CustomerClusetring

June 13, 2024

1 Machine Learning Internship at Prodigy InfoTech - Customer Clusetring Project

In this project, we are using unsupervised learning algorithms Kmeans to explore the possibilities of grouping customers dataset into similar clusters according to similar demands or purchasing habits to aid in targeted and effecient marketing campaigns

1.1 import important libraries

[124]:
```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import MinMaxScaler
from sklearn.decomposition import PCA
from sklearn.metrics import silhouette_score,davies_bouldin_score
from sklearn.cluster import KMeans
sns.set_style('darkgrid')
```

1.2 Load Data

[125]:
```python
data=pd.read_csv('Mall_Customers.csv')
data.info()
```

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

1.3 Checking Missing Values

[126]:
```
data.isnull().sum()
```

[126]:
```
CustomerID                0
Gender                    0
Age                       0
Annual Income (k$)        0
Spending Score (1-100)    0
dtype: int64
```

Nice, There is no missing values

1.4 Checking duplicate Values

[127]:
```
data.duplicated().sum()
```

[127]: 0

Nice , There is no duplicate values

1.5 Get Statistical Information

[128]:
```
data.describe()
```

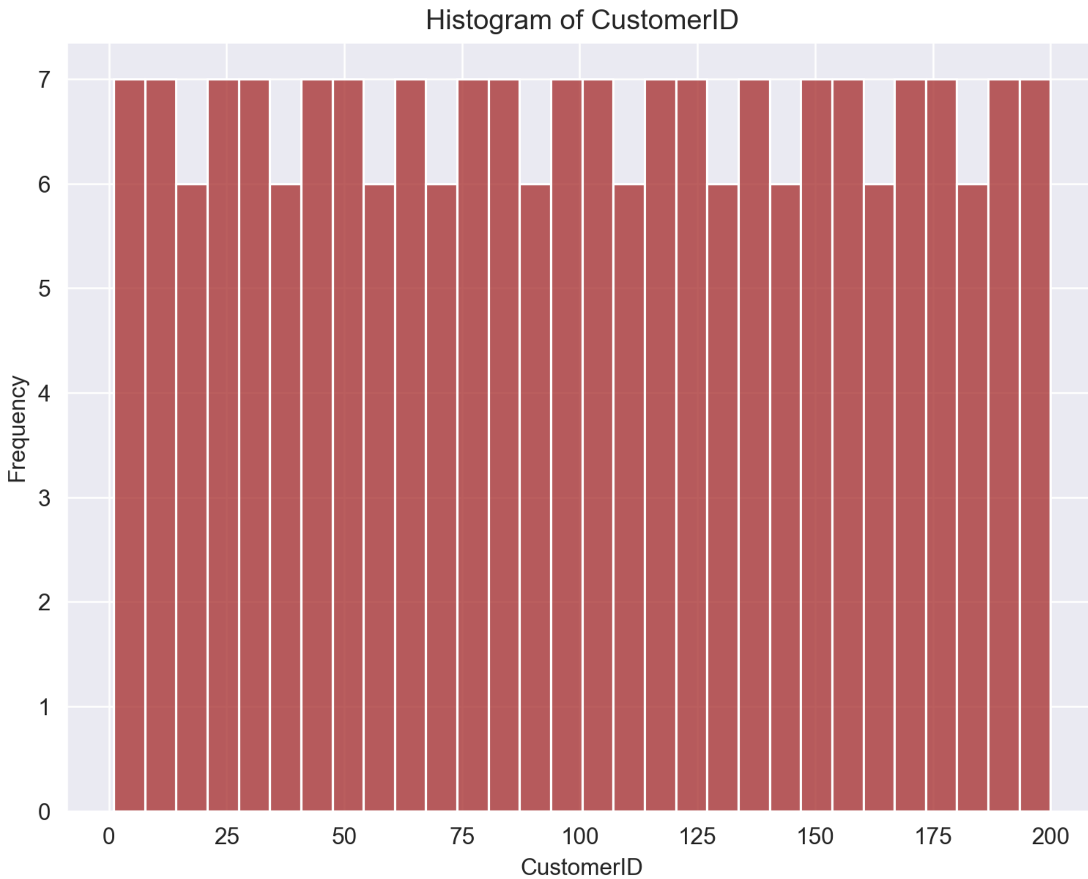
[128]:

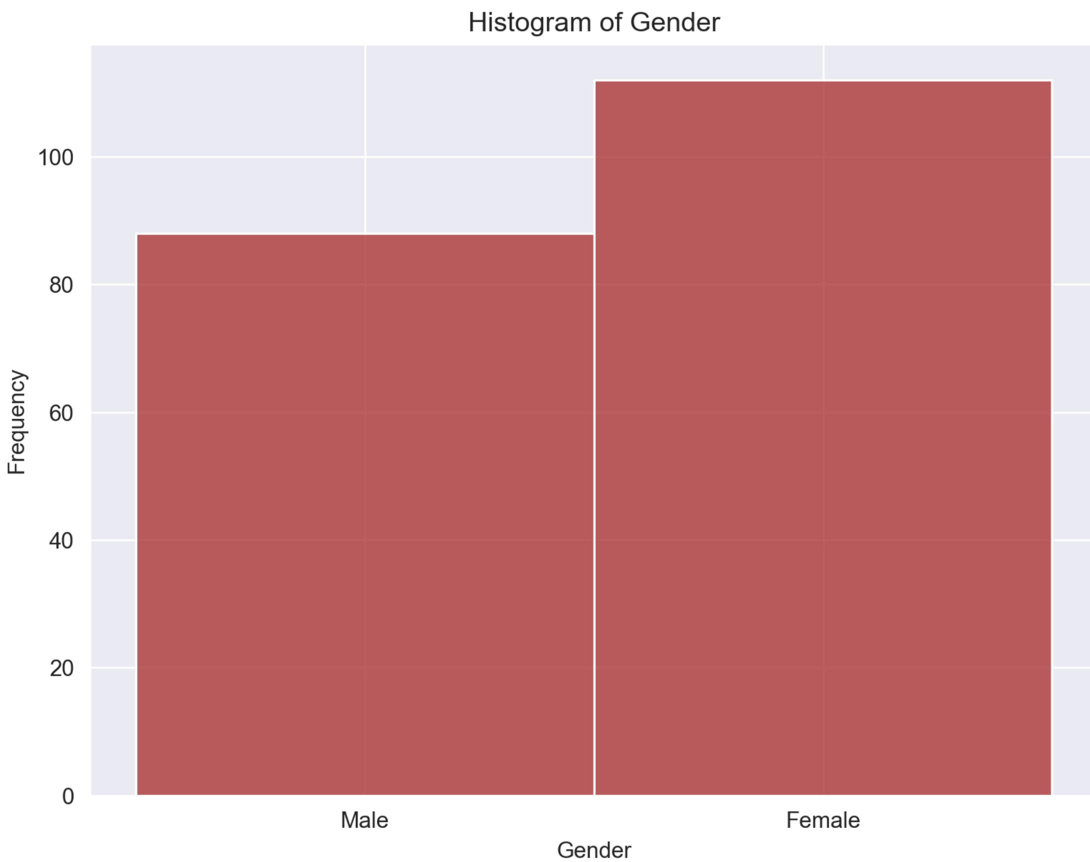
	CustomerID	Age	Annual Income (k$)	Spending Score (1-100)
count	200.000000	200.000000	200.000000	200.000000
mean	100.500000	38.850000	60.560000	50.200000
std	57.879185	13.969007	26.264721	25.823522
min	1.000000	18.000000	15.000000	1.000000
25%	50.750000	28.750000	41.500000	34.750000
50%	100.500000	36.000000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

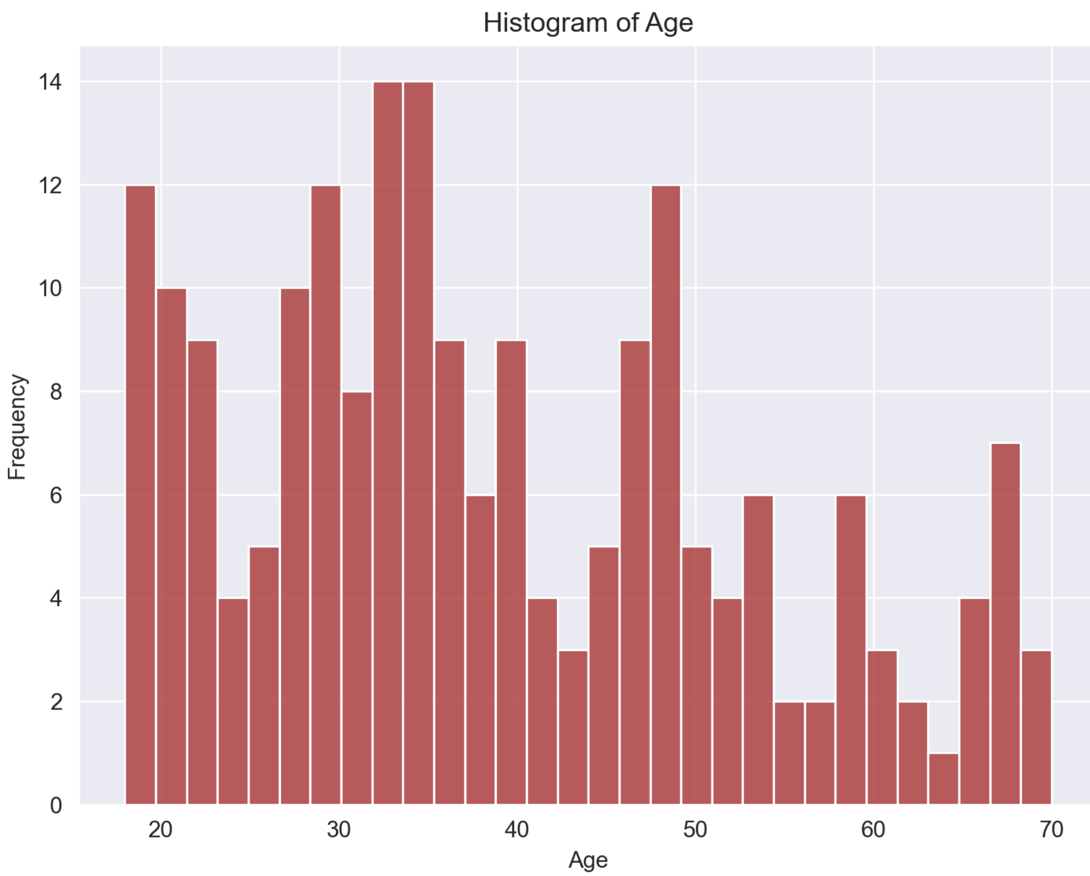
1.6 Distributions of columns :-

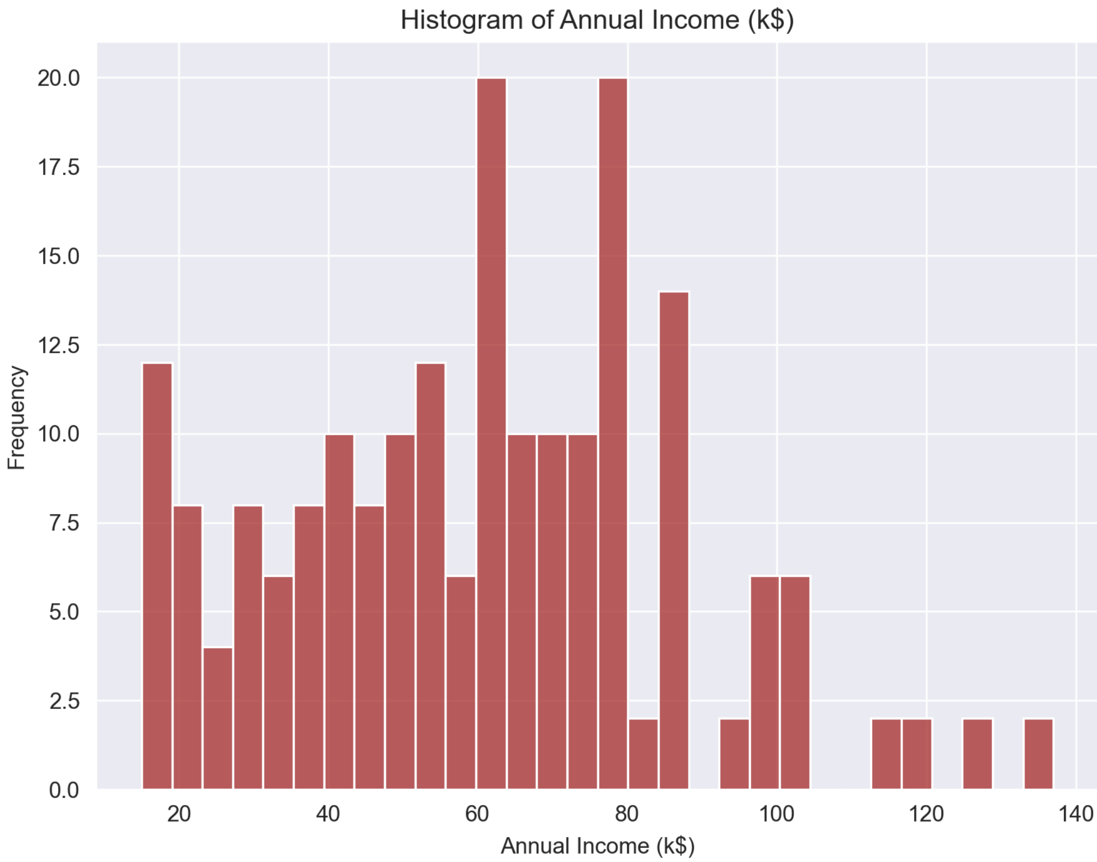
In this step, we plot histograms for various numerical columns within our dataset to gain a deeper understanding of the data distribution.

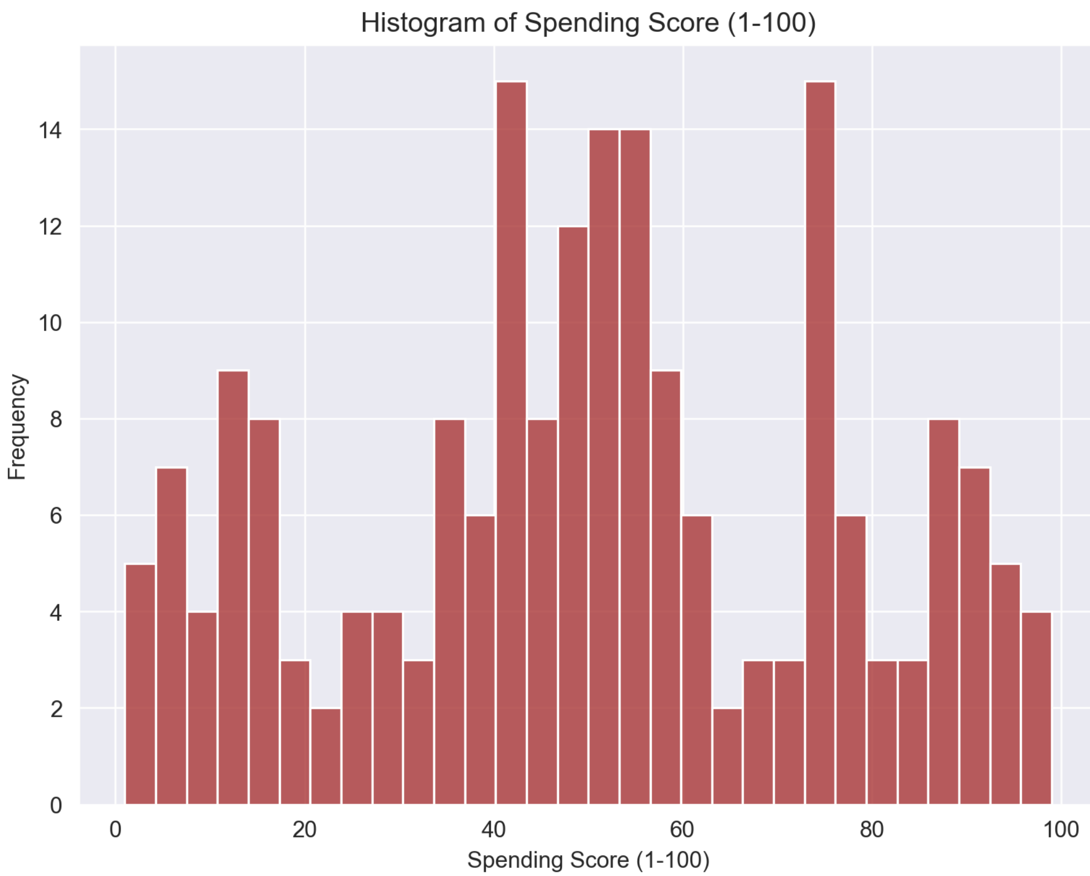
[129]:
```
for col in data:
    plt.figure(figsize=(8, 6))
    sns.histplot(data[col], bins=30,color='brown')
    plt.title(f'Histogram of {col}')
    plt.xlabel(col)
    plt.ylabel('Frequency')
    plt.show()
```


Histogram of CustomerID
Frequency
CustomerID
0
1
2
3
4
5
6
7
0
25
50
75
100
125
150
175
200


Histogram of Gender
100
80
60
40
20
0
Frequency
Male
Female
Gender


Histogram of Age
Frequency
Age
0
2
4
6
8
10
12
14
20
30
40
50
60
70


Histogram of Annual Income (k$)
Frequency
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
20
40
60
80
100
120
140
Annual Income (k$)



1.7 Encoding Categorical Features

[130]:
```
new_data=pd.get_dummies(data)
```

[131]:
```
new_data.head()
```

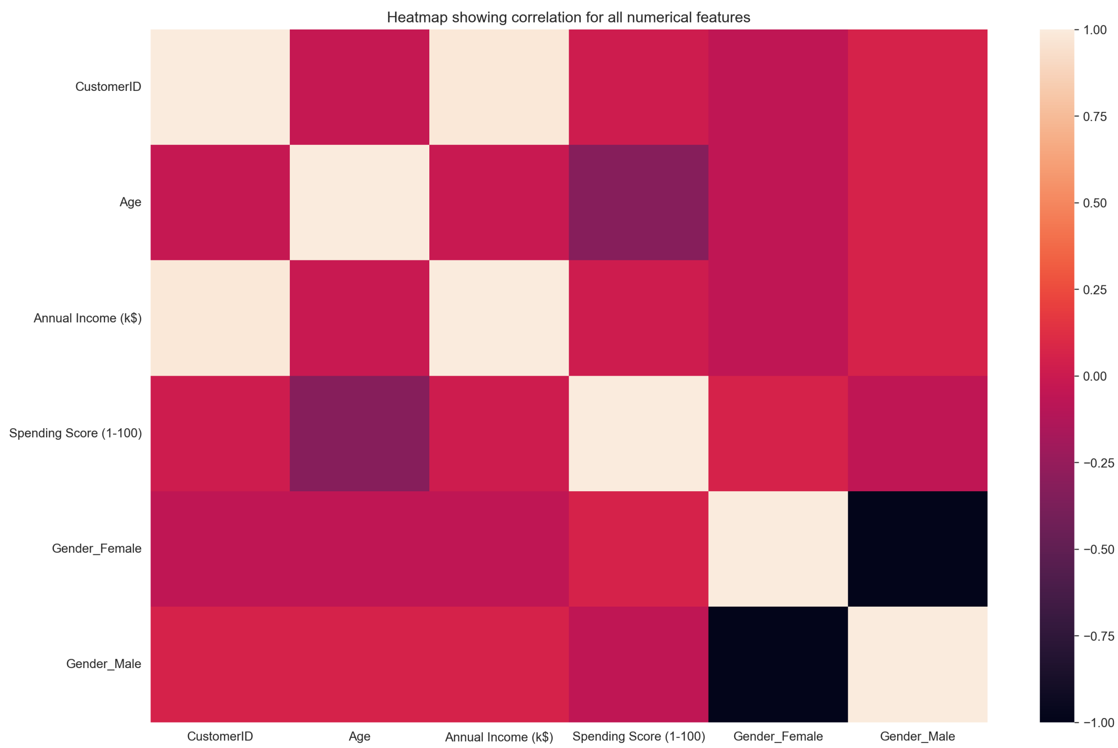
[131]:
```
   CustomerID  Age  Annual Income (k$)  Spending Score (1-100)  Gender_Female  \
0           1   19                  15                      39          False
1           2   21                  15                      81          False
2           3   20                  16                       6           True
3           4   23                  16                      77           True
4           5   31                  17                      40           True

   Gender_Male
0         True
1         True
2        False
3        False
4        False
```

2 Computing correlation between features

[132]:
```
corr_matrix=new_data.corr()
plt.figure(figsize = (15,10))
plt.title("Heatmap showing correlation for all numerical features")
sns.heatmap(corr_matrix)
```

[132]:
```
<Axes: title={'center': 'Heatmap showing correlation for all numerical
features'}>
```



2.1 Performing MinMax Scaling

[133]:
```
scaler = MinMaxScaler()
new_data= pd.DataFrame(scaler.fit_transform(new_data), columns=new_data.columns)
```

[134]:
```
new_data.head()
```

[134]:

	CustomerID	Age	Annual Income (k$)	Spending Score (1-100)	\
0	0.000000	0.019231	0.000000	0.387755	
1	0.005025	0.057692	0.000000	0.816327	
2	0.010050	0.038462	0.008197	0.051020	
3	0.015075	0.096154	0.008197	0.775510	
4	0.020101	0.250000	0.016393	0.397959	

```
   Gender_Female  Gender_Male
0            0.0          1.0
1            0.0          1.0
2            1.0          0.0
3            1.0          0.0
4            1.0          0.0
```

2.2 Performing PCA Algorithm

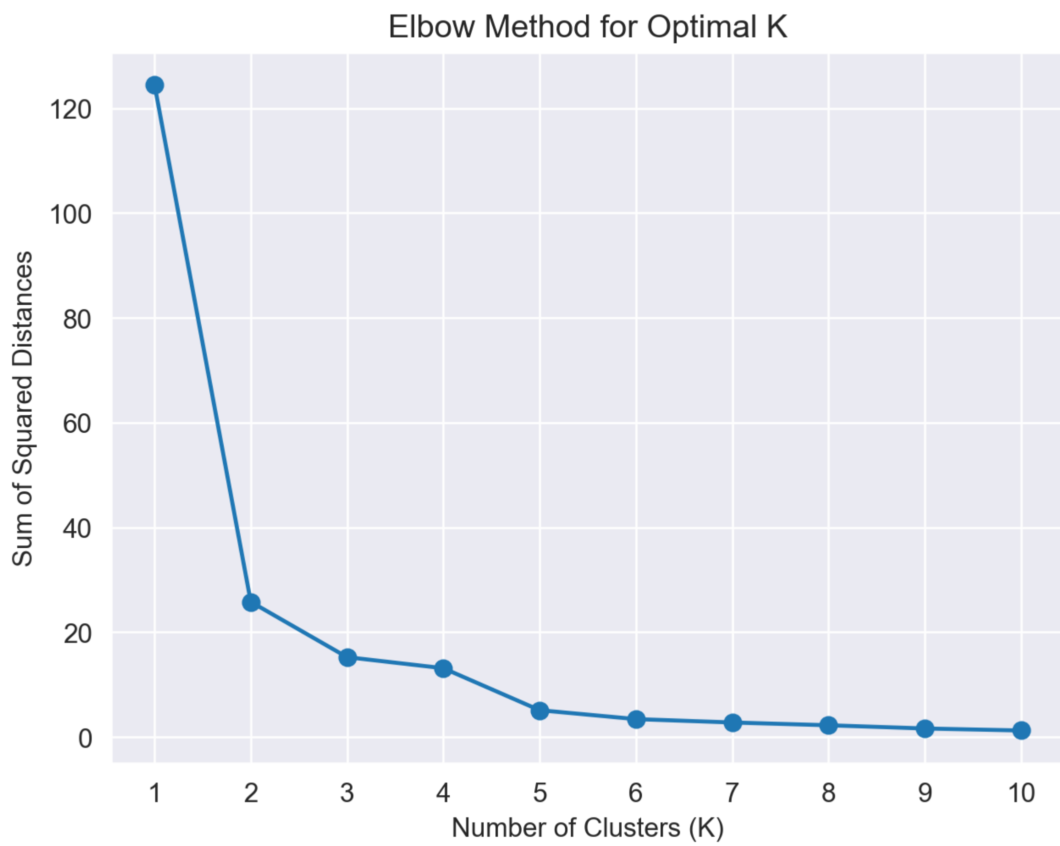
[135]:
```
pca = PCA(n_components=2)
new_data = pca.fit_transform(new_data)
```

2.3 Using the Elbow Method for finding the optimal number of clusters (K)

[136]:
```
k_values = range(1, 11)  # Evaluate from 1 to 20 clusters

# Calculate the Within-Clusters Sum of Squares (WCSS) for each value of K
costs = []
for k in k_values:
    kmeans = KMeans(n_clusters=k, random_state=0)
    kmeans.fit(new_data)
    costs.append( kmeans .inertia_)

# Plot the elbow curve
plt.rcParams['figure.dpi'] = 227
plt.plot(k_values, costs, marker='o')
plt.xlabel('Number of Clusters (K)')
plt.ylabel('Sum of Squared Distances')
plt.title('Elbow Method for Optimal K')
plt.xticks(k_values)
plt.show()
```



2.3.1 Then, 2 is the optimal number of clusters to train our data , so we will apply Kmeans

2.4 Applying Kmeans Clustering

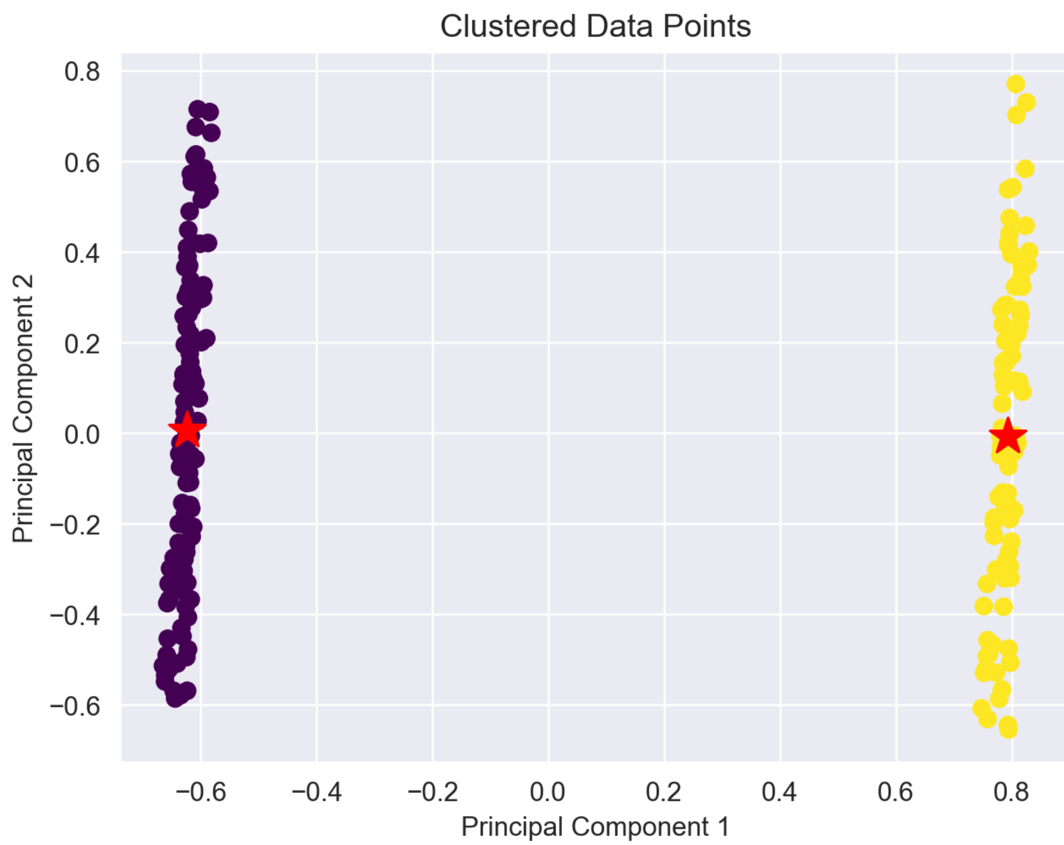
[137]:
```python
kmeans=KMeans(n_clusters=2).fit(new_data)
clusters=kmeans.cluster_centers_
labels=kmeans.labels_
print("clusters : ",clusters)
print(50*"*")
print("labels : ",labels)
```

```
clusters :  [[-0.62278988  0.00584397]
 [ 0.79264167 -0.00743777]]
**************************************************
labels :  [1 1 0 0 0 0 0 0 1 0 1 0 0 0 1 1 0 1 1 0 1 1 0 1 0 1 0 1 0 0 1 0 1 1 0
0 0
 0 0 0 0 1 1 0 0 0 0 0 0 0 0 1 0 1 0 1 0 1 0 1 1 1 0 0 1 1 0 0 1 0 1 0 0 0
 1 1 0 1 0 0 1 1 1 0 0 1 0 0 0 0 0 1 1 0 0 1 0 0 1 1 0 0 1 1 1 0 0 1 1 1 1
 0 0 1 0 0 0 0 0 0 1 0 0 1 0 0 1 1 1 1 1 1 0 0 1 0 0 1 1 0 0 1 0 0 1 1 1 0
```

0 1 1 1 0 0 0 0 1 0 1 0 0 0 1 0 1 0 1 0 0 1 1 1 1 1 0 0 1 1 1 1 0 0 1 0 0
1 0 1 0 0 0 0 1 0 0 0 0 1 1 1]

2.5 Plotting the Produced Clusters

[138]:
```
# Plot clustered data points
plt.scatter(new_data[:, 0], new_data[:, 1], c=labels, cmap='viridis')
plt.scatter(clusters[:, 0], clusters[:, 1], c='red', marker='*', s=200)
plt.title('Clustered Data Points')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.show()
```



2.6 Evaluate clustering performance using silhouette score and davies_bouldin_score

[139]:
```
k_medoids_silhouette = silhouette_score(new_data, labels)
print("Silhouette Score:", k_medoids_silhouette)
```

```python
k_medoids_davies_bouldin = davies_bouldin_score(new_data, labels)
print("davies_bouldin:", k_medoids_davies_bouldin)
```

```
Silhouette Score: 0.7232082660328618
davies_bouldin: 0.4304392332949172
```

2.7 "silhouette_score" :-

2.7.1 measures how similar an object is to its own cluster compared to other clusters.

```
Higher silhouette scores indicate better clustering .
```

2.8 "davies_bouldin_score" :-

2.8.1 quantifies the average "similarity" between clusters

```
and the "dissimilarity" between clusters. Lower Davies-Bouldin scores indicate
better clustering
```