JawadAhmed_20P-0165

March 20, 2023

Name: Jawad Ahmed, Roll No: 20P-0165, Section: BCS-6A, Lab Task 7 (K-Mean Clustering) # 1. Load the customer segmentation dataset.

[1]:
```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
from sklearn.feature_selection import SelectKBest, f_regression
```

[2]:
```python
customer_data = pd.read_csv("Cust_Segmentation.csv")
```

[3]:
```python
customer_data
```

[3]:
```
     Customer Id  Age  Edu  Years Employed  Income  Card Debt  Other Debt  \
0              1   41    2               6      19      0.124       1.073
1              2   47    1              26     100      4.582       8.218
2              3   33    2              10      57      6.111       5.802
3              4   29    2               4      19      0.681       0.516
4              5   47    1              31     253      9.308       8.908
..           …  …    …             …     …        …         …
845          846   27    1               5      26      0.548       1.220
846          847   28    2               7      34      0.359       2.021
847          848   25    4               0      18      2.802       3.210
848          849   32    1              12      28      0.116       0.696
849          850   52    1              16      64      1.866       3.638

     Defaulted Address  DebtIncomeRatio
0          0.0  NBA001              6.3
1          0.0  NBA021             12.8
2          1.0  NBA013             20.9
3          0.0  NBA009              6.3
4          0.0  NBA008              7.2
..         …     …              …
845        NaN  NBA007              6.8
846        0.0  NBA002              7.0
847        1.0  NBA001             33.4
848        0.0  NBA012              2.9
```

```
849         0.0   NBA025              8.6

[850 rows x 10 columns]
```

[4]:
```
customer_data.shape
```

[4]:
```
(850, 10)
```

[5]:
```
customer_data.describe()
```

[5]:
```
       Customer Id         Age         Edu  Years Employed      Income  \
count    850.00000  850.000000  850.000000      850.000000  850.000000
mean     425.50000   35.029412    1.710588        8.565882   46.675294
std      245.51816    8.041432    0.927784        6.777884   38.543054
min        1.00000   20.000000    1.000000        0.000000   13.000000
25%      213.25000   29.000000    1.000000        3.000000   24.000000
50%      425.50000   34.000000    1.000000        7.000000   35.000000
75%      637.75000   41.000000    2.000000       13.000000   55.750000
max      850.00000   56.000000    5.000000       33.000000  446.000000

        Card Debt  Other Debt   Defaulted  DebtIncomeRatio
count  850.000000  850.000000  700.000000       850.000000
mean     1.576820    3.078773    0.261429        10.171647
std      2.125843    3.398799    0.439727         6.719441
min      0.012000    0.046000    0.000000         0.100000
25%      0.382500    1.045750    0.000000         5.100000
50%      0.885000    2.003000    0.000000         8.700000
75%      1.898500    3.903250    1.000000        13.800000
max     20.561000   35.197000    1.000000        41.300000
```

[6]:
```
customer_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 850 entries, 0 to 849
Data columns (total 10 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   Customer Id      850 non-null    int64
 1   Age              850 non-null    int64
 2   Edu              850 non-null    int64
 3   Years Employed   850 non-null    int64
 4   Income           850 non-null    int64
 5   Card Debt        850 non-null    float64
 6   Other Debt       850 non-null    float64
 7   Defaulted        700 non-null    float64
 8   Address          850 non-null    object
 9   DebtIncomeRatio  850 non-null    float64
```

```
dtypes: float64(4), int64(5), object(1)
memory usage: 66.5+ KB
```

1 2. Clean the data by removing any duplicates, and missing values.

[7]:
```
# Check for duplicates
# No duplicates in the data
duplicates = customer_data[customer_data.duplicated()]

if duplicates.empty:
    print("No duplicates found.")
else:
    print("Duplicates found:")
    print(duplicates)
```

```
No duplicates found.
```

[8]:
```
# Check for missing values
missing_values = customer_data.isnull().sum()

if missing_values.sum() == 0:
    print("No missing values found.")
else:
    print("Missing values found:")
    print(missing_values)
```

```
Missing values found:
Customer Id          0
Age                  0
Edu                  0
Years Employed       0
Income               0
Card Debt            0
Other Debt           0
Defaulted          150
Address              0
DebtIncomeRatio      0
dtype: int64
```

[9]:
```
# remove missing values from defaulted column
customer_data = customer_data.dropna(subset=['Defaulted'])
```

[10]:
```
customer_data
```

[10]:
```
     Customer Id  Age  Edu  Years Employed  Income  Card Debt  Other Debt  \
0              1   41    2               6      19      0.124       1.073
```

```
1              2   47   1              26     100      4.582        8.218
2              3   33   2              10      57      6.111        5.802
3              4   29   2               4      19      0.681        0.516
4              5   47   1              31     253      9.308        8.908
..           …   …   …            …      …       …        …
844          845   41   1               7      43      0.694        1.198
846          847   28   2               7      34      0.359        2.021
847          848   25   4               0      18      2.802        3.210
848          849   32   1              12      28      0.116        0.696
849          850   52   1              16      64      1.866        3.638

     Defaulted Address  DebtIncomeRatio
0          0.0  NBA001              6.3
1          0.0  NBA021             12.8
2          1.0  NBA013             20.9
3          0.0  NBA009              6.3
4          0.0  NBA008              7.2
..         …      …            …
844        0.0  NBA011              4.4
846        0.0  NBA002              7.0
847        1.0  NBA001             33.4
848        0.0  NBA012              2.9
849        0.0  NBA025              8.6

[700 rows x 10 columns]
```

[11]:
```
customer_data.dtypes
```

[11]:
```
Customer Id          int64
Age                  int64
Edu                  int64
Years Employed       int64
Income               int64
Card Debt          float64
Other Debt         float64
Defaulted          float64
Address             object
DebtIncomeRatio    float64
dtype: object
```

2 3. Preprocess the data by scaling the features to ensure they are on the same scale. You can use standardization or normalization techniques for this step.

[12]:
```
# Convert the Address from object datatype to numeric
customer_data['Address'] = pd.to_numeric(customer_data['Address'],␣
↪errors='coerce')
```

```
/home/jad/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:1:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  """Entry point for launching an IPython kernel.
```

[13]:
```
customer_data.dtypes
```

[13]:
```
Customer Id          int64
Age                  int64
Edu                  int64
Years Employed       int64
Income               int64
Card Debt          float64
Other Debt         float64
Defaulted          float64
Address            float64
DebtIncomeRatio    float64
dtype: object
```

3 # Standardize the features

[15]:
```
scaler = StandardScaler()
customer_data_std = scaler.fit_transform(customer_data)
# Convert the standardized features back to a dataframe
columns = customer_data.columns
customer_data_std_df = pd.DataFrame(customer_data_std, columns=columns)
```

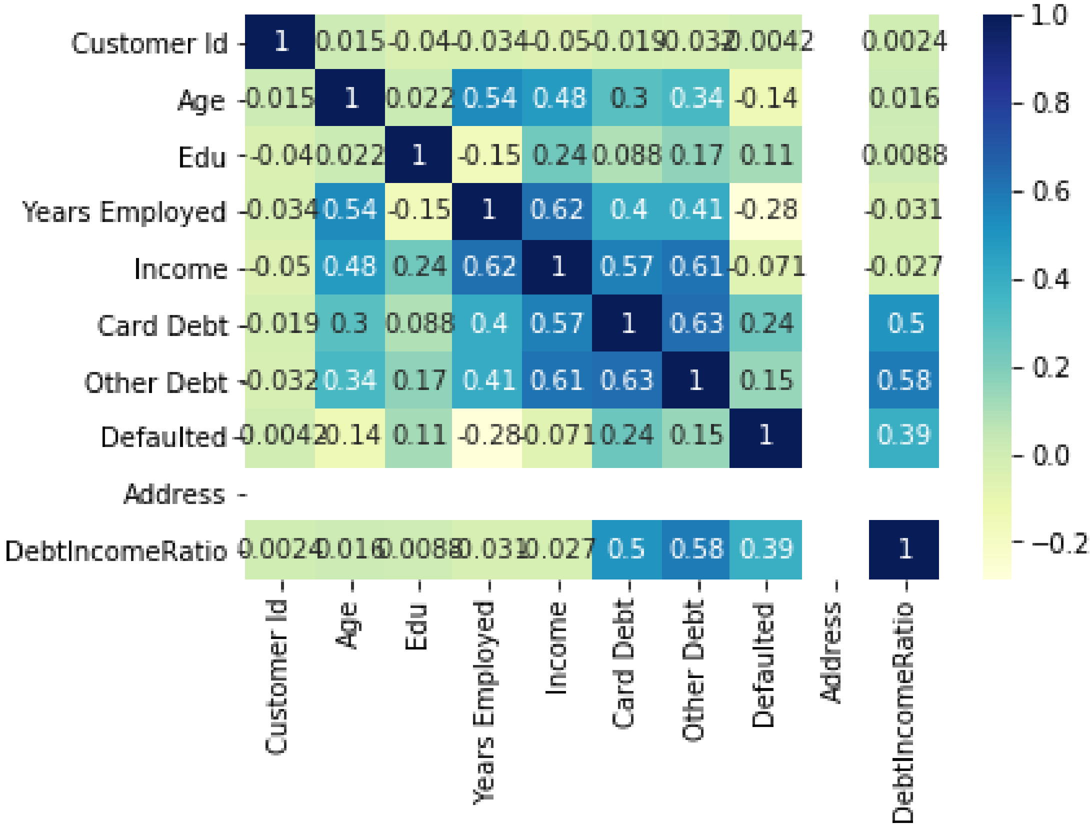
```
/home/jad/anaconda3/lib/python3.7/site-packages/sklearn/utils/extmath.py:765:
RuntimeWarning: invalid value encountered in true_divide
  updated_mean = (last_sum + new_sum) / updated_sample_count
/home/jad/anaconda3/lib/python3.7/site-packages/sklearn/utils/extmath.py:706:
RuntimeWarning: Degrees of freedom <= 0 for slice.
  result = op(x, *args, **kwargs)
```

[16]:
```
customer_data_std_df
```

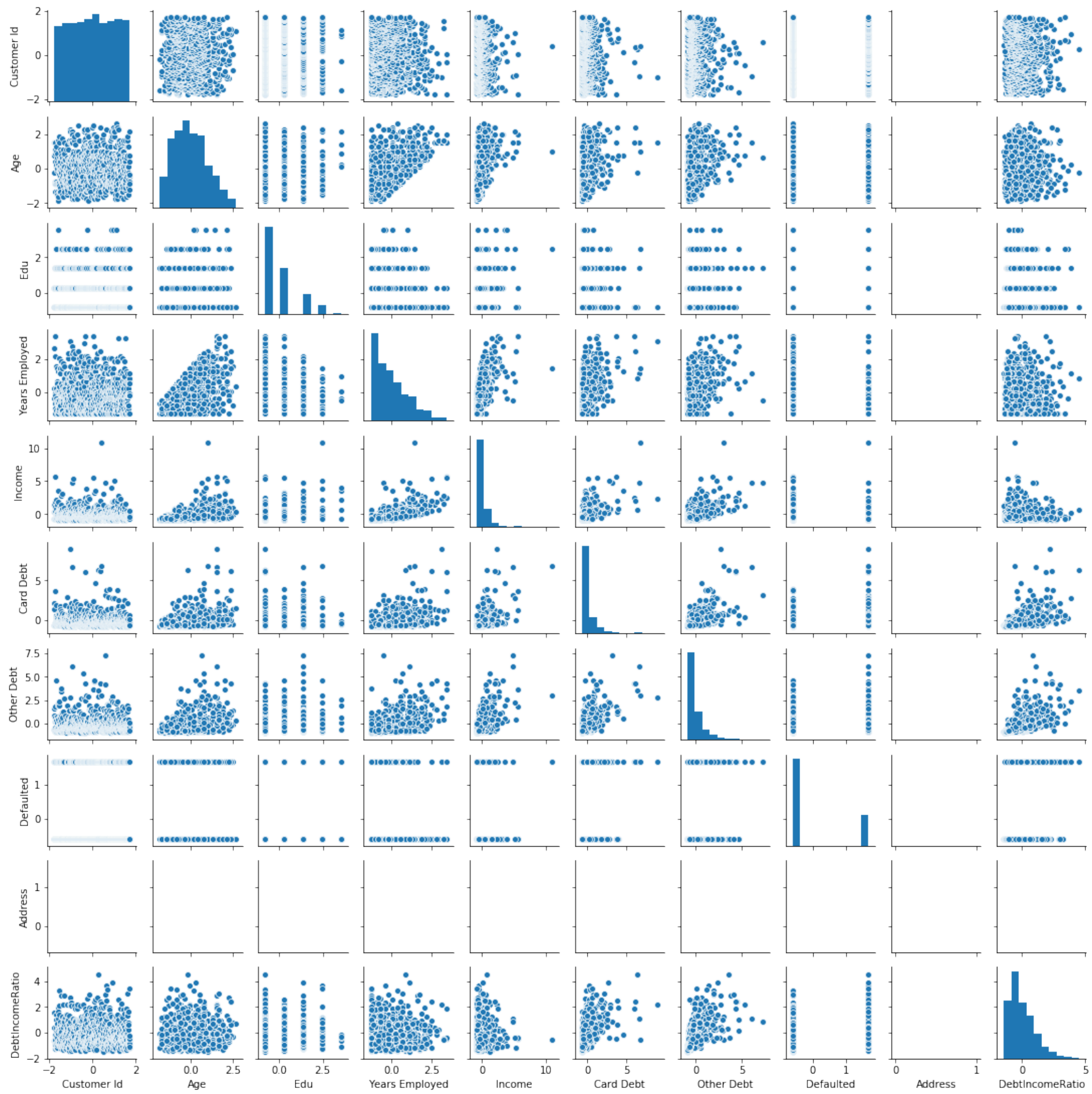
[16]:
```
      Customer Id       Age       Edu  Years Employed    Income  Card Debt  \
0       -1.766243  0.768304  0.298793       -0.359007 -0.723102  -0.675699
1       -1.762130  1.519090 -0.779325        2.647029  1.478707   1.431421
2       -1.758018 -0.232744  0.298793        0.242201  0.309845   2.154119
3       -1.753905 -0.733267  0.298793       -0.659610 -0.723102  -0.412427
4       -1.749792  1.519090 -0.779325        3.398538  5.637681   3.665215
..            …         …         …              …         …          …
695      1.704870  0.768304 -0.779325       -0.208705 -0.070714  -0.406283
696      1.713095 -0.858398  0.298793       -0.208705 -0.315360  -0.564624
697      1.717208 -1.233791  2.455029       -1.260817 -0.750285   0.590086
698      1.721321 -0.357875 -0.779325        0.542804 -0.478457  -0.679481
699      1.725434  2.144745 -0.779325        1.144011  0.500125   0.147675

     Other Debt  Defaulted  Address  DebtIncomeRatio
0     -0.604284  -0.594950      NaN        -0.580528
1      1.570620  -0.594950      NaN         0.372222
2      0.835201   1.680814      NaN         1.559495
3     -0.773833  -0.594950      NaN        -0.580528
4      1.780653  -0.594950      NaN        -0.448609
..          …          …     …              …
695   -0.566235  -0.594950      NaN        -0.859025
696   -0.315718  -0.594950      NaN        -0.477925
697    0.046209   1.680814      NaN         3.391707
698   -0.719041  -0.594950      NaN        -1.078890
699    0.176490  -0.594950      NaN        -0.243401

[700 rows x 10 columns]
```

4 4. Select the relevant features that are most important in determining customer behavior.

[18]:
```python
import seaborn as sns

# .corr() method will get rid of the columns that are not suited for␣
↪correlation for you
sns.heatmap(customer_data_std_df.corr(), annot=True, cmap="YlGnBu")
plt.show()
```



[19]:
```
sns.pairplot(customer_data_std_df, height=1.5)
# Corelation is not the only thing but if any feature is positively or
 ↪negatively highly corelated with survival then
# you can say that is important feature
plt.show()
```



[20]:
```
customer_data_std_df.columns
```

[20]:
```
Index(['Customer Id', 'Age', 'Edu', 'Years Employed', 'Income', 'Card Debt',
       'Other Debt', 'Defaulted', 'Address', 'DebtIncomeRatio'],
      dtype='object')
```

5 Selecting the Best Feature using PCA

[22]:
```
from sklearn.decomposition import PCA
```

[23]:
```
customer_data_std_df = customer_data_std_df.drop('Address', axis=1)
```

[25]:
```python
# Instantiate the PCA object with 2 components
pca = PCA(n_components=2)

# Fit and transform the data
X_pca = pca.fit_transform(customer_data_std_df)

# Create a new DataFrame with the reduced dimensions and selected best 2␣
↪features
df_pca = pd.DataFrame(data=X_pca, columns=['PC1', 'PC2'])
```

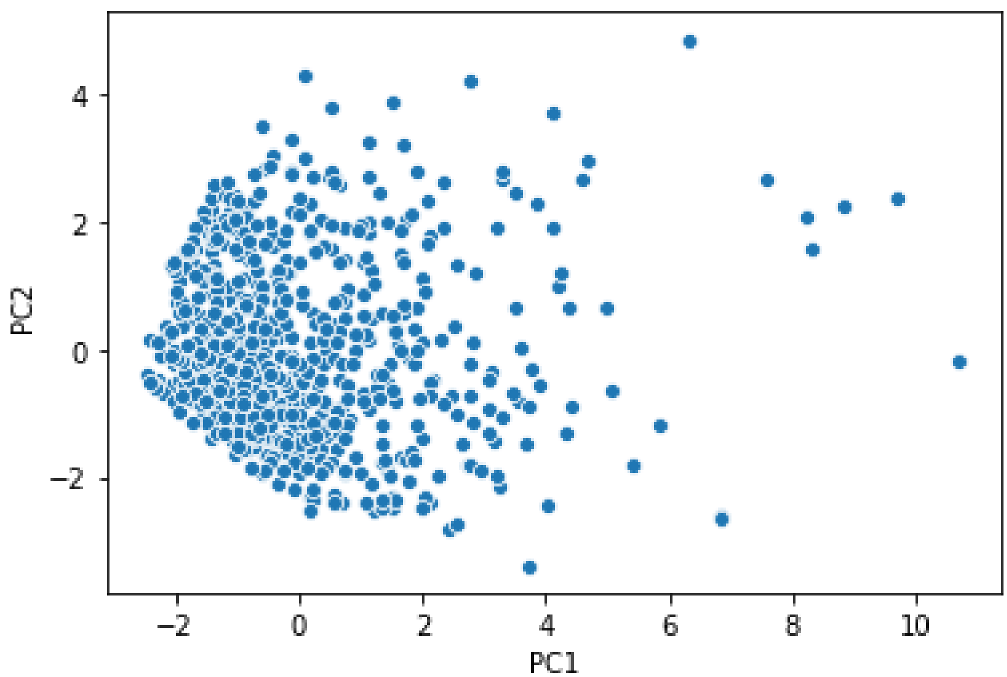
[26]:
```python
df_pca
```

[26]:
```
          PC1       PC2
0   -0.903986 -0.801849
1    3.701303 -1.480005
2    2.083050  2.337761
3   -1.501904 -0.207251
4    6.850336 -2.591303
..        …         …
695 -0.660098 -1.233261
696 -1.101576 -0.311298
697  0.098533  4.298702
698 -1.199449 -1.308140
699  1.369928 -1.718054

[700 rows x 2 columns]
```

6 Visualize the result

[28]:
```python
sns.scatterplot(x='PC1', y='PC2', data=df_pca)
```

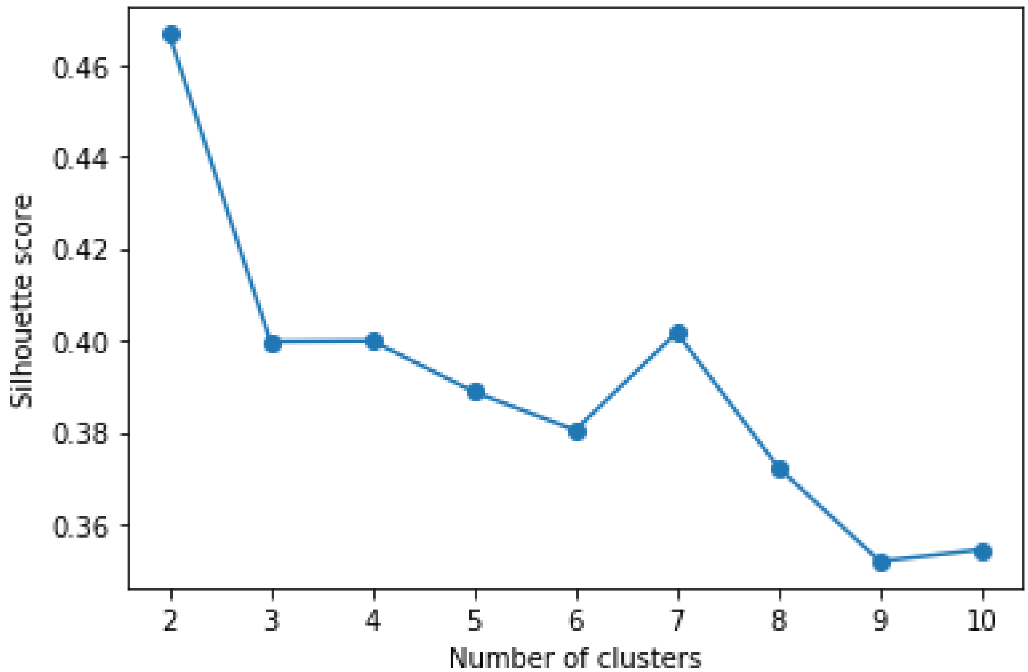
[28]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f13b178a610>
```



7 5. Apply K-means clustering to the preprocessed and selected features to identify customer segments with similar behavior and demographics. Choose the optimal number of clusters using techniques like the elbow method.

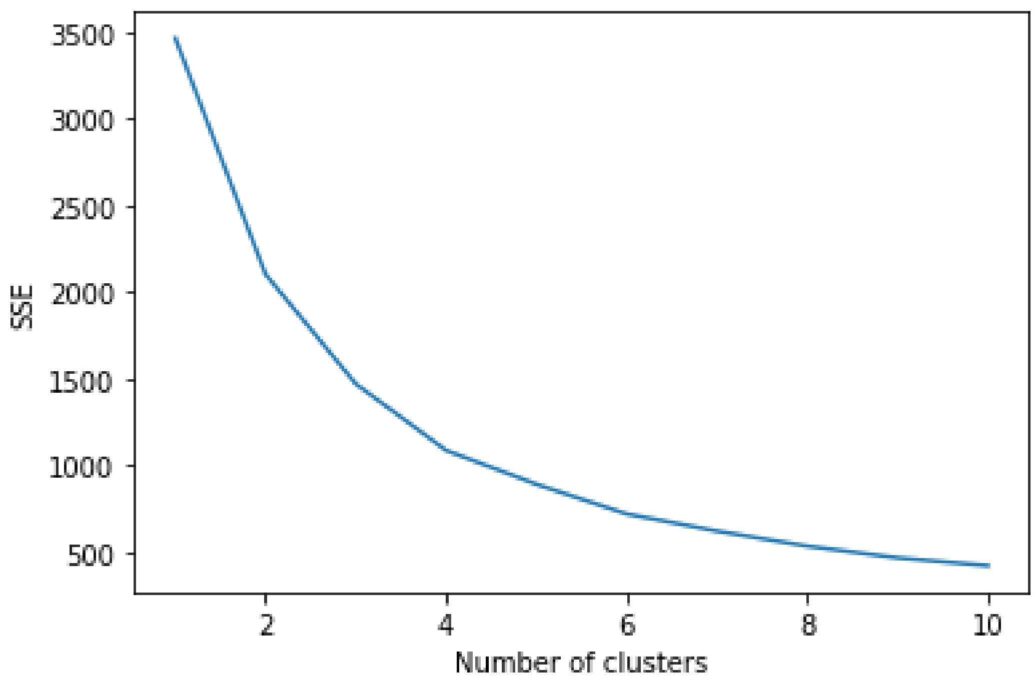
[29]:

```
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
# Apply K-means clustering with different numbers of clusters
scores = []
for k in range(2, 11):
    kmeans = KMeans(n_clusters=k)
    kmeans.fit(df_pca)
    score = silhouette_score(df_pca, kmeans.labels_)
    scores.append(score)

# Plot the results
plt.plot(range(2, 11), scores, marker='o')
plt.xlabel('Number of clusters')
plt.ylabel('Silhouette score')
plt.show()
```



8 Plotting Sum of Square Error

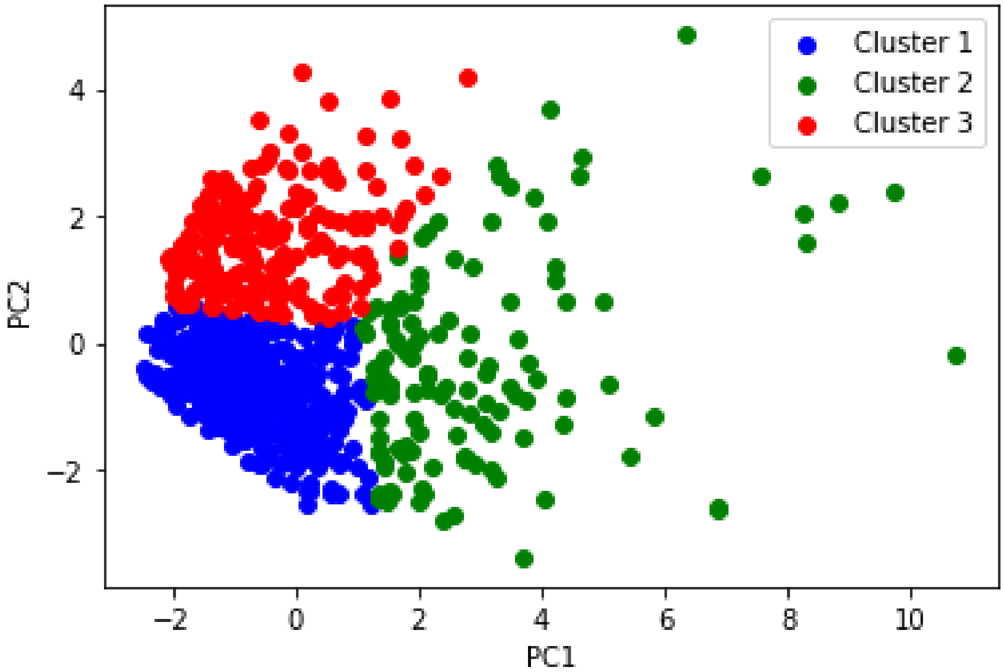
[30]:
```
# Determine the optimal number of clusters using the elbow method
sse = []
for k in range(1, 11):
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(df_pca)
    sse.append(kmeans.inertia_)

# Plot the SSE for different values of k
plt.plot(range(1, 11), sse)
plt.xlabel('Number of clusters')
plt.ylabel('SSE')
plt.show()
```



9 6. Visualize the resulting clusters using techniques like scatter plots.

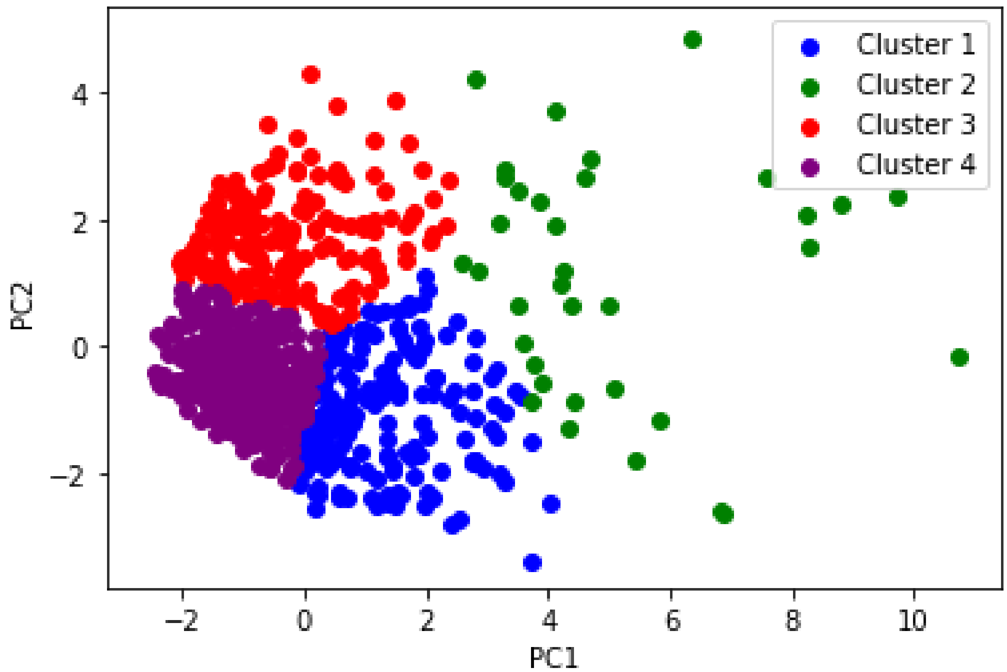
10 Apply K-means clustering with k=3

[35]:
```
kmeans = KMeans(n_clusters=3, random_state=42)
kmeans.fit(df_pca)
labels = kmeans.labels_

# Visualize the clusters using scatter plots
plt.scatter(df_pca['PC1'][labels==0], df_pca['PC2'][labels==0], c='blue',␣
 ↪label='Cluster 1')
plt.scatter(df_pca['PC1'][labels==1], df_pca['PC2'][labels==1], c='green',␣
 ↪label='Cluster 2')
plt.scatter(df_pca['PC1'][labels==2], df_pca['PC2'][labels==2], c='red',␣
 ↪label='Cluster 3')
plt.legend()
plt.xlabel('PC1')
plt.ylabel('PC2')
plt.show()
```



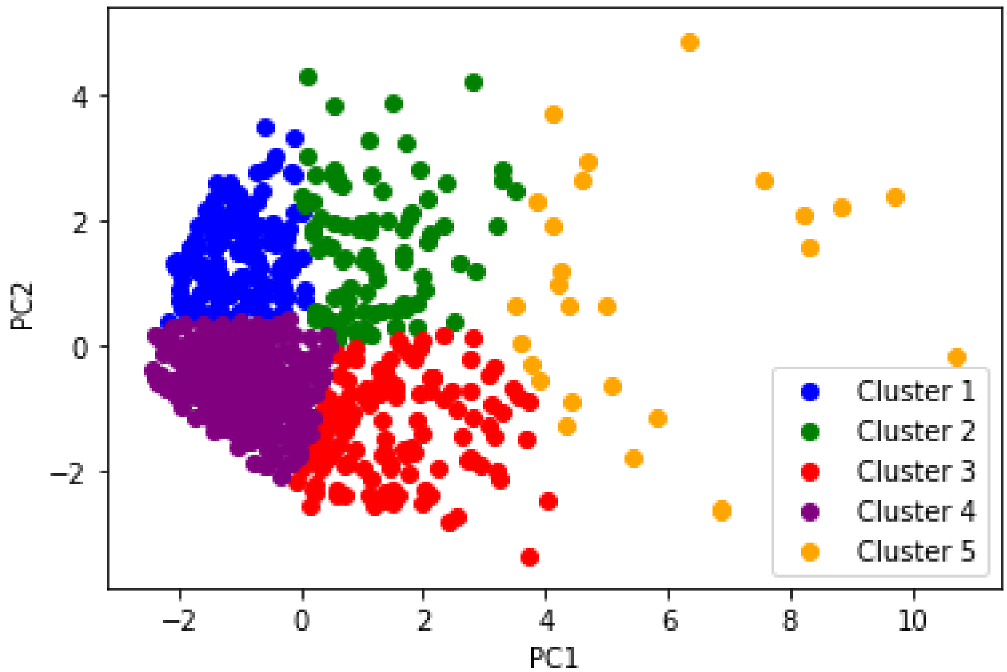
11 Apply K-means clustering with k=4

[37]:
```
kmeans = KMeans(n_clusters=4, random_state=42)
kmeans.fit(df_pca)
labels = kmeans.labels_

# Visualize the clusters using scatter plots
plt.scatter(df_pca['PC1'][labels==0], df_pca['PC2'][labels==0], c='blue',␣
 ↪label='Cluster 1')
plt.scatter(df_pca['PC1'][labels==1], df_pca['PC2'][labels==1], c='green',␣
 ↪label='Cluster 2')
plt.scatter(df_pca['PC1'][labels==2], df_pca['PC2'][labels==2], c='red',␣
 ↪label='Cluster 3')
plt.scatter(df_pca['PC1'][labels==3], df_pca['PC2'][labels==3], c='purple',␣
 ↪label='Cluster 4')
plt.legend()
plt.xlabel('PC1')
plt.ylabel('PC2')
plt.show()
```



12 Apply K-means clustering with k=5

[41]:
```
kmeans = KMeans(n_clusters=5, random_state=42)
kmeans.fit(df_pca)
labels = kmeans.labels_

# Visualize the clusters using scatter plots
plt.scatter(df_pca['PC1'][labels==0], df_pca['PC2'][labels==0], c='blue',␣
 ↪label='Cluster 1')
plt.scatter(df_pca['PC1'][labels==1], df_pca['PC2'][labels==1], c='green',␣
 ↪label='Cluster 2')
plt.scatter(df_pca['PC1'][labels==2], df_pca['PC2'][labels==2], c='red',␣
 ↪label='Cluster 3')
plt.scatter(df_pca['PC1'][labels==3], df_pca['PC2'][labels==3], c='purple',␣
 ↪label='Cluster 4')
plt.scatter(df_pca['PC1'][labels==4], df_pca['PC2'][labels==4], c='orange',␣
 ↪label='Cluster 5')
plt.legend()
plt.xlabel('PC1')
plt.ylabel('PC2')
plt.show()
```



13 Q1. When should we split the data into training and testing sets when using K-means clustering, and why?

Ans: K-means clustering is a unsupervised learning algorithm that does not require of splitting data into testing and tranning data set. ## Two cases in which split the Data into traning and testing in K-means clustering 1. Split the data into traning and testing when evaluating the performance of K-means clutering.In this case, a subset of the data can be randomly selected as a testing set, and the remaining data can be used for training the K-means model. The model can then be applied to the testing set to evaluate its performance in terms of clustering accuracy or other relevant metrics. 2. Using K-means clustering as a preprocessing step for a supervised learning task.In this case, the data can be split into training and testing sets, and K-means clustering can be applied to the training data to generate cluster labels. The resulting clusters can then be used as features for a supervised learning model, which can be trained and evaluated on the testing set.

14 Q2. Why do we need to scale the features before performing K-means clustering?

Ans: The reasons for scaling the features are listed below: 1. K-mean clustering calculate distances if the features are not scaled equally then with larger magnitudes will dominate the distance metric, and the algorithm will be biased towards those features. 2. Scaling the features will ensure that all features have equal importance. This is important because features with large magnitudes may not necessarily be more important than features with smaller magnitudes. 3. Scaling the features also helps in improving the convergence of the algorithm. If the features are not scaled, then the algorithm may take longer to converge, or may not converge at all.