Credit Card Segmentation

Himesh Nagar

18 September 2020

Contents

Chapter 1

Introduction

1.1 Problem Statement

This case requires to develop a customer segmentation to define marketing strategy. The sample dataset summarizes the usage behaviour of about 9000 active credit card holders during the last 6 months. The file is at a customer level with 18 behavioural variables.

1.2 Expectations

1. Advanced data preparation. Build an 'enriched' customer profile by deriving 'intelligent' KPIs such as monthly average purchase and cash advance amount, purchases by type (one-off, instalments), average amount per purchase and cash advance transaction, limit usage (balance to credit limit ratio), payments to minimum payments ratio etc.
2. Advanced reporting. Use the derived KPI's to gain insight on the customer profiles.
3. Clustering. Apply a data reduction technique factor analysis for variable reduction technique and a clustering algorithm to reveal the behavioural segments of credit card holders.

1.3 Data

Given below is a sample of the data set:

Table 1.1: Credit Card Segmentation Sample data (Columns: 1-5)

CUST_ID	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES
C10001	40.900749	0.818182	95.4	0
C10002	3202.467416	0.909091	0	0
C10003	2495.148862	1	773.17	773.17
C10004	1666.670542	0.636364	1499	1499
C10005	817.714335	1	16	16
C10006	1809.828751	1	1333.28	0

Table 1.2: Credit Card Segmentation Sample data (Columns: 6-8)

INSTALLMENTS_PURCHASES	CASH_ADVANCE	PURCHASES_FREQUENCY
95.4	0	0.166667
0	6442.945483	0
0	0	1
0	205.788017	0.083333
0	0	0.083333
1333.28	0	0.666667

Table 1.3: Credit Card Segmentation Sample data (Columns: 9-10)

ONEOFF_PURCHASES_FREQUENCY	PURCHASES_INSTALLMENTS_FREQUENCY
0	0.083333
0	0
1	0
0.083333	0
0.083333	0
0	0.583333

Table 1.4: Credit Card Segmentation Sample data (Columns: 11-15)

CASH_ADVANCE_FREQUENCY	CASH_ADVANCE_TRX	PURCHASES_TRX	CREDIT_LIMIT	PAYMENTS
0	0	2	1000	201.802084
0.25	4	0	7000	4103.032597
0	0	12	7500	622.066742
0.083333	1	1	7500	0
0	0	1	1200	678.334763
0	0	8	1800	1400.05777

Table 1.5: Credit Card Segmentation Sample data (Columns: 16-18)

MINIMUM_PAYMENTS	PRC_FULL_PAYMENT	TENURE
139.509787	0	12
1072.340217	0.222222	12
627.284787	0	12
NaN	0	12
244.791237	0	12
2407.246035	0	12

As you can see in the table 1.6 below we have the following 18 variables, using which we have to do the segmentation of customers:

Table 1.6: Predictor Variables

S.No.	Predictor
1	CUST_ID
2	BALANCE
3	BALANCE_FREQUENCY
4	PURCHASES
5	ONEOFF_PURCHASES
6	INSTALLMENTS_PURCHASES
7	CASH_ADVANCE
8	PURCHASES_FREQUENCY
9	ONEOFF_PURCHASES_FREQUENCY
10	PURCHASES_INSTALLMENTS_FREQUENCY
11	CASH_ADVANCE_FREQUENCY
12	CASH_ADVANCE_TRX
13	PURCHASES_TRX
14	CREDIT_LIMIT
15	PAYMENTS
16	MINIMUM_PAYMENTS
17	PRC_FULL_PAYMENT
18	TENURE

Chapter 2

Methodology

2.1 Advanced Data Preparation (KPIs)

Here, we will build an ‘enriched’ customer profile by deriving ‘intelligent’ KPIs such as monthly average purchase and cash advance amount, purchases by type (one-off, instalments), average amount per purchase and cash advance transaction, limit usage (balance to credit limit ratio), payments to minimum payments ratio etc.

- Monthly average purchase - We can get monthly average purchase for every customer by dividing their purchases with their tenure.
- Monthly average Cash Advance Amount - This can be achieved by dividing Cash advance by their tenure.
- Monthly Purchases by type (on-off, installments) - We have two types of purchases (on-off and installments), we can get monthly average for them by dividing it with tenure individually.
- Average Amount per purchase - This can be obtained by dividing purchase amount with number of transactions for purchase.
- Average Cash advance transaction - Dividing Cash Advance by number of times cash advance is taken by customers.
- Limit usage (balance to credit limit ratio) - We can get ratio by dividing balance with credit limit. This will help us to know how many users are spending more than their limit.
- Payments to minimum payments ratio - To know how much money do the customers pay respective to their minimum payments. Can be obtained by dividing Payments with minimum payments.

2.2 Insights of KPIs

We will plot some useful charts using these KPIs.

2.2.1 Number of users who used Purchases, cash advance and both

We can get this details using KPIs, Monthly average purchase and Monthly average Cash Advance Amount. The figure is below:

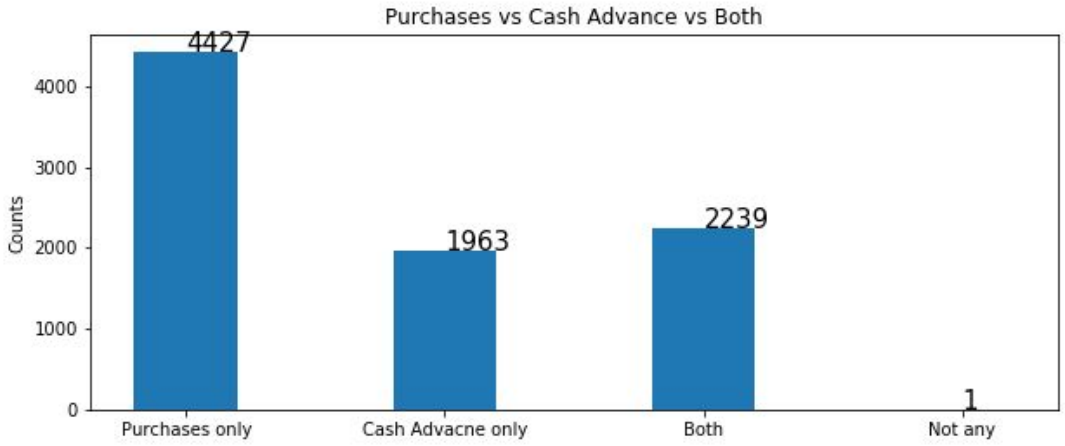

Figure 2.1 (See Python code in Appendix)

2.2.2 The total of average for purchase and cash advance

We can get this details using KPIs, Monthly average purchase and Monthly average Cash Advance Amount. The figure is below:

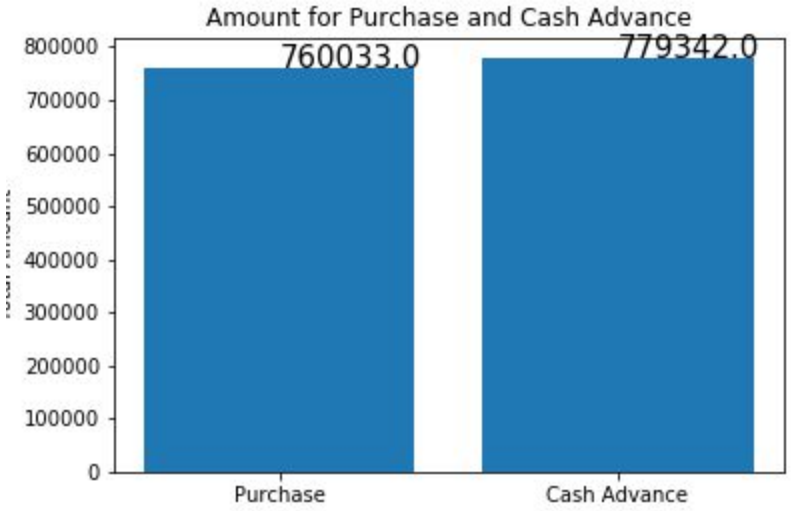

Figure 2.2 (See Python code in appendix)

2.2.3 Purchases comparison (one off, installments, both, not any)

This can be visualized using Monthly Purchases by type (on-off, installments). Figure is below:

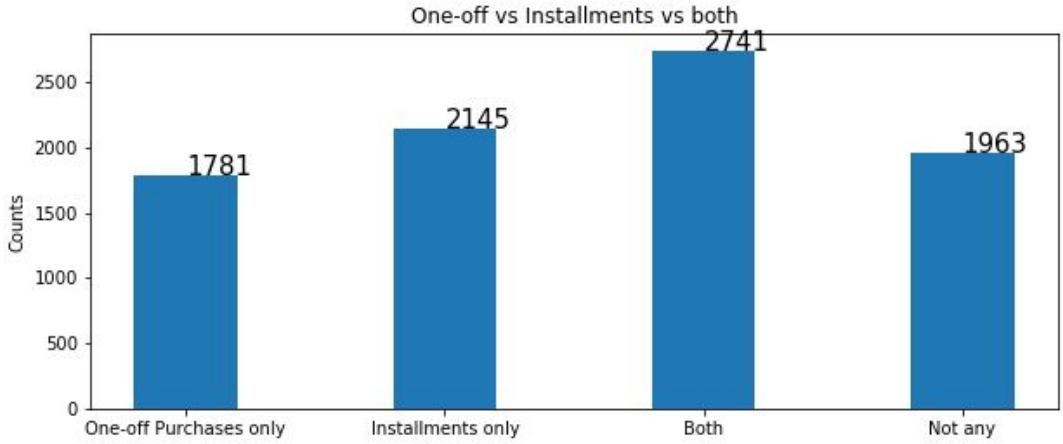

Figure 2.3 (See Python code in Appendix)

2.2.4 Total amount for each type of Purchases

This can be visualized using Monthly Purchases by type (on-off, installments). Figure is below:

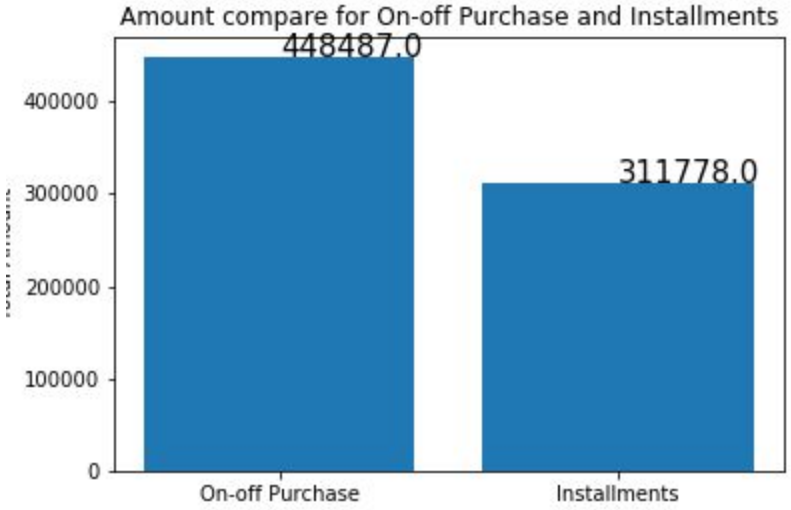

Figure 2.4 (See Python Code in Appendix)

2.2.5 Average amount per purchase and cash advance comparison

Here we will use the total of Average Amount per purchase and Average Cash advance transaction. Figure is below:

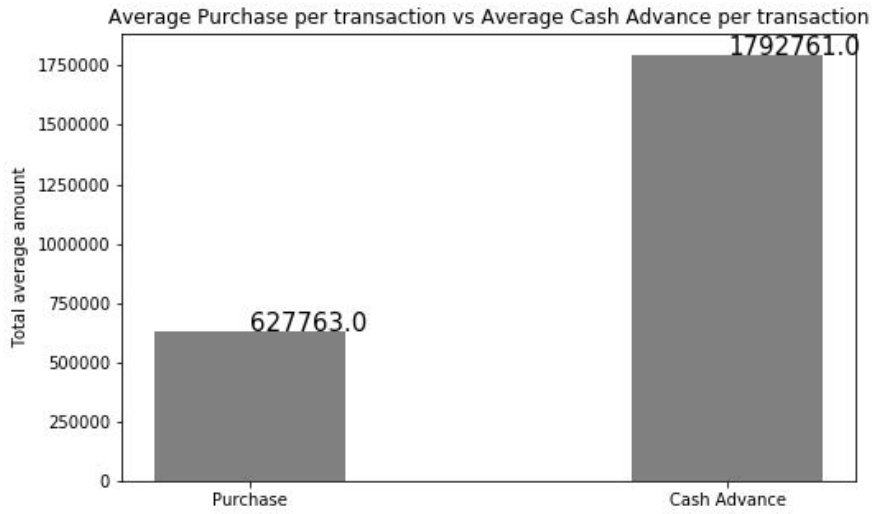

Figure 2.5 (See Python Code in Appendix)

2.2.6 Spending limit comparisons

Here, Limit usage (balance to credit limit ratio) KPI will help to find this. Figure is below:

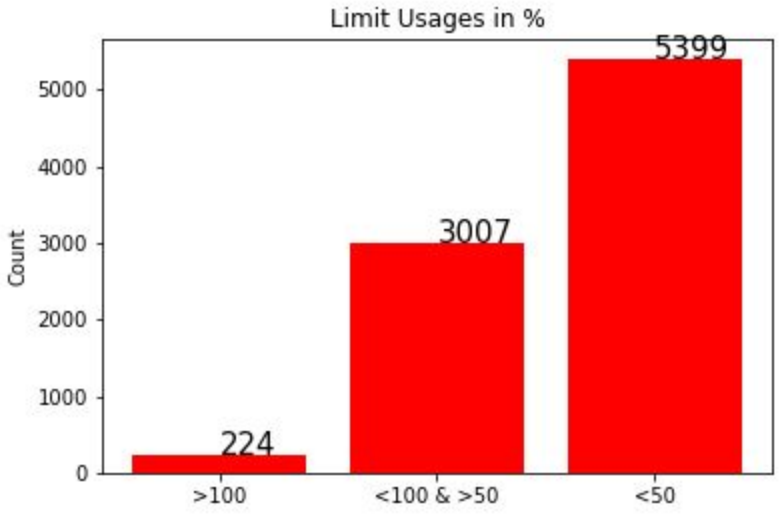

Figure 2.6 (See Python code in Appendix)

2.2.7 Payments more than minimum vs less than minimum

Payments to minimum payments ratio KPI is used.

Here, in the figure '<100' = Not paying enough, '>100' = Paying more than required.

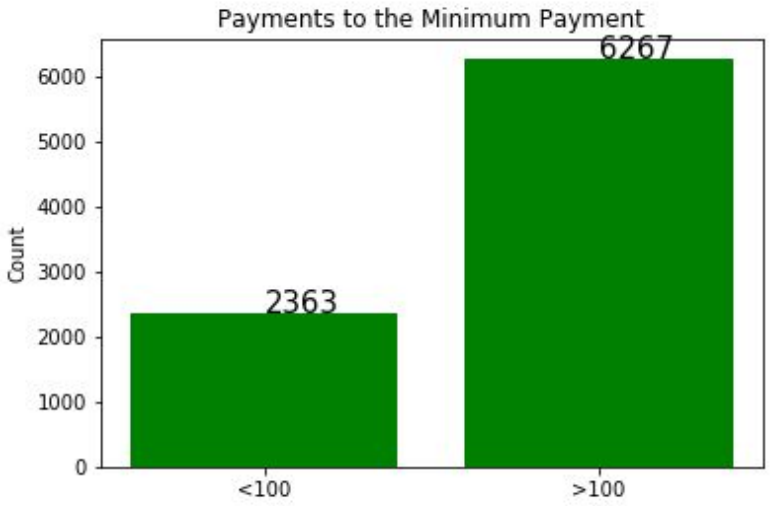

Figure 2.7 (See Python code in Appendix)

2.3 Preprocessing for Clustering

Here, we will do clustering as we have to segment the customers, but before that we have to do feature scaling, dimensionality reduction and selecting the number of clusters.

2.3.1 Feature Scaling

Here, normalization is used as the data is not normally distributed.

2.3.2 Dimensionality Reduction

For the dimensionality reduction Factor Analysis is used.

To decide how many features would be appropriate, scree plot is used. Figure is below:

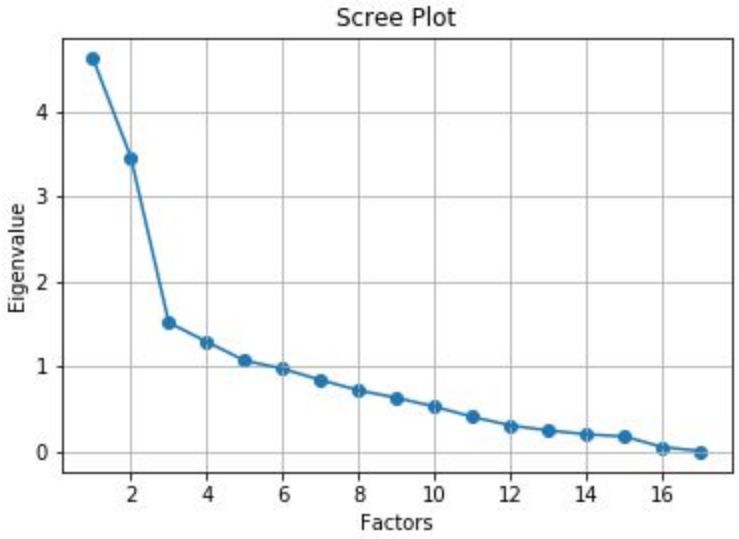

Figure 2.8 (See Python code in Appendix)

- From the figure, it is visible that there are 5 factors which have eigenvalue more than 1. So 5 features would be selected.

Chapter 3

Model

3.1 Methods to choose the number of clusters

To find optimum number of clusters WCSS (Within Cluster Sum of Squares) and Dendrogram are used.

1. WCSS score can be obtained using K-Means. And to choose the optimal number of clusters, elbow method would be used. Figure is below:

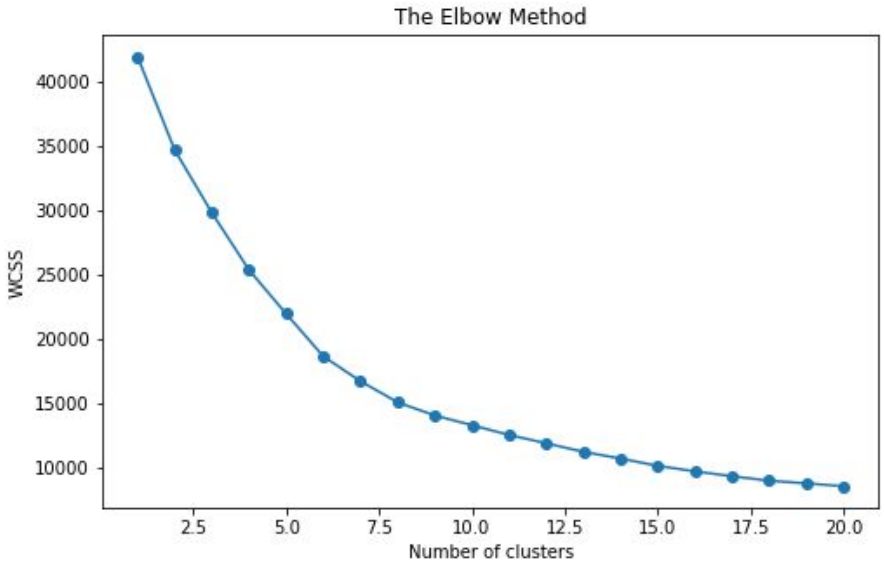

Figure 3.1 (See Python code in Appendix)

2. Dendrogram is one of the techniques which follows the bottom-up approach of hierarchical clustering. Figure is below:

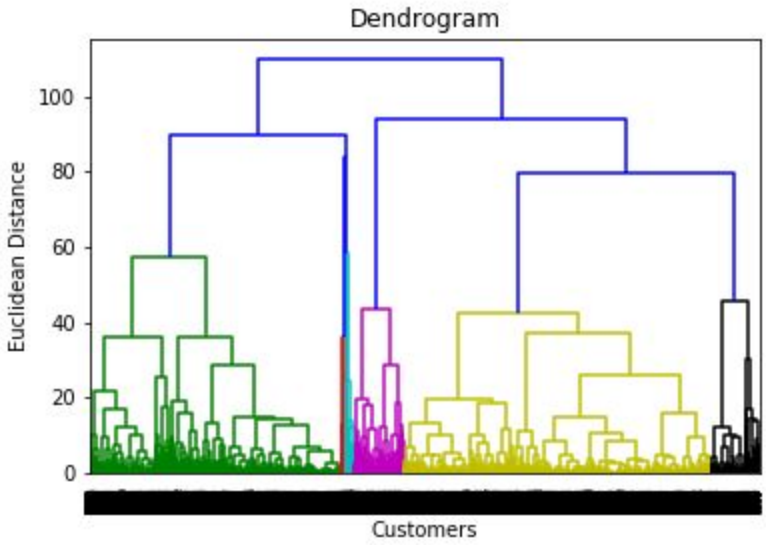

Figure 3.2 (See Python code in Appendix)

From both the techniques it is visible that 7 numbers of clusters would be appropriate.

3.2 K-Means Clustering

```
kmeans = KMeans(n_clusters=7, init='k-means++', n_init=10, max_iter=300, random_state=0)
y_kmeans = kmeans.fit_predict(X_fa)
pd.Series(y_kmeans).unique()

Output :
array([2, 0, 1, 4, 5, 6, 3], dtype=int64)
```

Here, we can see that we have got 7 clusters by applying K-Means Clustering.

Appendix A - Figures

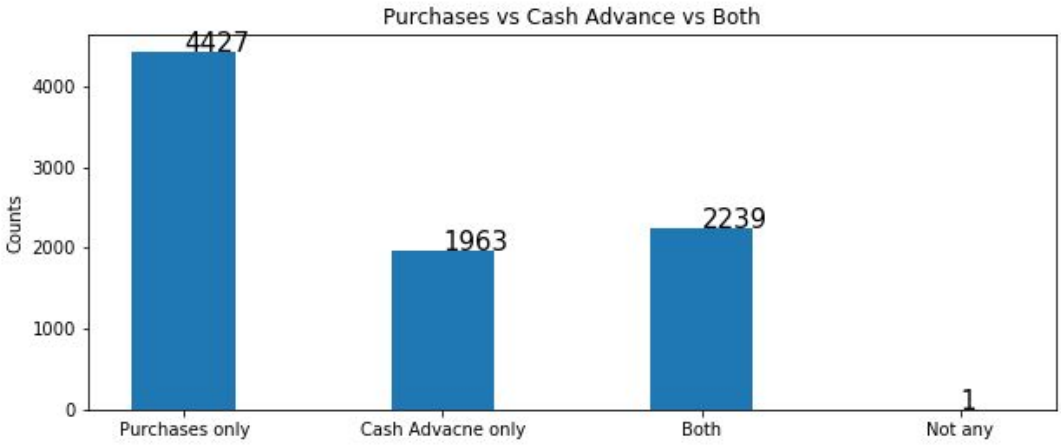

Figure 2.1

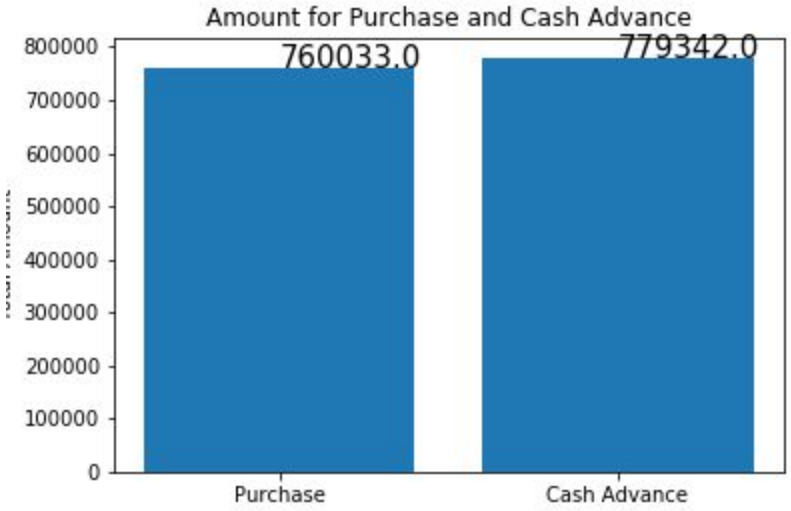

Figure 2.2

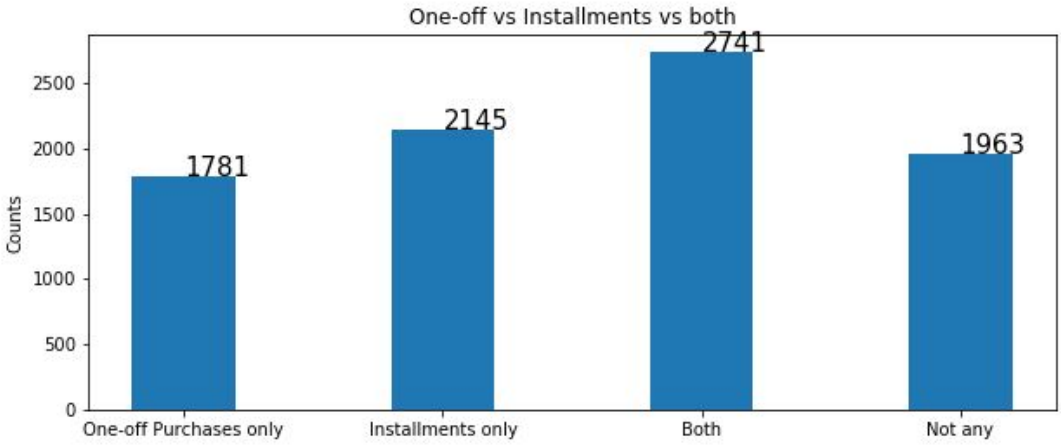

Figure 2.3

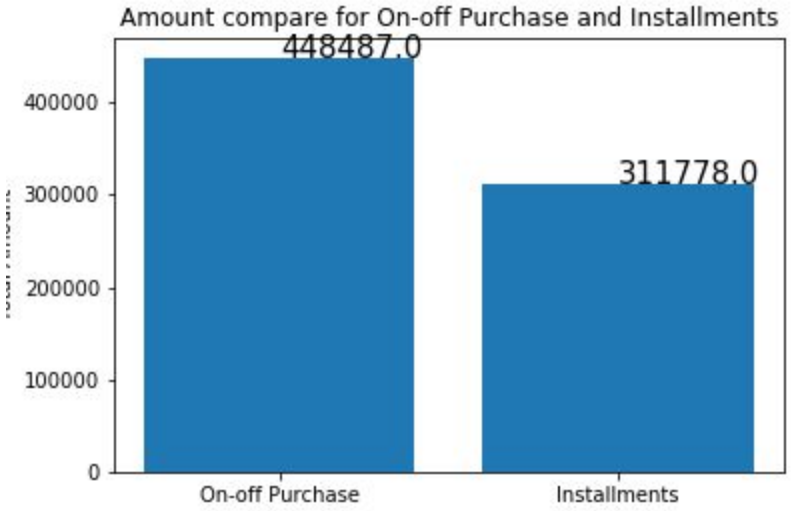

Figure 2.4

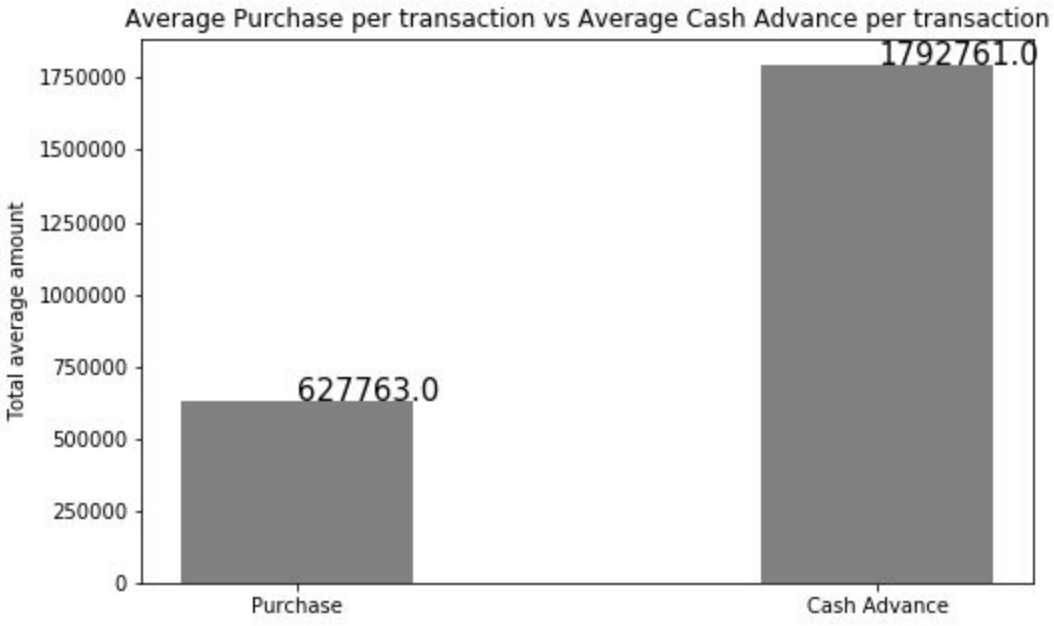

Figure 2.5

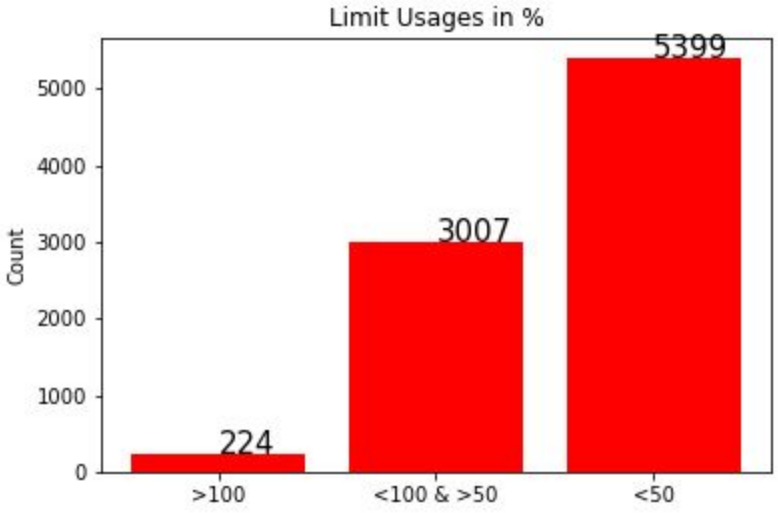

Figure 2.6

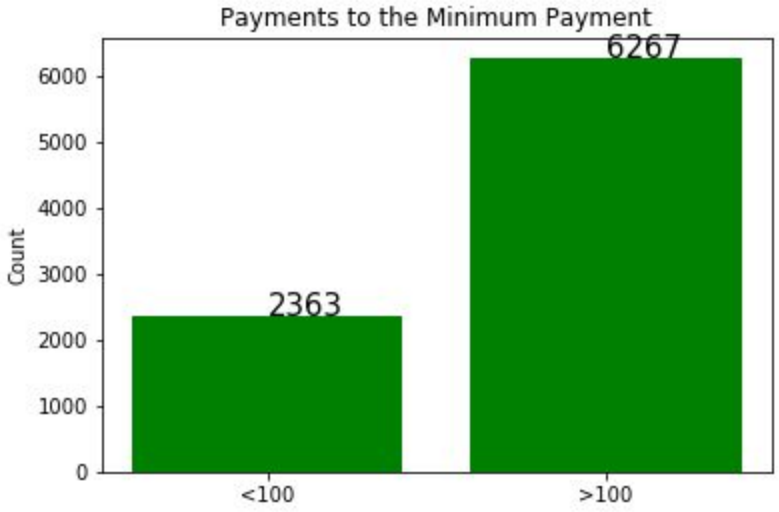

Figure 2.7

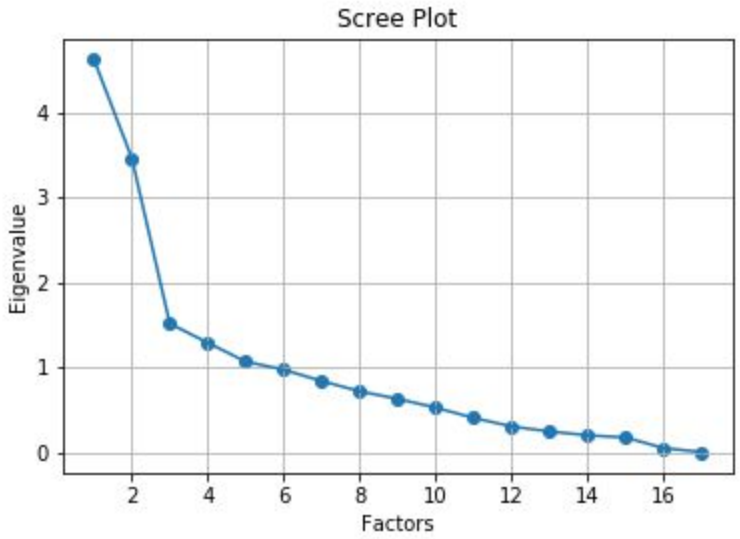

Figure 2.8

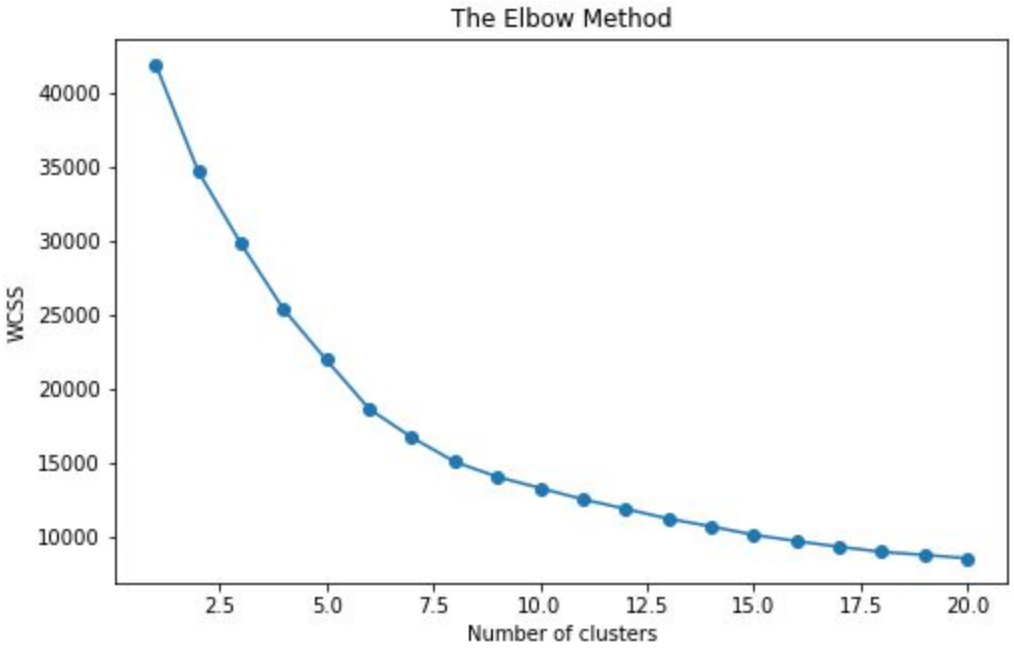

Figure 3.1

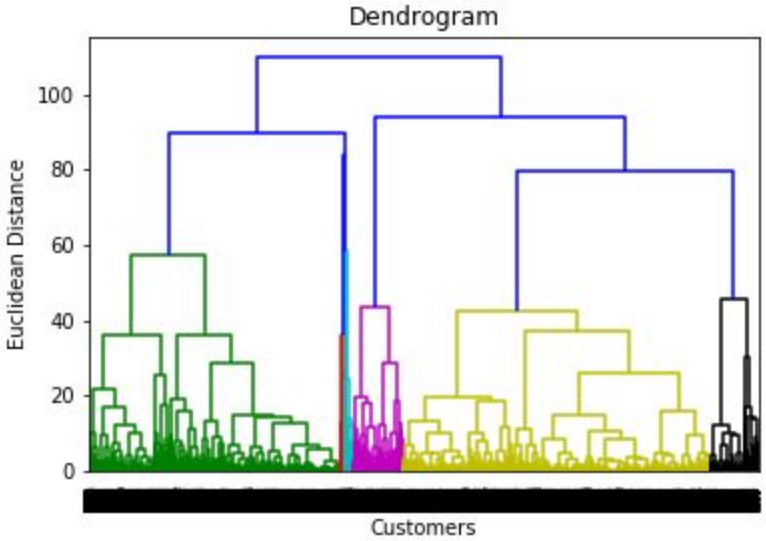

Figure 3.2

Appendix B - Python Code

Number of users who used Purchases, cash advance and both (Figure: 2.1)

```
counts = {
   'Purchases only':((card_df_KPI['monthly_avg_purchase']!=0) &
(card_df_KPI['monthly_cash_advance']==0)).sum(),
   'Cash Advance only':((card_df_KPI['monthly_cash_advance']!=0) &
(card_df_KPI['monthly_avg_purchase']==0)).sum(),
   'Both':((card_df_KPI['monthly_avg_purchase']!=0) &
(card_df_KPI['monthly_cash_advance']!=0)).sum(),
   'Not any':((card_df_KPI['monthly_avg_purchase']==0) &
(card_df_KPI['monthly_cash_advance']==0)).sum()
}
labels = counts.keys()
values = counts.values()
fig = plt.figure(figsize=(10,4))
plt.bar(labels,values, width=0.4)
for index,data in enumerate(values):
   plt.text(x=index , y =data+1 , s=f"{data}" , fontdict=dict(fontsize=15))
plt.ylabel('Counts')
plt.title("Purchases vs Cash Advance vs Both")
plt.show()
```

The total of average for purchase and cash advance (Figure: 2.2)

```
labels = ['Purchase','Cash Advance']
values = [card_df_KPI['monthly_avg_purchase'].sum(),
card_df_KPI['monthly_cash_advance'].sum()]
plt.bar(labels, values)
for index,data in enumerate(values):
   plt.text(x=index , y =data+1 , s=f"{round(data)}" , fontdict=dict(fontsize=15))
plt.title('Amount for Purchase and Cash Advance')
plt.ylabel("Total Amount")
plt.show()
```

Purchases comparison (one off, installments, both, not any) (Figure: 2.3)

```
counts = {
   'One-off Purchases only':((card_df_KPI['monthly_oneoff_purchase']!=0) &
(card_df_KPI['monthly_installment_purchase']==0)).sum(),
   'Installments only':((card_df_KPI['monthly_installment_purchase']!=0) &
(card_df_KPI['monthly_oneoff_purchase']==0)).sum(),
   'Both':((card_df_KPI['monthly_oneoff_purchase']!=0) &
(card_df_KPI['monthly_installment_purchase']!=0)).sum(),
   'Not any':((card_df_KPI['monthly_oneoff_purchase']==0) &
(card_df_KPI['monthly_installment_purchase']==0)).sum()
}
labels = counts.keys()
values = counts.values()
fig = plt.figure(figsize=(10,4))
plt.bar(labels,values, width=0.4)
for index,data in enumerate(values):
   plt.text(x=index , y =data+1 , s=f"{data}" , fontdict=dict(fontsize=15))
plt.title("One-off vs Installments vs both")
plt.ylabel('Counts')
plt.show()
```

Total amount for each type of Purchases (Figure: 2.4)

```
labels = ['On-off Purchase','Installments']
values = [card_df_KPI['monthly_oneoff_purchase'].sum(),
card_df_KPI['monthly_installment_purchase'].sum()]
plt.bar(labels, values)
for index,data in enumerate(values):
   plt.text(x=index , y =data+1 , s=f"{round(data)}" , fontdict=dict(fontsize=15))
plt.title('Amount compare for On-off Purchase and Installments')
plt.ylabel("Total Amount")
plt.show()
```

Average amount per purchase and cash advance comparison (Figure: 2.5)

```
avg_purchase = card_df_KPI['average_amount_per_purchase'].sum()
avg_cash = card_df_KPI['average_amount_per_cash_advance'].sum()
labels = ['Purchase','Cash Advance']
values = [avg_purchase, avg_cash]
fig = plt.figure(figsize=(8,5))
plt.bar(labels, values, color='grey', width=0.4)
for index,data in enumerate(values):
    plt.text(x=index , y =data+1 , s=f"{round(data)}" , fontdict=dict(fontsize=15))
plt.title('Average Purchase per transaction vs Average Cash Advance per transaction')
plt.ylabel("Total average amount")
plt.show()
```

Spending limit comparisons (Figure: 2.6)

```
more_limit = (card_df_KPI['b2c_lr']>100).sum()
less_limit = ((card_df_KPI['b2c_lr']>50) & (card_df_KPI['b2c_lr']<100)).sum()
less_50 = (card_df_KPI['b2c_lr']<50).sum()
labels = ['>100','<100 & >50','<50']
values = [more_limit, less_limit, less_50]
plt.bar(labels,values, color='red')
plt.title("Limit Usages in %")
plt.ylabel("Count")
for index,data in enumerate(values):
    plt.text(x=index , y =data+1 , s=f"{round(data)}" , fontdict=dict(fontsize=15))
plt.show()
```

Payments more than minimum vs less than minimum (Figure: 2.7)

```
paying_less = (card_df_KPI['pay_to_minpay_ratio']<100).sum()
paying_more = (card_df_KPI['pay_to_minpay_ratio']>100).sum()
values = [paying_less,  paying_more]
labels = ['<100', '>100']
plt.bar(labels,values, color='green')
plt.title("Payments to the Minimum Payment")
```

```
plt.ylabel("Count")
for index,data in enumerate(values):
    plt.text(x=index , y =data+1 , s=f"{round(data)}" , fontdict=dict(fontsize=15))
plt.show()
```

Scree Plot (Figure: 2.8)

```
plt.scatter(range(1,card_df_norm.shape[1]+1),ev)
plt.plot(range(1,card_df_norm.shape[1]+1),ev)
plt.title('Scree Plot')
plt.xlabel('Factors')
plt.ylabel('Eigenvalue')
plt.grid()
plt.show()
```

WCSS Score **(Figure: 3.1)**

```
wcss = [] # Within cluster sum of squares
num = 21
for i in range(1,num):
    kmeans = KMeans(n_clusters=i)
    kmeans.fit(X_fa)
    wcss.append(kmeans.inertia_)
fig = plt.figure(figsize=(8,5))
plt.plot(range(1,num), wcss, marker="o")
plt.title('The Elbow Method')
plt.xlabel('Number of clusters')
plt.ylabel('WCSS')
plt.show()
```

Dendrogram (Figure: 3.2)

```
dendrogram = sch.dendrogram(sch.linkage(X_fa, method='ward'))
plt.title('Dendrogram')
plt.xlabel('Customers')
plt.ylabel('Euclidean Distance')
```

Complete Python Code

```
# # Import libraries


# In[1]:



import pandas as pd
from factor_analyzer.factor_analyzer import calculate_bartlett_sphericity,
calculate_kmo
from sklearn.decomposition import FactorAnalysis
from factor_analyzer import FactorAnalyzer
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
import scipy.cluster.hierarchy as sch
from sklearn.preprocessing import MinMaxScaler



# # Load data


# In[2]:



card_df = pd.read_csv("credit-card-data.csv")
card_df.head()



# In[3]:



total_rows = len(card_df)
```

```
card_df.shape



# # Missing Value Analysis

# In[4]:



missing_val = pd.DataFrame(card_df.isnull().sum(), columns = ['Count'])
missing_val['Percentage'] = (missing_val.loc[:,'Count']*100)/len(card_df)



# In[5]:



missing_val



# ### Remove Missing values

# In[6]:



# card_df.drop((card_df[card_df['CREDIT_LIMIT'].isnull() |
card_df['MINIMUM_PAYMENTS'].isnull()]).index, inplace=True)
card_df.dropna(inplace=True)
print(f"DataLoss After removing missing values :
{total_rows-len(card_df)}")



# In[7]:
```

```
card_df.info()


# Data where purchases are zero but they have pruchase transactions

# In[8]:


card_df[(card_df['PURCHASES']==0) & (card_df['PURCHASES_TRX']!=0)]


# Data where pruchase transactions are zero but they have purchases

# In[9]:


card_df[(card_df['PURCHASES']!=0) & (card_df['PURCHASES_TRX']==0)]


# In[10]:


card_df[((card_df['PURCHASES']==0) & (card_df['PURCHASES_TRX']!=0)) |
((card_df['PURCHASES']!=0) & (card_df['PURCHASES_TRX']==0))]


# #### Remove above data as it seems to be a false data

# In[11]:
```

```
print("Rows before delete : ",len(card_df))
card_df.drop(card_df[((card_df['PURCHASES']==0) &
(card_df['PURCHASES_TRX']!=0)) | ((card_df['PURCHASES']!=0) &
(card_df['PURCHASES_TRX']==0))].index, inplace=True)
print("Rows after delete : ",len(card_df))



# Data where cash advance is zero but they have cash advance transactions,
# Data where cash advance transactions is zero but they have cash advance

# In[12]:



card_df[((card_df['CASH_ADVANCE']==0) & (card_df['CASH_ADVANCE_TRX']!=0))
| ((card_df['CASH_ADVANCE']!=0) & (card_df['CASH_ADVANCE_TRX']==0))]



# # 1. Deriving key performance indicators(KPI)

# - 1.1 - monthly average purchase
# - 1.2 - monthly cash advance amount
# - 1.3 - purchases by type (one-off, instalments)
# - 1.4 - average amount per purchase
# - 1.5 - cash advance transaction
# - 1.6 - limit usage (balance to credit limit ratio)
# - 1.7 - payments to minimum payments ratio
#

# In[13]:
```

```
# TENURE Number of months as a customer
card_df_KPI = pd.DataFrame(card_df['CUST_ID'])
card_df.TENURE.unique()


# ### 1.1 - monthly average purchase

# In[14]:


# PURCHASES Total purchase amount spent during last 12 months
card_df_KPI['monthly_avg_purchase'] = card_df.PURCHASES / card_df.TENURE
card_df_KPI.head()


# ### 1.2 - monthly cash advance amount

# In[15]:


card_df_KPI['monthly_cash_advance'] = card_df.CASH_ADVANCE /
card_df.TENURE
card_df_KPI.head()


# ### 1.3 - monthly purchases by type (one-off, instalments)

# In[16]:
```

```
card_df_KPI['monthly_oneoff_purchase'] = card_df.ONEOFF_PURCHASES /
card_df.TENURE
card_df_KPI['monthly_installment_purchase'] =
card_df.INSTALLMENTS_PURCHASES / card_df.TENURE
card_df_KPI.head()



# ### 1.4 - average amount per purchase


# In[17]:



card_df_KPI['average_amount_per_purchase'] =
card_df.PURCHASES/card_df.PURCHASES_TRX
card_df_KPI.head()



# Fill nan with 0


# In[18]:



card_df_KPI['average_amount_per_purchase'] =
card_df_KPI['average_amount_per_purchase'].fillna(0)
card_df_KPI.head()



# ### 1.5 - average cash advance transaction


# In[19]:
```

```
card_df_KPI['average_amount_per_cash_advance'] =
card_df.CASH_ADVANCE/card_df.CASH_ADVANCE_TRX
card_df_KPI.head()



# Fill NaN with 0

# In[20]:



card_df_KPI['average_amount_per_cash_advance'] =
card_df_KPI['average_amount_per_cash_advance'].fillna(0)
card_df_KPI.head()



# ### 1.6 - limit usage (balance to credit limit ratio)

# In[21]:



card_df_KPI['b2c_lr'] = card_df.BALANCE/ card_df.CREDIT_LIMIT * 100
card_df_KPI.head()



# ### 1.7 - payments to minimum payments ratio

# In[22]:
```

```
card_df_KPI['pay_to_minpay_ratio'] = card_df.PAYMENTS/
card_df.MINIMUM_PAYMENTS * 100
card_df_KPI.head()



# # 2. Insights using KPIs
#


# ### 2.1 Number of users who used Purchases, cash advance and both


# In[23]:



counts = {
    'Purchases only':((card_df_KPI['monthly_avg_purchase']!=0) &
(card_df_KPI['monthly_cash_advance']==0)).sum(),
    'Cash Advance only':((card_df_KPI['monthly_cash_advance']!=0) &
(card_df_KPI['monthly_avg_purchase']==0)).sum(),
    'Both':((card_df_KPI['monthly_avg_purchase']!=0) &
(card_df_KPI['monthly_cash_advance']!=0)).sum(),
    'Not any':((card_df_KPI['monthly_avg_purchase']==0) &
(card_df_KPI['monthly_cash_advance']==0)).sum()
}
labels = counts.keys()
values = counts.values()
fig = plt.figure(figsize=(10,4))
plt.bar(labels,values, width=0.4)
for index,data in enumerate(values):
    plt.text(x=index , y =data+1 , s=f"{data}" ,
fontdict=dict(fontsize=15))
plt.ylabel('Counts')
```

```
plt.title("Purchases vs Cash Advance vs Both")
plt.savefig('images/2.1.png')
plt.show()



# ### 2.2 Let's compare the amounts of purchases and cash advance of
monthly usages


# In[24]:



labels = ['Purchase','Cash Advance']
values = [card_df_KPI['monthly_avg_purchase'].sum(),
card_df_KPI['monthly_cash_advance'].sum()]
plt.bar(labels, values)
for index,data in enumerate(values):
    plt.text(x=index , y =data+1 , s=f"{round(data)}" ,
fontdict=dict(fontsize=15))
plt.title('Amount for Purchase and Cash Advance')
plt.ylabel("Total Amount")
plt.savefig('images/2.2.png')
plt.show()



# ### Now, Let's Dig into purchases


# #### 2.3 We have two types of purchases, so let's find out how many uses
oneoff, installments or both


# In[25]:
```

```
counts = {
    'One-off Purchases only':((card_df_KPI['monthly_oneoff_purchase']!=0)
& (card_df_KPI['monthly_installment_purchase']==0)).sum(),
    'Installments only':((card_df_KPI['monthly_installment_purchase']!=0)
& (card_df_KPI['monthly_oneoff_purchase']==0)).sum(),
    'Both':((card_df_KPI['monthly_oneoff_purchase']!=0) &
(card_df_KPI['monthly_installment_purchase']!=0)).sum(),
    'Not any':((card_df_KPI['monthly_oneoff_purchase']==0) &
(card_df_KPI['monthly_installment_purchase']==0)).sum()
}
labels = counts.keys()
values = counts.values()
fig = plt.figure(figsize=(10,4))
plt.bar(labels,values, width=0.4)
for index,data in enumerate(values):
    plt.text(x=index , y =data+1 , s=f"{data}" ,
fontdict=dict(fontsize=15))
plt.title("One-off vs Installments vs both")
plt.ylabel('Counts')
plt.savefig('images/2.3.png')
plt.show()



# #### 2.4 Total amount for each type of Purchases


# In[26]:



labels = ['On-off Purchase','Installments']
```

```
values = [card_df_KPI['monthly_oneoff_purchase'].sum(),
card_df_KPI['monthly_installment_purchase'].sum()]
plt.bar(labels, values)
for index,data in enumerate(values):
    plt.text(x=index , y =data+1 , s=f"{round(data)}" ,
fontdict=dict(fontsize=15))
plt.title('Amount compare for On-off Purchase and Installments')
plt.ylabel("Total Amount")
plt.savefig('images/2.4.png')
plt.show()



# ### 2.5 Average amount per purchase and per cash advance comparison


# In[27]:



avg_purchase = card_df_KPI['average_amount_per_purchase'].sum()
avg_cash = card_df_KPI['average_amount_per_cash_advance'].sum()
labels = ['Purchase','Cash Advance']
values = [avg_purchase, avg_cash]
fig = plt.figure(figsize=(8,5))
plt.bar(labels, values, color='grey', width=0.4)
for index,data in enumerate(values):
    plt.text(x=index , y =data+1 , s=f"{round(data)}" ,
fontdict=dict(fontsize=15))
plt.title('Average Purchase per transaction vs Average Cash Advance per
transaction')
plt.ylabel("Total average amount")
plt.savefig('images/2.5.png')
plt.show()
```

```
# ### 2.6 People spending more than credit limit vs under credit limit but more than 50% vs less than 50%

# In[28]:



# card_df_KPI['b2c_lr']
more_limit = (card_df_KPI['b2c_lr']>100).sum()
less_limit = ((card_df_KPI['b2c_lr']>50) & (card_df_KPI['b2c_lr']<100)).sum()
less_50 = (card_df_KPI['b2c_lr']<50).sum()
labels = ['>100','<100 & >50','<50']
values = [more_limit, less_limit, less_50]
plt.bar(labels,values, color='red')
plt.title("Limit Usages in %")
plt.ylabel("Count")
for index,data in enumerate(values):
    plt.text(x=index , y =data+1 , s=f"{round(data)}" , fontdict=dict(fontsize=15))
plt.savefig('images/2.6.png')
plt.show()



# ### 2.7 People who pay more than their minimum payments vs less than the minimum payment range

# Here,
# - <100 = Not paying enough
# - '>100 = Paying more than required
```

```
#


# In[29]:



paying_less = (card_df_KPI['pay_to_minpay_ratio']<100).sum()
paying_more = (card_df_KPI['pay_to_minpay_ratio']>100).sum()
values = [paying_less,  paying_more]
labels = ['<100', '>100']
plt.bar(labels,values, color='green')
plt.title("Payments to the Minimum Payment")
plt.ylabel("Count")
for index,data in enumerate(values):
    plt.text(x=index , y =data+1 , s=f"{round(data)}" ,
fontdict=dict(fontsize=15))
plt.savefig('images/2.7.png')
plt.show()



# # 3. Clustering

# Droping CUST_ID as there is no need

# In[30]:



card_df.drop('CUST_ID', axis=1,inplace=True)



# ## Feature scaling
```

```
# Normalizing the data

# In[35]:



card_df_norm = MinMaxScaler().fit_transform(card_df)
pd.DataFrame(card_df_norm).head()



# ## 3.1 Dimensionality Reduction - Factor Analysis

# Bartlett’s test of sphericity checks whether or not the observed
variables intercorrelate at all using the observed correlation matrix
against the identity matrix. If the test found statistically
insignificant, you should not employ a factor analysis
#

# In[36]:



chi_square_value,p_value=calculate_bartlett_sphericity(card_df_norm)
print(chi_square_value, p_value)



# ##### In this Bartlett ’s test, the p-value is 0. The test was
statistically significant, indicating that the observed correlation matrix
is not an identity matrix.
#

# Kaiser-Meyer-Olkin (KMO) Test measures the suitability of data for
factor analysis. It determines the adequacy for each observed variable and
for the complete model. KMO estimates the proportion of variance among all
```

```
the observed variable. Lower proportion id more suitable for factor
analysis. KMO values range between 0 and 1. Value of KMO less than 0.6 is
considered inadequate.


# In[37]:



kmo_all,kmo_model=calculate_kmo(card_df_norm)
kmo_model



# ###### Here kmo_model value is 0.64, so it is adequate.


# ### Choosing the Number of Factors


# In[38]:



fa = FactorAnalyzer()
fa.set_params(n_factors=25, rotation=None)
fa.fit(card_df_norm)
# Check Eigenvalues
ev, v = fa.get_eigenvalues()
ev



# In[39]:



# Create scree plot using matplotlib
plt.scatter(range(1,card_df_norm.shape[1]+1),ev)
```

```
plt.plot(range(1,card_df_norm.shape[1]+1),ev)
plt.title('Scree Plot')
plt.xlabel('Factors')
plt.ylabel('Eigenvalue')
plt.grid()
plt.savefig('images/Scree Plot.png')
plt.show()



# The scree plot method draws a straight line for each factor and its
eigenvalues. Number eigenvalues greater than one considered as the number
of factors.

# Here, you can see only for 5-factors eigenvalues are greater than one.
It means we need to choose only 5 factors (or unobserved variables).

# In[40]:



sk_fa = FactorAnalysis(n_components=5, random_state=0)
X_fa = sk_fa.fit_transform(card_df_norm)
# sk_fa.components_



# In[41]:



X_fa.shape
# sk_fa.components_.shape
```

```
# ## Let's find out how mant clusters we can make


# #### K-Means wcss score


# In[55]:



wcss = [] # Within cluster sum of squares
num = 21
for i in range(1,num):
    kmeans = KMeans(n_clusters=i)
    kmeans.fit(X_fa)
    wcss.append(kmeans.inertia_)
fig = plt.figure(figsize=(8,5))
plt.plot(range(1,num), wcss, marker="o")
plt.title('The Elbow Method')
plt.xlabel('Number of clusters')
plt.ylabel('WCSS')
plt.savefig('images/Wcss.png')
plt.show()



# ### Dendrogram


# In[43]:



dendrogram = sch.dendrogram(sch.linkage(X_fa, method='ward'))
plt.title('Dendrogram')
plt.xlabel('Customers')
```

```
plt.ylabel('Euclidean Distance')
plt.savefig('images/Dendrogram.png')
plt.show()



# #### Based on elbow method and dendrogram we can see that  7 clusters
are required


# In[68]:



kmeans = KMeans(n_clusters=7, init='k-means++', n_init=10, max_iter=300,
random_state=0)
y_kmeans = kmeans.fit_predict(X_fa)



# In[69]:



pd.Series(y_kmeans).unique()



# In[70]:



clustered_data = pd.concat([card_df_KPI['CUST_ID'],pd.DataFrame(X_fa),
pd.DataFrame(y_kmeans)], axis=1)
clustered_data.head()



# In[71]:
```

```
card_df_clustered = pd.concat([card_df_KPI['CUST_ID'],card_df,
pd.DataFrame(y_kmeans)], axis=1).dropna()
card_df_clustered.rename(columns={0:'Clusters'}, inplace=True)
card_df_clustered.head()
```