Problem 1: Clustering

A leading bank wants to develop a customer segmentation to give promotional offers to its customers. They collected a sample that summarizes the activities of users during the past few months. You are given the task to identify the segments based on credit card usage. 1.1 Read the data and do exploratory data analysis. Describe the data briefly.

1.2 Do you think scaling is necessary for clustering in this case? Justify

1.3 Apply hierarchical clustering to scaled data. Identify the number of optimum clusters using Dendrogram and briefly describe them

1.4 Apply K-Means clustering on scaled data and determine optimum clusters. Apply elbow curve and silhouette score.

1.5 Describe cluster profiles for the clusters defined. Recommend different promotional strategies for different clusters.

Dataset for Problem 1: bank_marketing_part1_Data.csv

Data Dictionary for Market Segmentation:

spending: Amount spent by the customer per month (in 1000s) advance_payments: Amount paid by the customer in advance by cash (in 100s) probability_of_full_payment: Probability of payment done in full by the customer to the bank current_balance: Balance amount left in the account to make purchases (in 1000s) credit_limit: Limit of the amount in credit card (10000s) min_payment_amt : minimum paid by the customer while making payments for purchases made monthly (in 100s) max_spent_in_single_shopping: Maximum amount spent in one purchase (in 1000s)

1.1 Read the data and do exploratory data analysis. Describe the data briefly.

In [1]:

```
import numpy as np
import pandas as pd
import seaborn as sns
from matplotlib import pyplot as plt
%matplotlib inline
```

In [2]:

```
df=pd.read_csv('bank_marketing_part1_Data.csv')
```

In [3]:

```
df.head()
```

Out[3]:

	spending	advance_payments	probability_of_full_payment	current_balance	credit_limit	min_payment_amt	max_spent_in_sir
0	19.94	16.92	0.8752	6.675	3.763	3.252	
1	15.99	14.89	0.9064	5.363	3.582	3.336	
2	18.95	16.42	0.8829	6.248	3.755	3.368	
3	10.83	12.96	0.8099	5.278	2.641	5.182	
4	17.99	15.86	0.8992	5.890	3.694	2.068	

In [4]:

```
df.shape
```

Out[4]:

```
(210, 7)
```

In [5]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 210 entries, 0 to 209
Data columns (total 7 columns):
 #   Column                        Non-Null Count  Dtype  
---  ------                        --------------  -----  
 0   spending                      210 non-null    float64
 1   advance_payments              210 non-null    float64
 2   probability_of_full_payment   210 non-null    float64
 3   current_balance               210 non-null    float64
 4   credit_limit                  210 non-null    float64
 5   min_payment_amt               210 non-null    float64
 6   max_spent_in_single_shopping  210 non-null    float64
dtypes: float64(7)
memory usage: 11.6 KB
```

In [6]:

```
df.dtypes.value_counts()
```

Out[6]:

```
float64    7
dtype: int64
```

All data types are integer type and there is no need for datatype converting

In [7]:

```
df.isna().sum()
```

Out[7]:

```
spending                        0
advance_payments                0
probability_of_full_payment     0
current_balance                 0
credit_limit                    0
min_payment_amt                 0
max_spent_in_single_shopping    0
dtype: int64
```

There are no missing values in the dataset.

In [8]:

```
dupes=df.duplicated()
sum(dupes)
```

Out[8]:

```
0
```

There are no duplicates in the dataset.

In [9]:

```
df.describe()
```

Out[9]:

	spending	advance_payments	probability_of_full_payment	current_balance	credit_limit	min_payment_amt	max_spent
count	210.000000	210.000000	210.000000	210.000000	210.000000	210.000000	

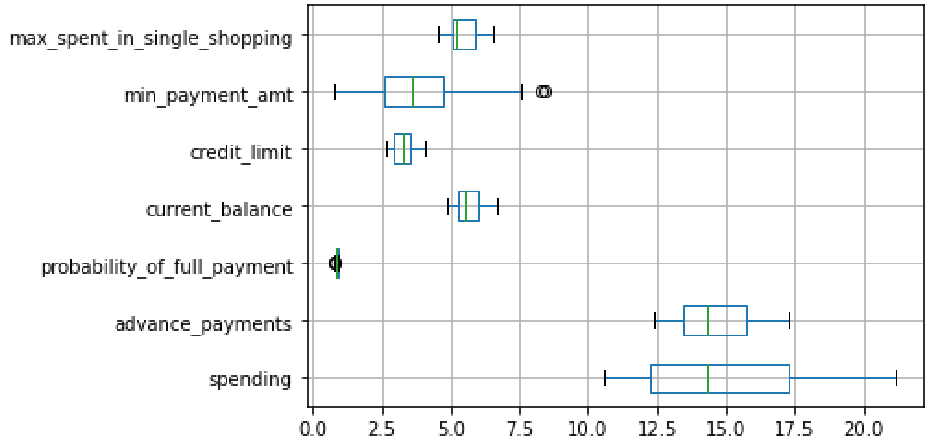
	spending	advance_payments	probability_of_full_payment	current_balance	credit_limit	min_payment_amt	max_spent
mean	14.847524	14.559286	0.870999	5.628533	3.258605	3.700201	
std	2.909699	1.305959	0.023629	0.443063	0.377714	1.503557	
min	10.590000	12.410000	0.808100	4.899000	2.630000	0.765100	
25%	12.270000	13.450000	0.856900	5.262250	2.944000	2.561500	
50%	14.355000	14.320000	0.873450	5.523500	3.237000	3.599000	
75%	17.305000	15.715000	0.887775	5.979750	3.561750	4.768750	
max	21.180000	17.250000	0.918300	6.675000	4.033000	8.456000	

In [10]:

```
plt.figure(figsize=(6,4))
df.boxplot(vert=0)
```

Out[10]:

<AxesSubplot:>



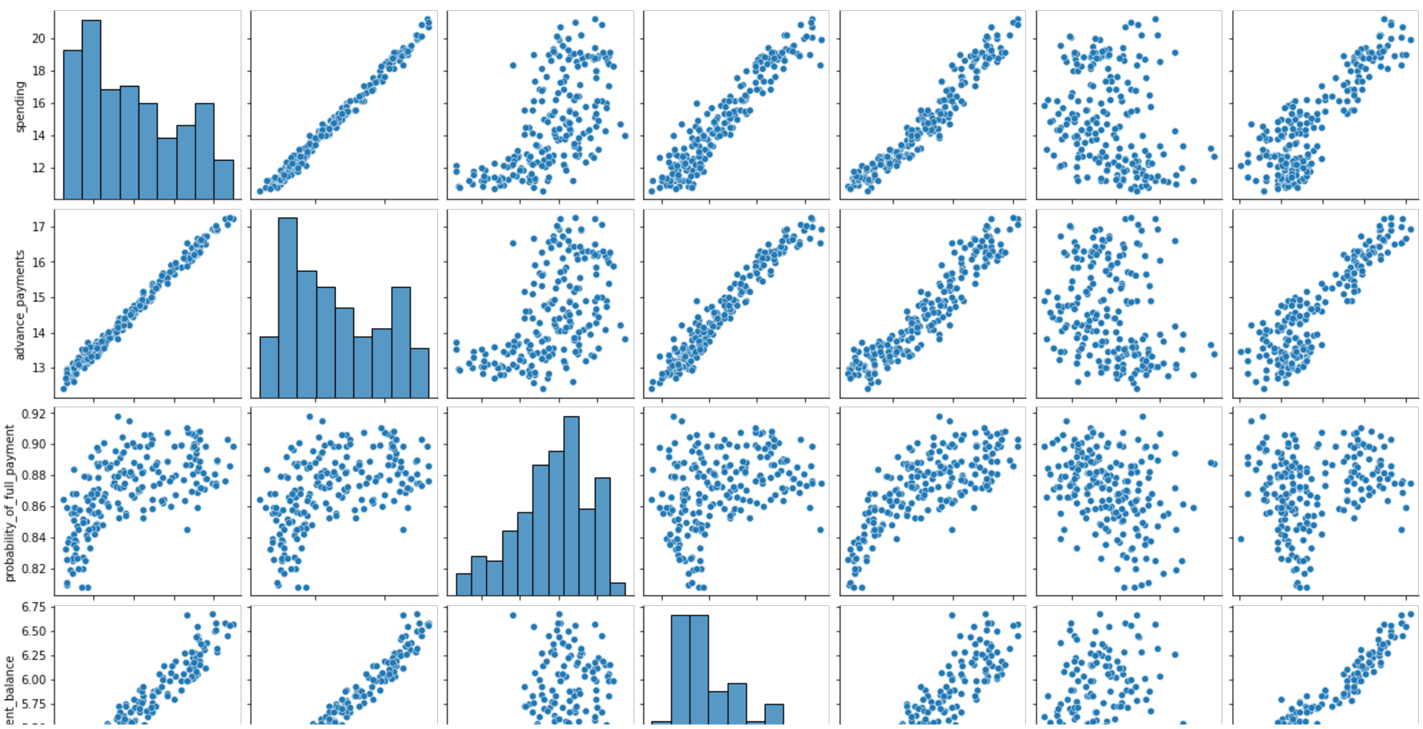
Very few outliers are there i.e., for min_payment_amount and probability_of_full_payment. Since clustering is not affected by outliers there is no need to treat outliers.

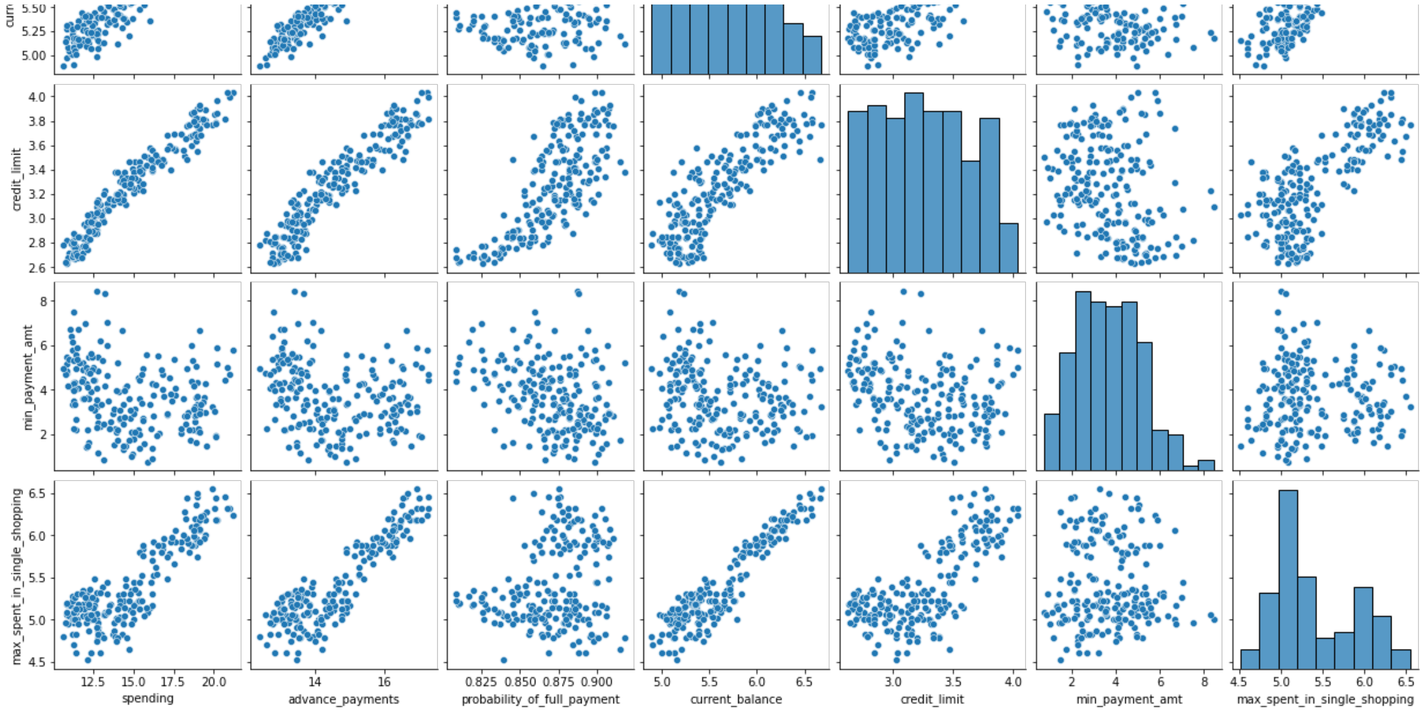
In [11]:

```
sns.pairplot(df)
```

Out[11]:

<seaborn.axisgrid.PairGrid at 0x1d03b291550>





In [12]:

```
df.corr()
```

Out[12]:

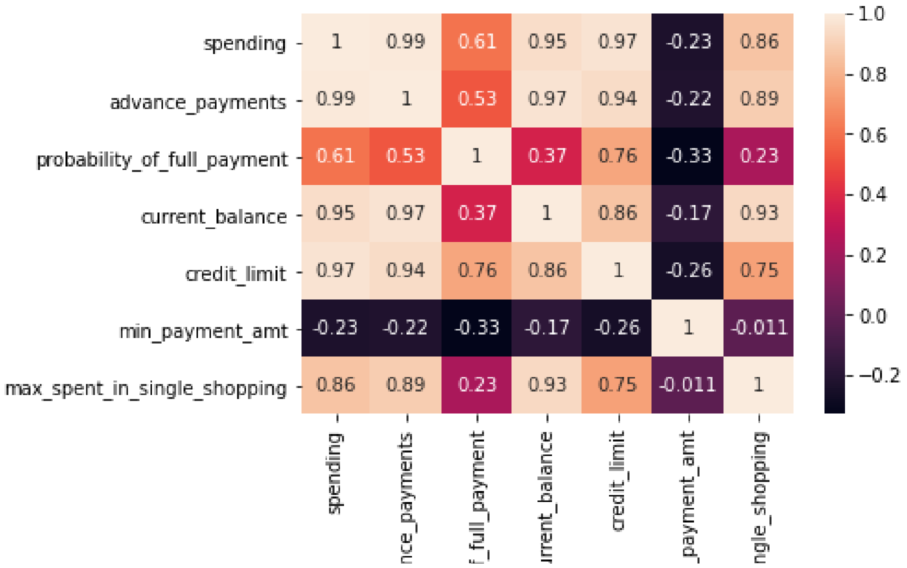
	spending	advance_payments	probability_of_full_payment	current_balance	credit_limit	min_pa
spending	1.000000	0.994341	0.608288	0.949985	0.970771	
advance_payments	0.994341	1.000000	0.529244	0.972422	0.944829	
probability_of_full_payment	0.608288	0.529244	1.000000	0.367915	0.761635	
current_balance	0.949985	0.972422	0.367915	1.000000	0.860415	
credit_limit	0.970771	0.944829	0.761635	0.860415	1.000000	
min_payment_amt	-0.229572	-0.217340	-0.331471	-0.171562	-0.258037	
max_spent_in_single_shopping	0.863693	0.890784	0.226825	0.932806	0.749131	

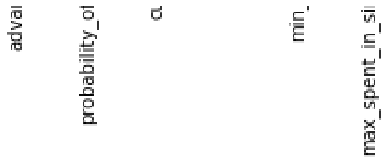
In [13]:

```
sns.heatmap(df.corr(), annot=True)
```

Out[13]:

<AxesSubplot:>





1.2 Do you think scaling is necessary for clustering in this case? Justify

Yes. Scaling is required for the data.

In [14]:

```
from sklearn.preprocessing import StandardScaler
```

In [15]:

```
X = StandardScaler()
```

In [16]:

```
s_df = pd.DataFrame(X.fit_transform(df.iloc[:,0:7]),columns=df.columns[0:])
```

In [17]:

```
s_df
```

Out[17]:

	spending	advance_payments	probability_of_full_payment	current_balance	credit_limit	min_payment_amt	max_spent_in_
0	1.754355	1.811968	0.178230	2.367533	1.338579	-0.298806	
1	0.393582	0.253840	1.501773	-0.600744	0.858236	-0.242805	
2	1.413300	1.428192	0.504874	1.401485	1.317348	-0.221471	
3	-1.384034	-1.227533	-2.591878	-0.793049	-1.639017	0.987884	
4	1.082581	0.998364	1.196340	0.591544	1.155464	-1.088154	
...	...	...	...	...	...	...	
205	-0.329866	-0.413929	0.721222	-0.428801	-0.158181	0.190536	
206	0.662292	0.814152	-0.305372	0.675253	0.476084	0.813214	
207	-0.281636	-0.306472	0.364883	-0.431064	-0.152873	-1.322158	
208	0.438367	0.338271	1.230277	0.182048	0.600814	-0.953484	
209	0.248893	0.453403	-0.776248	0.659416	-0.073258	-0.706813	

210 rows × 7 columns

In [18]:

```
s_df.describe()
```

Out[18]:

	spending	advance_payments	probability_of_full_payment	current_balance	credit_limit	min_payment_amt	max_s
count	2.100000e+02	2.100000e+02	2.100000e+02	2.100000e+02	2.100000e+02	2.100000e+02	
mean	9.148766e-16	1.097006e-16	1.260896e-15	-1.358702e-16	-2.790757e-16	5.418946e-16	
std	1.002389e+00	1.002389e+00	1.002389e+00	1.002389e+00	1.002389e+00	1.002389e+00	
min	-1.466714e+00	-1.649686e+00	-2.668236e+00	-1.650501e+00	-1.668209e+00	-1.956769e+00	

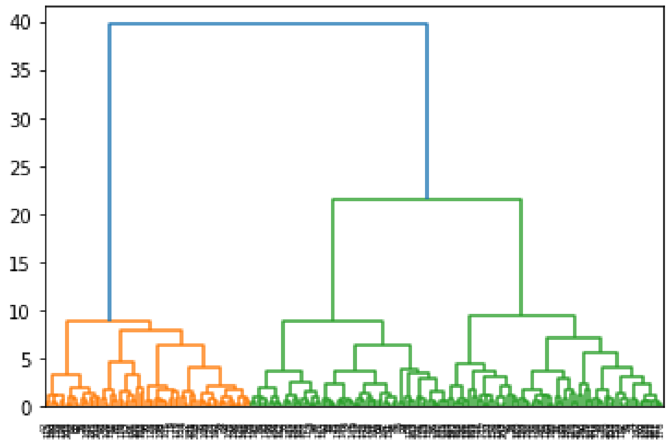
	spending	advance_payments	probability_of_full_payment	current_balance	credit_limit	min_payment_amt	max_s
25%	-8.879552e-01	-8.514330e-01	-5.980791e-01	-8.286816e-01	-8.349072e-01	-7.591477e-01	
50%	-1.696741e-01	-1.836639e-01	1.039927e-01	-2.376280e-01	-5.733534e-02	-6.746852e-02	
75%	8.465989e-01	8.870693e-01	7.116771e-01	7.945947e-01	8.044956e-01	7.123789e-01	
max	2.181534e+00	2.065260e+00	2.006586e+00	2.367533e+00	2.055112e+00	3.170590e+00	

Scaled data is in s_df dataframe which will be used for further hierarchical clustering and also kmeans clustering.

1.3 Apply hierarchical clustering to scaled data. Identify the number of optimum clusters using Dendrogram and briefly describe them

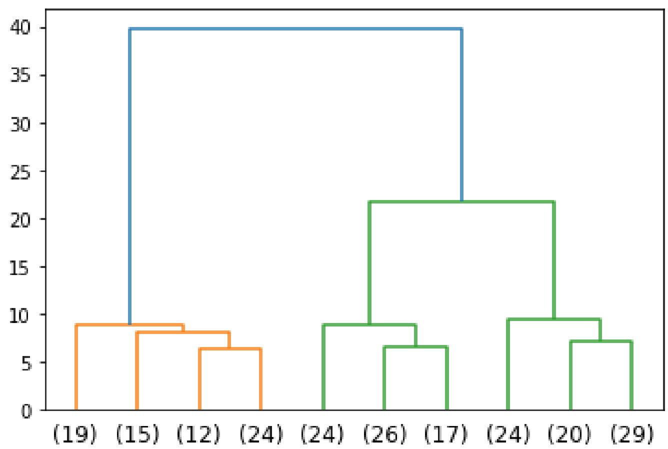
In [19]:

```
from scipy.cluster.hierarchy import dendrogram, linkage
HClust = linkage(s_df, method = 'ward')
dend = dendrogram(HClust)
```



In [20]:

```
dend = dendrogram(HClust,
                  truncate_mode='lastp',
                  p = 10,
                  )
```



Identifying number of optimal clusters.

Importing flcuster module to create clusters

In [21]:

```
from scipy.cluster.hierarchy import fcluster
```

In [22]:

```
clusters = fcluster(HClust, 2, criterion='maxclust')
clusters
```

Out[22]:

```
array([1, 2, 1, 2, 1, 2, 2, 2, 1, 2, 1, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2,
       1, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 1, 2, 1, 1,
       2, 2, 2, 1, 1, 1, 2, 1, 1, 1, 1, 1, 2, 2, 2, 1, 2, 2, 2, 2, 2, 1,
       1, 2, 1, 2, 2, 2, 1, 1, 2, 1, 2, 2, 1, 2, 2, 2, 2, 1, 2, 2, 2, 1,
       1, 2, 2, 1, 2, 2, 2, 1, 1, 1, 2, 1, 2, 1, 2, 1, 2, 1, 1, 2, 2, 1,
       2, 2, 1, 2, 2, 1, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2,
       2, 1, 2, 1, 1, 2, 1, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 1, 1, 2, 1, 1, 1, 2, 1, 2, 2, 2, 2, 2, 2, 1, 1, 1,
       2, 2, 1, 2, 2, 2, 2, 2, 1, 1, 2, 2, 2, 2, 2, 2, 2, 1, 2, 1, 1, 2,
       1, 2, 2, 1, 2, 2, 1, 2, 1, 2, 1, 2], dtype=int32)
```

Appending clusters to original data set

In [23]:

```
df['clusters']=clusters
```

Cluster frequency

In [24]:

```
df.clusters.value_counts().sort_index()
```

Out[24]:

```
1     70
2    140
Name: clusters, dtype: int64
```

In [25]:

```
clusters1 = fcluster(HClust, 3, criterion='maxclust')
clusters1
```

Out[25]:

```
array([1, 3, 1, 2, 1, 2, 2, 3, 1, 2, 1, 3, 2, 1, 3, 2, 3, 2, 3, 2, 2, 2,
       1, 2, 3, 1, 3, 2, 2, 2, 3, 2, 2, 3, 2, 2, 2, 2, 2, 1, 1, 3, 1, 1,
       2, 2, 3, 1, 1, 1, 2, 1, 1, 1, 1, 1, 2, 2, 2, 1, 3, 2, 2, 3, 3, 1,
       1, 3, 1, 2, 3, 2, 1, 1, 2, 1, 3, 2, 1, 3, 3, 3, 3, 1, 2, 3, 3, 1,
       1, 2, 3, 1, 3, 2, 2, 1, 1, 1, 2, 1, 2, 1, 3, 1, 3, 1, 1, 2, 2, 1,
       3, 3, 1, 2, 2, 1, 3, 3, 2, 1, 3, 2, 2, 2, 3, 3, 1, 2, 3, 3, 2, 3,
       3, 1, 2, 1, 1, 2, 1, 3, 3, 3, 2, 2, 3, 2, 1, 2, 3, 2, 3, 2, 3, 3,
       3, 3, 3, 2, 3, 1, 1, 2, 1, 1, 1, 2, 1, 3, 3, 3, 3, 2, 3, 1, 1, 1,
       3, 3, 1, 2, 3, 3, 3, 3, 1, 1, 3, 3, 3, 2, 3, 3, 2, 1, 3, 1, 1, 2,
       1, 2, 3, 1, 3, 2, 1, 3, 1, 3, 1, 3], dtype=int32)
```

Appending clusters to original data set

In [26]:

```
df['clusters1']=clusters1
```

Cluster frequency

In [27]:

```
df.clusters1.value_counts().sort_index()
```

Out[27]:

```
1    70
2    67
```

```
3    73
Name: clusters1, dtype: int64
```

By default, dendrogram is showing 2 clusters cluster1: 70 and cluster2:with 140 observations. But if created 3 clusters, almost similar no. of observations are falling in each cluster and i think that the profile can be best compared from technical and business angle if 3 clusters are taken rather than taking two clusters i.e., one high and one low. so, 3 clusters are taken.

df.columns

In [28]:

```
df.columns
```

Out[28]:

```
Index(['spending', 'advance_payments', 'probability_of_full_payment',
       'current_balance', 'credit_limit', 'min_payment_amt',
       'max_spent_in_single_shopping', 'clusters', 'clusters1'],
      dtype='object')
```

Cluster profile

In [29]:

```
aggdata=df.drop(['clusters'],axis=1).groupby('clusters1').mean()
aggdata['Freq']=df.clusters1.value_counts().sort_index()
aggdata
```

Out[29]:

	spending	advance_payments	probability_of_full_payment	current_balance	credit_limit	min_payment_amt	max_spe
clusters1							
1	18.371429	16.145429	0.884400	6.158171	3.684629	3.639157	
2	11.872388	13.257015	0.848072	5.238940	2.848537	4.949433	
3	14.199041	14.233562	0.879190	5.478233	3.226452	2.612181	

HIERARCHICAL CLUSTERING: RECOMMENDATIONS: Cluster 1: This cluster has maximum credit_limit as they are spending more, maximum spent in single_shopping (max_spent_in_single_shopping) and also with highest advance_payments options with highest current_balance and are having highest probability of full payment (probability_of_full_payment) For this cluster, min_payment_amount is moderate. For the cluster 1 type customers, certain incentives of increasing their credit-limits can be an option to attract and maintain customers. Cluster 2: This cluster has minimum spending and the credit_limit, advance_payments, probability_of_full_payment,current_balance are also less. For this segment of clusters, attractive offers can increase their credit usage. Cluster 3:This cluster has moderate credit_limit and all the variable means are showing moderate, except for min_payment_amount.

1.4 Apply K-Means clustering on scaled data and determine optimum clusters. Apply elbow curve and silhouette score.

In [30]:

```
df.columns
```

Out[30]:

```
Index(['spending', 'advance_payments', 'probability_of_full_payment',
       'current_balance', 'credit_limit', 'min_payment_amt',
       'max_spent_in_single_shopping', 'clusters', 'clusters1'],
      dtype='object')
```

In [31]:

```
s_df.columns
```

Out[31]:

```
Index(['spending', 'advance_payments', 'probability_of_full_payment',
       'current_balance', 'credit_limit', 'min_payment_amt',
       'max_spent_in_single_shopping'],
      dtype='object')
```

In [32]:

```
s_df.head()
```

Out[32]:

	spending	advance_payments	probability_of_full_payment	current_balance	credit_limit	min_payment_amt	max_spent_in_sir
0	1.754355	1.811968	0.178230	2.367533	1.338579	-0.298806	
1	0.393582	0.253840	1.501773	-0.600744	0.858236	-0.242805	
2	1.413300	1.428192	0.504874	1.401485	1.317348	-0.221471	
3	-1.384034	-1.227533	-2.591878	-0.793049	-1.639017	0.987884	
4	1.082581	0.998364	1.196340	0.591544	1.155464	-1.088154	

creating clusters with kMeans

In [33]:

```
from sklearn.cluster import KMeans
```

In [34]:

```
k_means = KMeans(n_clusters = 2,random_state=1)
```

In [35]:

```
k_means.fit(s_df)
```

Out[35]:

```
KMeans(n_clusters=2, random_state=1)
```

cluster output for all observations

In [36]:

```
k_means.labels_
```

Out[36]:

```
array([1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0,
       1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1,
       0, 0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1,
       1, 0, 1, 0, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 1,
       1, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1, 1, 0, 0, 1,
       1, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0,
       0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1,
       0, 0, 0, 0, 0, 1, 1, 0, 1, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1, 1,
       0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0,
       1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 1])
```

within cluster sum of squares

In [37]:

```
k_means.inertia_
```

Out[37]:

```
659.171754487041
```

Calculating WSS for other values of K - Elbow Method

In [38]:

```
wss =[]
```

In [39]:

```
for i in range(1,6):
    KM = KMeans(n_clusters=i,random_state=1)
    KM.fit(s_df)
    wss.append(KM.inertia_)
```

In [40]:

```
wss
```

Out[40]:

```
[1469.9999999999998,
 659.171754487041,
 430.6589731513006,
 371.38509060801096,
 327.21278165661346]
```

In [41]:

```
a=[1,2,3,4,5]
```

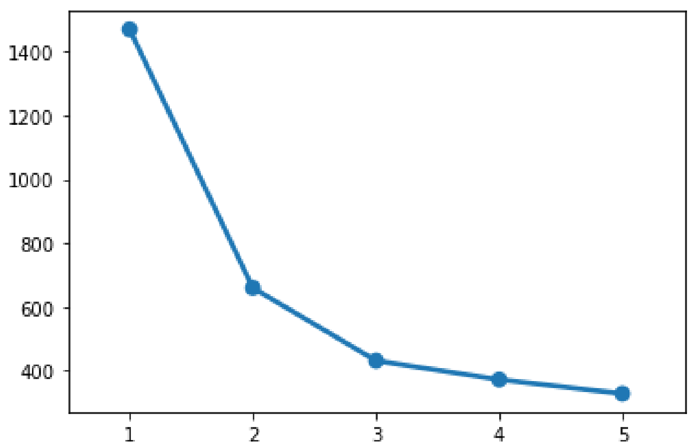
In [42]:

```
sns.pointplot(a, wss)
```

```
C:\Users\hp\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass th
e following variables as keyword args: x, y. From version 0.12, the only valid positional
argument will be `data`, and passing other arguments without an explicit keyword will res
ult in an error or misinterpretation.
  warnings.warn(
```

Out[42]:

```
<AxesSubplot:>
```



KMeans with K=2

In [43]:

```
k_means = KMeans(n_clusters = 2,random_state=1)
k_means.fit(s_df)
labels = k_means.labels_
```

In [44]:

```
labels
```

Out[44]:

```
array([1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0,
       1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1,
       0, 0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1,
       1, 0, 1, 0, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 1,
       1, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1, 1, 0, 0, 1,
       1, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0,
       0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1,
       0, 0, 0, 0, 0, 1, 1, 0, 1, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1, 1,
       0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0,
       1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 1])
```

Cluster evaluation for 2 clusters: the silhouette score

In [45]:

```
from sklearn.metrics import silhouette_samples, silhouette_score
```

In [46]:

```
# Calculating silhouette_score
silhouette_score(s_df,labels,random_state=1)
```

Out[46]:

```
0.46577247686580914
```

KMeans with K=3

In [47]:

```
k_means = KMeans(n_clusters = 3,random_state=1)
k_means.fit(s_df)
labels = k_means.labels_
```

Cluster evaluation for 3 clusters: the silhouette score

In [48]:

```
silhouette_score(s_df,labels,random_state=1)
```

Out[48]:

```
0.4007270552751299
```

Appending Clusters to the original dataset

In [49]:

```
df["Clus_kmeans3"] = labels
df.head()
```

Out[49]:

	spending	advance_payments	probability_of_full_payment	current_balance	credit_limit	min_payment_amt	max_spent_in_sin
0	19.94	16.92	0.8752	6.675	3.763	3.252	
1	15.99	14.89	0.9064	5.363	3.582	3.336	
2	18.95	16.42	0.8829	6.248	3.755	3.368	
3	10.83	12.96	0.8099	5.278	2.641	5.182	
4	17.99	15.86	0.8992	5.890	3.694	2.068	

Cluster Profiling

In [50]:

```
df.Clus_kmeans3.value_counts().sort_index()
```

Out[50]:

```
0    71
1    72
2    67
Name: Clus_kmeans3, dtype: int64
```

In [51]:

```
clust_profile=df.drop(['clusters','clusters1'], axis=1).groupby('Clus_kmeans3').mean()
clust_profile['freq']=df.Clus_kmeans3.value_counts().sort_index()
clust_profile
```

Out[51]:

Clus_kmeans3	spending	advance_payments	probability_of_full_payment	current_balance	credit_limit	min_payment_amt	ma
0	14.437887	14.337746	0.881597	5.514577	3.259225	2.707341	
1	11.856944	13.247778	0.848253	5.231750	2.849542	4.742389	
2	18.495373	16.203433	0.884210	6.175687	3.697537	3.632373	

Cluster 0: This cluster has moderate credit_limit and all the variable means are showing moderate, except for min_payment_amount..

Cluster 1: This cluster has minimum spending and the credit_limit, advance_payments, probability_of_full_payment,current_balance are also less. For this segment of clusters, attractive offers can increase their credit usage.

Cluster 2: This cluster has maximum credit_limit as they are spending more, maximum spent in single_shopping (max_spent_in_single_shopping) and also with highest advance_payments options with highest current_balance and are having highest probability of full payment (probability_of_full_payment) For this cluster, min_payment_amount is moderate. For the cluster 2 type customers, certain incentives of increasing their credit-limits can be an option to attract and maintain customers

1.5 Describe cluster profiles for the clusters defined. Recommend different promotional strategies for different clusters

In [52]:

```
df.columns
```

Out[52]:

```
Index(['spending', 'advance_payments', 'probability_of_full_payment',
       'current_balance', 'credit_limit', 'min_payment_amt',
       'max_spent_in_single_shopping', 'clusters', 'clusters1',
       'Clus_kmeans3'],
      dtype='object')
```

The results of the cluster profile were compared and shown in Table 1.5 The frequency count of the clusters revealed almost similar counts. The mean for the clusters with both hierarchical and KMeans showed similar trend. The ouput for the cluster profiles are given in Table 1.3 and 1.4 as shown in the above questions. Table 1.5 given in pdf TABLE 1.5 HIERARCHICAL KMEANS segment 1 2 HIGH 2 1 LOW 3 0 MODERATE

HIGH SEGMENT:This cluster has maximum credit_limit as they are spending more, maximum spent in

single_shopping (max_spent_in_single_shopping) and also with highest advance_payments options with highest current_balance and are having highest probability of full payment (probability_of_full_payment)

Promotional Strategy: Credit card usage is maximum and certain incentives of increasing their credit-limits can be an option to attract and maintain these customers.

MODERATE SEGMENT: This cluster has moderate credit_limit and all the variable means are showing moderate, except for min_payment_amount.

Promotional Strategy: To these segment interest rate deductions can be the best strategy so that they can increase their credit usage.

LOW SEGMENT: This cluster has minimum spending and the credit_limit, advance_payments, probability_of_full_payment,current_balance are also less.

Promotional Strategy: For this segment of clusters, attractive offers can increase their credit usage. To these segment interest rate deductions and long term payment options without interest can be the strategies so that they can increase their credit usage.

In [ ]:

```

```