AllLife Bank Customer Segmentation - Problem Statement

Description:

AllLife Bank wants to focus on its credit card customer base in the next financial year. They have been advised by their marketing research team, that the penetration in the market can be improved. Based on this input, the Marketing team proposes to run personalized campaigns to target new customers as well as upsell to existing customers. Another insight from the market research was that the customers perceive the support services of the back poorly. Based on this, the Operations team wants to upgrade the service delivery model, to ensure that customer queries are resolved faster. Head of Marketing and Head of Delivery both decide to reach out to the Data Science team for help

Objective:

To identify different segments in the existing customer, based on their spending patterns as well as past interaction with the bank, using clustering algorithms, and provide recommendations to the bank on how to better market to and service these customers.

Data Description:

The data provided is of various customers of a bank and their financial attributes like credit limit, the total number of credit cards the customer has, and different channels through which customers have contacted the bank for any queries (including visiting the bank, online and through a call center).

Data Dictionary:

- **Sl_No:** Primary key of the records
- **Customer Key:** Customer identification number
- **Average Credit Limit:** Average credit limit of each customer for all credit cards
- **Total credit cards:** Total number of credit cards possessed by the customer
- **Total visits bank:** Total number of visits that customer made (yearly) personally to the bank
- **Total visits online:** Total number of visits or online logins made by the customer (yearly)
- **Total calls made:** Total number of calls made by the customer to the bank or its customer service department (yearly)

Importing Necessary Libraries

In [280…

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

#KMeans clustering
from sklearn.cluster import KMeans
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from scipy.spatial.distance import cdist
from sklearn import metrics
from sklearn.metrics import silhouette_score

#Hierarchical clustering
from scipy.cluster.hierarchy import cophenet, dendrogram, linkage
from sklearn.cluster import AgglomerativeClustering
from scipy.cluster.hierarchy import fcluster
from scipy.spatial.distance import pdist
from sklearn.preprocessing import StandardScaler
```

Read the dataset

In [281…

```
data = pd.read_excel('Credit+Card+Customer+Data.xlsx')
```

Summary of the dataset

In [282…

```
data.describe()
```

Out[282…

	Sl_No	Customer Key	Avg_Credit_Limit	Total_Credit_Cards	Total_visits_bank	Total_visits_online	Total_calls_made

count	660.000000	660.000000	660.000000	660.000000	660.000000	660.000000	660.000000
mean	330.500000	55141.443939	34574.242424	4.706061	2.403030	2.606061	3.583333
std	190.669872	25627.772200	37625.487804	2.167835	1.631813	2.935724	2.865317
min	1.000000	11265.000000	3000.000000	1.000000	0.000000	0.000000	0.000000
25%	165.750000	33825.250000	10000.000000	3.000000	1.000000	1.000000	1.000000
50%	330.500000	53874.500000	18000.000000	5.000000	2.000000	2.000000	3.000000
75%	495.250000	77202.500000	48000.000000	6.000000	4.000000	4.000000	5.000000
max	660.000000	99843.000000	200000.000000	10.000000	5.000000	15.000000	10.000000

Understand the shape of the dataset.

In [283…

```
data.shape
```

Out[283…

```
(660, 7)
```

Check the data types of the columns for the dataset.

In [284…

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 660 entries, 0 to 659
Data columns (total 7 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   Sl_No                660 non-null    int64
 1   Customer Key         660 non-null    int64
 2   Avg_Credit_Limit     660 non-null    int64
 3   Total_Credit_Cards   660 non-null    int64
 4   Total_visits_bank    660 non-null    int64
 5   Total_visits_online  660 non-null    int64
 6   Total_calls_made     660 non-null    int64
dtypes: int64(7)
memory usage: 36.2 KB
```

In [285…

```
# To check number of unique elements in each columns
data.nunique()
```

Out[285…

```
Sl_No                  660
Customer Key           655
Avg_Credit_Limit       110
Total_Credit_Cards      10
Total_visits_bank        6
Total_visits_online     16
Total_calls_made        11
dtype: int64
```

Check for missing values

In [286…

```
data.isnull().sum()
```

Out[286…

```
Sl_No                  0
Customer Key           0
Avg_Credit_Limit       0
Total_Credit_Cards     0
Total_visits_bank      0
Total_visits_online    0
Total_calls_made       0
dtype: int64
```

In [287…

```
#Drop columns 'Sl_No' & 'Customer_Key' since its unique id
data.drop(['Sl_No','Customer Key'],axis=1,inplace=True)
```

In [288…

```
# drop duplicated rows
data.drop_duplicates(inplace=True)
```

In [289…

```
data = data.reset_index(drop=True)
```

In [290…

```
data
```

Out[290…

	Avg_Credit_Limit	Total_Credit_Cards	Total_visits_bank	Total_visits_online	Total_calls_made
0	100000	2	1	1	0
1	50000	3	0	10	9
2	50000	7	1	3	4
3	30000	5	1	1	4
4	100000	6	0	12	3
...	...	...	...	...	...
644	99000	10	1	10	0
645	84000	10	1	13	2
646	145000	8	1	9	1
647	172000	10	1	15	0
648	167000	9	0	12	2

649 rows × 5 columns

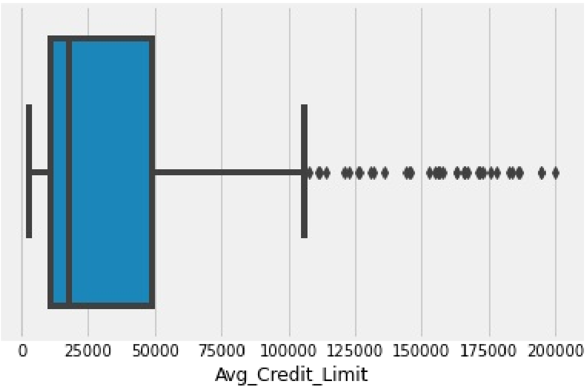
Data Visualization - Univariate analysis

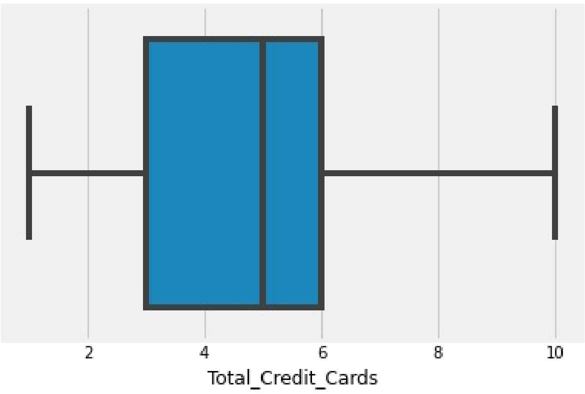
- Univariate analysis refer to the analysis of a single variable. The main purpose of univariate analysis is to summarize and find patterns in the data. The key point is that there is only one variable involved in the analysis.

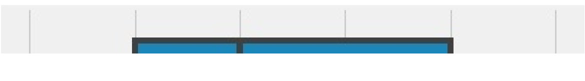
  Let us take the loan dataset and work on that for the univariate analysis.

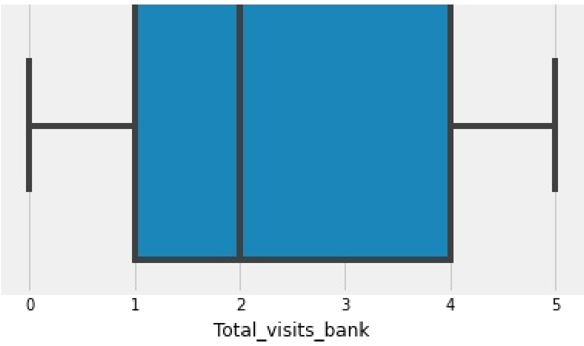
In [291…

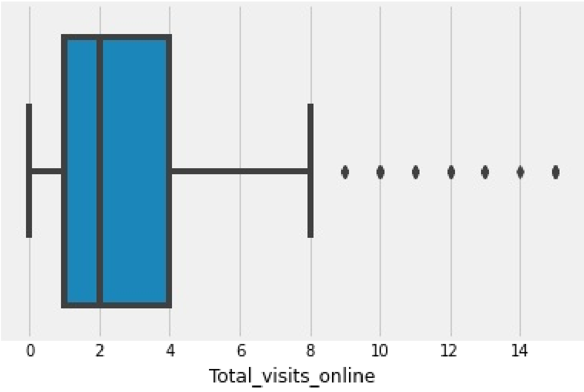
```
for column in data.columns:
    sns.boxplot(x=data[column])
    plt.show()
```

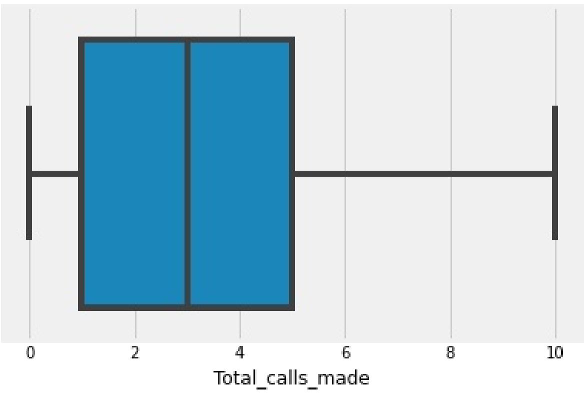












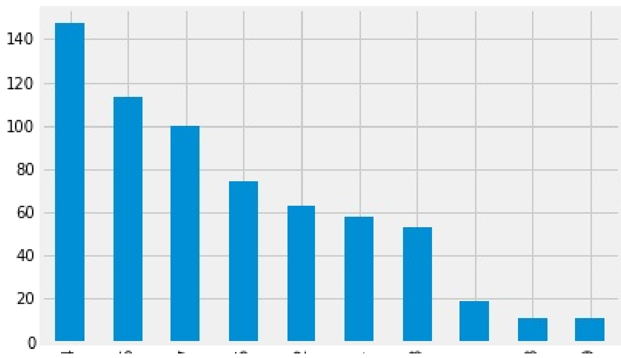
Observations:

- Avg_Credit_Limit & Total_visits_online have outliers.However this doesn't mean that we have to deal with these outliers, it could be just the fact that customer has more credit limit and more visits online than others.

In [292…

```
data['Total_Credit_Cards'].value_counts().plot(kind='bar');
print(data['Total_Credit_Cards'].value_counts(normalize=True))
```

```
4     0.226502
6     0.174114
7     0.154083
5     0.114022
2     0.097072
1     0.089368
3     0.081664
10    0.029276
8     0.016949
9     0.016949
Name: Total_Credit_Cards, dtype: float64
```



In [293…

```
data.loc[data['Total_Credit_Cards']>=4].shape[0] / data.shape[0]
```

Out[293…

```
0.7318952234206472
```

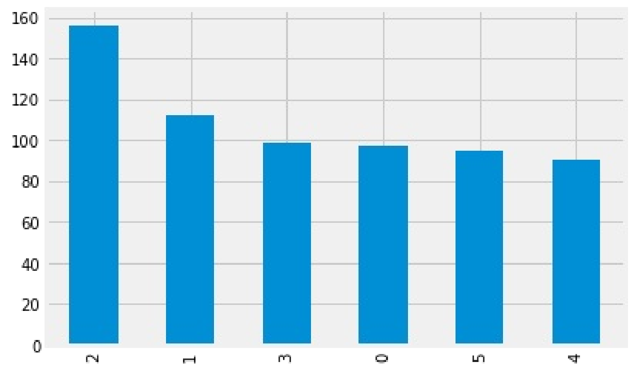
Observations:

- Approx. 73% of the customers has at least 4 credit cards or more.

In [294…

```
data['Total_visits_bank'].value_counts().plot(kind='bar');
print(data['Total_visits_bank'].value_counts(normalize=True))
```

```
2    0.240370
1    0.172573
3    0.152542
0    0.149461
5    0.146379
4    0.138675
Name: Total_visits_bank, dtype: float64
```



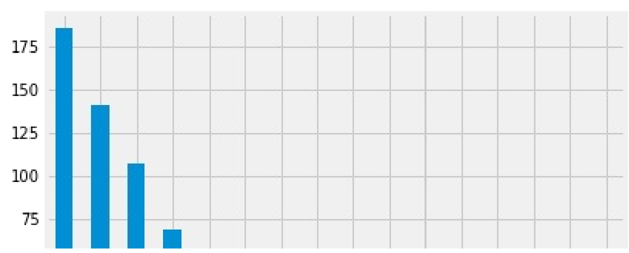
Observations:

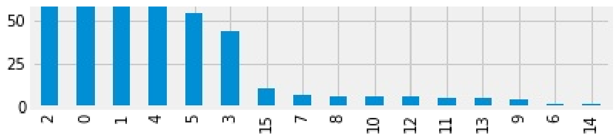
- approx. 24%(High) of the customer visited bank 2 times.
- approx. 15% of the customer never visited the bank.

In [295…

```
data['Total_visits_online'].value_counts().plot(kind='bar');
print(data['Total_visits_online'].value_counts(normalize=True))
```

```
2     0.285054
0     0.217257
1     0.164869
4     0.104777
5     0.083205
3     0.066256
15    0.015408
7     0.010786
8     0.009245
10    0.009245
12    0.009245
11    0.007704
13    0.007704
9     0.006163
6     0.001541
14    0.001541
Name: Total_visits_online, dtype: float64
```



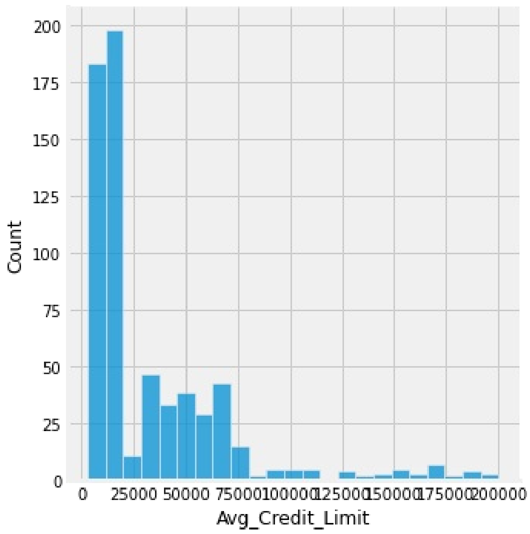


In [219…
```
data.loc[data['Total_visits_online']<=5].shape[0] / data.shape[0]
```
Out[219… 0.9214175654853621

In [220…
```
data.loc[data['Total_visits_online']==0].shape[0] / data.shape[0]
```
Out[220… 0.2172573189522342

Observations:

- Approx. 22% of the customer never visit online
- Approx. 92% of the customer visits online 5 times or less.

In [221…
```
sns.displot(data['Avg_Credit_Limit']);
```


In [222…
```
data.loc[data['Avg_Credit_Limit']<25000].shape[0] / data.shape[0]
```
Out[222… 0.5870570107858244

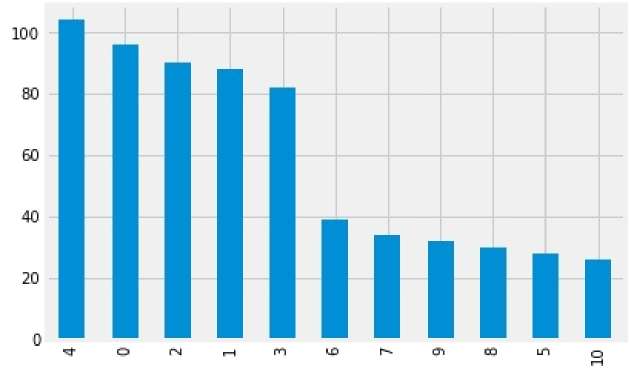
In [223…
```
data.loc[data['Avg_Credit_Limit']>=75000].shape[0] / data.shape[0]
```
Out[223… 0.08166409861325115

Observations:

- average credit limit is right skewed.
- 59.3% customer with low average credit limit (<25,000)
- 7.6% customer with high average credit limit (>=75,000)

In [224…
```
data['Total_calls_made'].value_counts().plot(kind='bar');
print(data['Total_calls_made'].value_counts(normalize=True))
```
```
4    0.160247
0    0.147920
2    0.138675
1    0.135593
```

```
3     0.126348
6     0.060092
7     0.052388
9     0.049307
8     0.046225
5     0.043143
10    0.040062
Name: Total_calls_made, dtype: float64
```



In [225…
```
data.loc[data['Total_calls_made']==0].shape[0] / data.shape[0]
```

Out[225… 0.14791987673343607

In [226…
```
data.loc[data['Total_calls_made']<=4].shape[0] / data.shape[0]
```

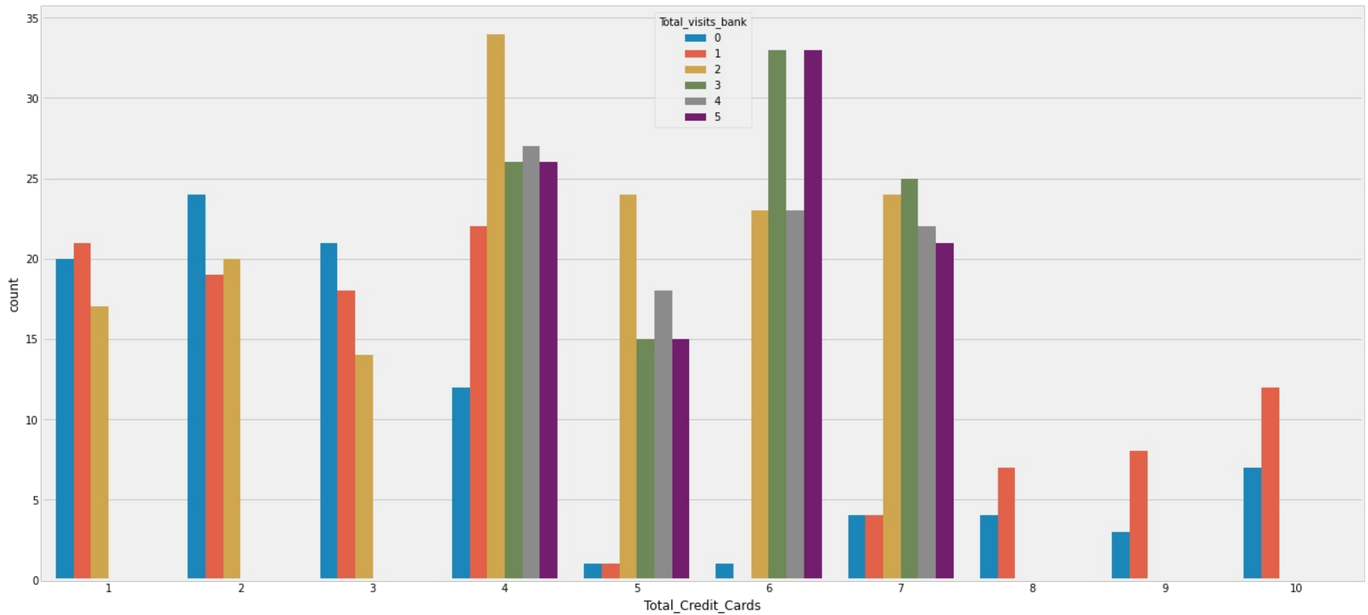
Out[226… 0.7087827426810478

Observations:

- Approx. 15% of the customers nver made calls to the bank.
- Approx. 70% of the customers made calls 4 times or less.

Bivariate analysis

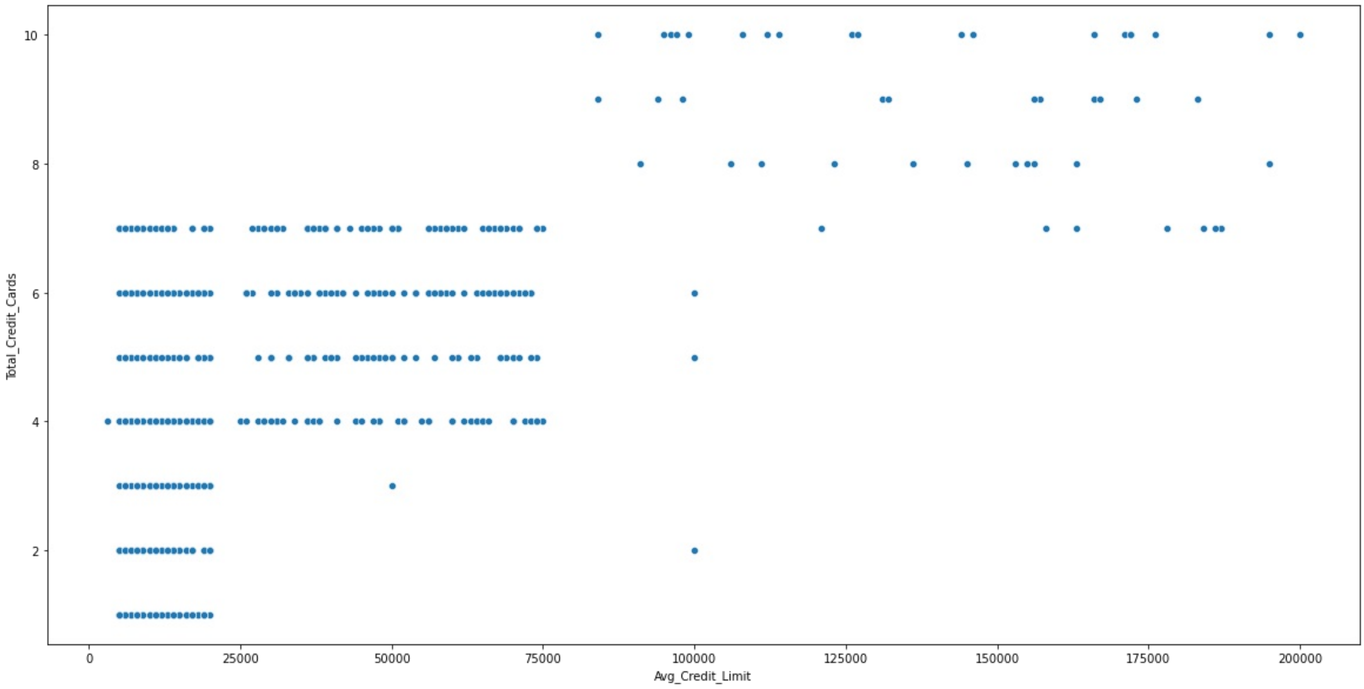
In [227…
```
plt.figure(figsize=(20,10))
sns.countplot(x='Total_Credit_Cards',hue='Total_visits_bank', data=data);
```



Observations:

- Customers who have less than 4 credit cards visited bank less - 2 times or less.
- Customers who have more than 7 credit cards visited bank less - 1 time or less.
- Customers who have 4 to 7 credit cards visits banks more than others - up to 5 times.

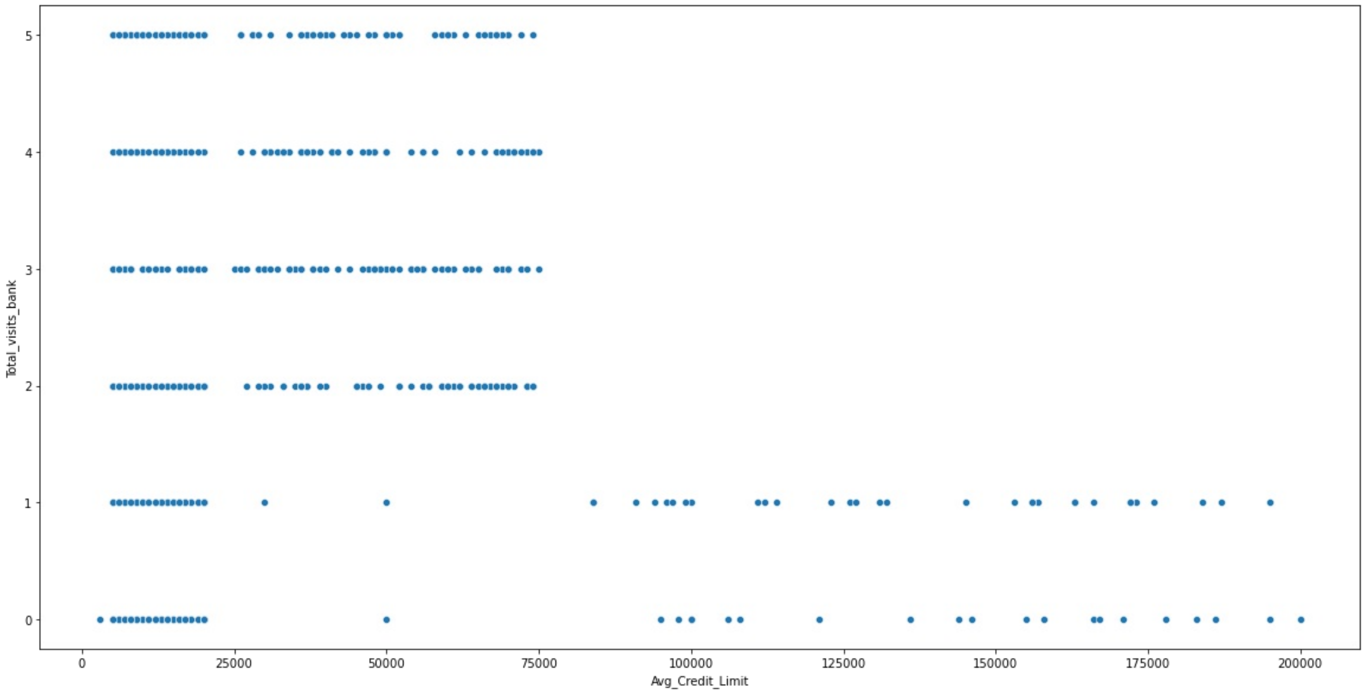
In [66]:
```
plt.figure(figsize=(20,10))
sns.scatterplot(x='Avg_Credit_Limit',y='Total_Credit_Cards', data=data);
```



Observations:

- Customer has average credit limit more than 75,000 has 7-10 credit cards.
- Customer has average credit limit between 25,000 and 75,000 has 4-7 credit cards. There are some outliers in this limits as well which needs to be handled.
- Customer has average credit limit less than 25,000 has at least 1 credit card to maximum 7 credit cards.

In [40]:
```
plt.figure(figsize=(20,10))
sns.scatterplot(x='Avg_Credit_Limit',y='Total_visits_bank', data=data);
```



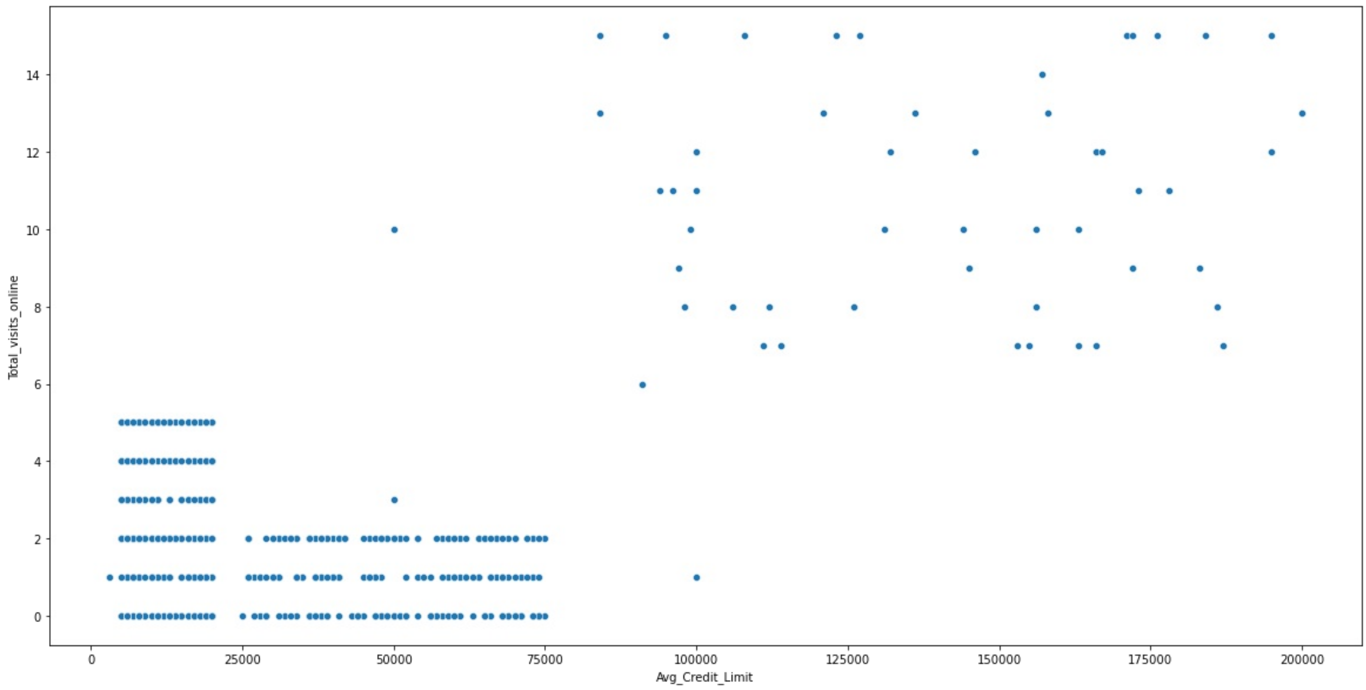
Observations:

- Customer has average credit limit more than 75,000 has 0-1 visits

- Customer has average credit limit between 25,000 and 75,000 has 2-5visits.
- Customer has average credit limit less than 25,000 visited banks more than others.

In [42]:

```
plt.figure(figsize=(20,10))
sns.scatterplot(x='Avg_Credit_Limit',y='Total_visits_online', data=data);
```

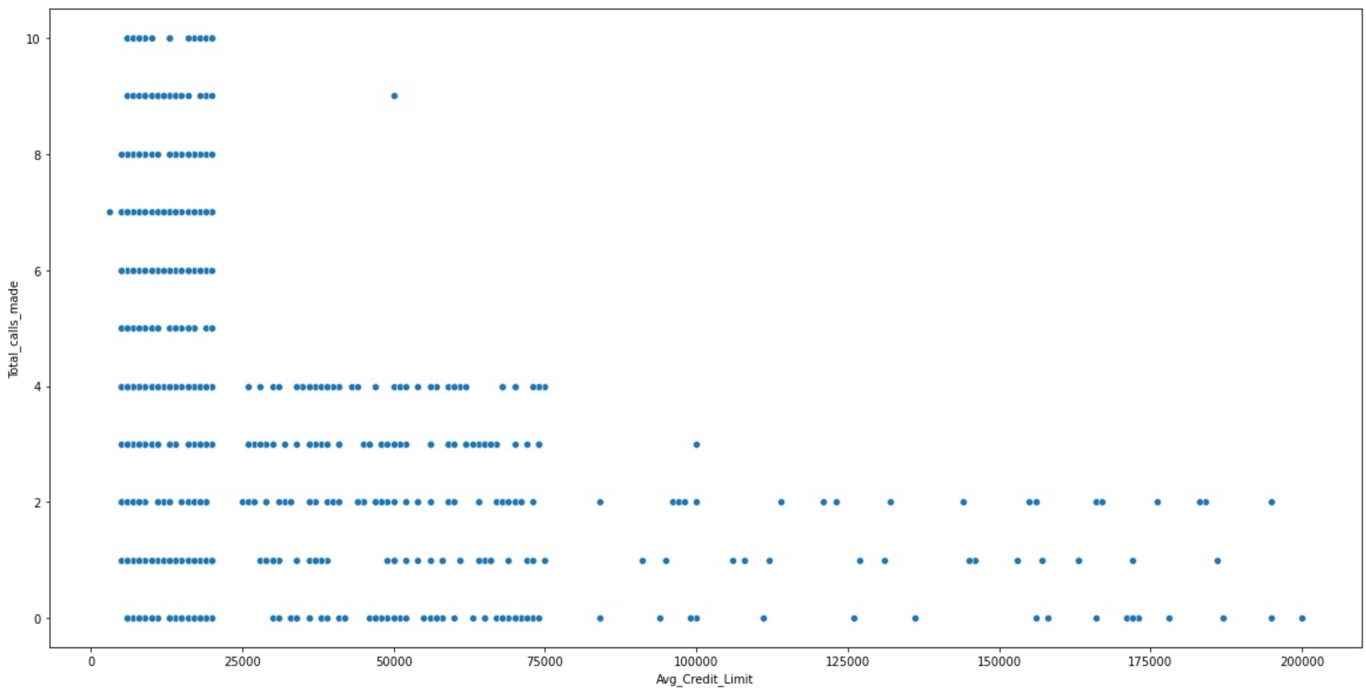


Observations:

- Customers have average credit limit more than 75,000 has 6-14(High) visits.
- Customers have average credit limit between 25,000 and 75,000 has 2 or less online visits.
- Customers have average credit limit less than 25,000 has 0 to 5(Less) online visits.

In [45]:

```
plt.figure(figsize=(20,10))
sns.scatterplot(x='Avg_Credit_Limit',y='Total_calls_made', data=data);
```



Obseravtions

- Customers have average credit limit more than 75,000 has made 0-2(Less) calls.
- Customers have average credit limit between 25,000 and 75,000 has 0-4 total calls made.
- Customers have average credit limit less than 25,000 has made 0 to 10(High) total calls.

Outlier treatment

In [296…

```
# Avg_Credit_Limit vs Total_Credit_Cards
filt = (data['Avg_Credit_Limit']>25000) & (data['Total_Credit_Cards'] < 4)
data.loc[(filt)]
```

Out[296…

	Avg_Credit_Limit	Total_Credit_Cards	Total_visits_bank	Total_visits_online	Total_calls_made
0	100000	2	1	1	0
1	50000	3	0	10	9

In [297…

```
filt = (data['Avg_Credit_Limit']>75000) & (data['Total_Credit_Cards'] < 7)
data.loc[(filt)]
```

Out[297…

	Avg_Credit_Limit	Total_Credit_Cards	Total_visits_bank	Total_visits_online	Total_calls_made
0	100000	2	1	1	0
4	100000	6	0	12	3
6	100000	5	0	11	2

Observations:

- rows 0,1,4 and 6 are the outliers in this data so we need to handle this

In [298…

```
# Avg_Credit_Limit vs Avg_Credit_Limit
filt = (data['Avg_Credit_Limit']>25000) & (data['Avg_Credit_Limit']<75000) & (data['Total_visits_bank'] < 2)
data.loc[(filt)]
```

Out[298…

	Avg_Credit_Limit	Total_Credit_Cards	Total_visits_bank	Total_visits_online	Total_calls_made
1	50000	3	0	10	9
2	50000	7	1	3	4
3	30000	5	1	1	4

Observations:

- rows 1,2 and 3 are the outliers in this data so we need to handle this

In [299…

```
filt = (data['Avg_Credit_Limit']>75000) & (data['Total_visits_online'] < 7)
data.loc[(filt)]
```

Out[299…

	Avg_Credit_Limit	Total_Credit_Cards	Total_visits_bank	Total_visits_online	Total_calls_made
0	100000	2	1	1	0
614	91000	8	1	6	1

In [300…

```
filt = (data['Avg_Credit_Limit']>25000) & (data['Avg_Credit_Limit']<75000) & (data['Total_visits_online'] >2)
data.loc[(filt)]
```

Out[300…

	Avg_Credit_Limit	Total_Credit_Cards	Total_visits_bank	Total_visits_online	Total_calls_made
1	50000	3	0	10	9
2	50000	7	1	3	4

Observations:

- rows 0,1,2 and 614 are the outliers in this data so we need to handle this

In [301…

```
filt = (data['Avg_Credit_Limit']>75000) & (data['Total_calls_made'] > 2)
data.loc[(filt)]
```

Out[301…

	Avg_Credit_Limit	Total_Credit_Cards	Total_visits_bank	Total_visits_online	Total_calls_made

4	100000	6	0	12	3

In [302…

```python
filt = (data['Avg_Credit_Limit']>25000) & (data['Total_calls_made'] > 4)
data.loc[(filt)]
```

Out[302…

	Avg_Credit_Limit	Total_Credit_Cards	Total_visits_bank	Total_visits_online	Total_calls_made
1	50000	3	0	10	9

Observations:

- rows 1 and 4 are the outliers in this data so we need to handle this

In [303…

```python
# based on the above observations, drop rows 0,1,2,3,4,6,614
data.drop(data.index[[0,1,2,3,4,6,614]],inplace=True)
```

In [304…

```python
data
```

Out[304…

	Avg_Credit_Limit	Total_Credit_Cards	Total_visits_bank	Total_visits_online	Total_calls_made
5	20000	3	0	1	8
7	15000	3	0	1	1
8	5000	2	0	2	2
9	3000	4	0	1	7
10	10000	4	0	5	5
...	...	...	...	...	...
644	99000	10	1	10	0
645	84000	10	1	13	2
646	145000	8	1	9	1
647	172000	10	1	15	0
648	167000	9	0	12	2

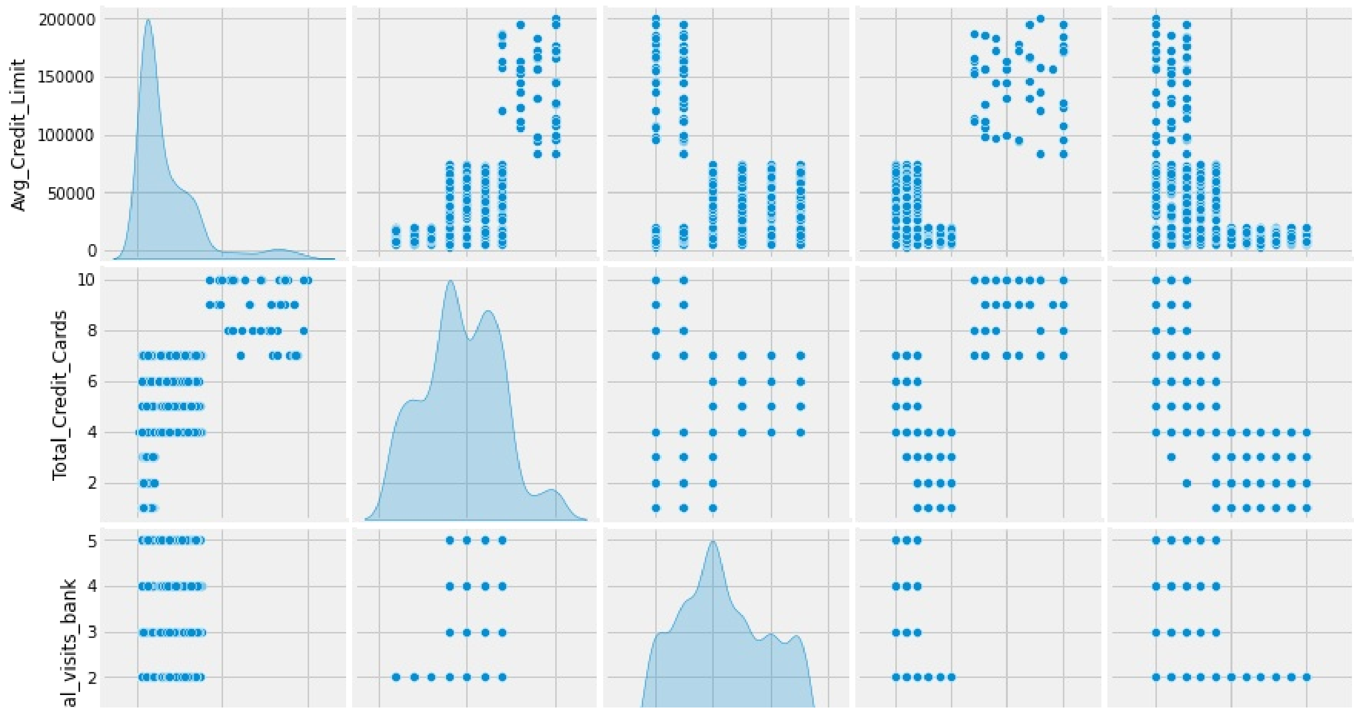
642 rows × 5 columns

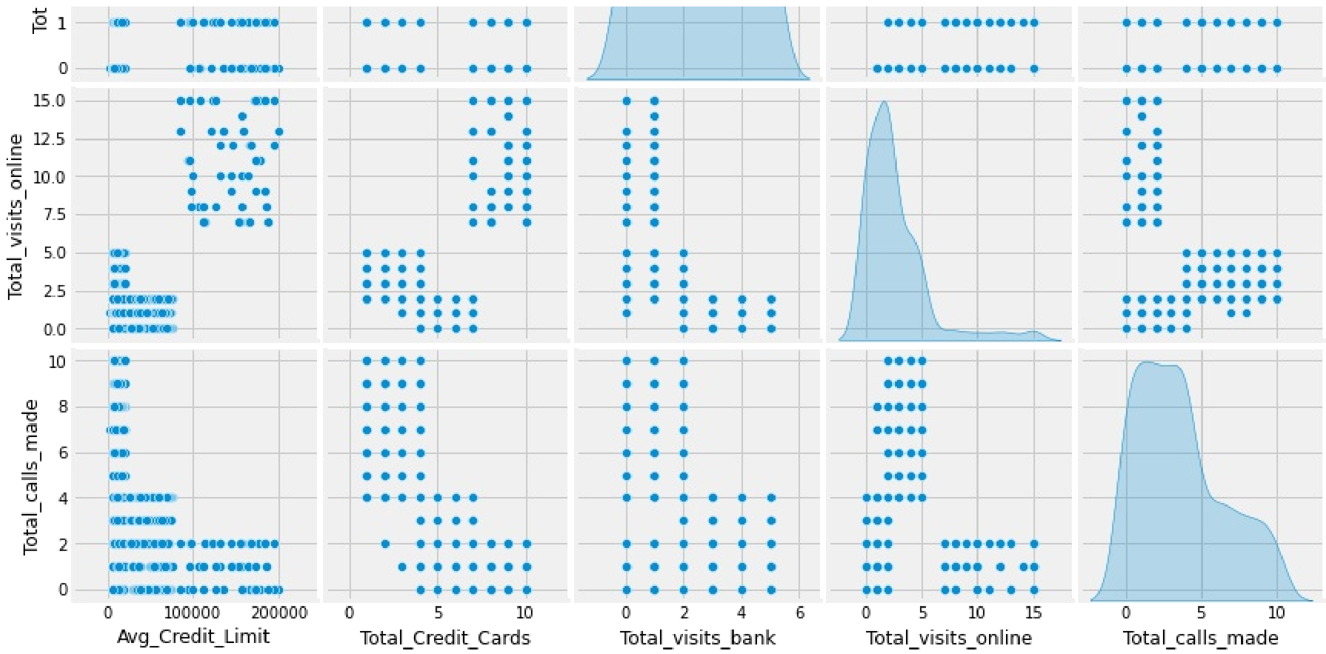
In [305…

```python
# Reset Index Id to compensate the deletion of the rows
data = data.reset_index(drop=True)
```

In [306…

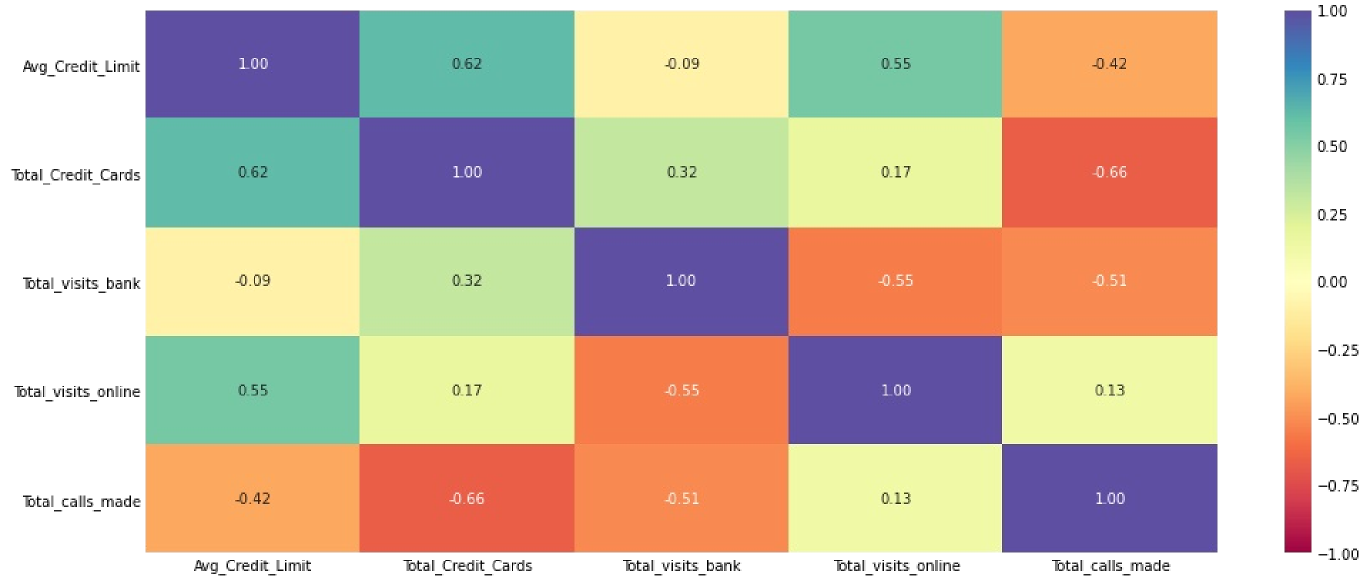
```python
sns.pairplot(data,diag_kind='kde');
```





In [307…

```
# Heatmap

plt.figure(figsize=(15, 7))
sns.heatmap(
    data.corr(), annot=True, vmin=-1, vmax=1, fmt=".2f", cmap="Spectral"
)
plt.show()
```



Observations:

- Total_Credit_Cards has comparatively high positive correlation with Avg_Credit_Limit which is 0.62.
- Total_Credit_Cards has high negative correlation with Total_calls_made which is -0.66.

Scaling the data

In [308…

```
# Scaling the features by zscore
from scipy.stats import zscore
data_z = data.apply(zscore)
data_z = pd.DataFrame(data_z,columns=data.columns)
```

In [309…

```
data_z.shape
```

Out[309… (642, 5)

In [310…
```
data_z
```

Out[310…

	Avg_Credit_Limit	Total_Credit_Cards	Total_visits_bank	Total_visits_online	Total_calls_made
0	-0.383711	-0.783896	-1.491693	-0.545200	1.531504
1	-0.516511	-0.783896	-1.491693	-0.545200	-0.901362
2	-0.782109	-1.243916	-1.491693	-0.201032	-0.553810
3	-0.835229	-0.323877	-1.491693	-0.545200	1.183952
4	-0.649310	-0.323877	-1.491693	0.831470	0.488847
...	...	...	...	...	...
637	1.714517	2.436240	-0.874639	2.552306	-1.248915
638	1.316119	2.436240	-0.874639	3.584808	-0.553810
639	2.936270	1.516201	-0.874639	2.208139	-0.901362
640	3.653386	2.436240	-0.874639	4.273142	-1.248915
641	3.520587	1.976221	-1.491693	3.240640	-0.553810

642 rows × 5 columns

K-means Clustering

`KMeans` is a clustering algorithm that groups data points together based on how similar they are to each other. When we specify the number of clusters, K, that number of data points are randomly chosen as cluster centroids, and all the other data points are assigned to the cluster of the closest centroid. The centroid is then reassigned so that it becomes the average of the cluster.

This process is repeated until the size of the clusters becomes stable.

When using KMeans, we have to specify the number of clusters the algorithm will use. One way to find the ideal number of clusters is the elbow method.

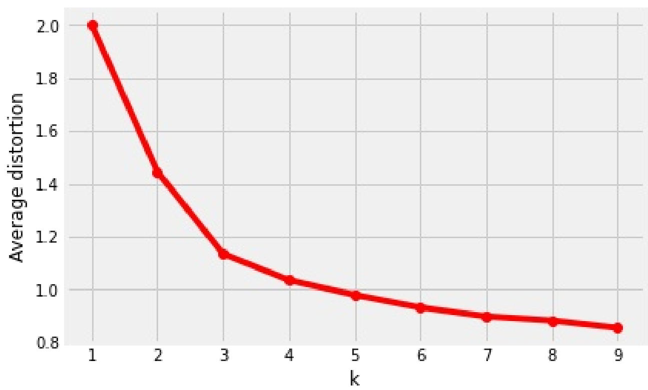
The **elbow method** allows us to identify at which K value the sum of squared distance, or the distance between data points and their respective centroids, begins to level off.

The sum of squared distance flattening indicates that increasing the amount of clusters is not leading to better-defined clusters, so it is a good method to use when trying to find an optimal value for K.

Let's use the elbow method to select our value for K.

Elbow Method

In [312…
```
clusters = range(1,10)

meanDistortions = []

for k in clusters:
    km = KMeans(n_clusters=k)
    km.fit(data_z)
    predict=km.predict(data_z)
    meanDistortions.append(sum(np.min(cdist(data_z,km.cluster_centers_, 'euclidean'), axis=1)) /
                           data_z.shape[0])

plt.plot(clusters, meanDistortions, 'ro-')
plt.xlabel('k');
plt.ylabel('Average distortion');
```



Observations:

It looks like the optimal number of clusters is 3. The Silhoutte score is a measure of how well defined clusters are, with scores near 1 indicating well-defined clusters, and scores near 0 indicating overlapping clusters.

In [313…
```
km = KMeans(n_clusters=3, n_init = 15, random_state=38)
```

It looks like we were able to create and fit our model.

Now let's add the cluster labels to our data and see how well our clusters are defined.

In [314…
```
km.fit(data_z)
```

Out[314… `KMeans(n_clusters=3, n_init=15, random_state=38)`

In [315…
```
predict = km.fit_predict(data_z)
```

In [316…
```
km_silhouette_score = silhouette_score(data_z, predict)
```

In [317…
```
km_silhouette_score
```

Out[317…
```
0.5207269512698913
```

While ideally the Silhouette score would be higher, given the somewhat non-distinct groups in the data we're using we will consider this as an acceptable score.

Let's take a look at the clusters on a scatterplot.

In [318…
```
centroids = km.cluster_centers_
```

In [319…
```
centroids
```

Out[319…
```
array([[-0.01548547,  0.37780641,  0.67046205, -0.55345955, -0.55844414],
       [ 2.90575468,  1.92728253, -1.1109577 ,  2.88915036, -0.89396776],
       [-0.59437918, -1.05572582, -0.90549208,  0.32616939,  1.14287744]])
```

In [320…
```
centroid_df = pd.DataFrame(centroids, columns = list(data_z) )
```

In [321…
```
centroid_df
```

Out[321…

	Avg_Credit_Limit	Total_Credit_Cards	Total_visits_bank	Total_visits_online	Total_calls_made
0	-0.015485	0.377806	0.670462	-0.553460	-0.558444
1	2.905755	1.927283	-1.110958	2.889150	-0.893968
2	-0.594379	-1.055726	-0.905492	0.326169	1.142877

Observations::

- Cluster 2 has the highest value for Avg_Credit_Limit,Total_Credit_Cards & Total_visits_online.
- Cluster 2 has the lowest value for Total_visits_bank.
- Cluster 1 has the highest value for Total_calls_made.
- Cluster 1 has the lowest value for Avg_Credit_Limit & Total_Credit_Cards.
- Cluster 0 has the highest value for Total_visits_bank.
- Cluster 0 has the lowest value for Total_visits_online.
- K-means silhouette_score is 0.52072.

In [322…
```
km.labels_
```

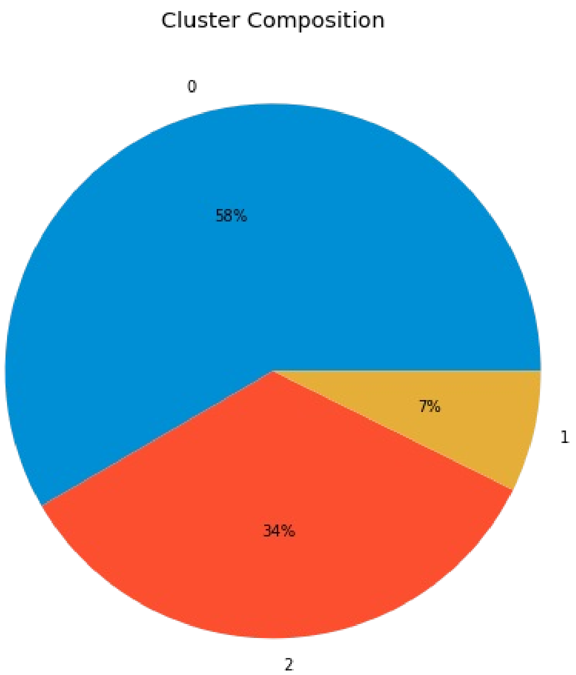
```
Out[322… array([2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 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, 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, 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, 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, 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, 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, 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, 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, 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, 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, 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, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1], dtype=int32)
```

```
In [323… data_group = data.copy()
```

```
In [324… data_group['Group'] = predict
         data_z['Group'] = predict
```

```
In [325… data_group['Group'] = data_group['Group'].astype('category')
         data_z['Group'] = data_z['Group'].astype('category')
```

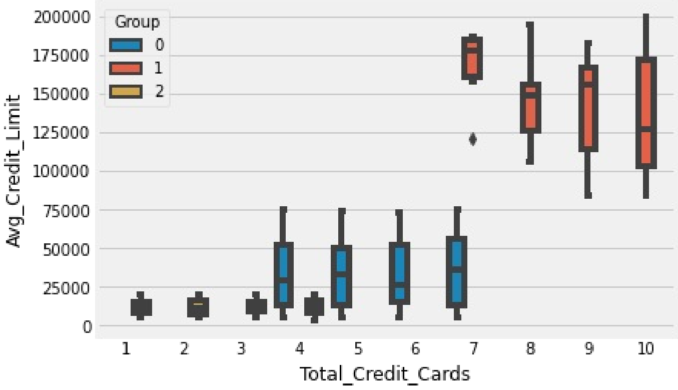
```
In [330… fig, axs = plt.subplots(figsize=(12,8))
         ax = data_group['Group'].value_counts().plot.pie(title='Cluster Composition', autopct='%1.0f%%')
         plt.title=False
         ax.set_ylabel('')
         plt.show()
```

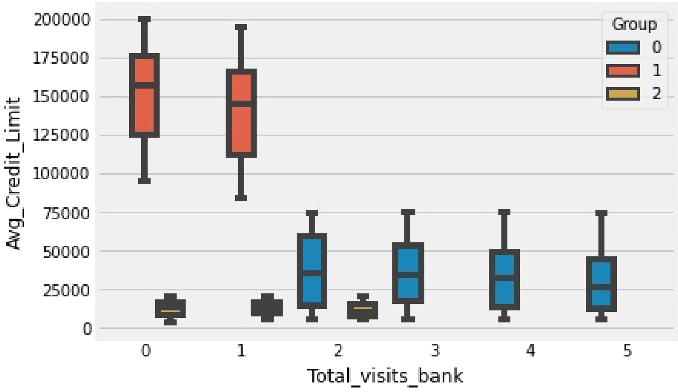


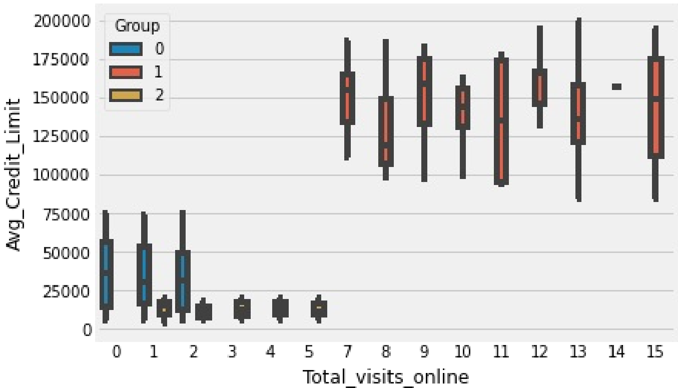
- Most users fall into Group 0.

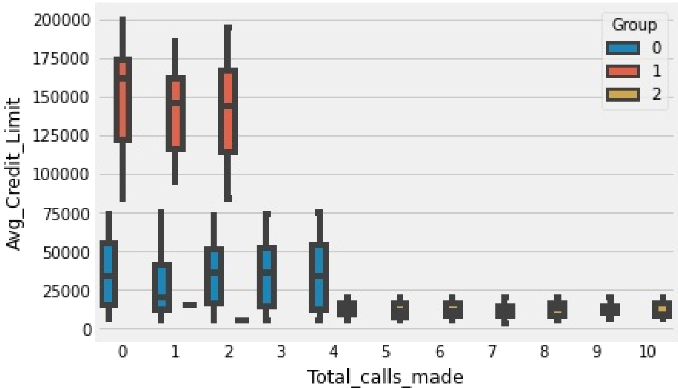
In [335…

```
for i in data_group.columns[(data_group.columns!='Group') & (data_group.columns!='Avg_Credit_Limit')]:
    sns.boxplot(x=data_group[i],y=data_group['Avg_Credit_Limit'], hue=data_group['Group'])
    plt.show()
```



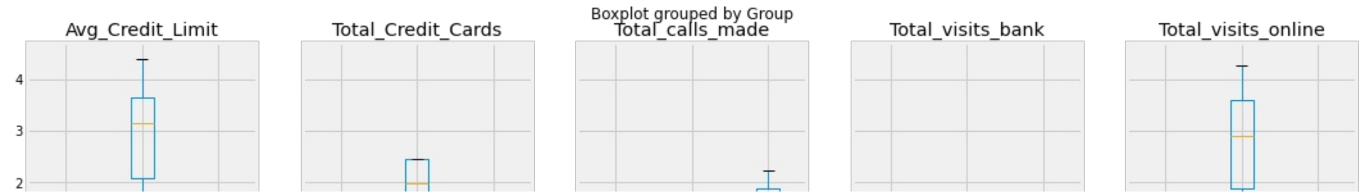


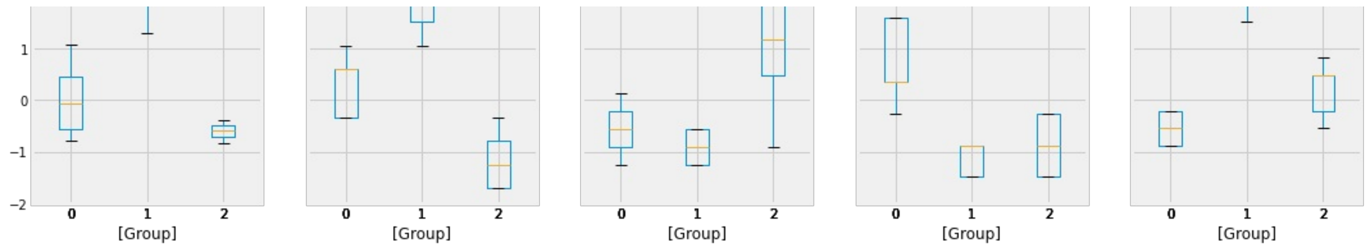




In [337…

```
data_z.boxplot(by='Group',layout=(1,5),figsize=(18,5));
```





Observations:

- Customers who have average credit limit of 75k and above are in group 1.
- Customers who have more than 7 credit cards are in group 1.
- Customers who have visited online more than 6 times are in group 1.
- Group 2 has less credit cards compare to other groups.
- Group 2 has one attribute that is distinct from other group namely, customers who make calls more than 4 times.
- Group 0 visited bank more than other groups - more than 2 times and up to 5 times.
- Group 0 also less visited bank online compare to other groups.

Hierarchical clustering

In [365…

```
scalar = StandardScaler()
X_std = pd.DataFrame(scalar.fit_transform(data),columns=data.columns)
X_std.head()
```

Out[365…

	Avg_Credit_Limit	Total_Credit_Cards	Total_visits_bank	Total_visits_online	Total_calls_made
0	-0.383711	-0.783896	-1.491693	-0.545200	1.531504
1	-0.516511	-0.783896	-1.491693	-0.545200	-0.901362
2	-0.782109	-1.243916	-1.491693	-0.201032	-0.553810
3	-0.835229	-0.323877	-1.491693	-0.545200	1.183952
4	-0.649310	-0.323877	-1.491693	0.831470	0.488847

In [366…

```
model = AgglomerativeClustering(n_clusters=3, affinity='euclidean',  linkage='ward')
model.fit(X_std)
```

Out[366… `AgglomerativeClustering(n_clusters=3)`

In [367…

```
Z = linkage(X_std, metric='euclidean', method='ward')
ward_c, coph_dists = cophenet(Z , pdist(X_std))
```

In [368…

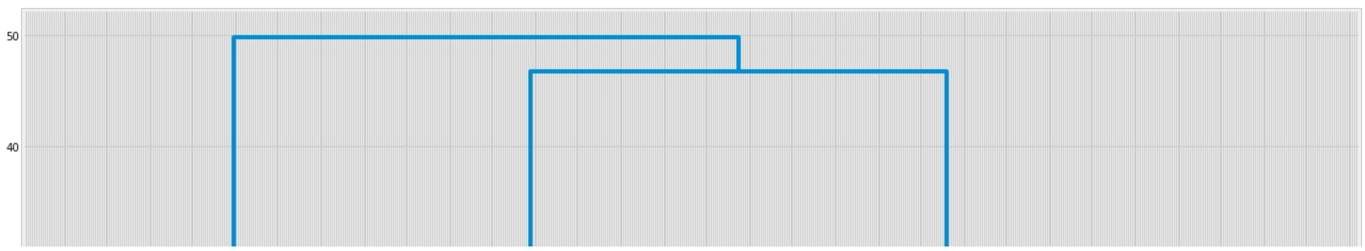
```
#Store for final comparison

results = pd.DataFrame({'linkage':['ward'], 'cophenetic coeff': ward_c},index={'0'})
results = results[['linkage', 'cophenetic coeff']]
results
```

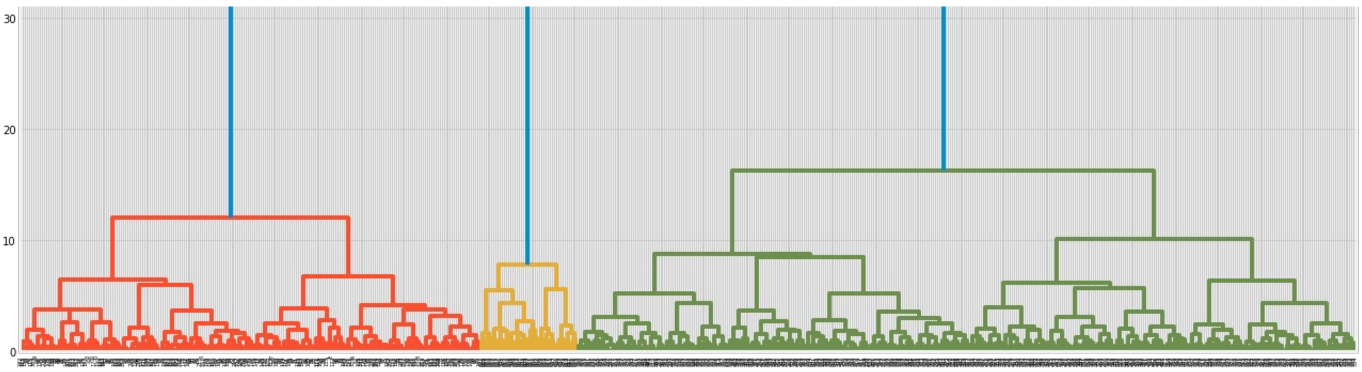
Out[368…

	linkage	cophenetic coeff
0	ward	0.742861

In [369…

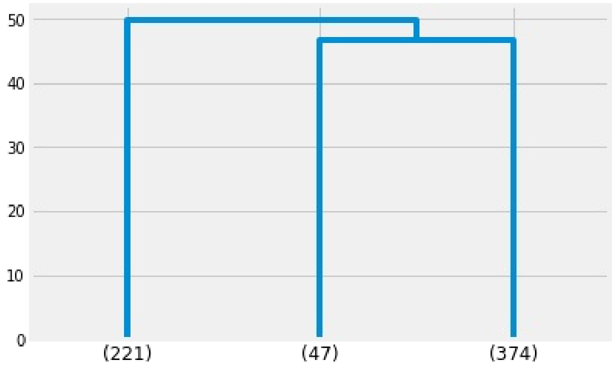
```
plt.figure(figsize=(20, 10))
dendrogram(Z)
plt.show()
```





In [370…

```
dendrogram(Z, truncate_mode='lastp',p=3)
plt.show()
```



In [371…

```
max_d = 30
clusters = fcluster(Z, max_d, criterion='distance')
ward_sc = silhouette_score(X_std,clusters)
```

In [372…

```
#Store  for final comparison

results1 = pd.DataFrame({'linkage':['ward'], 'silhouette_score': ward_sc},index={'0'})
results1 = results1[['linkage', 'silhouette_score']]
results = pd.merge(results,results1, on='linkage')
results
```

Out[372…

	linkage	cophenetic coeff	silhouette_score
0	ward	0.742861	0.519784

In [373…

```
model = AgglomerativeClustering(n_clusters=3, affinity='euclidean',  linkage='average')
model.fit(X_std)
```

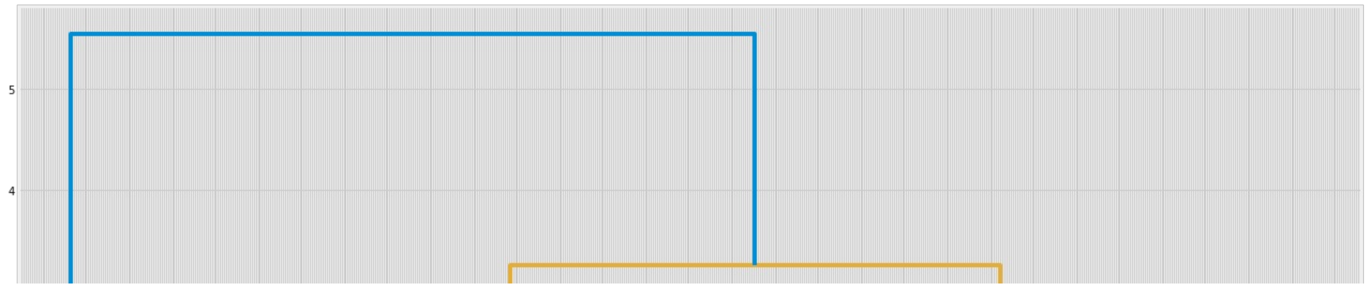
Out[373… AgglomerativeClustering(linkage='average', n_clusters=3)

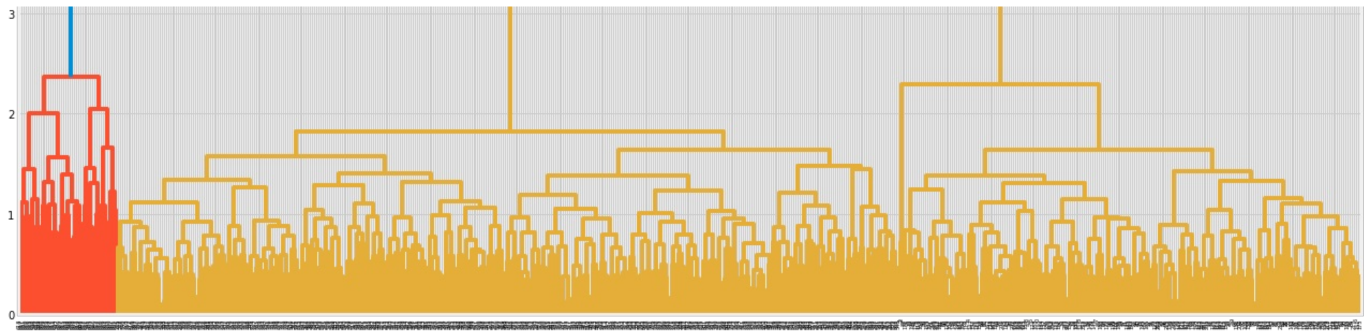
In [374…

```
Z = linkage(X_std, metric='euclidean', method='average')
average_c, coph_dists = cophenet(Z , pdist(X_std))
```

In [375…

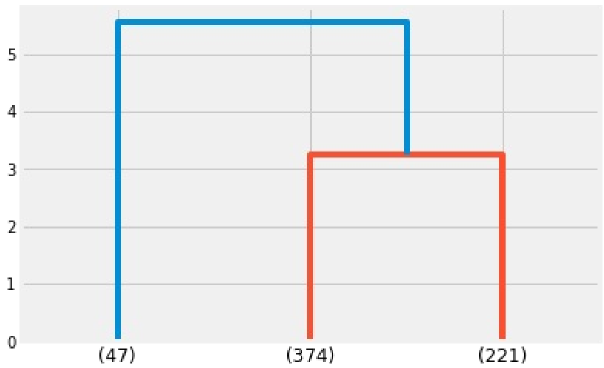
```
plt.figure(figsize=(20, 10))
dendrogram(Z)
plt.show()
```





In [376…

```
dendrogram(Z, truncate_mode='lastp',p=3)
plt.show()
```



In [377…

```
max_d = 3
clusters = fcluster(Z, max_d, criterion='distance')
average_sc = silhouette_score(X_std,clusters)
```

In [378…

```
tempResultsDf = pd.DataFrame({'linkage':['average'],
                              'cophenetic coeff': average_c,
                              'silhouette_score': average_sc},index={'1'})
results = pd.concat([results, tempResultsDf])
results = results[['linkage','cophenetic coeff', 'silhouette_score']]
results
```

Out[378…

	linkage	cophenetic coeff	silhouette_score
0	ward	0.742861	0.519784
1	average	0.902956	0.519784

In [379…

```
model = AgglomerativeClustering(n_clusters=3, affinity='euclidean',  linkage='complete')
model.fit(X_std)
```

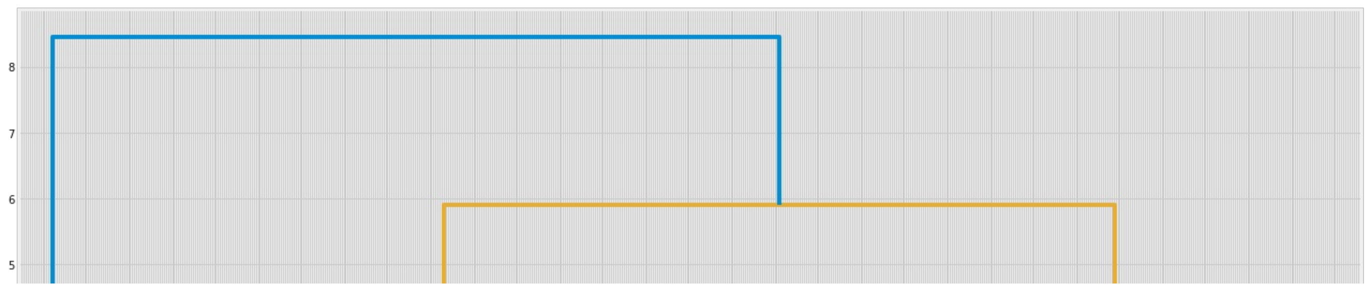
Out[379… `AgglomerativeClustering(linkage='complete', n_clusters=3)`

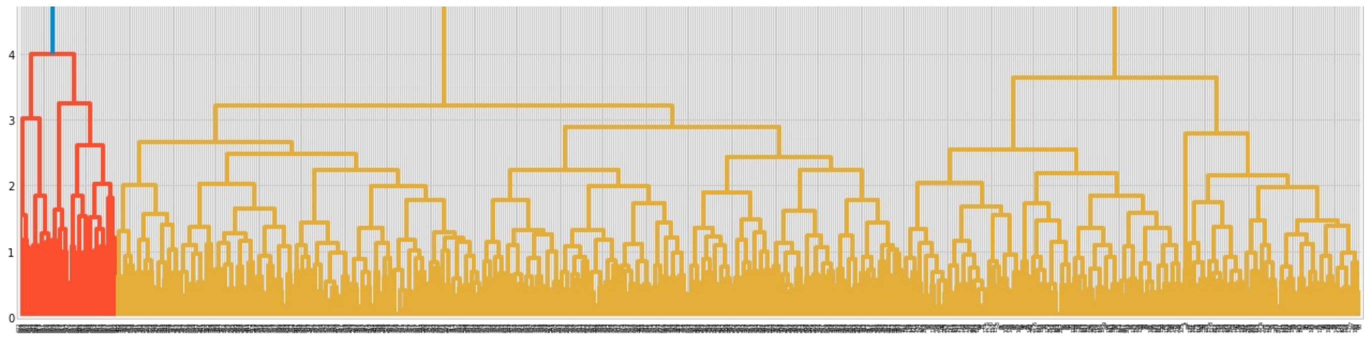
In [380…

```
Z = linkage(X_std, metric='euclidean', method='complete')
complete_c, coph_dists = cophenet(Z , pdist(X_std))
```

In [381…

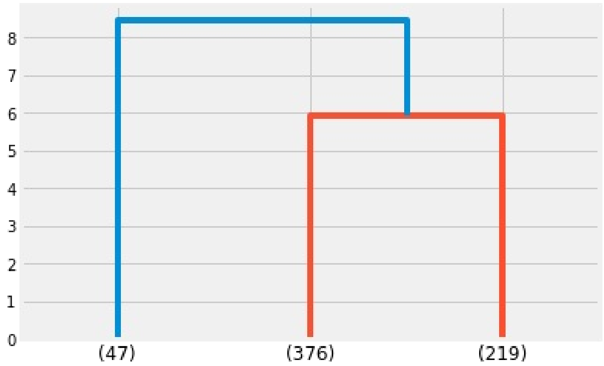
```
plt.figure(figsize=(20, 10))
dendrogram(Z)
plt.show()
```





In [382…

```
dendrogram(Z, truncate_mode='lastp',p=3)
plt.show()
```



In [383…

```
max_d = 5
clusters = fcluster(Z, max_d, criterion='distance')
complete_sc = silhouette_score(X_std,clusters)
```

In [384…

```
tempResultsDf = pd.DataFrame({'linkage':['complete'],
                              'cophenetic coeff': complete_c,
                              'silhouette_score': complete_sc},index={'2'})
results = pd.concat([results, tempResultsDf])
results = results[['linkage','cophenetic coeff', 'silhouette_score']]
results
```

Out[384…

	linkage	cophenetic coeff	silhouette_score
0	ward	0.742861	0.519784
1	average	0.902956	0.519784
2	complete	0.886412	0.520901

In [385…

```
model = AgglomerativeClustering(n_clusters=3, affinity='euclidean',  linkage='single')
model.fit(X_std)
```

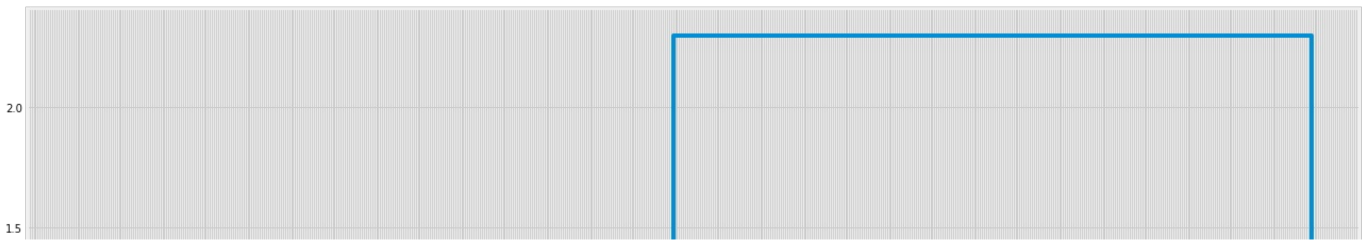
Out[385… AgglomerativeClustering(linkage='single', n_clusters=3)

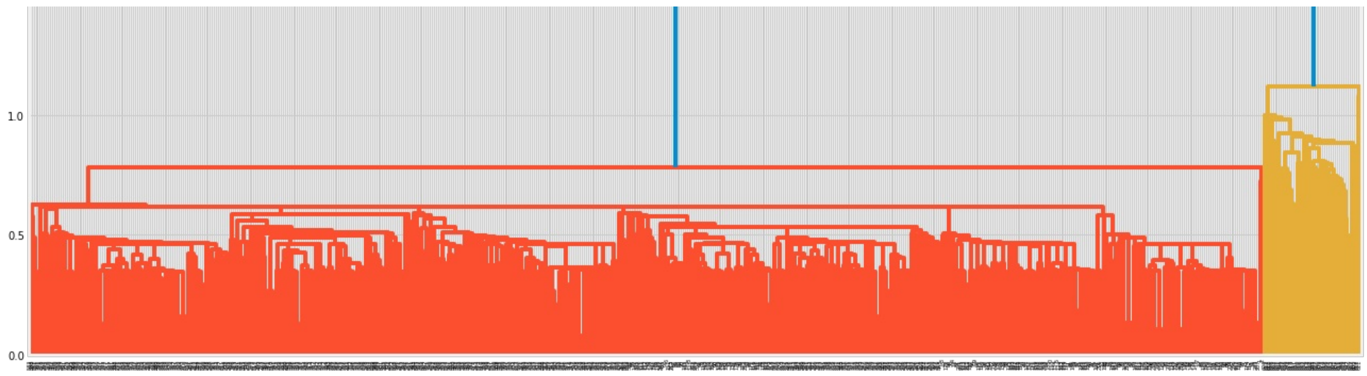
In [386…

```
Z = linkage(X_std, metric='euclidean', method='single')
single_c, coph_dists = cophenet(Z , pdist(X_std))
```

In [387…

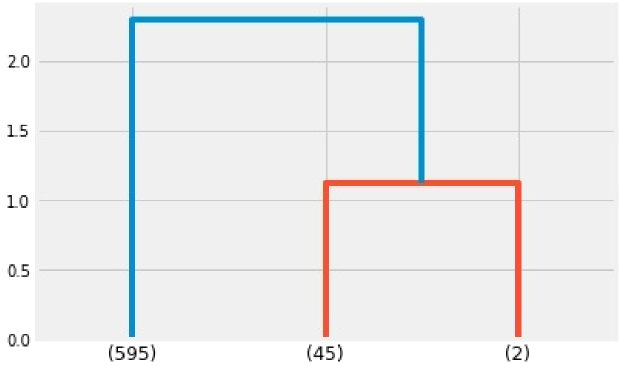
```
plt.figure(figsize=(20, 10))
dendrogram(Z)
plt.show()
```





In [388…

```
dendrogram(Z, truncate_mode='lastp',p=3)
plt.show()
```



In [389…

```
max_d = 1
clusters = fcluster(Z, max_d, criterion='distance')
single_sc = silhouette_score(X_std,clusters)
```

In [390…

```
tempResultsDf = pd.DataFrame({'linkage':['single'],
                              'cophenetic coeff': single_c,
                              'silhouette_score': single_sc},index={'3'})
results = pd.concat([results, tempResultsDf])
results = results[['linkage','cophenetic coeff', 'silhouette_score']]
results
```

Out[390…

	linkage	cophenetic coeff	silhouette_score
0	ward	0.742861	0.519784
1	average	0.902956	0.519784
2	complete	0.886412	0.520901
3	single	0.744817	0.512156

Observations:

- linkage method average give higher cophenetic coeff - 0.902956 and with silhouette score 0.519784

In [391…

```
model = AgglomerativeClustering(n_clusters=3, affinity='euclidean',  linkage='average')
model.fit(X_std)
```

Out[391… AgglomerativeClustering(linkage='average', n_clusters=3)

In [392…

```
Z = linkage(X_std, metric='euclidean', method='average')
average_c, coph_dists = cophenet(Z , pdist(X_std))
```

In [393…

```
max_d = 3
clusters = fcluster(Z, max_d, criterion='distance')
average_sc = silhouette_score(X_std,clusters)
```

In [394…

```python
# creating a new dataframe only for labels and converting it into categorical variable

data_labels = pd.DataFrame(model.labels_ , columns = list(['labels']))

data_labels['labels'] = model.labels_
```

In [395…

```python
data_labels['labels'] = data_labels['labels'].astype('category')
```

In [396…

```python
data_labels = data.join(data_labels)
```

In [397…

```python
data_labels
```

Out[397…

	Avg_Credit_Limit	Total_Credit_Cards	Total_visits_bank	Total_visits_online	Total_calls_made	labels
0	20000	3	0	1	8	1
1	15000	3	0	1	1	1
2	5000	2	0	2	2	1
3	3000	4	0	1	7	1
4	10000	4	0	5	5	1
...	...	...	...	...	...	...
637	99000	10	1	10	0	0
638	84000	10	1	13	2	0
639	145000	8	1	9	1	0
640	172000	10	1	15	0	0
641	167000	9	0	12	2	0

642 rows × 6 columns

In [398…

```python
X_std['labels'] = model.labels_
```

In [399…

```python
# Hierarchical clustering method

X_std_Clust = X_std.groupby(['labels'])
X_std_Clust.mean()
```

Out[399…

labels	Avg_Credit_Limit	Total_Credit_Cards	Total_visits_bank	Total_visits_online	Total_calls_made
0	2.905755	1.927283	-1.110958	2.889150	-0.893968
1	-0.594988	-1.050333	-0.902560	0.323784	1.138345
2	-0.013578	0.378453	0.672943	-0.554402	-0.560315

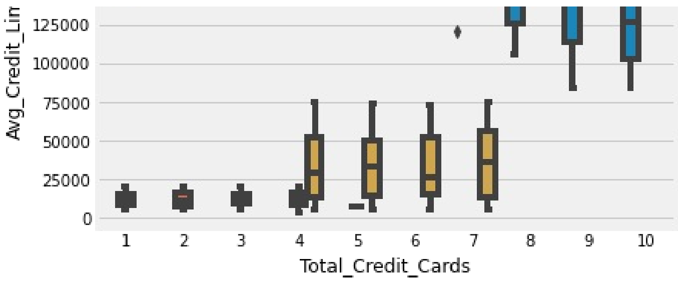
Observations:

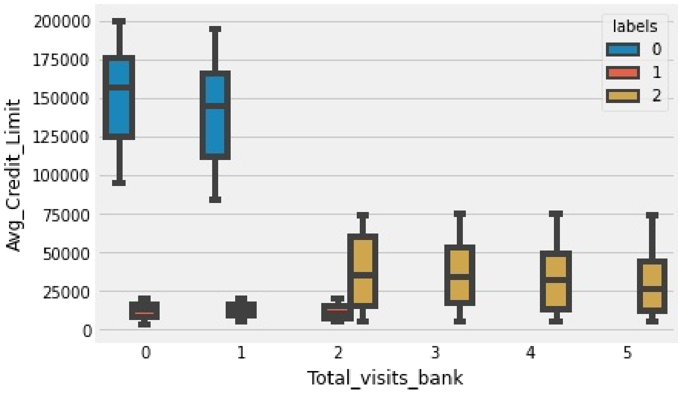
- Cluster 0 has the highest value for Avg_Credit_Limit,Total_Credit_Cards & Total_visits_online.
- Cluster 0 has the lowest value for Total_visits_bank.
- Cluster 1 has the highest value for Total_calls_made.
- Cluster 1 has the lowest value for Avg_Credit_Limit & Total_Credit_Cards.
- Cluster 2 has the highest value for Total_visits_bank.
- Cluster 2 has the lowest value for Total_visits_online.

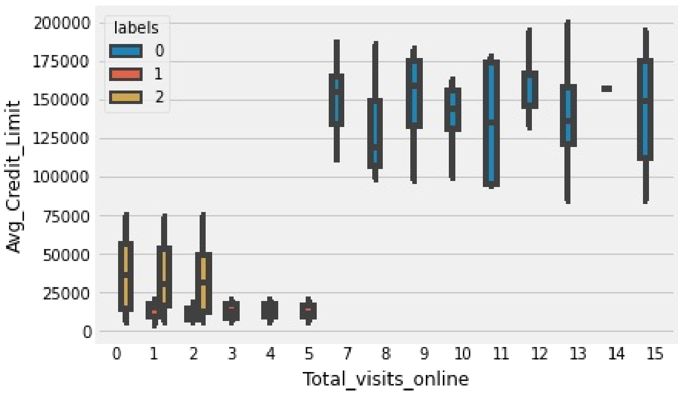
- Seems like when compare with K-means centroid, the values are almost the same but only difference is cluster lables are interchanged.

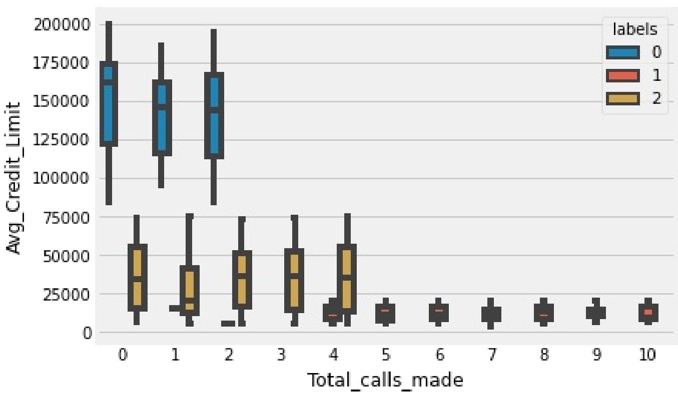
In [400…

```python
# Hierarchical clusters boxplot

for i in data_labels.columns[(data_labels.columns!='labels') & (data_labels.columns!='Avg_Credit_Limit')]:
    sns.boxplot(x=data_labels[i],y=data_labels['Avg_Credit_Limit'], hue=data_labels['labels'])
    plt.show()
```









Observations:

- Customers who have average credit limit of 75k and above are in Label 0.
- Customers who have more than 7 credit cards are in Label 0.
- Customers who have visited online more than 6 times are in Label 0.
- Label 1 has less credit cards compare to other labels.
- Label 1 has more customers who make calls more than 4 times.
- Label 2 visited bank more than other labels - more than 2 times and up to 5 times.
- Label 2 also less visited bank online compare to other labels.

In [401…
```
# Hierarchical clusters
data_labels['labels'].value_counts()
```

Out[401…
```
2    374
1    221
0     47
Name: labels, dtype: int64
```

In [402…

```
# K-means clusters
data_group['Group'].value_counts()
```

Out[402…

```
0    375
2    220
1     47
Name: Group, dtype: int64
```

In [403…

```
# K-means clusters
data_group['Group'].value_counts(normalize=True)
```

Out[403…

```
0    0.584112
2    0.342679
1    0.073209
Name: Group, dtype: float64
```

In [404…

```
# Hierarchical clusters silhouette score (linkage method:'average')
average_sc
```

Out[404…

```
0.5197840914842371
```

In [405…

```
# K-means clusters silhouette score
km_silhouette_score
```

Out[405…

```
0.5207269512698913
```

Obseravtions:

- Hierarchical cluster has almost the same cluster as K-means clusters.
- The labeling of cluster is different between K-means clusters & Hierarchical cluster.
- Hierarchical clusters silhouette score is almost same as K-means clusters.
- Silhouette score closer to 1 indicate the clustering is better. In this case, we can say that K-means clusters is slightly better than k-means cluster.

Observations:

- Based on KMeans cluster there are 3 different segements of customers in AllLife Bank credit card customer base.

`Group 0:`

- Customers who have average credit limit between 25k-75k.
- Customers who own 4-7 credit cards.
- Customers who visited bank 2-5 times.
- Customers who least visit bank online 0-2 times.
- Customers who make phone calls 0-4 times.
- 58.4% of the customers are in this group.

`Group 1:`

- Customers who have average credit limit above 75k
- Customers who own 7-10 credit cards.
- Customers who visit bank 0-1 times.
- Customers who visit bank online 7-15 times.
- Customers who make least phone calls 0-2 times.
- Only 7.3% of customers are in this group.

`Group 2:`

- Customers who have average credit limit below 25k.
- Customers who own 1-4 credit cards
- Customers who visit bank 0-2 times.
- Customers who visit bank online 1-5 times.

- Customers who make phone calls 4-10 times.
- 34% of customers are in this group.

Recommendations:

- Group 2 own less credit card than others, bank should target group 2 more.
- Bank should increase the credit limit to group 0 where most of the customers are. So they can spend more.
- Group 0 using online portal very less so need to promote the online service more to this group.
- Group 2 prefers phone instead of online portal so we can promote online customer service among this group.