In [1]:
```
## importing the Libraries

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

from sklearn.metrics import silhouette_score

import warnings
warnings.filterwarnings("ignore")
%matplotlib inline
```

In [2]:
```
# import datasets

df=pd.read_csv("https://raw.githubusercontent.com/NelakurthiSudheer/Mall-Customers-Segmentation/main/Dataset/Mall_Customers.csv")
```

In [3]:
```
# top 5 datasets
df.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]:
```
# check shape
df.shape
```

Out[4]:
```
(200, 5)
```

In [5]:
```
# check info
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   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
```

In [6]:
```
#check null values
df.isnull().sum()
```

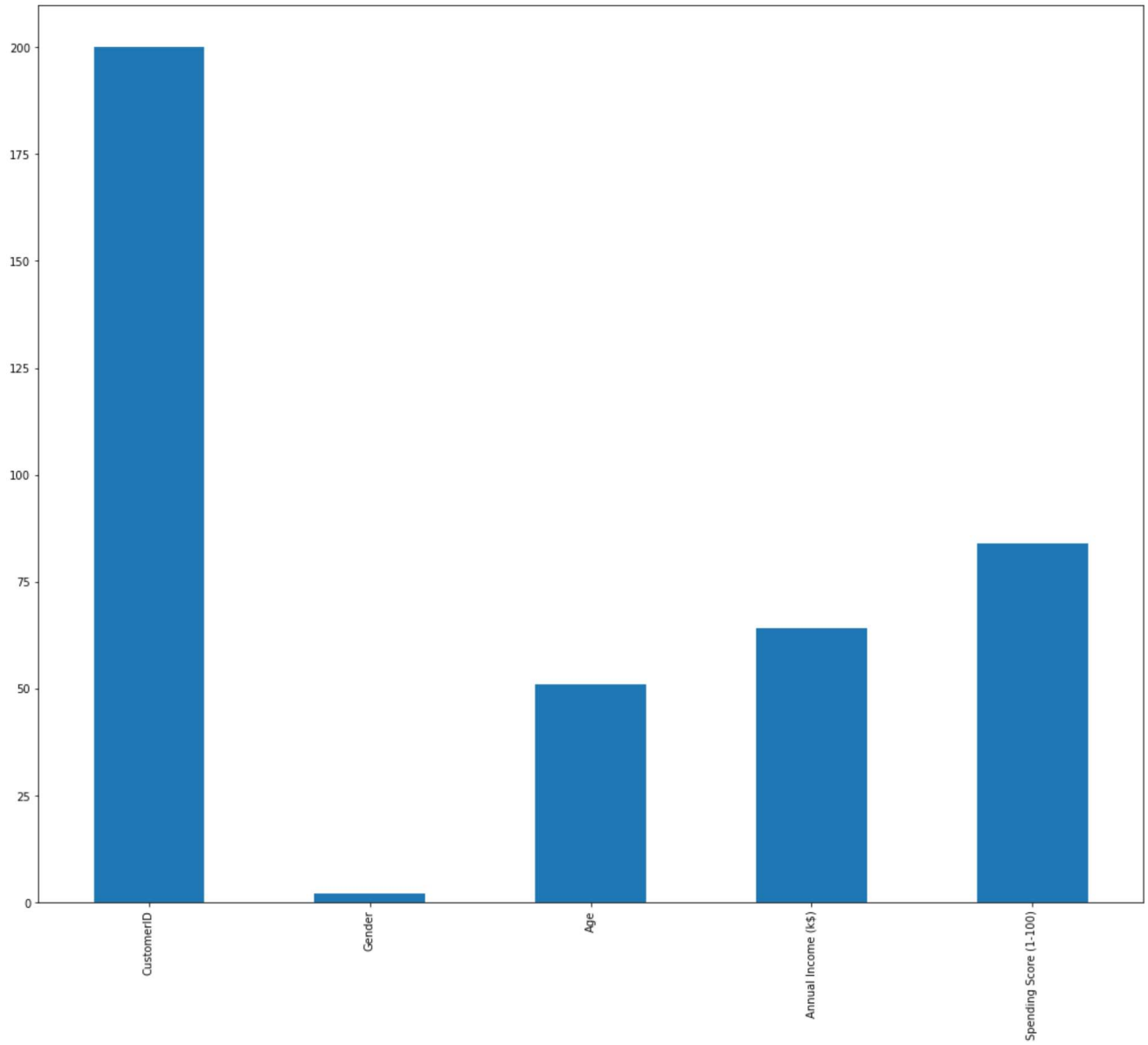
Out[6]:
```
CustomerID              0
Gender                  0
Age                     0
Annual Income (k$)      0
Spending Score (1-100)  0
dtype: int64
```

In [7]:
```
#check unique values
df.nunique()
```

Out[7]:
```
CustomerID              200
Gender                    2
Age                      51
Annual Income (k$)       64
Spending Score (1-100)   84
dtype: int64
```

In [8]:
```
df.nunique().plot.bar(figsize=(18,15))
```

Out[8]: <AxesSubplot:>



In [9]:
```
# check duplicated
df.duplicated().sum()
```

Out[9]: 0

Statical Based Analysis

In [10]:
```
# check the statistical data
df.describe()
```

Out[10]:

	CustomerID	Age	Annual Income (k$)	Spending Score (1-100)
count	200.000000	200.000000	200.000000	200.000000
mean	100.500000	38.850000	60.560000	50.200000
std	57.879185	13.969007	26.264721	25.823522
min	1.000000	18.000000	15.000000	1.000000
25%	50.750000	28.750000	41.500000	34.750000
50%	100.500000	36.000000	61.500000	50.000000
75%	150.250000	49.000000	78.000000	73.000000
max	200.000000	70.000000	137.000000	99.000000

In [11]:
```
df.describe().T
```

Out[11]:

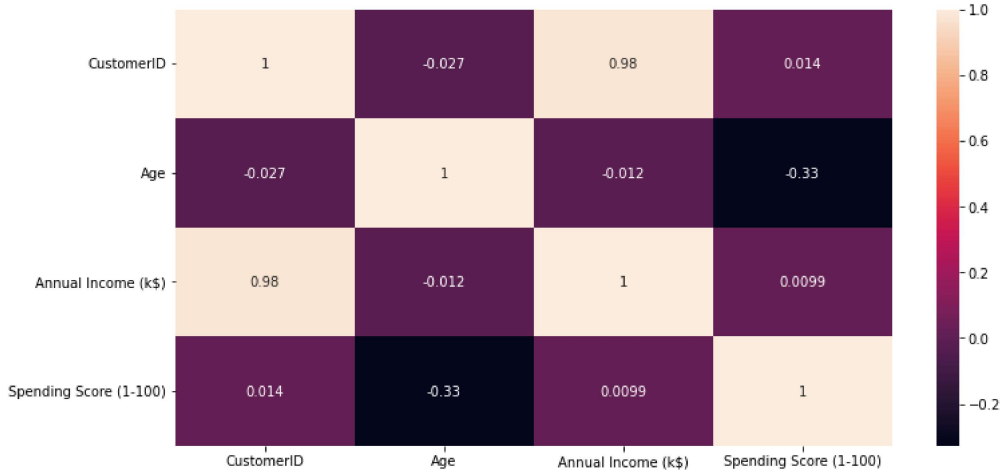
	count	mean	std	min	25%	50%	75%	max
CustomerID	200.0	100.50	57.879185	1.0	50.75	100.5	150.25	200.0
Age	200.0	38.85	13.969007	18.0	28.75	36.0	49.00	70.0
Annual Income (k$)	200.0	60.56	26.264721	15.0	41.50	61.5	78.00	137.0
Spending Score (1-100)	200.0	50.20	25.823522	1.0	34.75	50.0	73.00	99.0

In [12]:
```
# check correlation
df.corr()
```

Out[12]:

	CustomerID	Age	Annual Income (k$)	Spending Score (1-100)
CustomerID	1.000000	-0.026763	0.977548	0.013835
Age	-0.026763	1.000000	-0.012398	-0.327227
Annual Income (k$)	0.977548	-0.012398	1.000000	0.009903
Spending Score (1-100)	0.013835	-0.327227	0.009903	1.000000

In [13]:
```
plt.figure(figsize=(12,6))
sns.heatmap(df.corr() , annot=True)
plt.show()
```



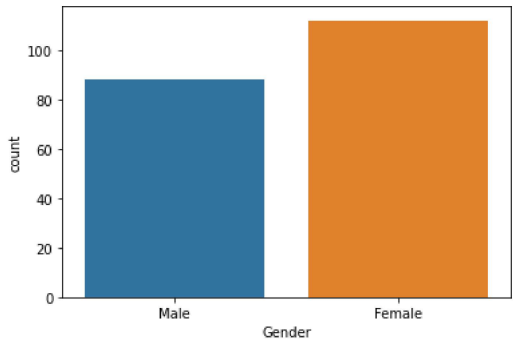
In [14]:
```
# check skewness
df.skew()
```

Out[14]:
```
CustomerID              0.000000
Age                     0.485569
Annual Income (k$)      0.321843
Spending Score (1-100)  -0.047220
dtype: float64
```

Graph Based Analysis

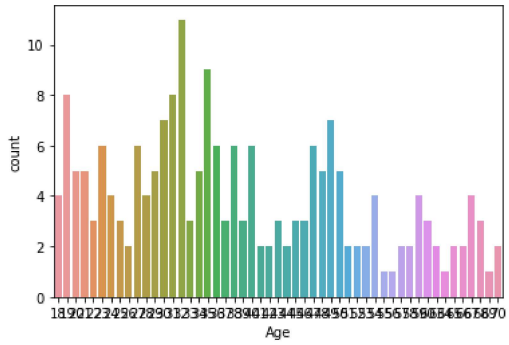
In [15]:
```
sns.countplot(df['Gender'])
```

Out[15]:
```
<AxesSubplot:xlabel='Gender', ylabel='count'>
```



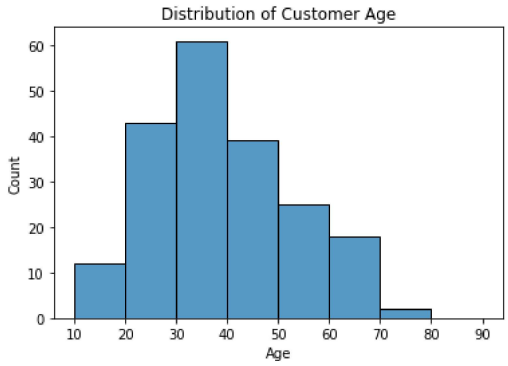
In [16]:
```
sns.countplot(df['Age'])
```

Out[16]:
```
<AxesSubplot:xlabel='Age', ylabel='count'>
```



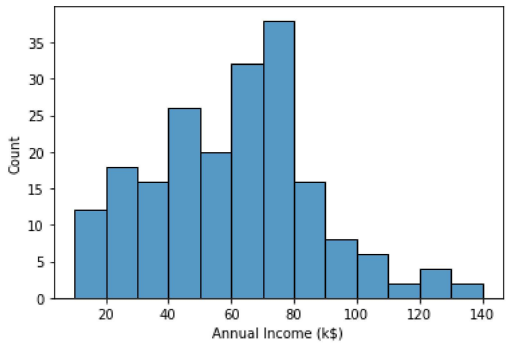
In [17]:
```
# Age distribution plot
sns.histplot(df['Age'],bins=list(range(10,100,10)))
plt.title("Distribution of Customer Age")
```

Out[17]:
```
Text(0.5, 1.0, 'Distribution of Customer Age')
```



In [18]:
```
# Annual Income Distribution
sns.histplot(df['Annual Income (k$)'] , bins=list(range(10,150,10)))
```

Out[18]:
```
<AxesSubplot:xlabel='Annual Income (k$)', ylabel='Count'>
```

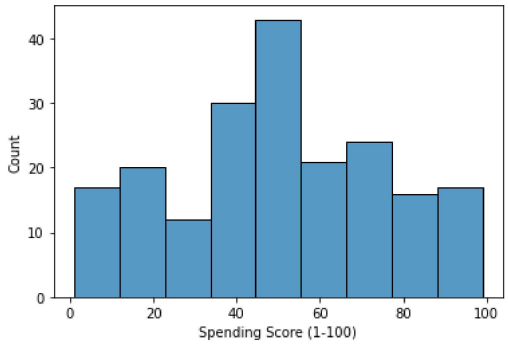


In [19]:
```
df['Annual Income (k$)'].unique()
```

Out[19]:
```
array([ 15,  16,  17,  18,  19,  20,  21,  23,  24,  25,  28,  29,  30,
        33,  34,  37,  38,  39,  40,  42,  43,  44,  46,  47,  48,  49,
        50,  54,  57,  58,  59,  60,  61,  62,  63,  64,  65,  67,  69,
        70,  71,  72,  73,  74,  75,  76,  77,  78,  79,  81,  85,  86,
        87,  88,  93,  97,  98,  99, 101, 103, 113, 120, 126, 137],
      dtype=int64)
```

In [20]:
```
# Spending Score Distribution
sns.histplot(df['Spending Score (1-100)'])
```

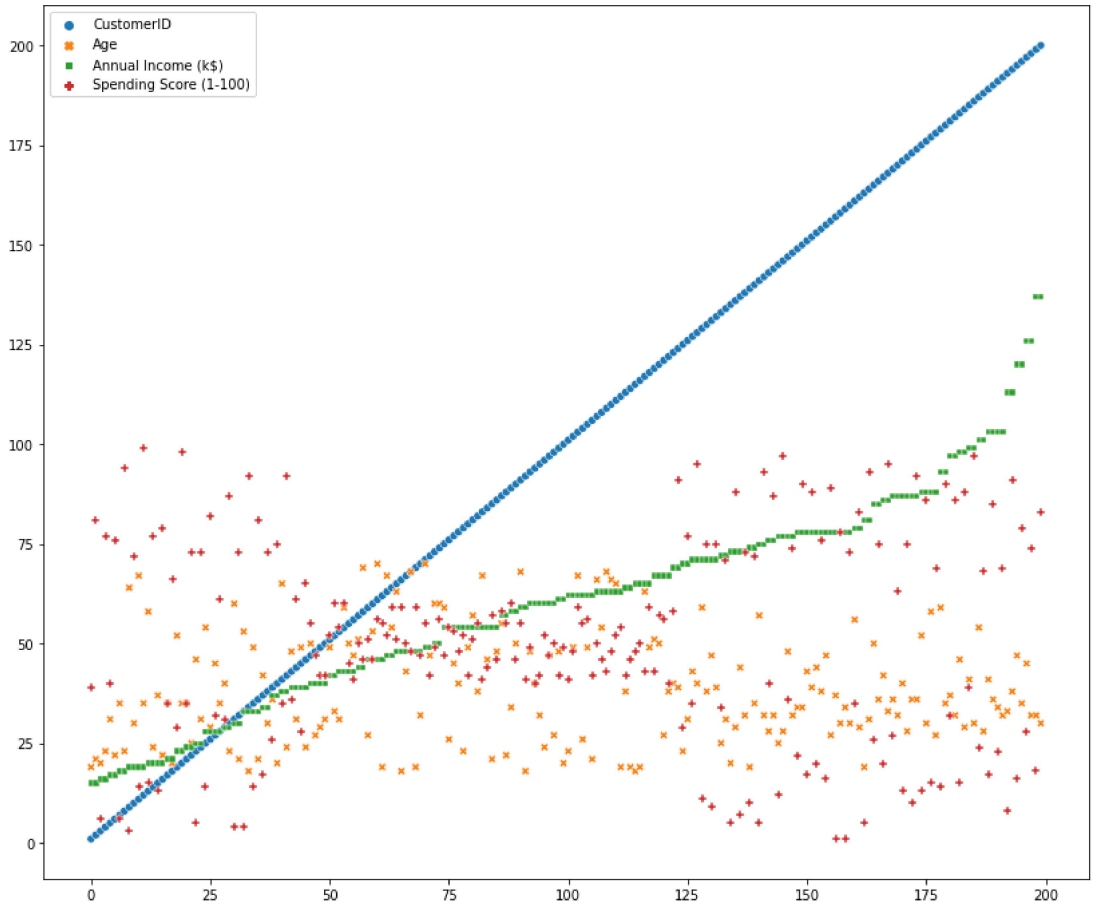
Out[20]:
```
<AxesSubplot:xlabel='Spending Score (1-100)', ylabel='Count'>
```



In [21]:
```
df.columns
```

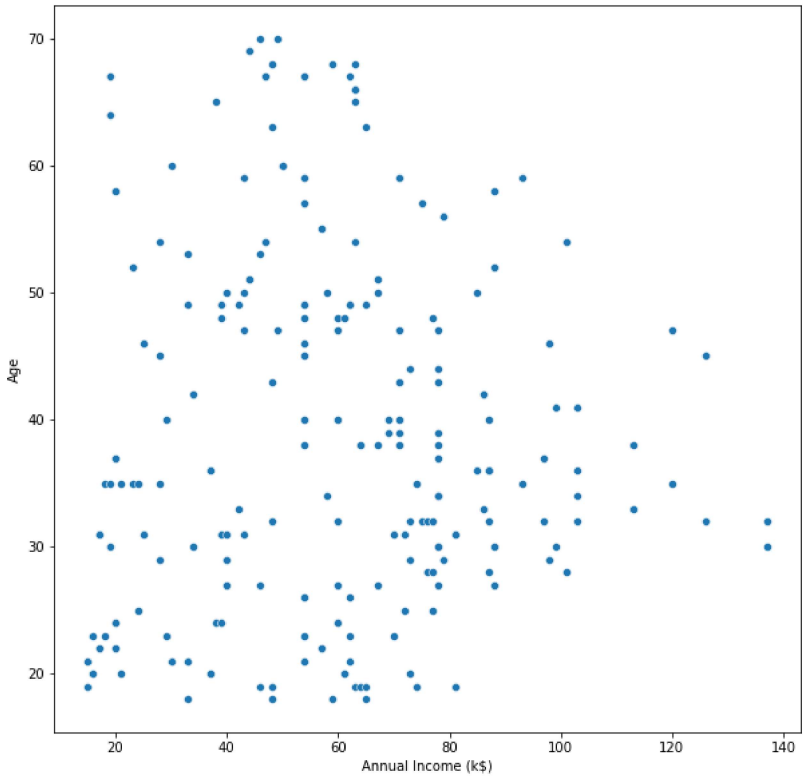
Out[21]:
```
Index(['CustomerID', 'Gender', 'Age', 'Annual Income (k$)',
       'Spending Score (1-100)'],
      dtype='object')
```

In [22]:
```
#  Scatter plot
plt.figure(figsize=(14,12))
sns.scatterplot(data=df)
plt.show()
```



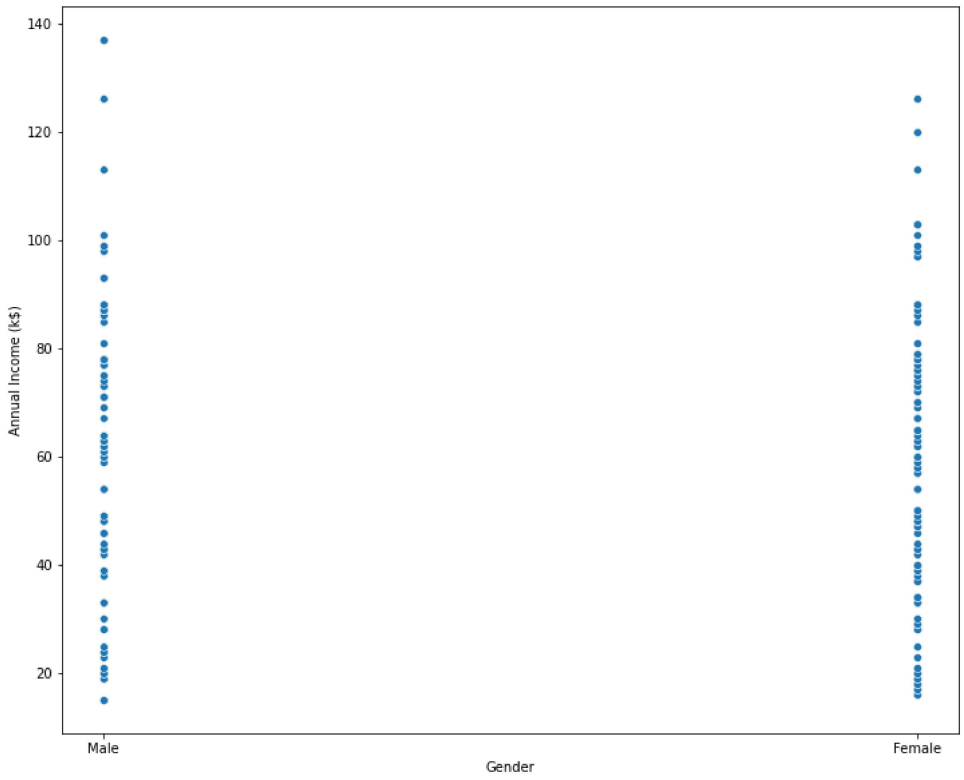
In [23]:
```
# Relation ship between Age and income
plt.figure(figsize=(10,10))
sns.scatterplot(data=df, x='Annual Income (k$)' , y='Age')
```

Out[23]: <AxesSubplot:xlabel='Annual Income (k$)', ylabel='Age'>



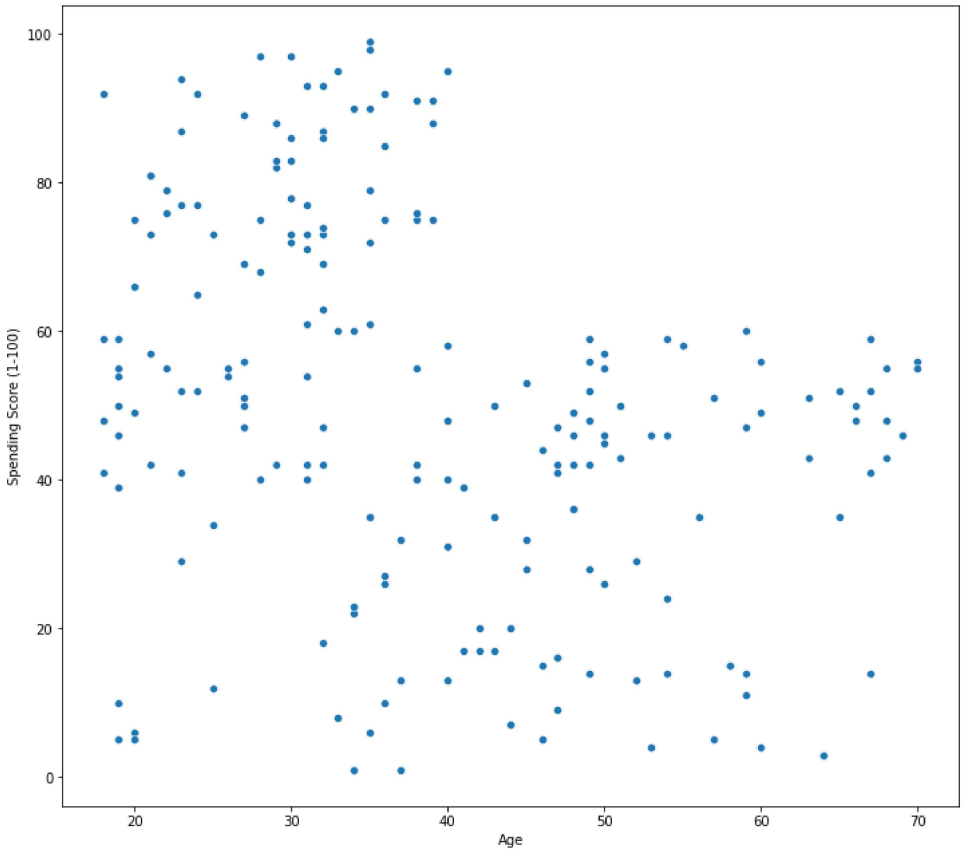
In [24]:
```
# check relationship between Gender and Annual Income
plt.figure(figsize=(12,10))
sns.scatterplot(data=df , x= 'Gender' , y='Annual Income (k$)')
```

Out[24]: <AxesSubplot:xlabel='Gender', ylabel='Annual Income (k$)'>



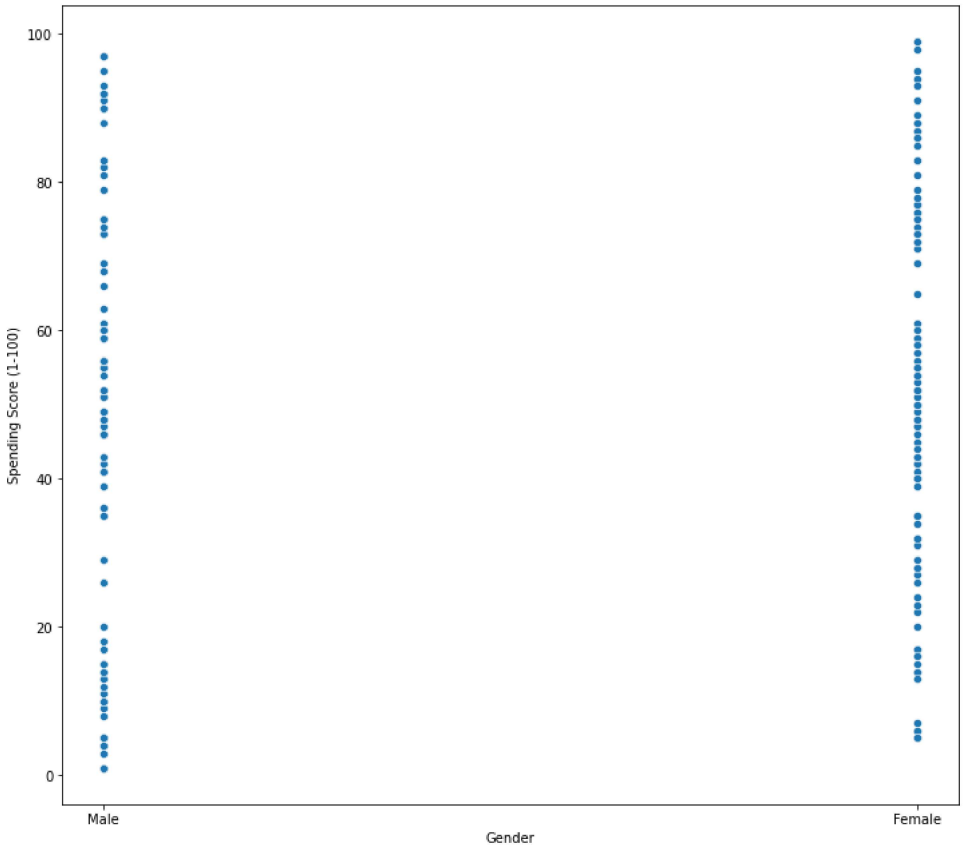
In [25]:
```
# check the relationship between age and spending score
plt.figure(figsize=(12,11))
sns.scatterplot(data=df , x="Age" , y='Spending Score (1-100)')
```

Out[25]:
```
<AxesSubplot:xlabel='Age', ylabel='Spending Score (1-100)'>
```



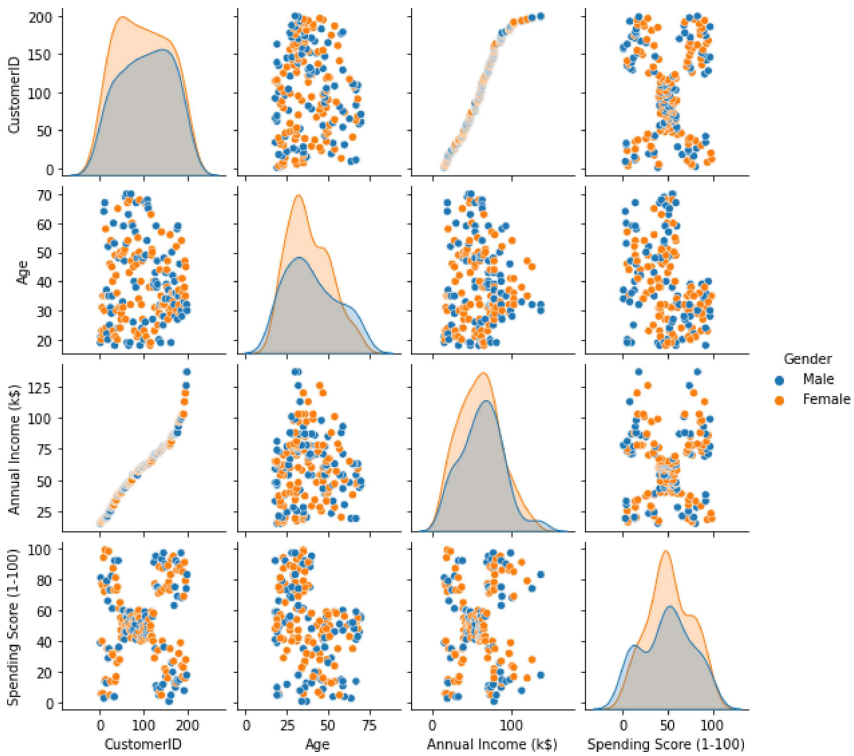
In [26]:
```
# check the relationship between Gender and Spending score
plt.figure(figsize=(12,11))
sns.scatterplot(data=df , x="Gender" , y='Spending Score (1-100)')
```

Out[26]: <AxesSubplot:xlabel='Gender', ylabel='Spending Score (1-100)'>



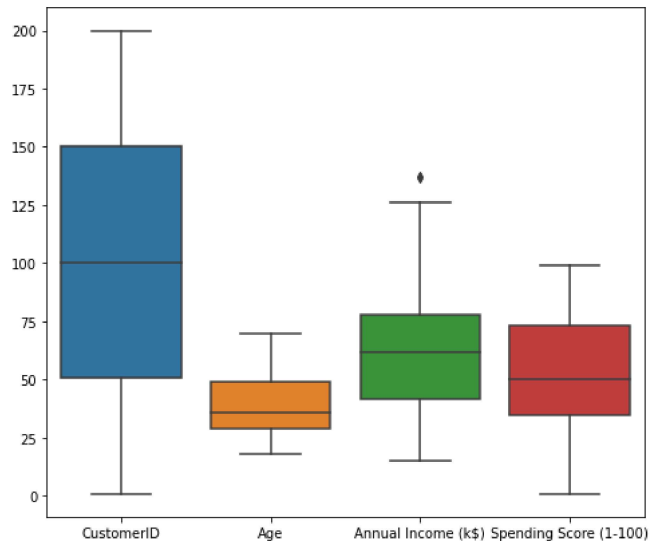
In [27]:
```
# Pair plot
sns.pairplot(df , hue='Gender' ,size=2.0)
```

Out[27]: <seaborn.axisgrid.PairGrid at 0x1c177b02c70>



In [28]:
```
# BOX PLOT
plt.figure(figsize=(8,7))
sns.boxplot(data=df , orient='v')
```

Out[28]: <AxesSubplot:>



In [29]:
```
df['Annual Income (k$)'].unique()
```

Out[29]:
```
array([ 15,  16,  17,  18,  19,  20,  21,  23,  24,  25,  28,  29,  30,
        33,  34,  37,  38,  39,  40,  42,  43,  44,  46,  47,  48,  49,
        50,  54,  57,  58,  59,  60,  61,  62,  63,  64,  65,  67,  69,
        70,  71,  72,  73,  74,  75,  76,  77,  78,  79,  81,  85,  86,
        87,  88,  93,  97,  98,  99, 101, 103, 113, 120, 126, 137],
      dtype=int64)
```

In [30]:
```
df['Annual Income (k$)'].value_counts()
```

Out[30]:
```
54     12
78     12
48      6
71      6
63      6
       ..
58      2
59      2
16      2
64      2
137     2
Name: Annual Income (k$), Length: 64, dtype: int64
```

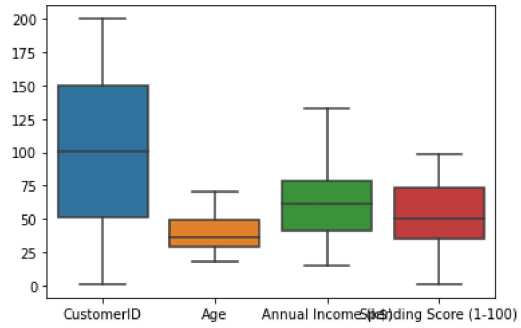
In [31]:
```
# Handle the Outlier
IQR=df['Annual Income (k$)'].quantile(0.75) - df['Annual Income (k$)'].quantile(0.25)
lower_fence = df['Annual Income (k$)'].quantile(0.25) - 1.5*IQR
upper_fence= df['Annual Income (k$)'].quantile(0.75) + 1.5*IQR

lower_fence , upper_fence
```

Out[31]: (-13.25, 132.75)

In [32]:
```
df['Annual Income (k$)']=np.where(df['Annual Income (k$)']>upper_fence , upper_fence ,np.where(df['Annual Income (k$)']<lower_fer
```

In [33]:
```
sns.boxplot(data=df , orient='v')
```

Out[33]:
```
<AxesSubplot:>
```



In [ ]:
```

```

Categorical to Numerical columns

In [34]:
```
# Gender columns to numerical columns
df=pd.get_dummies(df , columns=['Gender'])
```

In [35]:
```
# dropping the Coustomer ID
df.drop(['CustomerID'], axis=1)
```

Out[35]:

	Age	Annual Income (k$)	Spending Score (1-100)	Gender_Female	Gender_Male
0	19	15.00	39	0	1
1	21	15.00	81	0	1
2	20	16.00	6	1	0
3	23	16.00	77	1	0
4	31	17.00	40	1	0
...	...	...	...	...	...
195	35	120.00	79	1	0
196	45	126.00	28	1	0
197	32	126.00	74	0	1
198	32	132.75	18	0	1
199	30	132.75	83	0	1

200 rows × 5 columns

Standardization

In [36]:
```
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans

scaler=StandardScaler()
cl_scaler=scaler.fit_transform(df)
cl_scaler
```

Out[36]:
```
array([[-1.7234121 , -1.42456879, -1.74542941, -0.43480148, -1.12815215,
         1.12815215],
       [-1.70609137, -1.28103541, -1.74542941,  1.19570407, -1.12815215,
         1.12815215],
       [-1.68877065, -1.3528021 , -1.70708307, -1.71591298,  0.88640526,
        -0.88640526],
       ...,
       [ 1.68877065, -0.49160182,  2.51101403,  0.92395314, -1.12815215,
         1.12815215],
       [ 1.70609137, -0.49160182,  2.76985181, -1.25005425, -1.12815215,
         1.12815215],
       [ 1.7234121 , -0.6351352 ,  2.76985181,  1.27334719, -1.12815215,
         1.12815215]])
```

Non-Hierarchical - Distance - Centroid Based Clustering

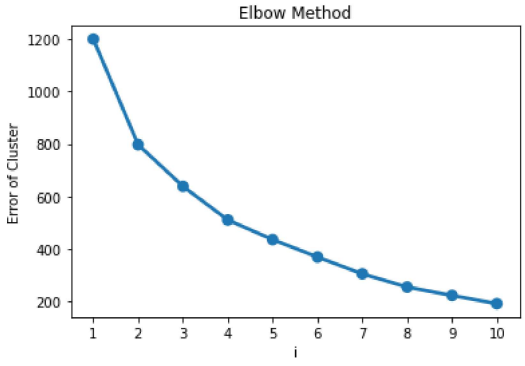
In [37]:
```
# finding the optimal No of clusters for K-Means implementation using WCSS(INertia)
```

In [38]:
```
wcss=[]

# The Elbow methods runs k-means clustering on datasets for a range of values between 1 to 11.
# Then for each values of k computes an average score for all cluster
```

In [156]:
```
wcss=[]
for i in range (1 , 11):
    model = KMeans(n_clusters= i , random_state=40)
    model.fit(cl_scaler)
    wcss.append(model.inertia_) # WCSS-- inertia
```

In [43]:
```
sns.pointplot(x=list(range(1,11)) , y=wcss)
plt.xlabel('i')
plt.ylabel("Error of Cluster")
plt.title(" Elbow Method ")
```

Out[43]:
```
Text(0.5, 1.0, ' Elbow Method ')
```



In [44]:
```
# i=3,4,5,6 centriod point
model =KMeans(n_clusters=3 , random_state=40)
model.fit(cl_scaler)
```

Out[44]:
```
KMeans(n_clusters=3, random_state=40)
```

In [45]:
```
model.labels_
```

Out[45]:
```
array([0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0,
       1, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1,
       1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0,
       1, 1, 0, 1, 0, 1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 1, 0, 1, 1,
       1, 1, 1, 0, 0, 1, 1, 0, 1, 2, 0, 0, 2, 1, 0, 0, 0, 2, 1, 0, 0, 0,
       0, 2, 2, 0, 2, 2, 2, 2, 2, 2, 0, 2, 2, 0, 2, 2, 0, 0, 0, 0, 0, 0,
       2, 2, 0, 2, 2, 0, 0, 2, 2, 0, 2, 2, 0, 0, 0, 2, 2, 0, 0, 0, 2, 2,
       2, 2, 0, 2, 0, 2, 2, 2, 0, 2, 0, 2, 0, 2, 2, 0, 0, 0, 0, 0, 2, 2,
       0, 0, 0, 0, 2, 2, 0, 2, 2, 0, 2, 0, 2, 2, 2, 2, 0, 2, 2, 2, 2, 0,
       0, 0])
```

In [46]:
```
df=df.assign(ClusterLabel=model.labels_)
```

In [47]:
```
df.groupby('ClusterLabel')[['Age','Annual Income (k$)' , 'Spending Score (1-100)' , 'Gender_Female' ,'Gender_Male']]
```

Out[47]:
```
<pandas.core.groupby.generic.DataFrameGroupBy object at 0x000001C17B010FA0>
```

In [48]:
```
from sklearn.metrics import silhouette_score
silhouette_score(df , model.labels_ , metric='euclidean')
```

Out[48]:
```
0.04697487175528268
```

In [51]:
```
#2.....
model=KMeans(n_clusters=4 , random_state=40)
model.fit(cl_scaler)
```

Out[51]:
```
KMeans(n_clusters=4, random_state=40)
```

In [53]:
```
model.labels_
```

Out[53]:
```
array([0, 0, 3, 3, 3, 3, 3, 3, 0, 3, 0, 3, 3, 3, 0, 0, 3, 0, 0, 3, 0, 0,
       3, 0, 3, 0, 3, 0, 3, 3, 0, 3, 0, 0, 3, 3, 3, 3, 3, 3, 3, 3, 0, 0, 3,
       3, 3, 3, 3, 3, 3, 3, 0, 3, 0, 3, 0, 3, 0, 3, 0, 0, 0, 3, 3, 0, 0,
       3, 3, 0, 3, 0, 3, 3, 3, 0, 0, 3, 0, 3, 3, 0, 0, 0, 3, 3, 0, 3, 3,
       3, 3, 3, 2, 0, 3, 3, 2, 3, 1, 0, 2, 1, 3, 0, 2, 0, 1, 3, 0, 0, 0,
       0, 1, 1, 2, 1, 1, 1, 1, 1, 1, 2, 1, 1, 2, 1, 1, 2, 2, 2, 2, 2, 2,
       1, 1, 2, 1, 1, 2, 2, 1, 1, 2, 1, 1, 2, 2, 2, 1, 1, 2, 2, 2, 1, 1,
       1, 1, 2, 1, 2, 1, 1, 1, 2, 1, 2, 1, 2, 1, 1, 2, 2, 2, 2, 1, 1,
       2, 2, 2, 2, 1, 1, 2, 1, 1, 2, 1, 2, 1, 1, 1, 1, 2, 1, 1, 1, 1, 2,
       2, 2])
```

In [54]:
```
df=df.assign(ClusterLabel=model.labels_)
```

In [55]:
```
df.groupby('ClusterLabel')[['Age','Annual Income (k$)' , 'Spending Score (1-100)' , 'Gender_Female' ,'Gender_Male']]
```

Out[55]:
```
<pandas.core.groupby.generic.DataFrameGroupBy object at 0x000001C17A3A37F0>
```

In [56]:
```
silhouette_score(df , model.labels_ , metric='euclidean')
```

Out[56]:
```
0.00016388579170994623
```

In [67]:
```
# 3.....

model=KMeans(n_clusters=5 ,random_state=40)
model.fit(cl_scaler)
```

Out[67]:
```
KMeans(n_clusters=5, random_state=40)
```

In [68]:
```
model.labels_
```

Out[68]:
```
array([4, 4, 3, 3, 3, 3, 3, 3, 0, 3, 0, 3, 3, 3, 4, 4, 3, 4, 0, 3, 4, 4,
       3, 4, 3, 4, 3, 4, 3, 3, 0, 3, 0, 4, 3, 3, 3, 3, 3, 3, 3, 4, 0, 3,
       3, 3, 3, 3, 3, 3, 3, 4, 3, 0, 3, 0, 3, 0, 3, 0, 0, 4, 3, 3, 0, 4,
       3, 3, 4, 3, 0, 3, 3, 3, 0, 4, 3, 0, 3, 3, 0, 4, 0, 3, 3, 0, 3, 3,
       3, 3, 3, 4, 0, 3, 3, 4, 3, 2, 0, 4, 2, 2, 0, 4, 0, 2, 2, 0, 0, 0,
       0, 2, 2, 1, 2, 2, 2, 2, 2, 2, 1, 2, 2, 1, 2, 2, 1, 1, 0, 1, 0, 1,
       2, 2, 1, 2, 2, 1, 1, 2, 2, 1, 2, 2, 1, 1, 1, 2, 2, 1, 1, 1, 2, 2,
       2, 2, 1, 2, 1, 2, 2, 2, 1, 2, 1, 2, 1, 2, 2, 1, 1, 1, 1, 1, 2, 2,
       1, 1, 1, 1, 2, 2, 1, 2, 2, 1, 2, 1, 2, 2, 2, 2, 1, 2, 2, 2, 2, 1,
       1, 1])
```

In [69]:
```
df=df.assign(ClusterLabel=model.labels_)
```

In [70]:
```
df.groupby('ClusterLabel')[['Age','Annual Income (k$)' , 'Spending Score (1-100)' , 'Gender_Female' ,'Gender_Male']]
```

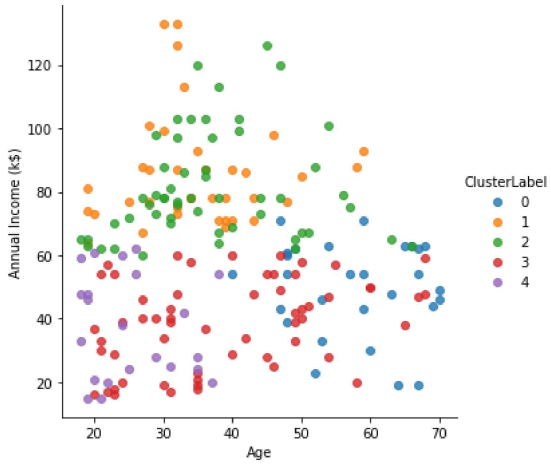
Out[70]:
```
<pandas.core.groupby.generic.DataFrameGroupBy object at 0x000001C17A41C580>
```

In [71]:
```
silhouette_score(df , model.labels_ , metric='euclidean')
```

Out[71]:
```
-0.010208990006408493
```

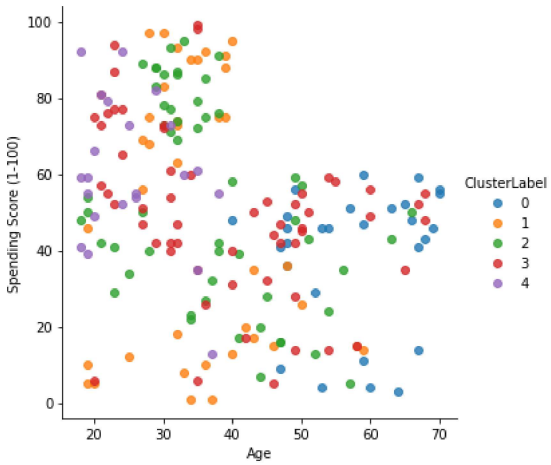
In [76]:
```
plt.figure(figsize=(12,6))  # number of centeriod--ClusterLabel
sns.lmplot(data=df , x='Age' , y='Annual Income (k$)' , hue='ClusterLabel',fit_reg=False , legend=True , legend_out=True)
```

Out[76]:
```
<seaborn.axisgrid.FacetGrid at 0x1c17dbc8280>

<Figure size 864x432 with 0 Axes>
```



In [75]:
```
plt.figure(figsize=(12,6))
sns.lmplot(data=df,x='Age' , y='Spending Score (1-100)' , hue='ClusterLabel' , fit_reg=False , legend=True , legend_out=True)
```

Out[75]:
```
<seaborn.axisgrid.FacetGrid at 0x1c17e4e6160>

<Figure size 864x432 with 0 Axes>
```



DB SCAN ---Density-Based Spatial Clustering of Applications with Noise

Density-Based Spatial Clustering of Applications with Noise (DBSCAN) is a base algorithm for density-based clustering. It can discover clusters of different shapes and sizes from a large amount of data, which is containing noise and outliers.

DBSCAN is a clustering method that is used in machine learning to separate clusters of high density from clusters of low density.

In [77]:
```
df1=pd.read_csv("https://raw.githubusercontent.com/NelakurthiSudheer/Mall-Customers-Segmentation/main/Dataset/Mall_Customers.csv"
```

In [78]:
```
df1.head()
```

Out[78]:

	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 [79]:
```
df1.drop(['CustomerID'],axis=1 , inplace=True)
```

In [85]:
```
df1=pd.get_dummies(df1 , columns=['Gender'])
```

In [86]:
```
df1.head(2)
```

Out[86]:

	Age	Annual Income (k$)	Spending Score (1-100)	Gender_Female	Gender_Male
0	19	15	39	0	1
1	21	15	81	0	1

In [87]:
```
from sklearn.cluster import DBSCAN
import numpy as np
```

In [110]:
```
DB_scan=DBSCAN(eps=12.5 , min_samples=4).fit(df1)

DB_cluster=df1.copy()
DB_cluster.loc[:,'Cluster']=DB_scan.labels_
```

In [111]:
```
y_pred=DB_scan.fit_predict(df)
```

In [112]:
```
y_pred
```

Out[112]:
```
array([-1,  0, -1,  0, -1,  0, -1, -1, -1,  0, -1, -1, -1,  0, -1,  0, -1,
        0, -1, -1, -1,  0, -1,  0,  1,  0, -1, -1, -1,  0, -1,  0,  1,  0,
        1,  0,  1,  0, -1,  0, -1,  0, -1,  0, -1,  0,  2,  0,  0,  0,  2,
        0,  0,  2,  2,  2,  2,  2,  0,  2,  2,  3,  2,  2,  2,  3,  2,  2,
        3,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2, -1,  2,  2,
        2,  2,  2,  2,  2, -1,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,
        2,  2,  2,  2,  2,  2,  2,  2,  2,  2, -1,  2,  2,  2,  2, -1,  2,
       -1,  2, -1, -1, -1, -1,  4, -1, -1, -1,  4, -1,  4, -1,  4, -1,  4,
       -1,  4, -1,  4, -1,  4, -1,  4, -1,  4, -1,  4,  5,  4,  5,  4,  5,
        4,  5,  4, -1,  4, -1,  4, -1,  4, -1,  4,  6,  4,  6,  4,  6, -1,
        6,  7, -1,  7, -1,  7, -1,  7, -1,  7, -1,  7, -1,  7, -1,  7, -1,
       -1, -1,  7, -1, -1, -1,  7, -1, -1, -1, -1, -1, -1], dtype=int64)
```

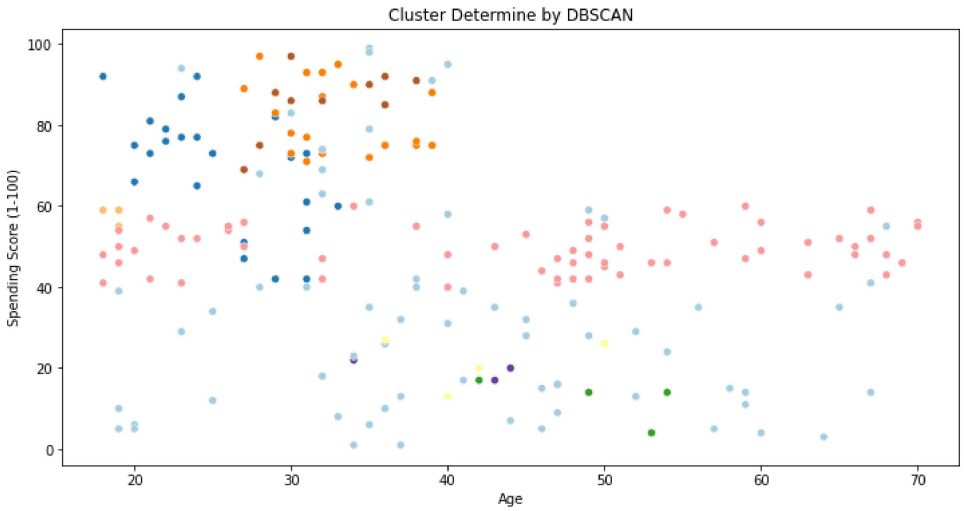
In [113]:
```
# Assigning Cluster label to Dataset
df1=df1.assign(ClusterLabel=DB_scan.labels_)
```

In [114]:
```
df1.groupby('ClusterLabel')[['Age' , 'Annual Income (k$)' ,'Spending Score (1-100)' ,'Gender_Female', 'Gender_Male']]
```

Out[114]:
```
<pandas.core.groupby.generic.DataFrameGroupBy object at 0x000001C10615A4C0>
```

In [115]:
```
# Cluster PLot of Age vs Spending Score

plt.figure(figsize=(12,6))
sns.scatterplot( x=df1['Age'] , y=df1['Spending Score (1-100)'] ,c=y_pred , cmap='Paired')
plt.title('Cluster Determine by DBSCAN')
```

Out[115]:
```
Text(0.5, 1.0, 'Cluster Determine by DBSCAN')
```



In [116]:
```
silhouette_score(df1,DB_scan.labels_)
```

Out[116]:
```
-0.03935257321229305
```

In [120]:
```
# No of Point in a cluster
DBSCAN_clu_size=DB_cluster.groupby('Cluster').size().to_frame()
DBSCAN_clu_size.columns=['DBSCAN_size']
DBSCAN_clu_size
```

Out[120]:

Cluster	DBSCAN_size
-1	27
0	109
1	8
2	33
3	4
4	4
5	9
6	6

observation

-1 is assigned to the point which are considered to be Noise . Therefore we have 27 Noise point here

ALGOMERATIVE Clustering

Agglomerative Clustering is a type of hierarchical clustering algorithm. It is an unsupervised machine learning technique that divides the population into several clusters such that data points in the same cluster are more similar and data points in different clusters are dissimilar. Agglomerative: This is a bottom-up approach: each observation starts in its own cluster, and pairs of clusters are merged as one moves up the hierarchy. Divisive: This is a top-down approach: all observations start in one cluster, and splits are performed recursively as one moves down the hierarchy.

Linkage Methods

Single Linkage: Minimum Distance between instances of two considered clusters. The distance calculation can be either Euclidean or Manhattan Distance

Complete Linkage: The farthest distance between the instances of two class.

Average Linkage: The average distance between all linkages is considered.

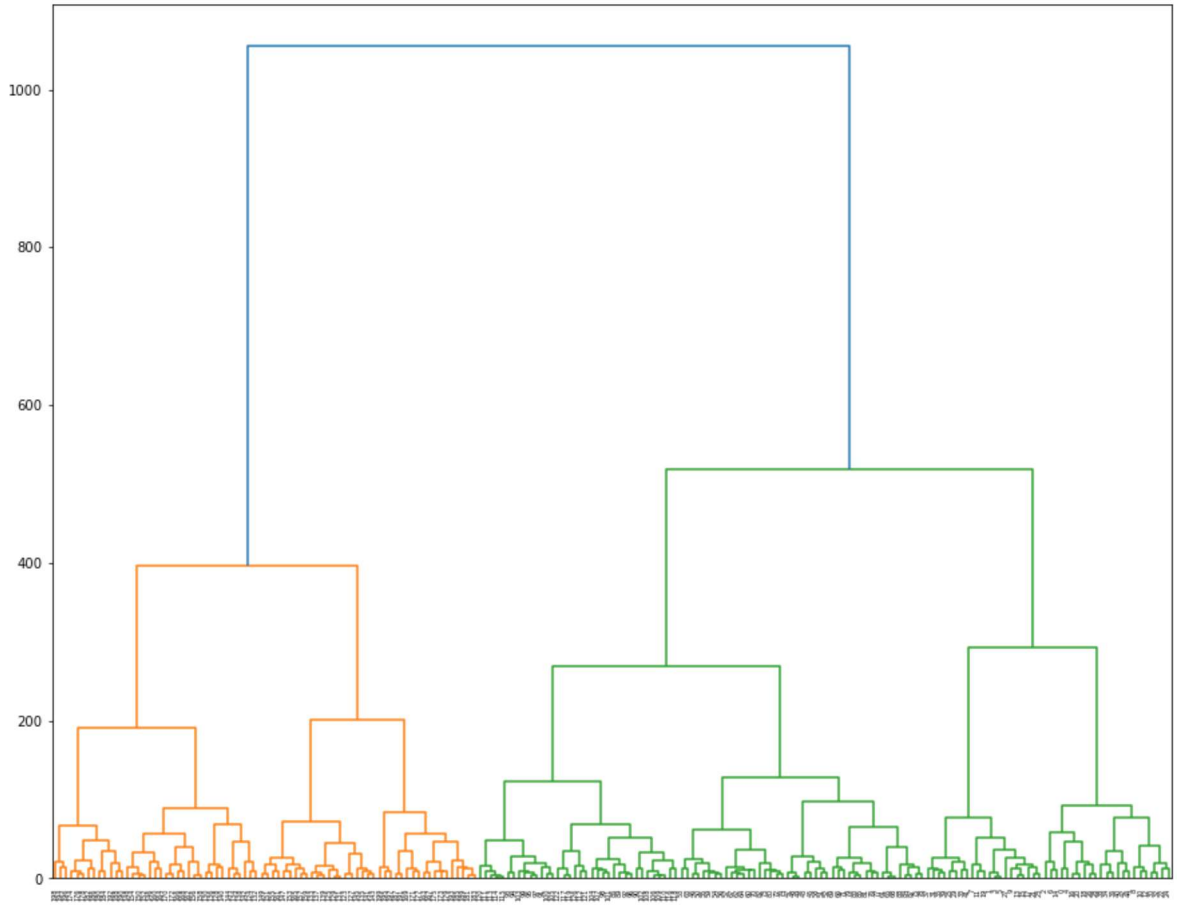
Centroid Linkage: The distance between the two centroid is considered.

Ward Linkage: The distance between two clusters say A and B are how much the sum of squares will increase when they both are merged.

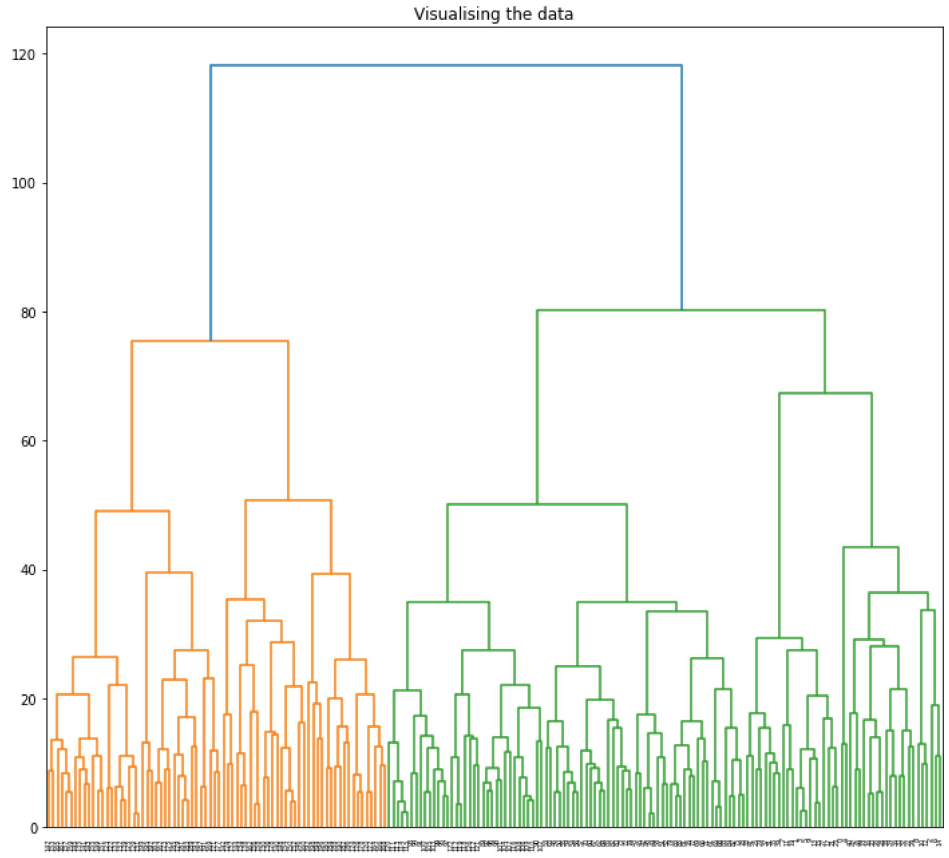
In [128]:
```
# Importing Libraries
from sklearn.decomposition import PCA
from sklearn.cluster import AgglomerativeClustering
from sklearn.preprocessing import StandardScaler, normalize
from sklearn.metrics import silhouette_score
import scipy.cluster.hierarchy as shc
```

Dendrogram Visualization using different linkage method

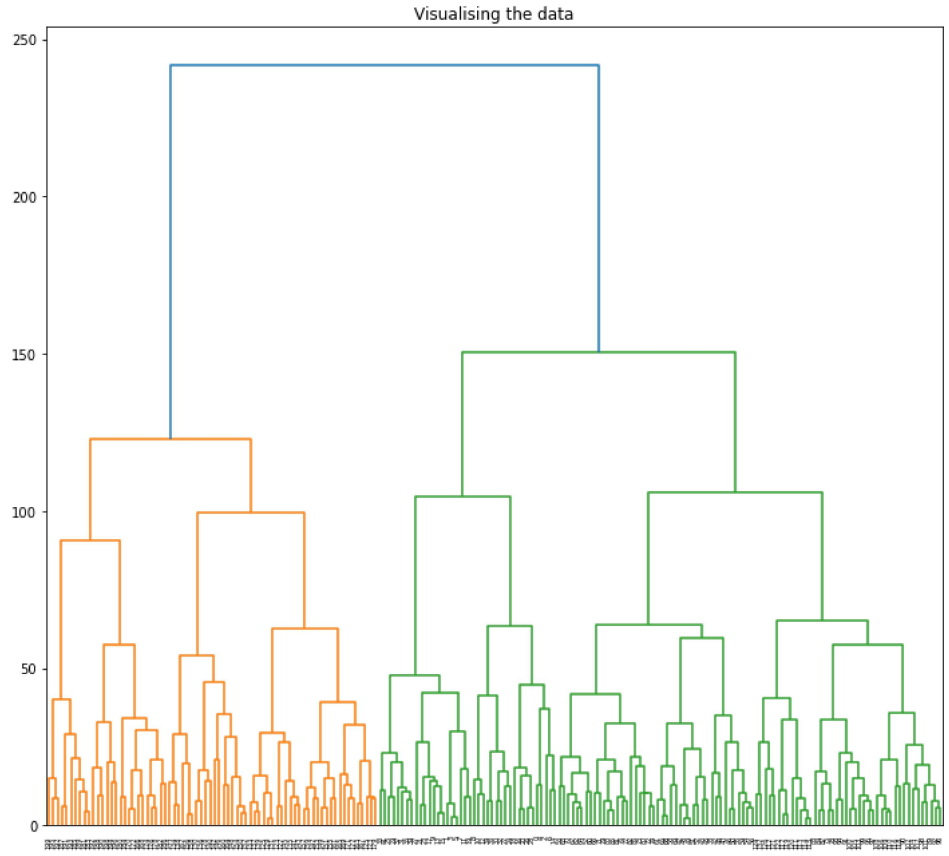
In [147]:
```
# ward linkage mathod

plt.figure(figsize=(15,12))
Dendrogram_ward= shc.dendrogram((shc.linkage(df,metric='euclidean' , method='ward')))
```



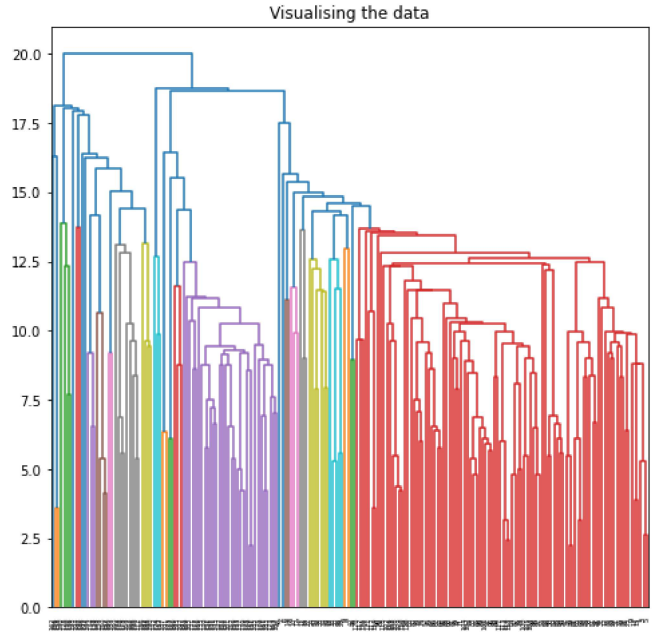
In [150]:
```
# AVERAGE LINKAGE METHOD
plt.figure(figsize =(12, 11))
plt.title('Visualising the data')
Dendrogram_average = shc.dendrogram((shc.linkage(df,metric='euclidean', method ='average')))
```



In [149]:
```
#COMPLETE LINKAGE METHOD
plt.figure(figsize =(12, 11))
plt.title('Visualising the data')
Dendrogram_complete = shc.dendrogram((shc.linkage(df, metric='euclidean',method ='complete')))
```

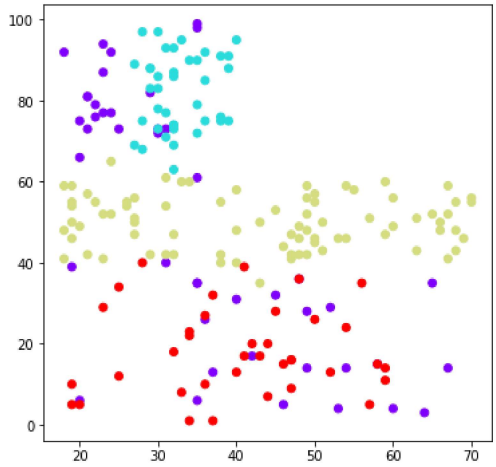


In [152]:
```
#SINGLE LINKAGE METHOD
plt.figure(figsize =(8, 8))
plt.title('Visualising the data')
Dendrogram_single = shc.dendrogram((shc.linkage(df,metric='euclidean', method ='single')))
```



In [ ]:

In [153]:
```
#  PLOT WITH 4 Cluster
# AGLOMERATIVE WITH 4 CLUSTERS
ac2 = AgglomerativeClustering(n_clusters = 4)

# Visualizing the clustering
plt.figure(figsize =(6, 6))
plt.scatter(df['Age'], df['Spending Score (1-100)'],
           c = ac2.fit_predict(df), cmap ='rainbow')
plt.show()
```



In [154]:
```
ac2.labels_
```

Out[154]:
```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2,
       0, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 3, 1, 2, 1, 3, 1, 3, 1,
       3, 1, 3, 1, 3, 1, 3, 1, 3, 1, 3, 1, 3, 1, 3, 1, 3, 1, 3, 1, 3, 1,
       3, 1, 3, 1, 3, 1, 3, 1, 3, 1, 3, 1, 3, 1, 3, 1, 3, 1, 3, 1, 3, 1,
       3, 1, 3, 1, 3, 1, 3, 1, 3, 1, 3, 1, 3, 1, 3, 1, 3, 1, 3, 1, 3, 1,
       3, 1], dtype=int64)
```

In [155]:
```
silhouette_score(df, ac2.labels_)
```

Out[155]:
```
0.41594677186060464
```

SUBMITTED By: Mukul Singh

Thank You

In [ ]: