Unsupervised Learning Project: AllLife Bank Customer Segmentation

Objective

Identify different segments in the existing customer base, taking into account their spending patterns as well as past interactions with the bank.

About the data

Data is available on customers of the bank with their credit limit, the total number of credit cards the customer has, and different channels through which the customer has contacted the bank for any queries. These different channels include visiting the bank, online, and through a call center.

- Sl_no - Customer Serial Number
- Customer Key - Customer identification
- Avg_Credit_Limit - Average credit limit (currency is not specified, you can make an assumption around this)
- Total_Credit_Cards - Total number of credit cards
- Total_visits_bank - Total bank visits
- Total_visits_online - Total online visits
- Total_calls_made - Total calls made

Importing libraries and overview of the dataset

In [2]:

```
# Importing all the necessary packages

import pandas as pd

import numpy as np

import matplotlib.pylab as plt

import seaborn as sns

# To scale the data using z-score
from sklearn.preprocessing import StandardScaler

# Importing clustering algorithms
from sklearn.cluster import KMeans

from sklearn.mixture import GaussianMixture

from sklearn_extra.cluster import KMedoids

import warnings
warnings.filterwarnings("ignore")
```

Loading the data

In [5]:

```
data = pd.read_excel('Credit+Card+Customer+Data (1).xlsx')

data.head()
```

Out[5]:

	Sl_No	Customer Key	Avg_Credit_Limit	Total_Credit_Cards	Total_visits_bank	Total_visits_online	Total_calls_made
0	1	87073	100000	2	1	1	0
1	2	38414	50000	3	0	10	9
2	3	17341	50000	7	1	3	4
3	4	40496	30000	5	1	1	4
4	5	47437	100000	6	0	12	3

Check the info of the data

In [6]:

```
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
```

Observations:

- There are **660 observations and 7 columns** in the dataset.
- All the columns have 660 non-null values, i.e., there are **no missing values**.
- All the columns are of integer data type.
- There are no missing values. Let us now figure out the number of unique values in each column

In [7]:

```
data.nunique()
```

Out[7]:

```
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
```

- Customer key, which is an identifier, has duplicate values. We will treat the duplicate customer keys before applying any algorithm.

Data Preprocessing and Exploratory Data Analysis

Checking duplicate customer keys

Drop the rows with duplicate customer keys

In [8]:

```
# There are some duplicates in the column 'Customer Key'. Let us explore

duplicate_keys = data['Customer Key'].duplicated()

data[duplicate_keys]
```

Out[8]:

	Sl_No	Customer Key	Avg_Credit_Limit	Total_Credit_Cards	Total_visits_bank	Total_visits_online	Total_calls_made
332	333	47437	17000	7	3	1	0
398	399	96929	67000	6	2	2	2
432	433	37252	59000	6	2	1	2
541	542	50706	60000	7	5	2	2
632	633	97935	187000	7	1	7	0

- There are **5 duplicate customer keys**. We can **drop these observations**.

In [9]:

```
#drop duplicate keys

data = data.drop_duplicates(subset= 'Customer Key', keep='first', inplace=False, ignore_index=False)
```

drop the variables that are not required for our analysis .

In [10]:

```
data.drop(columns = ['Sl_No', 'Customer Key'], inplace = True)
```

check for duplicates. **Duplicates would mean customers with identical features.**

In [11]:

```
data[data.duplicated()]
```

Out[11]:

	Avg_Credit_Limit	Total_Credit_Cards	Total_visits_bank	Total_visits_online	Total_calls_made
162	8000	2	0	3	4
175	6000	1	0	2	5
215	8000	4	0	4	7
295	10000	6	4	2	3
324	9000	4	5	0	4
361	18000	6	3	1	4
378	12000	6	5	2	1
385	8000	7	4	2	0
395	5000	4	5	0	1
455	47000	6	2	0	4
497	52000	4	2	1	2

- There are 11 duplicate rows.Drop these duplicate rows from the data.

In [12]:

```
data = data[~data.duplicated()]
```

In [13]:

```
data.shape
```

Out[13]:

(644, 5)

- After removing the duplicate keys, the duplicate rows, and dropping unnecessary columns, there are 644 unique observations and 5 columns in our data.

Summary Statistics

In [14]:

```
data.describe().T
```

Out[14]:

	count	mean	std	min	25%	50%	75%	max
Avg_Credit_Limit	644.0	34543.478261	37428.704286	3000.0	11000.0	18000.0	48000.00	200000.0
Total_Credit_Cards	644.0	4.694099	2.175338	1.0	3.0	5.0	6.00	10.0
Total_visits_bank	644.0	2.395963	1.626964	0.0	1.0	2.0	4.00	5.0
Total_visits_online	644.0	2.624224	2.957728	0.0	1.0	2.0	4.00	15.0
Total_calls_made	644.0	3.608696	2.880025	0.0	1.0	3.0	5.25	10.0

**Observations:

--There is a high standard deviation away from mean for average credit limit.

--Each customer had at least one credit card and 3000 pounds minimum limit.

--Visits online had the maximum number of use per customer

--while some customers didn't visit banks, some didn't use online service, some didn't use the call center but at least they all made use of one query channel or the other

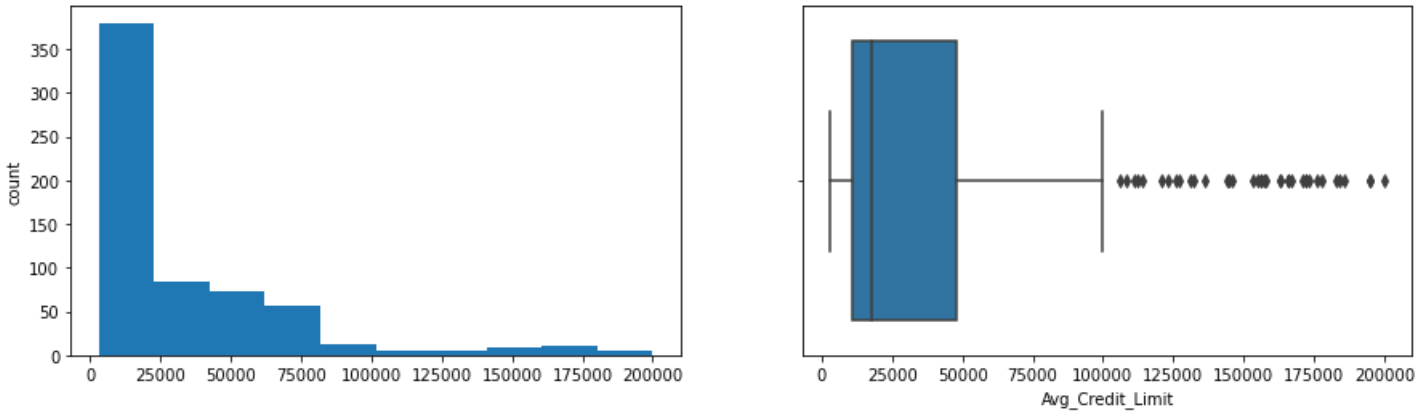
Check the distribution and outliers for each variable in the data.

In [15]:

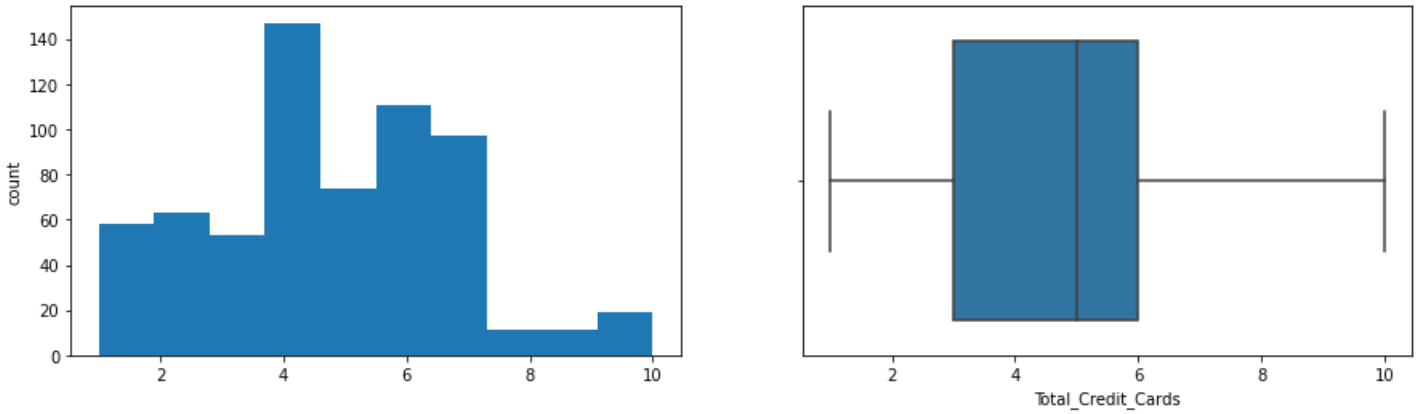
```
# complete the BELOW code by filling the blanks, before running the cell to avoid any errors

for col in data.columns:
    print(col)

    print('Skew :', round(data[col].skew(), 2))

    plt.figure(figsize = (15, 4))

    plt.subplot(1, 2, 1)

    data[col].hist(bins = 10, grid = False)

    plt.ylabel('count')
```

```
    plt.subplot(1, 2, 2)

    sns.boxplot(x = data[col])

    plt.show()
```

