusters=6))
    algorithms.append(AgglomerativeClustering(n_clusters = 6, affinity = 'euclidean', l
    data = []
    for algo in algorithms:
        algo.fit(X)
        data.append(({
            'ARI': metrics.adjusted_rand_score(y, algo.labels_),
            'AMI': metrics.adjusted_mutual_info_score(y, algo.labels_,
                                                     average_method='arithmetic'),
            'Homogenity': metrics.homogeneity_score(y, algo.labels_),
            'Completeness': metrics.completeness_score(y, algo.labels_),
            'V-measure': metrics.v_measure_score(y, algo.labels_),
            'Silhouette': metrics.silhouette_score(X, algo.labels_)
        }))
```

```
    results = pd.DataFrame(data=data1, columns=['ARI', 'AMI', 'Homogenity', 'Completene
                        index=['K-means', 'Agglomerative','Affinity',
                               'Spectral','DBSCAN','OPTICS',
                               'GaussianMixture',
                               'MiniBatchKMeans','Birch',
                              'AgglomerativeClustering'])
    return results
```

If in the sample the actual label is availible, then we can apply the above algrotham to evaluate and compare the predicted class and actual class of each row in terms of 'ARI', 'AMI', 'Homogenity', 'Completeness', 'V-measure','Silhouette', which are the various measures of comparison

Conclusion

The above results shows that the K-mean and Hirarchical clustering give almost similar results and seems better than the remaining clustering algrotham. This Clustering Analysis gives us a very clear insight about the different segments of the customers in the Mall. There are clearly Five segments of Customers namely Miser, General, Target, Spendthrift, Careful based on their Annual Income and Spending Score which are reportedly the best factors/attributes to determine the segments of a customer in a Mall.