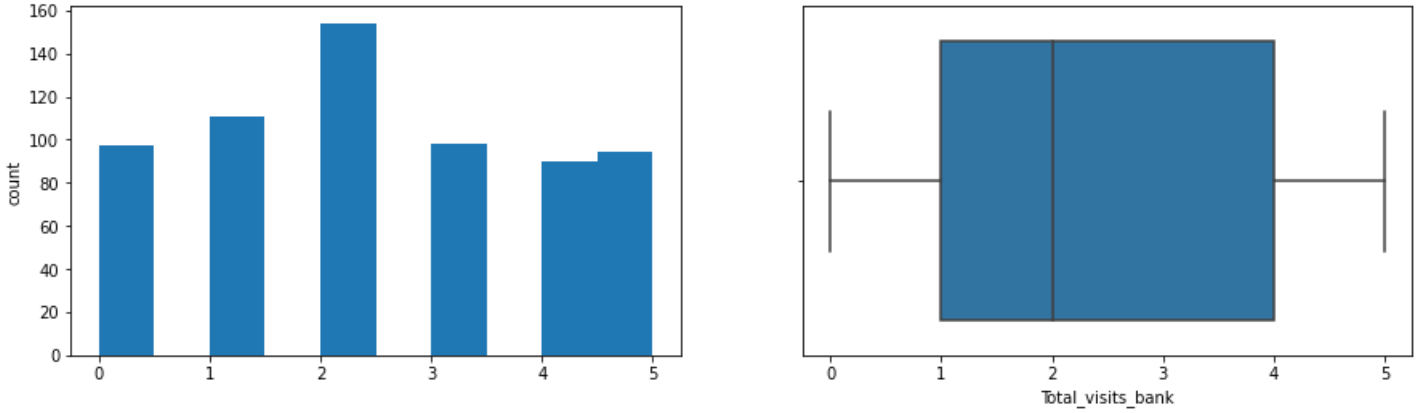
Avg_Credit_Limit
Skew : 2.19



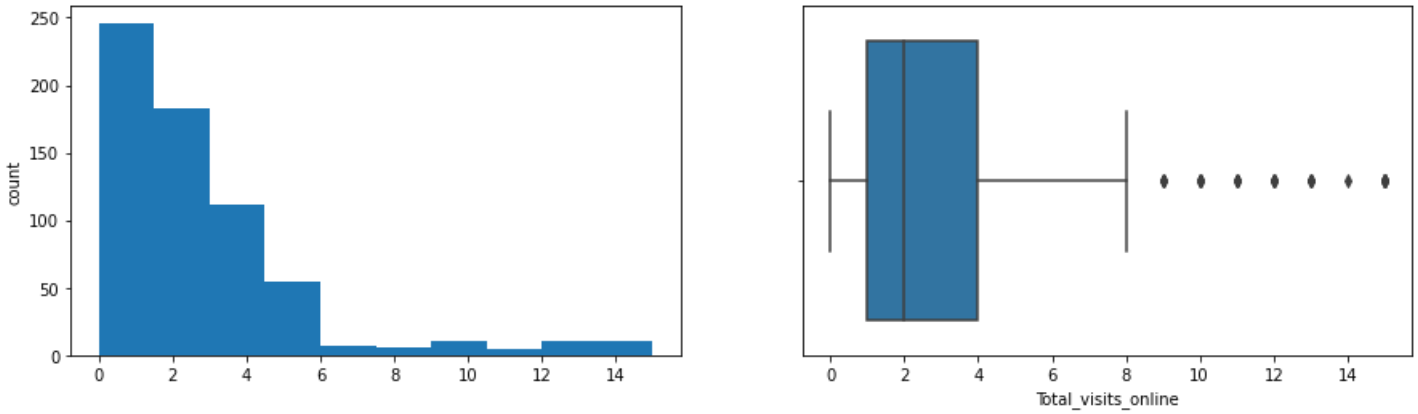
Total_Credit_Cards
Skew : 0.17



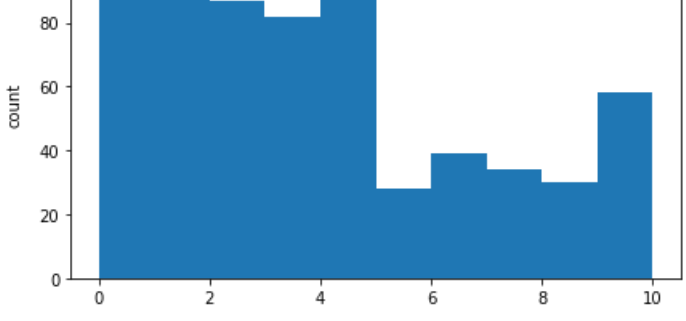
Total_visits_bank
Skew : 0.15

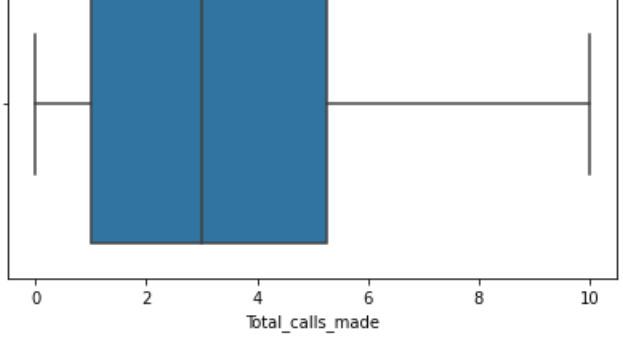


Total_visits_online
Skew : 2.21



Total_calls_made
Skew : 0.65





**Observation:

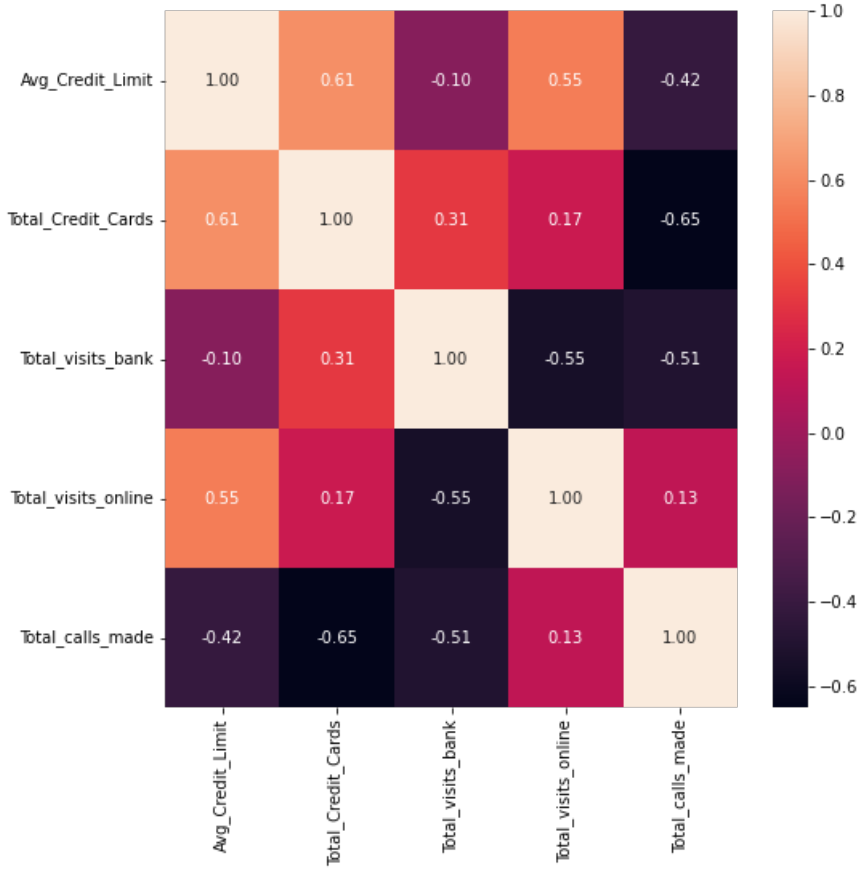
--most of the variables have right skewed distributions except for total bank visits.

--Average Credit limit has a lot of outliers and customers with average credit limit between 3000 and 20000 were the highest

Checking correlation

In [16]:

```
plt.figure(figsize = (8, 8))

sns.heatmap(data.corr(), annot = True, fmt = '0.2f')

plt.show()
```



Observations:

- Avg_Credit_Limit is positively correlated with Total_Credit_Cards and Total_visits_online which makes sense.
- Avg_Credit_Limit is negatively correlated with Total_calls_made and Total_visits_bank.
- Total_visits_bank, Total_visits_online, Total_calls_made are negatively correlated which implies that majority of customers use only one of these channels to contact the bank.

Scaling the data

In [17]:

```
scaler = StandardScaler()

data_scaled = pd.DataFrame(scaler.fit_transform(data), columns = data.columns)
```

In [18]:

```
data_scaled.head()
```

Out[18]:

	Avg_Credit_Limit	Total_Credit_Cards	Total_visits_bank	Total_visits_online	Total_calls_made
0	1.750192	-1.239437	-0.858684	-0.549573	-1.253982
1	0.413280	-0.779381	-1.473803	2.495669	1.873420
2	0.413280	1.060843	-0.858684	0.127148	0.135974
3	-0.121485	0.140731	-0.858684	-0.549573	0.135974
4	1.750192	0.600787	-1.473803	3.172390	-0.211515

In [19]:

```
# Creating copy of the data to store labels from each algorithm

data_scaled_copy = data_scaled.copy(deep = True)
```

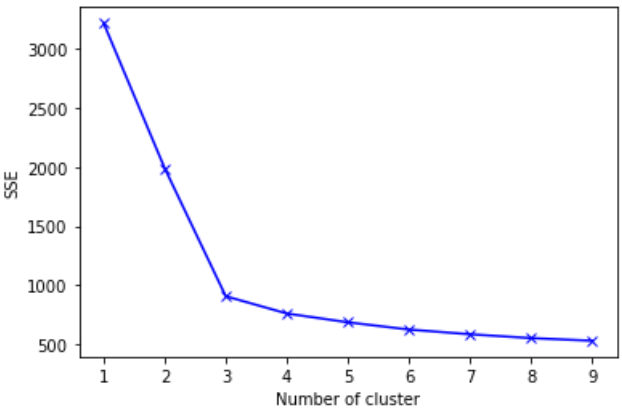
K-Means

3 steps:

1. Initialize a dictionary to store the Sum of Squared Error (SSE) for each K
2. Run for a range of Ks and store SSE for each run
3. Plot the SSE vs K and plot the elbow curve

In [20]:

```
# step 1
sse = {}

# step 2 - iterate for a range of Ks and fit the scaled data to the algorithm.
for k in range(1, 10):
    kmeans = KMeans(n_clusters = k, max_iter = 1000, random_state = 1).fit(data_scaled)
    sse[k] = kmeans.inertia_    # Use inertia attribute from the clustering object and store the inertia value for that K

# step 3
plt.figure()

plt.plot(list(sse.keys()), list(sse.values()), 'bx-')

plt.xlabel("Number of cluster")

plt.ylabel("SSE")

plt.show()
```



The above plot chose K= 3 because that is the optimal K value for decrease in SSE, after which there is no significant drop in SSE as number of clusters increase

In [22]:

```
kmeans = KMeans(n_clusters = 3, random_state = 1)                  # Apply the K-Means algorithm
kmeans.fit(data_scaled)                    # Fit the kmeans function on the scaled data

# Adding predicted labels to the original data and the scaled data
data_scaled_copy['Labels'] = kmeans.predict(data_scaled)    # Save the predictions on the scaled data from K-Means
data['Labels'] = kmeans.predict(data_scaled)          # Save the predictions on the scaled data from K-Means
```

look at the various features based on the labels.

Create the cluster profiles using the below summary statistics and box plots for each label

In [23]:

```
# Number of observations in each cluster
data.Labels.value_counts()
```

Out[23]:

```
1    374
0    221
2     49
Name: Labels, dtype: int64
```

In [24]:

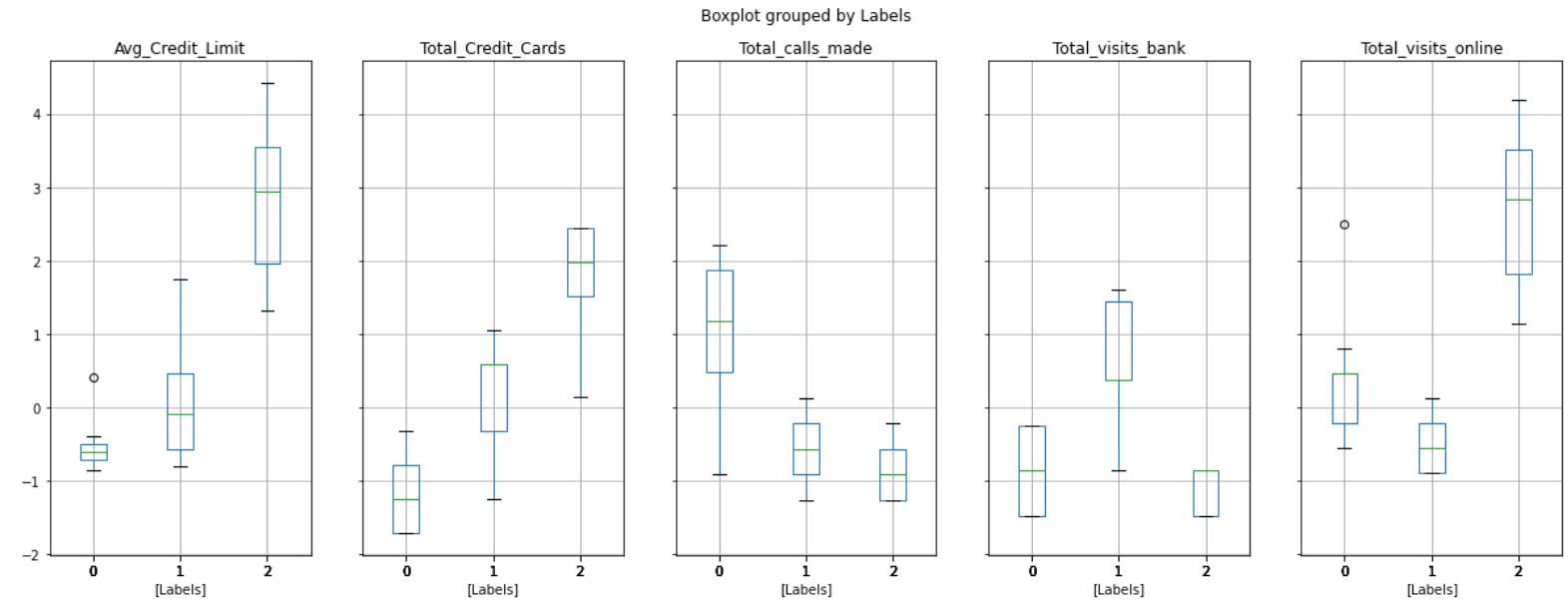
```
# Calculating summary statistics of the original data for each label
mean = data.groupby('Labels').mean()

median = data.groupby('Labels').median()

df_kmeans = pd.concat([mean, median], axis = 0)

df_kmeans.index = ['group_0 Mean', 'group_1 Mean', 'group_2 Mean', 'group_0 Median', 'group_1 Median', 'group_2 Median']

df_kmeans.T
```

Out[24]:

	group_0 Mean	group_1 Mean	group_2 Mean	group_0 Median	group_1 Median	group_2 Median
Avg_Credit_Limit	12239.819005	33893.048128	140102.040816	12000.0	31500.0	145000.0
Total_Credit_Cards	2.411765	5.508021	8.775510	2.0	6.0	9.0
Total_visits_bank	0.945701	3.489305	0.591837	1.0	3.0	1.0
Total_visits_online	3.561086	0.975936	10.979592	4.0	1.0	11.0
Total_calls_made	6.891403	1.997326	1.102041	7.0	2.0	1.0

In [25]:

```
# Visualizing different features w.r.t K-means labels
data_scaled_copy.boxplot(by = 'Labels', layout = (1, 5), figsize = (20, 7))

plt.show()
```



**Cluster Profiles:

--Cluster 2 customers had the highest average credit card limit, also had more credit cards and they used more of the online channel for query. Possibly these customers pay up there loans on time and likely to have better jobs. These customers should be given more attention and a better strategy should be used when responding to their queries because they use more of the online channel

--Cluster 0 customers had the least average credit limit and least number of credit cards. A lot of factors could give rise to customers in these group, e.g less use of loans, low income, not paying up loans and more. They used the call center channel more than other channels for query

--Cluster 1 indicates a mix of customers with low and high average credit card limit, the use of credit cards is on the average among the 3 clusters. They prefered visiting the bank as their best choice for submitting a query.

Gaussian Mixture Model

- **Apply the Gaussian Mixture Model algorithm on the scaled data with n_components=3 and random_state=1**
- **Create the cluster profiles using the below summary statistics and box plots for each label**
- **Compare the clusters from both algorithms - K-means and Gaussian Mixture Model**

In [26]:

```
gmm = GaussianMixture(n_components = 3, random_state = 1)    # Apply the Gaussian Mixture algorithm on the scaled data with n_components=3 and rand
```

```
gmm.fit(data_scaled)    # Fit the model on the scaled data

data_scaled_copy['GmmLabels'] = gmm.predict(data_scaled)

data['GmmLabels'] = gmm.predict(data_scaled)
```

In [27]:

```
# Number of observations in each cluster
data.GmmLabels.value_counts()
```

Out[27]:

```
1    374
0    221
2     49
Name: GmmLabels, dtype: int64
```

In [28]:

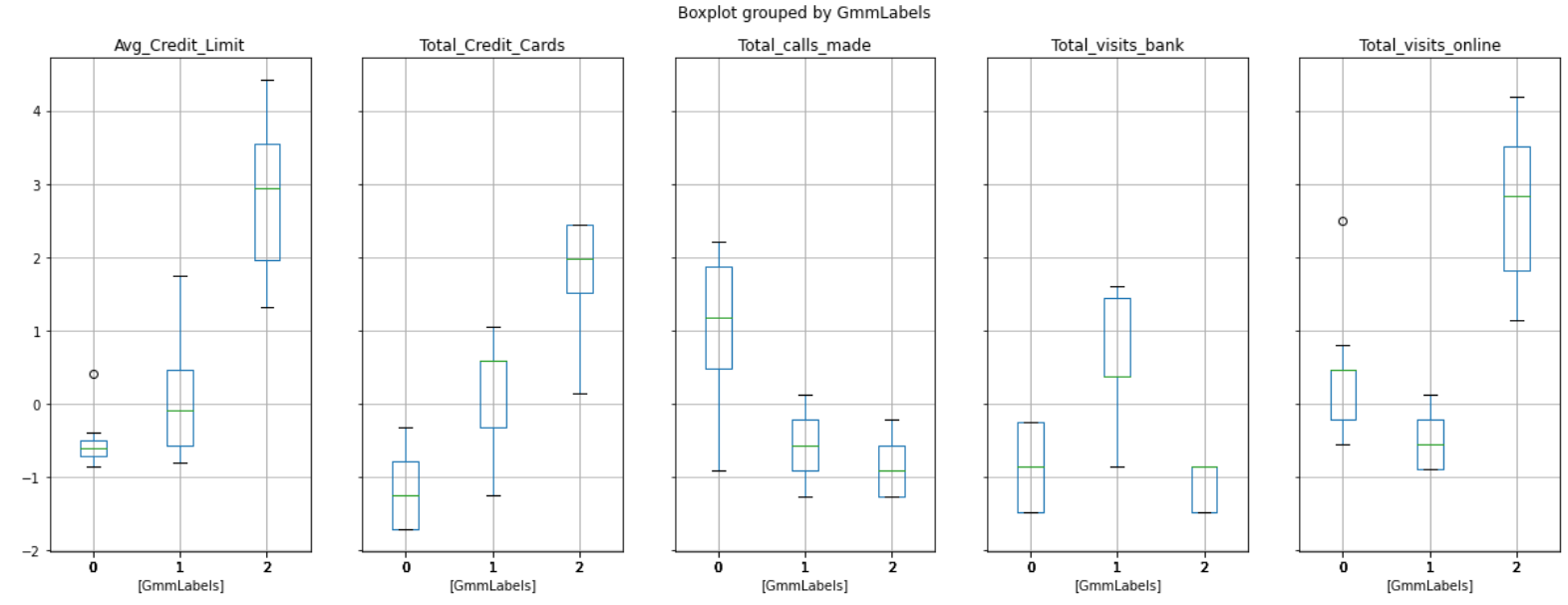
```
# Calculating the summary statistics of the original data for each label
original_features = ["Avg_Credit_Limit", "Total_Credit_Cards", "Total_visits_bank", "Total_visits_online", "Total_calls_made"]

mean = data.groupby('GmmLabels').mean()

median = data.groupby('GmmLabels').median()

df_gmm = pd.concat([mean, median], axis = 0)

df_gmm.index = ['group_0 Mean', 'group_1 Mean', 'group_2 Mean', 'group_0 Median', 'group_1 Median', 'group_2 Median']

df_gmm[original_features].T
```

Out[28]:

	group_0 Mean	group_1 Mean	group_2 Mean	group_0 Median	group_1 Median	group_2 Median
Avg_Credit_Limit	12239.819005	33893.048128	140102.040816	12000.0	31500.0	145000.0
Total_Credit_Cards	2.411765	5.508021	8.775510	2.0	6.0	9.0
Total_visits_bank	0.945701	3.489305	0.591837	1.0	3.0	1.0
Total_visits_online	3.561086	0.975936	10.979592	4.0	1.0	11.0
Total_calls_made	6.891403	1.997326	1.102041	7.0	2.0	1.0

In [29]:

```
# Plotting boxplots with the new GMM based labels

features_with_lables = ["Avg_Credit_Limit", "Total_Credit_Cards", "Total_visits_bank", "Total_visits_online", "Total_calls_made", "GmmLabels"]

data_scaled_copy[features_with_lables].boxplot(by = 'GmmLabels', layout = (1, 5),figsize = (20, 7))

plt.show()
```



**Cluster Profiles:

--Cluster 2 customers has the highest average credit limit, cluster 1 customers had an in between average credit limit and cluster 0 had the least average credit limit.

**Comparing Clusters:

--Comparing GMM to K-means algorithm shows no significant difference, so either of them could be used for grouping

K-Medoids

- **Apply the K-Medoids clustering algorithm on the scaled data with n_clusters=3 and random_state=1**
- **Create cluster profiles using the below summary statistics and box plots for each label**
- **Compare the clusters from both algorithms - K-Means and K-Medoids**

In [30]:

```
kmedo = KMedoids(n_clusters = 3, random_state = 1)     # Apply the K-Medoids algorithm on the scaled data with n_components=3 and random_state=1

kmedo.fit(data_scaled)     # Fit the model on the scaled data

data_scaled_copy['kmedoLabels'] = kmedo.predict(data_scaled)

data['kmedoLabels'] = kmedo.predict(data_scaled)
```

In [31]:

```
# Number of observations in each cluster
data.kmedoLabels.value_counts()
```

Out[31]:

```
2    289
0    222
1    133
Name: kmedoLabels, dtype: int64
```

In [32]:

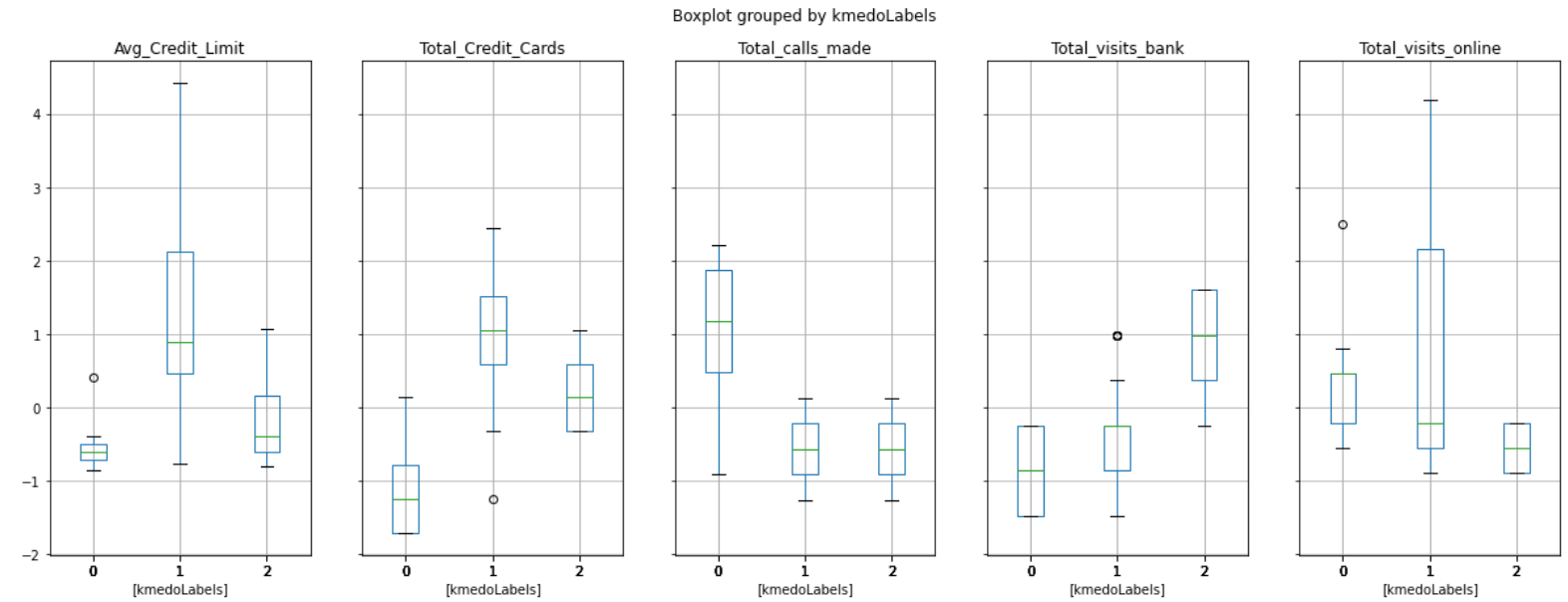
```
# Calculating summary statistics of the original data for each label
mean = data.groupby('kmedoLabels').mean()

median = data.groupby('kmedoLabels').median()

df_kmedoids = pd.concat([mean, median], axis = 0)

df_kmedoids.index = ['group_0 Mean', 'group_1 Mean', 'group_2 Mean', 'group_0 Median', 'group_1 Median', 'group_2 Median']

df_kmedoids[original_features].T
```

Out[32]:

	group_0 Mean	group_1 Mean	group_2 Mean	group_0 Median	group_1 Median	group_2 Median
Avg_Credit_Limit	12216.216216	85052.631579	28449.826990	12000.0	68000.0	20000.0
Total_Credit_Cards	2.423423	7.030075	5.363322	2.0	7.0	5.0
Total_visits_bank	0.950450	1.691729	3.830450	1.0	2.0	4.0
Total_visits_online	3.554054	4.639098	0.982699	4.0	2.0	1.0
Total_calls_made	6.878378	1.969925	1.851211	7.0	2.0	2.0

In [33]:

```
# Plotting boxplots with the new K-Medoids based labels

features_with_lables = ["Avg_Credit_Limit", "Total_Credit_Cards", "Total_visits_bank", "Total_visits_online", "Total_calls_made", "kmedoLabels"]

data_scaled_copy[features_with_lables].boxplot(by = 'kmedoLabels', layout = (1, 5), figsize = (20, 7))

plt.show()
```



**Cluster Profiles:

--Cluster 1 customers had the highest average credit card limit, also had more credit cards and they used more of the online channel for query. Possibly these customers pay up there loans on time and likely to have better jobs. These customers should be given more attention and a better strategy should be used when responding to their queries because they use more of the online channel

--Cluster 0 customers had the least average credit limit and least number of credit cards. A lot of factors could give rise to customers in these group, e.g less use of loans, low income, not paying up loans and more. They used the call center channel more than other channels for query

--Cluster 2 indicates a mix of customers with low and high average credit card limit, the use of credit cards is on the average among the 3 clusters. They prefered visiting the bank as their best choice for submitting a query.

Let's compare the clusters from K-Means and K-Medoids

In [34]:

```
comparison = pd.concat([df_kmedoids, df_kmeans], axis = 1)[original_features]

comparison
```

Out[34]:

	Avg_Credit_Limit	Avg_Credit_Limit	Total_Credit_Cards	Total_Credit_Cards	Total_visits_bank	Total_visits_bank	Total_visits_online	Total_visits_onli
group_0 Mean	12216.216216	12239.819005	2.423423	2.411765	0.950450	0.945701	3.554054	3.5610
group_1 Mean	85052.631579	33893.048128	7.030075	5.508021	1.691729	3.489305	4.639098	0.9759
group_2 Mean	28449.826990	140102.040816	5.363322	8.775510	3.830450	0.591837	0.982699	10.9795
group_0 Median	12000.000000	12000.000000	2.000000	2.000000	1.000000	1.000000	4.000000	4.0000
group_1 Median	68000.000000	31500.000000	7.000000	6.000000	2.000000	3.000000	2.000000	1.0000
group_2 Median	20000.000000	145000.000000	5.000000	9.000000	4.000000	1.000000	1.000000	11.0000

**Comparing Clusters:

-- Kmeans customers with the highest average credit limit are in Cluster 2 while Kmedoids customers with the highest average credit limit are in cluster 1.

-- Kmedoids values across all varibales are lower for than K means. E.g. when comparing the customers based on highest average credit limit in Kmediod with that of Kmeans, customers with the lowest average credit limits in kmeans and Kmedoids.

--Kmedoids tends to be more robust to outliers compared to Kmeans

In [ ]: