Case Study: AllLife Bank Customer Segmentation

Unsupervised Learning

Problem Statement:

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.

Objective:

1. Explore and visualize the dataset.
2. Identify different segments in the existing customer (based on patterns), using clustering algorithms;
3. 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 Credi*t 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 [1]:
```
# To supress warnings
import warnings

warnings.filterwarnings("ignore")

# This will help in making the Python code more structured automatically (goo
%load_ext nb_black

# To help with reading and manupulation data
import pandas as pd
import numpy as np

# To help with data visualization
import matplotlib.pyplot as plt
import seaborn as sns


# Removes the limit for the number of displayed columns
pd.set_option("display.max_columns", None)
# Sets the limit for the number of displayed rows
pd.set_option("display.max_rows", 200)


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

# To compute distances
from scipy.spatial.distance import cdist, pdist

# To perform kMean clustering
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score

# to perform hierarchical clustering, compute cophenetic correlation, and cre
from sklearn.cluster import AgglomerativeClustering
from scipy.cluster.hierarchy import dendrogram, linkage, cophenet

# to perform PCA
from sklearn.decomposition import PCA

# To visualize the elbow curve and silhouette scores
from yellowbrick.cluster import KElbowVisualizer, SilhouetteVisualizer
```

Load and overview the dataset

In [2]:
```
# Load the data into pandas dataframe
bank = pd.read_excel("Credit Card Customer Data (1).xlsx")
```

In [3]:
```
# Coping the data to another variable to avoid any changes to the original da
df = bank.copy()
```

In [4]:
```
# viewing a random sample of the dataset
df.sample(n=5, random_state=1)
```

Out[4]:

	Sl_No	Customer Key	Avg_Credit_Limit	Total_Credit_Cards	Total_visits_bank	Total_visits_o
547	548	38125	26000	4	5	
353	354	94437	9000	5	4	
499	500	65825	68000	6	4	
173	174	38410	9000	2	1	
241	242	81878	10000	4	5	

In [5]:
```
# Fixing Customer Key column name
df.columns = [col.replace(" ", "_") for col in df.columns]
```

In [6]:
```
print(f" The dataset has {df.shape[0]} values and {df.shape[1]} columns")
```

```
 The dataset has 660 values and 7 columns
```

In [7]:
```
# Let's look at the structure of the data
df.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
```

- All variables are int and with no null values;
- We can drop the column: `Sl_No` as it is a Primary Key (unique for each customer) and will not add value to the model.

In [8]:
```
# Dropping the Primary Key column Sl_No
df.drop("Sl_No", axis=1, inplace=True)
```

Data pre-processing

In [9]:
```
# Checking the number of unique values in each columns
df.nunique()
```

Out[9]:
```
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
```

Observation:

- We can see that 5 customers have duplicated Customer_Key, which is the customer identification and should be unique for each of them, let's check them:

In [10]:
```
dup = df.groupby("Customer_Key").count()
```

In [11]:
```
dup_Key = list(dup.loc[dup.Avg_Credit_Limit >= 2].index)
dup_Key
```

Out[11]:
```
[37252, 47437, 50706, 96929, 97935]
```

In [12]:
```
for k in dup_Key:
    display(df.loc[df["Customer_Key"] == k])
```

	Customer_Key	Avg_Credit_Limit	Total_Credit_Cards	Total_visits_bank	Total_visits_onlin
48	37252	6000	4	0	
432	37252	59000	6	2	

	Customer_Key	Avg_Credit_Limit	Total_Credit_Cards	Total_visits_bank	Total_visits_onlin
4	47437	100000	6	0	1
332	47437	17000	7	3	

	Customer_Key	Avg_Credit_Limit	Total_Credit_Cards	Total_visits_bank	Total_visits_onlin
411	50706	44000	4	5	
541	50706	60000	7	5	

	Customer_Key	Avg_Credit_Limit	Total_Credit_Cards	Total_visits_bank	Total_visits_onlin
391	96929	13000	4	5	
398	96929	67000	6	2	

	Customer_Key	Avg_Credit_Limit	Total_Credit_Cards	Total_visits_bank	Total_visits_onlin
104	97935	17000	2	1	
632	97935	187000	7	1	

Observations:

- The duplicate values might correspond to customer profile changes and as such, there is no need to delete these records as these are actual occurrences at some point in the

time.
- We can see that all of them increases the Total Number of Credit Cards.
- Customers with less than 65000 Avg Credit Limit visited the bank more after profile changes and the ones with Avg greater visited less.
- Customers with 60000 or more visited more online channels and kept the same or more calls made except for customer 97935 which visited more online and no calls made.

In [13]:
```
# Checking for missing values
df.isna().sum()
```

Out[13]:
```
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 [14]:
```
# Let's check for duplicate informations
print(f"There is \033[1m{df.duplicated().sum()}\033[m values duplicated")
```

```
There is 0 values duplicated
```

Observation:

- There are no missing value and/or duplicated rows on the data set.

Summary of the dataset

In [15]:
```
df.describe().T
```

Out[15]:

	count	mean	std	min	25%	50%	75%
Customer_Key	660.0	55141.443939	25627.772200	11265.0	33825.25	53874.5	77202.5
Avg_Credit_Limit	660.0	34574.242424	37625.487804	3000.0	10000.00	18000.0	48000.0
Total_Credit_Cards	660.0	4.706061	2.167835	1.0	3.00	5.0	6.0
Total_visits_bank	660.0	2.403030	1.631813	0.0	1.00	2.0	4.0
Total_visits_online	660.0	2.606061	2.935724	0.0	1.00	2.0	4.0
Total_calls_made	660.0	3.583333	2.865317	0.0	1.00	3.0	5.0

Observations:

- Customer_Key is only a customer identification, this information is not going to be used for clustering.
- Avg_Credit_Limit has a huge range going from min 3000 to max 200000, it needs further analysis to understand if it is a outlier our not.
- Total_Credit_Cards goes from 1 to 10, which can explaint the hieghts Avg_Credit_Limit (we will check on EDA).
- Mean Total_Visit_Bank is 2.4 with median of 2.

- Max Total_Visit_Online is 15 with min 0.
- Total_Calls-Made has a range of 0 to 10.

EDA

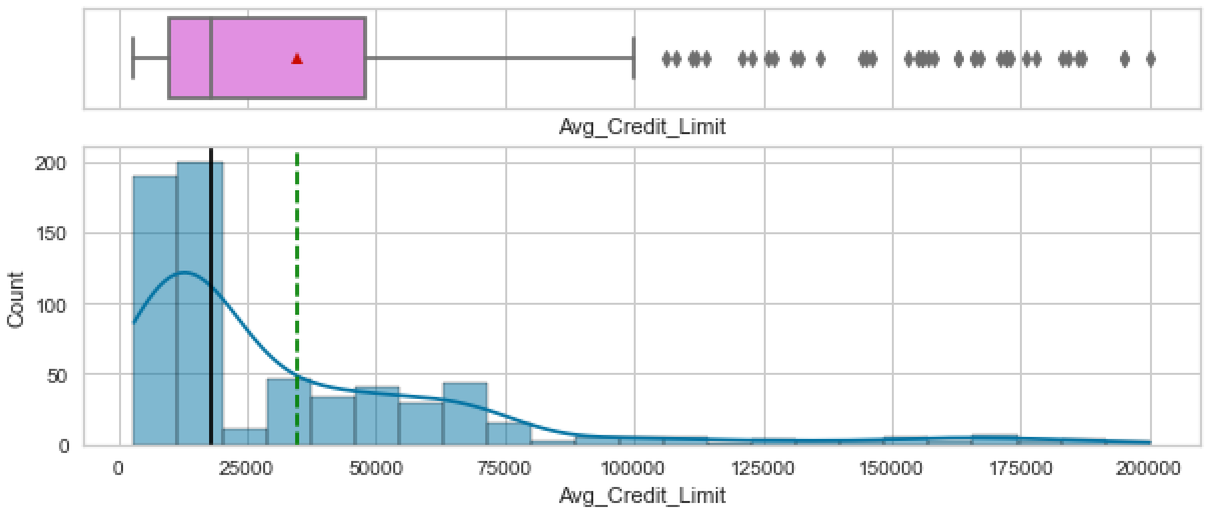
Univariate Analysis

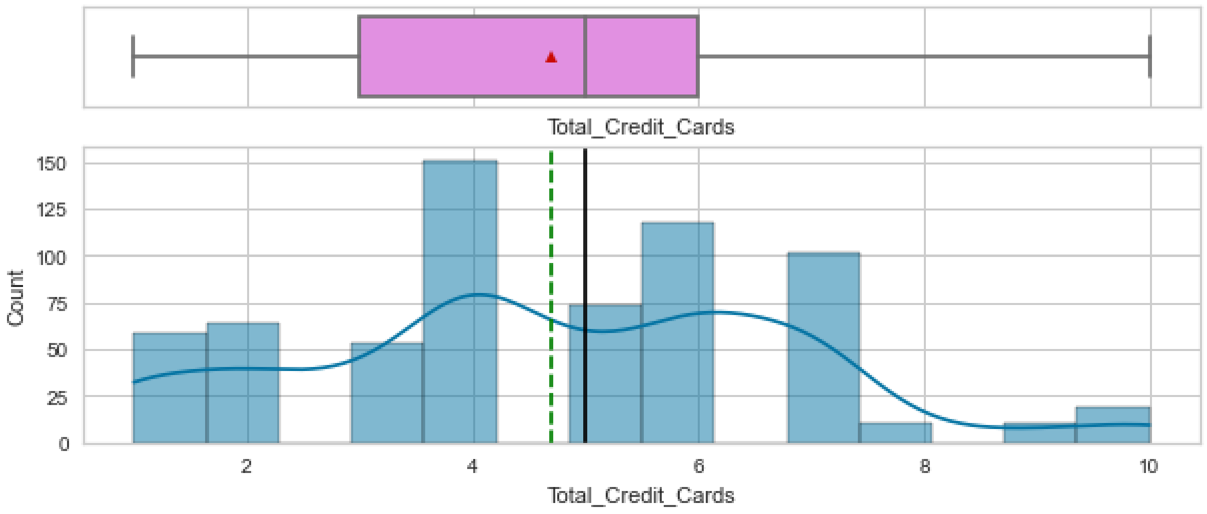
In [16]:
```
# function to plot a boxplot and a histogram along the same scale.


def histogram_boxplot(data, feature, figsize=(12, 7), kde=False, bins=None):
    """
    Boxplot and histogram combined

    data: dataframe
    feature: dataframe column
    figsize: size of figure (default (12,7))
    kde: whether to the show density curve (default False)
    bins: number of bins for histogram (default None)
    """
    f2, (ax_box2, ax_hist2) = plt.subplots(
        nrows=2,  # Number of rows of the subplot grid= 2
        sharex=True,  # x-axis will be shared among all subplots
        gridspec_kw={"height_ratios": (0.25, 0.75)},
        figsize=figsize,
    )  # creating the 2 subplots
    sns.boxplot(
        data=data, x=feature, ax=ax_box2, showmeans=True, color="violet"
    )  # boxplot will be created and a star will indicate the mean value of t
    sns.histplot(
        data=data, x=feature, kde=kde, ax=ax_hist2, bins=bins, palette="winte
    ) if bins else sns.histplot(
        data=data, x=feature, kde=kde, ax=ax_hist2
    )  # For histogram
    ax_hist2.axvline(
        data[feature].mean(), color="green", linestyle="--"
    )  # Add mean to the histogram
    ax_hist2.axvline(
        data[feature].median(), color="black", linestyle="-"
    )  # Add median to the histogram
```

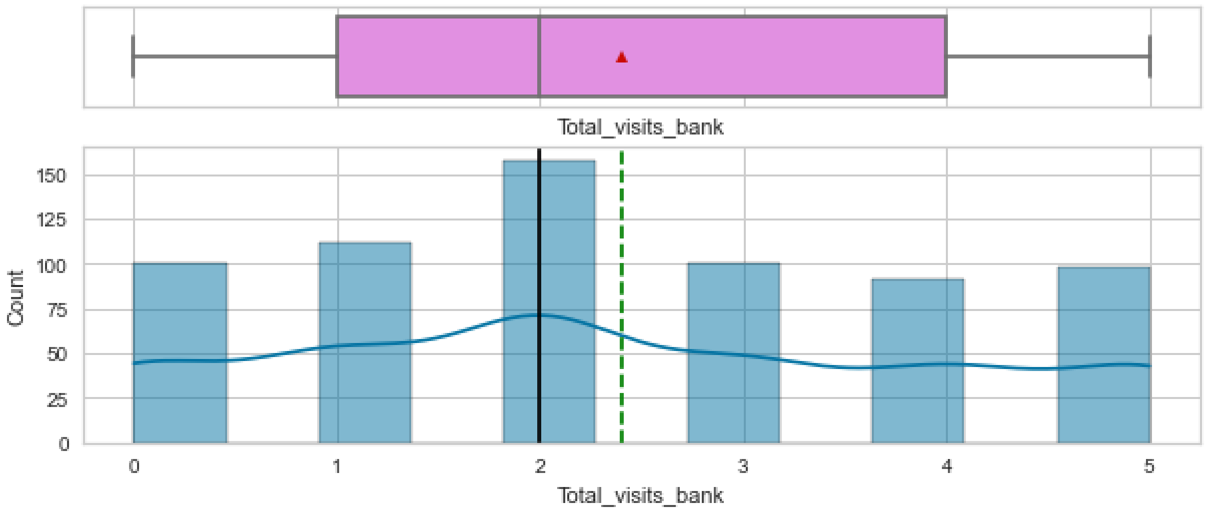
In [17]:
```
# function to create labeled barplots


def labeled_barplot(data, feature, perc=False, n=None):
    """
    Barplot with percentage at the top

    data: dataframe
    feature: dataframe column
    perc: whether to display percentages instead of count (default is False)
    n: displays the top n category levels (default is None, i.e., display all
    """

    total = len(data[feature])  # length of the column
    count = data[feature].nunique()
    if n is None:
        plt.figure(figsize=(count + 1, 5))
    else:
        plt.figure(figsize=(n + 1, 5))

    plt.xticks(rotation=90, fontsize=15)
    ax = sns.countplot(
        data=data,
        x=feature,
        palette="Paired",
        order=data[feature].value_counts().index[:n].sort_values(),
    )

    for p in ax.patches:
        if perc:
            label = "{:.1f}%".format(
                100 * p.get_height() / total
            )  # percentage of each class of the category
        else:
            label = p.get_height()  # count of each level of the category

        x = p.get_x() + p.get_width() / 2  # width of the plot
        y = p.get_height()  # height of the plot

        ax.annotate(
            label,
            (x, y),
            ha="center",
            va="center",
            size=12,
            xytext=(0, 5),
            textcoords="offset points",
        )  # annotate the percentage

    plt.show()  # show the plot
```

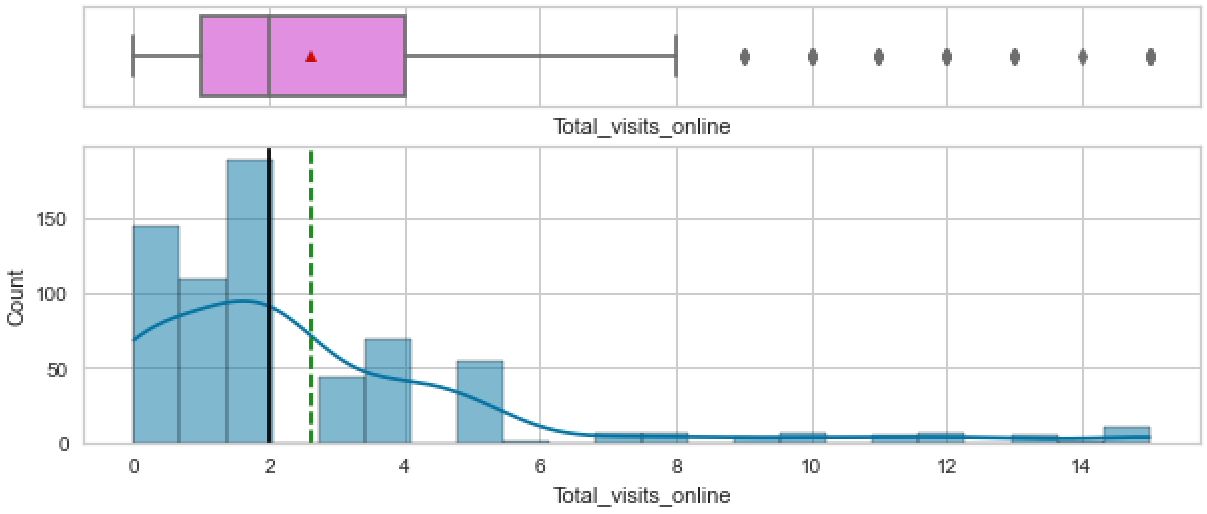
In [18]:

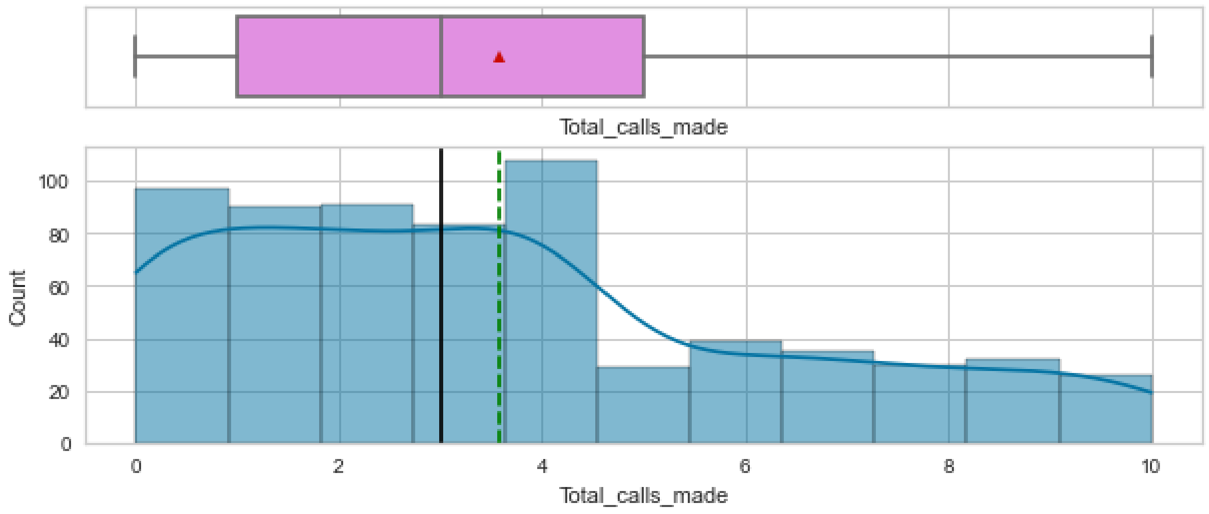
```
# selecting numerical columns
num_col = df.select_dtypes(include=np.number).columns.tolist()
num_col.remove(
    "Customer_Key"
)  # Its only a customer identification number, we don't need it on our analy


for item in num_col:
    histogram_boxplot(df, item, kde=True, figsize=(10, 4))
```



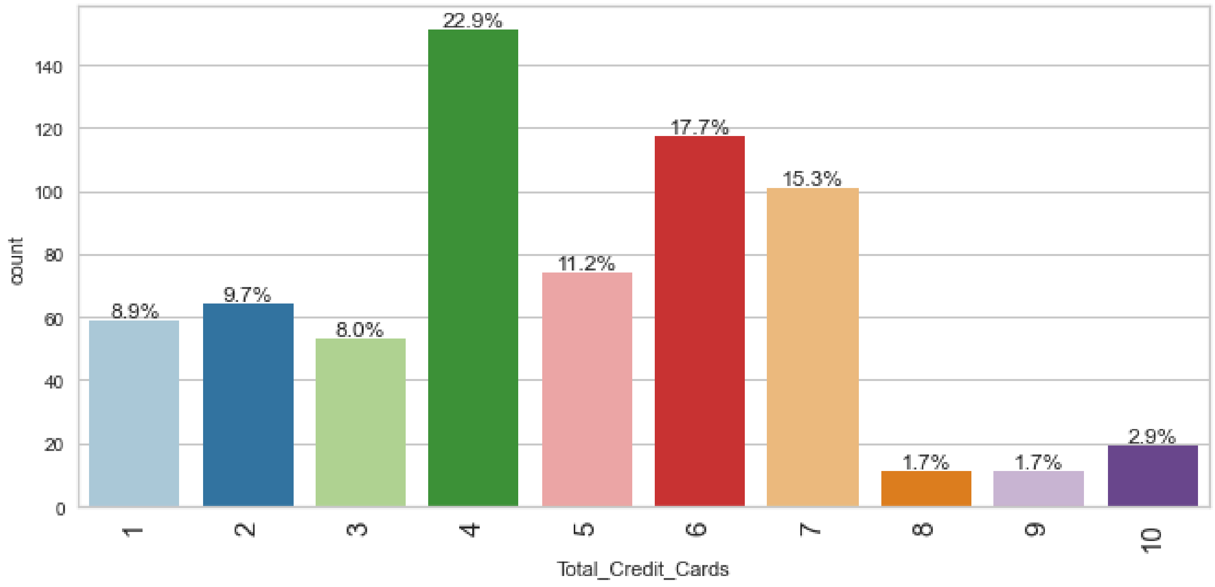

Total_Credit_Cards
Count
150
125
100
75
50
25
0
2
4
6
8
10
Total_Credit_Cards


Total_visits_bank
Count
150
125
100
75
50
25
0
0
1
2
3
4
5
Total_visits_bank

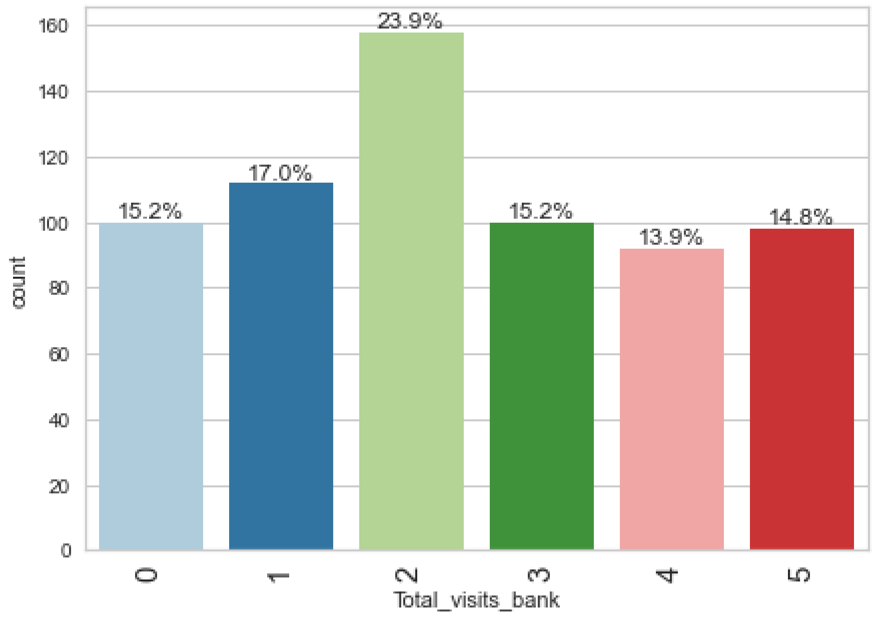

Total_visits_online
Count
150
100
50
0
0
2
4
6
8
10
12
14
Total_visits_online



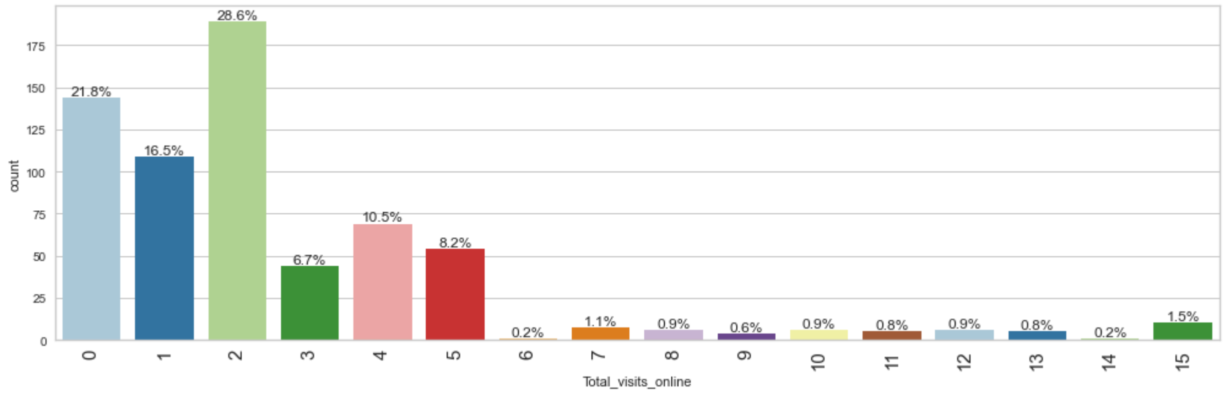
In [19]:
```
# let's explore discounts further
labeled_barplot(df, "Total_Credit_Cards", perc=True)
```



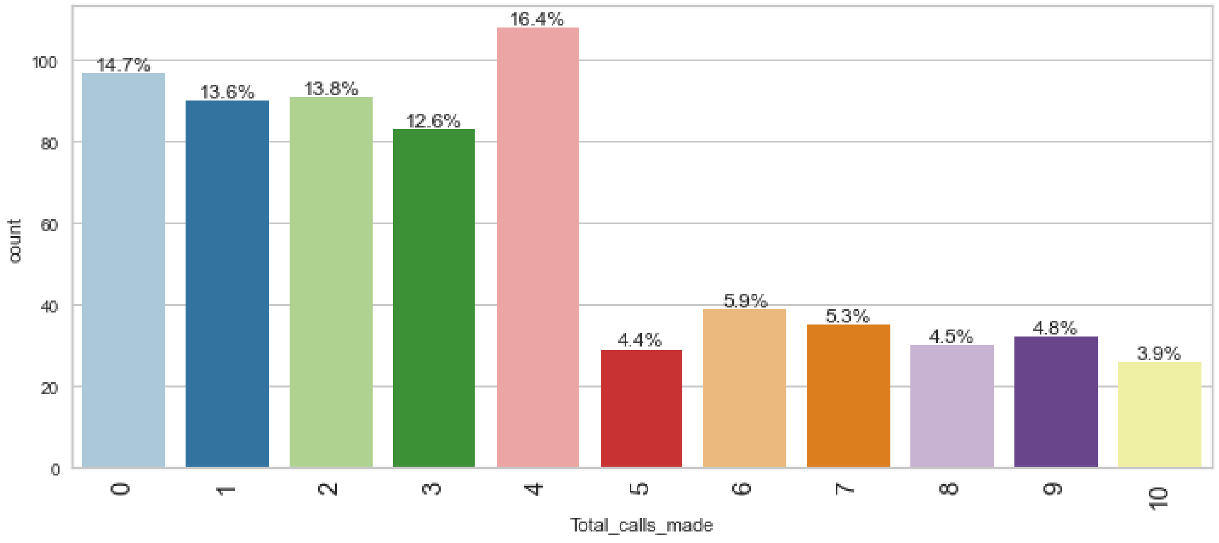
In [20]:
```
# let's explore discounts further
labeled_barplot(df, "Total_visits_bank", perc=True)
```



In [21]:
```
# let's explore discounts further
labeled_barplot(df, "Total_visits_online", perc=True)
```



In [22]:
```
# let's explore discounts further
labeled_barplot(df, "Total_calls_made", perc=True)
```



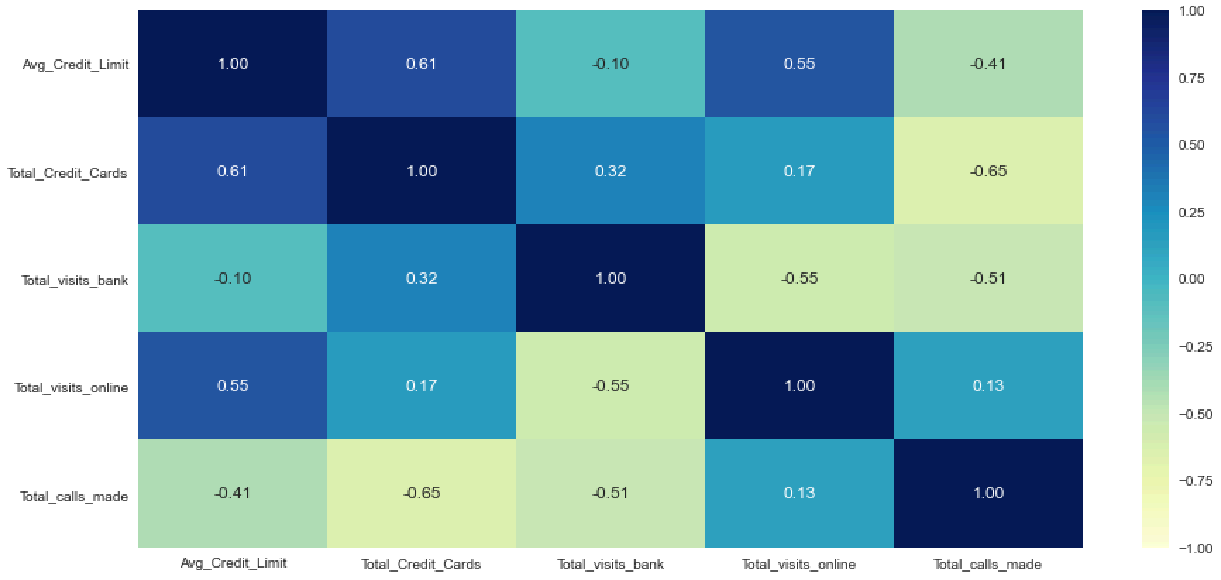
Observations:

- *Avg_Credit_Limit* and *Total_visits_online* are right-skewed and have upper outliers.
- 50% of customers have Avg_Credit_Limit less than 18000.0
- Most customers have 4 *Total_Credit_Cards* followed by 6 and 7.
- Median *Total_visits_bank* are 2 and is almost normally distributed with thick tails.
- 52% of customers have a *Total_visits_online* between 1 to 3, 22% neves used our online chanells, we can improve this number upgrading the service delivery model on online channels.
- 15% of our customers never called, at least 56% of the customers have Total_calls_made between 1 to 4 times.

Bivariate Analysis

Let's check for correlations.

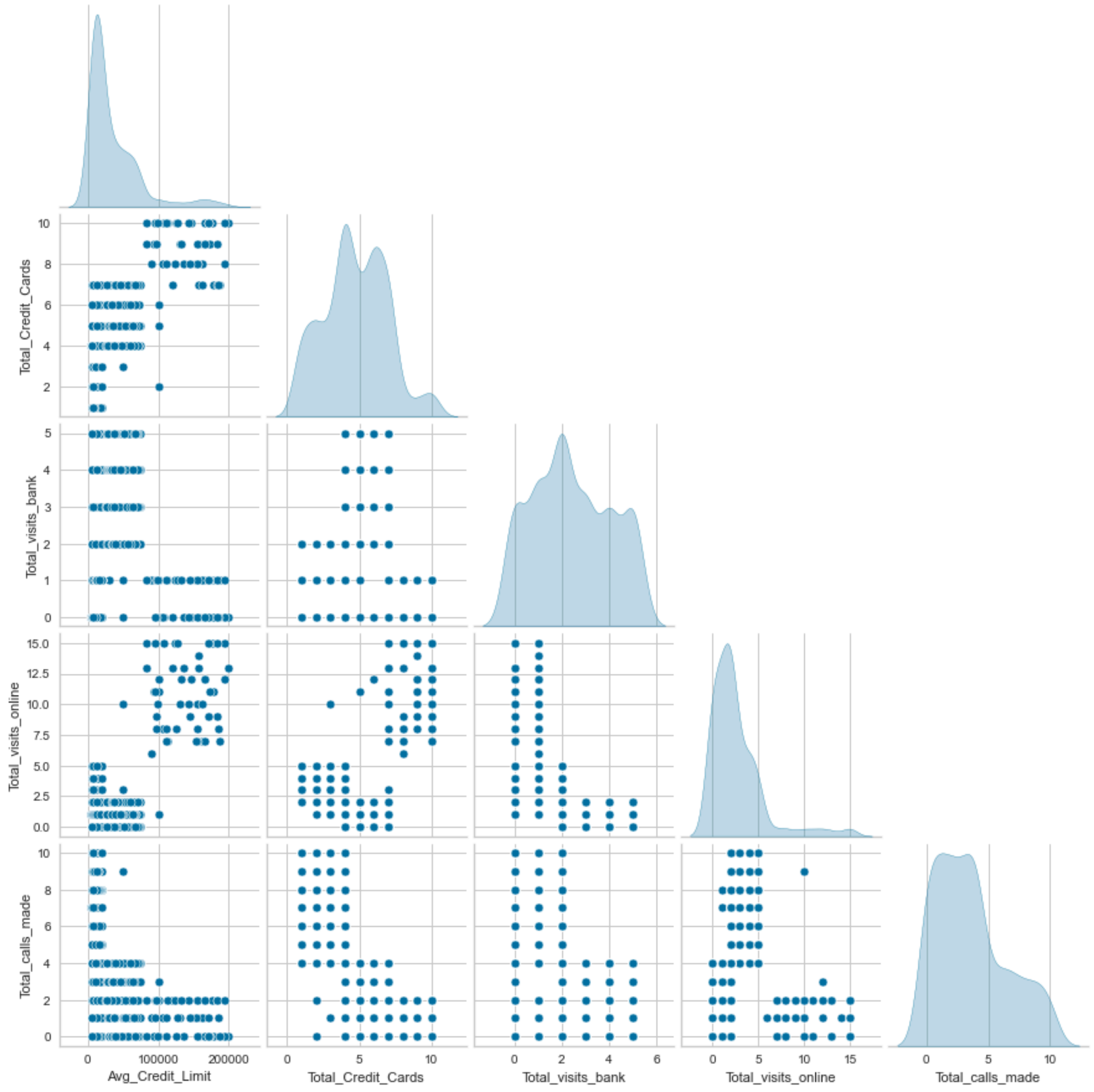
In [23]:
```
plt.figure(figsize=(15, 7))
sns.heatmap(df[num_col].corr(), annot=True, vmin=-1, vmax=1, fmt=".2f", cmap=
plt.show()
```



Observations:

- *Total_Credit_Cards* is positively correlated with *Avg_Credit_Limit*, as expected since more credit cards, more likehood to have a higher Average credit limit.
- *Total_visitis_online* is positivel correlated with *Avg_Credit_Limit*, suggesting that customers with higher Credit Limit tend to use more the Online tools and Negative Correlated with Total_visits_bank which indicates that online customers tend not to visit the bank.
- *Total_calls_made* is negative correlated with *Avg_Credit_Limit*, consequently negative correlated with *Total_Credit_Cards* and *Total_visitis_bank*, which indicate that customers who use more calls channels, are more likehood to visit less the bank and also have a small *Avg_Credit_Limit*.

In [24]:
```
sns.pairplot(data=df[num_col], diag_kind="kde", corner=True)
plt.show()
```



Observations:

- Customers with more than 7 credit cards are more likehood to have an avg Credit Limit greater than 100000 as expected, so customers with high Avg Credit Limit are not outliers, and it can also represent one different cluster.
- Customers that visit the bank more than 2 times have a Avg Credit Card low than 100000 and between 4 and 7 credit cards.
- Customers that have high Avg Credit Card so high total credit card are more likehood to do more online visits and less bank visit.
- Customers with low Avg Credit Limit, 1 to 4 total credit card, prefers to make calls, visiting less the bank and 1 to 5 visit on the online channels.

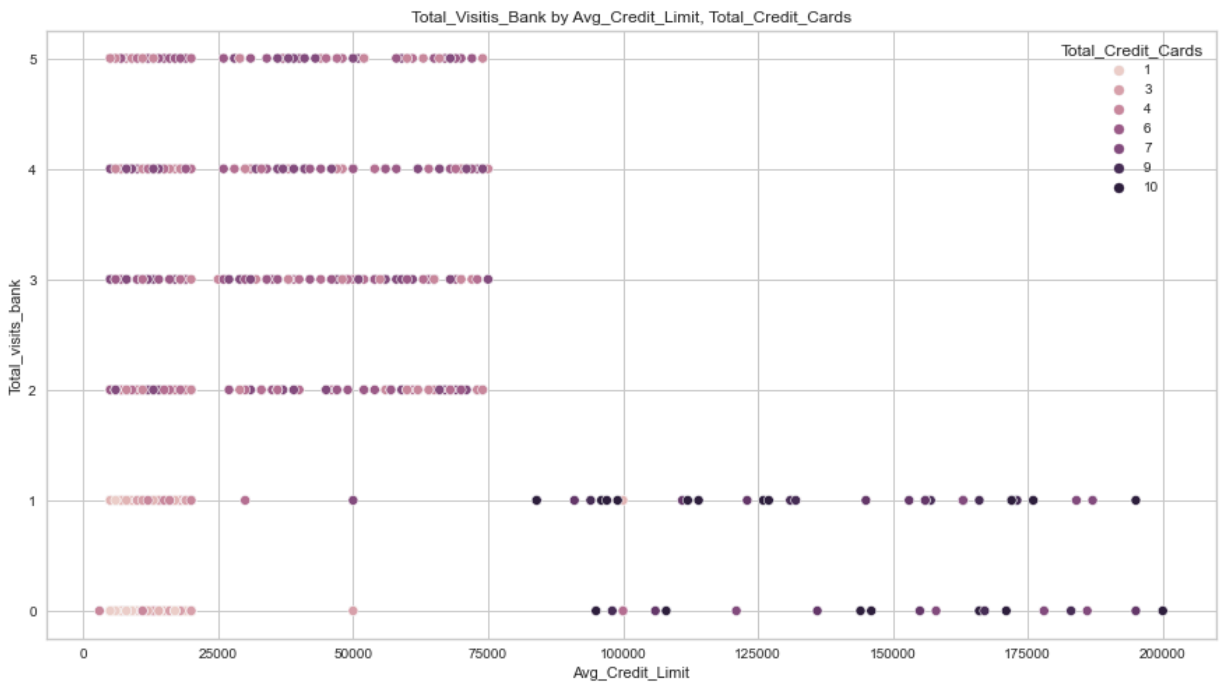
Multvariate Analysis

In [25]:
```
# Ploting scatterplot for analysis about correlation between Total_Visits_Ban
plt.figure(figsize=(15, 8))

sns.scatterplot(
    df["Avg_Credit_Limit"], df["Total_visits_bank"], hue=df["Total_Credit_Car
)
```

```
plt.title("Total_Visitis_Bank by Avg_Credit_Limit, Total_Credit_Cards")
```

Out[25]:

```
Text(0.5, 1.0, 'Total_Visitis_Bank by Avg_Credit_Limit, Total_Credit_Cards')
```



Observations:

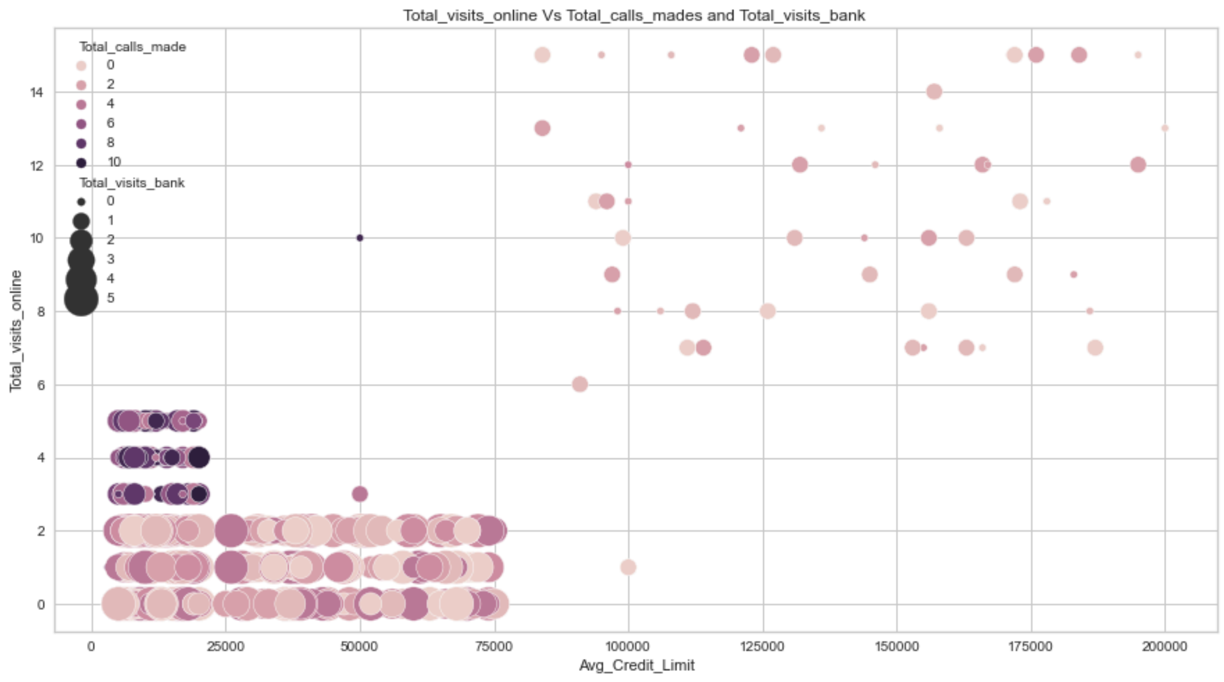
- Customers with more credit_cards and avg_limit visit the bank 0 or 1 time yearly.
- Customers with less than 4 credit_cards and avg_limit less than 50000 also visit the bank 0 or 1 time yearly.
- On the other hand, customers with more than 4 credit_cards and avg_limit less than 75000 are more likehood to visit the bank.

In [26]:

```
# Ploting scatterplot for analysis about correlation between Total_Visits_Ban
plt.figure(figsize=(15, 8))

sns.scatterplot(
    df["Avg_Credit_Limit"],
    df["Total_visits_online"],
    hue=df["Total_calls_made"],
    size=df["Total_visits_bank"],
    sizes=(30, 600),
)

plt.title("Total_visits_online Vs Total_calls_mades and Total_visits_bank ")
```

Out[26]:

```
Text(0.5, 1.0, 'Total_visits_online Vs Total_calls_mades and Total_visits_ban
k ')
```



Obsrvations:

- Customers that are more used to the online channels call less, we can work to improve the use of our online channels.
- Customers that are starting to use our online channels (3 to 5 access) they still calling between 4 to 10 times, and they have a low avg_credit_limit, we need to understand the calls, work for a better intuitive online channel with simple use and interface.
- Customers that use less the online channel and call less they are more likehood to visit the bank more often and have a avg_credit_limit less than 75000.

Data pre-processing

Let's scale the data before we proceed to cluster it.

In [27]:
```
# variables used for clustering
num_col
```

Out[27]:
```
['Avg_Credit_Limit',
 'Total_Credit_Cards',
 'Total_visits_bank',
 'Total_visits_online',
 'Total_calls_made']
```

In [28]:
```
# scaling the dataset before clustering
scaler = StandardScaler()
subset = df[num_col].copy()
subset_scaled = scaler.fit_transform(subset)
```

In [29]:
```
# creating a dataframe of the scaled columns
subset_scaled_df = pd.DataFrame(subset_scaled, columns=subset.columns)
```

In [30]:
```
subset_scaled_df.head()
```

Out[30]:

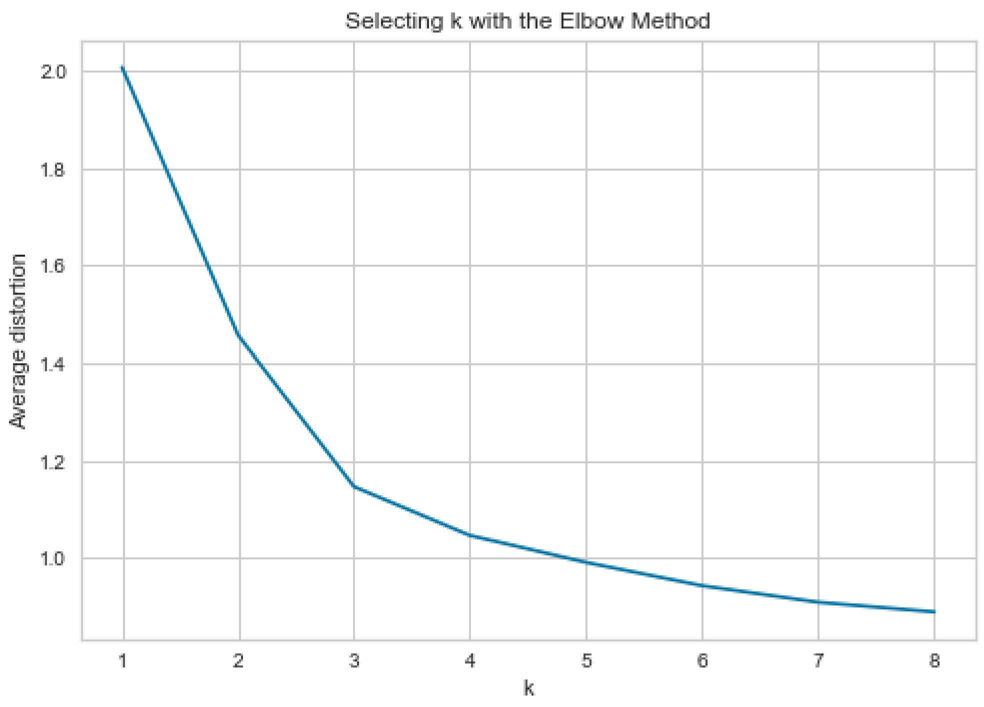
	Avg_Credit_Limit	Total_Credit_Cards	Total_visits_bank	Total_visits_online	Total_calls_mad
0	1.740187	-1.249225	-0.860451	-0.547490	-1.25153
1	0.410293	-0.787585	-1.473731	2.520519	1.89185
2	0.410293	1.058973	-0.860451	0.134290	0.14552
3	-0.121665	0.135694	-0.860451	-0.547490	0.14552
4	1.740187	0.597334	-1.473731	3.202298	-0.20373

1. K-means Clustering

Checking Elbow method

In [31]:
```
clusters = range(1, 9)
meanDistortions = []

for k in clusters:
    model = KMeans(n_clusters=k)
    model.fit(subset_scaled_df)
    prediction = model.predict(subset_scaled_df)
    distortion = (
        sum(
            np.min(cdist(subset_scaled_df, model.cluster_centers_, "euclidean
        )
        / subset_scaled_df.shape[0]
    )

    meanDistortions.append(distortion)

    print("Number of Clusters:", k, "\tAverage Distortion:", distortion)

plt.plot(clusters, meanDistortions, "bx-")
plt.xlabel("k")
plt.ylabel("Average distortion")
plt.title("Selecting k with the Elbow Method")
plt.show()
```

```
Number of Clusters: 1   Average Distortion: 2.0069222262503614
Number of Clusters: 2   Average Distortion: 1.4571553548514269
Number of Clusters: 3   Average Distortion: 1.1466276549150365
Number of Clusters: 4   Average Distortion: 1.0463825294774465
Number of Clusters: 5   Average Distortion: 0.9908683849620168
Number of Clusters: 6   Average Distortion: 0.9430693962124551
Number of Clusters: 7   Average Distortion: 0.9091362001500471
Number of Clusters: 8   Average Distortion: 0.8893698376158061
```



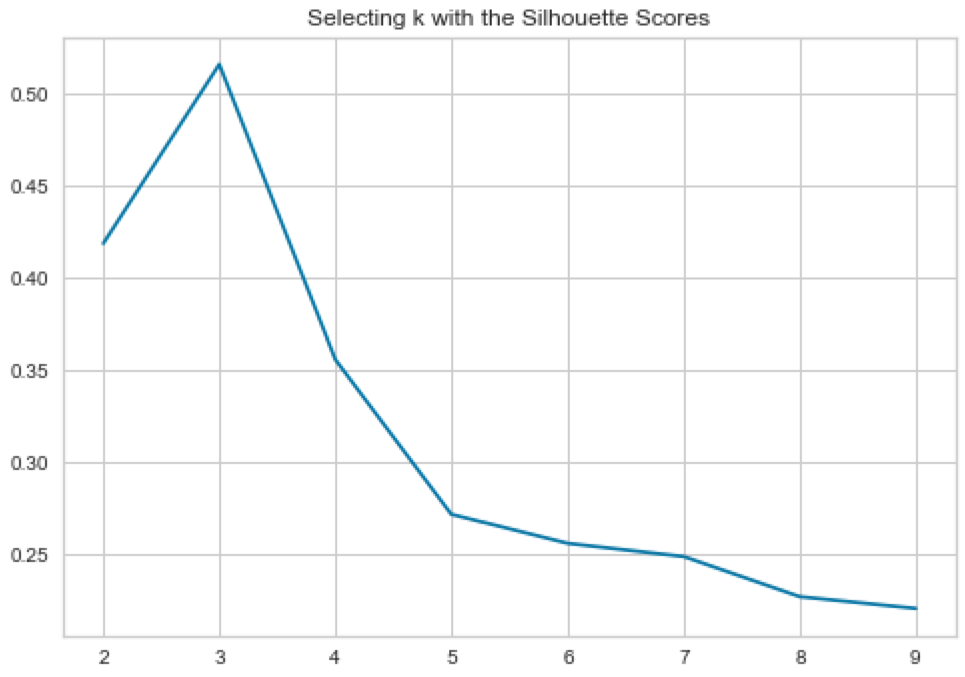
Observation:

- The distortion declines most at 3 and 4. Hence the optimal value for k seems to be 3 or 4.

Checking Silhouette scores

In [32]:
```
sil_score = []
cluster_list = list(range(2, 10))
for n_clusters in cluster_list:
    clusterer = KMeans(n_clusters=n_clusters)
    preds = clusterer.fit_predict((subset_scaled_df))
    # centers = clusterer.cluster_centers_
    score = silhouette_score(subset_scaled_df, preds)
    sil_score.append(score)
    print("For n_clusters = {}, the silhouette score is {})".format(n_cluster
plt.title("Selecting k with the Silhouette Scores")
plt.plot(cluster_list, sil_score)
plt.show()
```

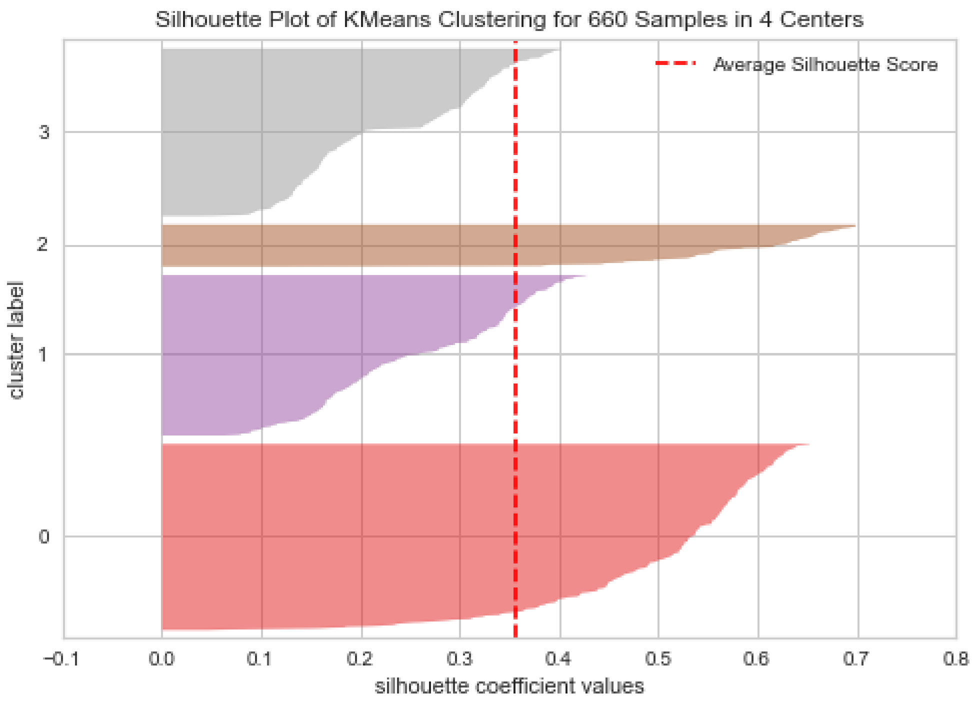
```
For n_clusters = 2, the silhouette score is 0.41842496663215445)
For n_clusters = 3, the silhouette score is 0.5157182558881063)
For n_clusters = 4, the silhouette score is 0.3556670619372605)
For n_clusters = 5, the silhouette score is 0.2715953810610515)
For n_clusters = 6, the silhouette score is 0.25583657571102003)
For n_clusters = 7, the silhouette score is 0.24865472065730512)
For n_clusters = 8, the silhouette score is 0.2268656542161218)
For n_clusters = 9, the silhouette score is 0.22048913325777397)
```



Observation:

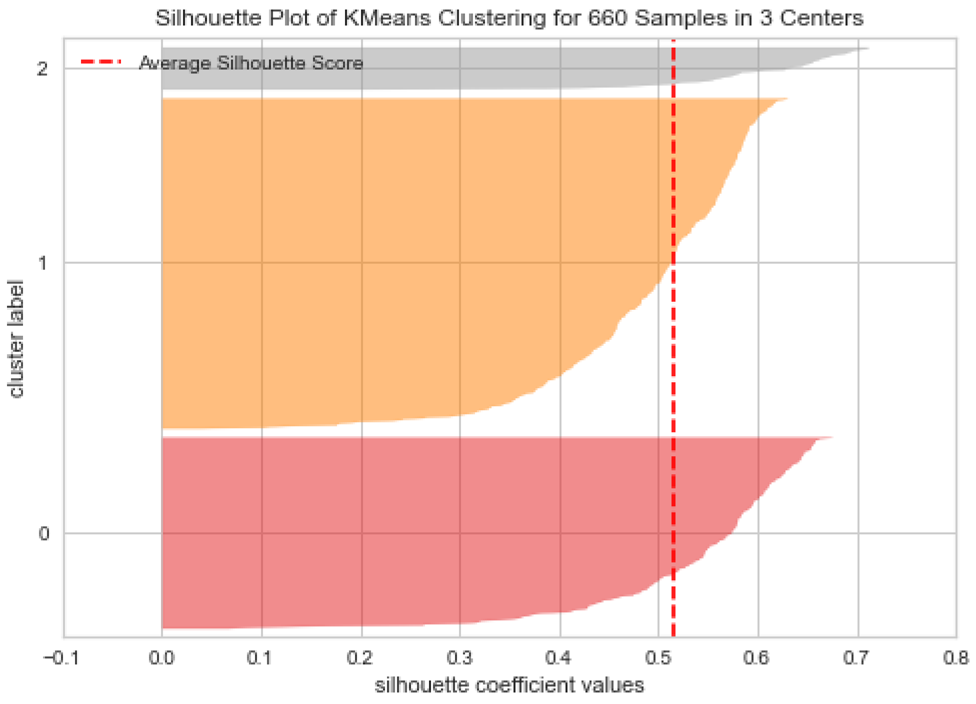
- Silhouette score for k=3 is higher than for 4. So, we will choose 3 as value of k.

In [33]:
```
# finding optimal no. of clusters with silhouette coefficients
visualizer = SilhouetteVisualizer(KMeans(4, random_state=1))
visualizer.fit(subset_scaled_df)
visualizer.show()
```



Out[33]:
```
<AxesSubplot:title={'center':'Silhouette Plot of KMeans Clustering for 660 Sa
mples in 4 Centers'}, xlabel='silhouette coefficient values', ylabel='cluster
label'>
```

In [34]:
```
# finding optimal no. of clusters with silhouette coefficients
visualizer = SilhouetteVisualizer(KMeans(3, random_state=1))
visualizer.fit(subset_scaled_df)
visualizer.show()
```



```
Out[34]: <AxesSubplot:title={'center':'Silhouette Plot of KMeans Clustering for 660 Sa
mples in 3 Centers'}, xlabel='silhouette coefficient values', ylabel='cluster
label'>
```

Observation:

- The Silhouette plot, with ***k = 3*** have all clusters plot is beyond average Silhouette score, with an okay thickness and do not have wide fluctuations in the size.

```
In [35]: %%time
kmeans = KMeans(n_clusters=3, random_state=0)
kmeans.fit(subset_scaled_df)
```

```
Wall time: 26.1 ms
Out[35]: KMeans(n_clusters=3, random_state=0)
```

```
In [36]: # adding kmeans cluster labels to the original and scaled dataframes

subset["K_means_segments"] = kmeans.labels_
subset_scaled_df["K_means_segments"] = kmeans.labels_
```

1.1 K-means Cluster Profiling

```
In [37]: cluster_profile = subset.groupby("K_means_segments").mean()
```

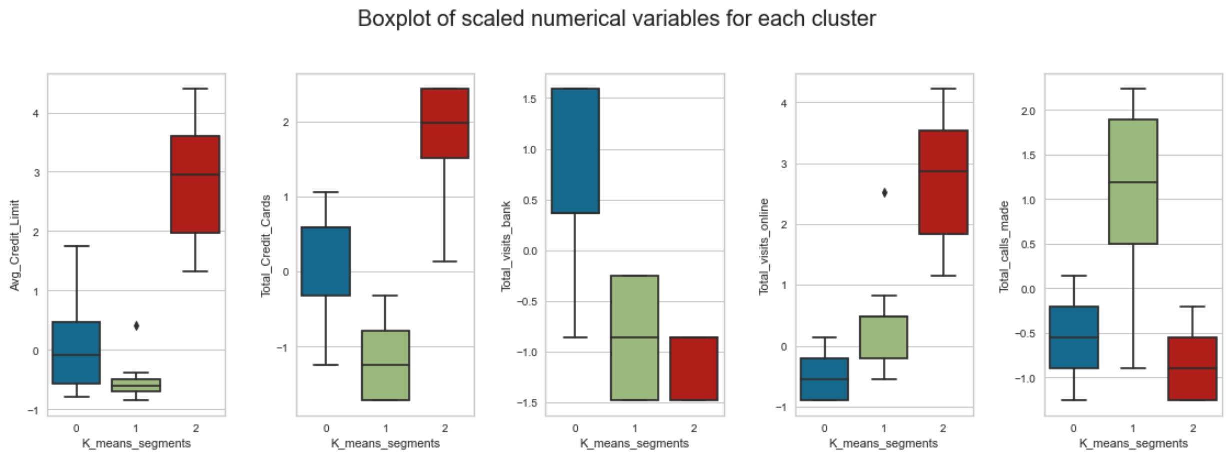
```
In [38]: cluster_profile["count_in_each_segments"] = (
    subset.groupby("K_means_segments")["Avg_Credit_Limit"].count().values
)
```

```
In [39]: # let's display cluster profiles
cluster_profile.style.highlight_max(color="lightgreen", axis=0)
```

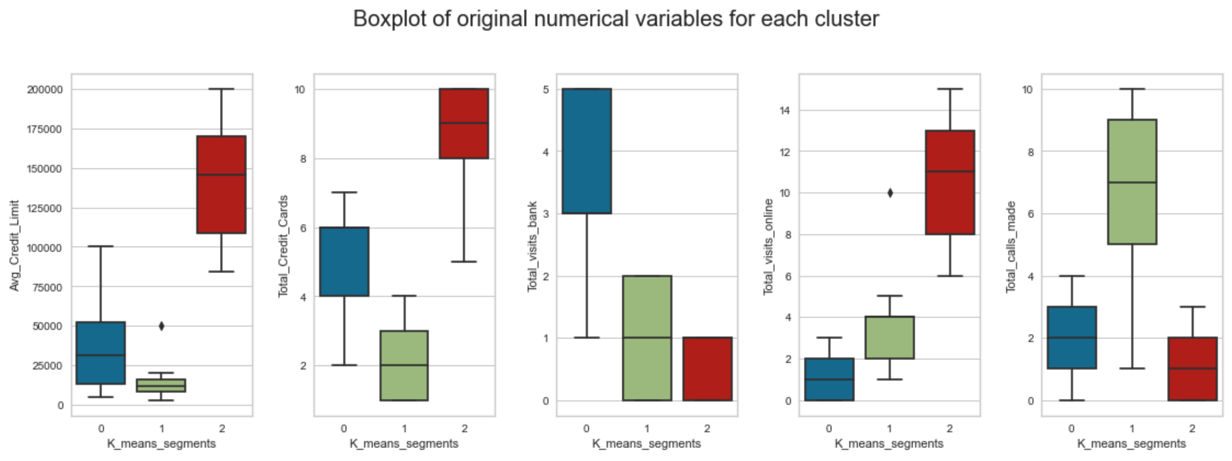
Out[39]:

	Avg_Credit_Limit	Total_Credit_Cards	Total_visits_bank	Total_visits_onlin
K_means_segments				
0	33782.383420	5.515544	3.489637	0.98186
1	12174.107143	2.410714	0.933036	3.55357
2	141040.000000	8.740000	0.600000	10.90000

In [40]:
```
fig, axes = plt.subplots(1, 5, figsize=(16, 6))
fig.suptitle("Boxplot of scaled numerical variables for each cluster", fontsi
counter = 0
for ii in range(5):
    sns.boxplot(
        ax=axes[ii],
        y=subset_scaled_df[num_col[counter]],
        x=subset_scaled_df["K_means_segments"],
    )
    counter = counter + 1

fig.tight_layout(pad=2.0)
```



In [41]:
```
fig, axes = plt.subplots(1, 5, figsize=(16, 6))
fig.suptitle("Boxplot of original numerical variables for each cluster", font
counter = 0
for ii in range(5):
    sns.boxplot(ax=axes[ii], y=subset[num_col[counter]], x=subset["K_means_se
    counter = counter + 1

fig.tight_layout(pad=2.0)
```



Insights

- **Cluster 0**:
  - We can say these are our *Gold* customers, with moderate Average Credit Limit and Total Credit Cards.
  - With an average of approximately 34 thousands Credit Limit and total of 5 Credit Cards.
  - These are the customer that visit the bank the most (4 times yearly), on the other hands are our customers that almost do not use our online chanells.
  - Customer in these cluster have a mean of 2 calls made per customer yearly.
  - Therefore this cluster is composed of moderate Average Limit and in person visit preferable.
- **Cluster 1**:
  - These are our *Silver* customers, with low Average Credit Limit (mean of 12174.11) and lowest Total of Credit Cards (mean of 2).
  - Customers in these cluster visit the bank 1 time yearly.
  - Silver Customers have a moderate use of Online Chanells, mean of almost 4 per customer per year oppositely these are the customer that made more calls, mean of almost 7.
  - Accordingly, this cluster is composed of low Average Limit and the preferable contact is to call customer service department.
- **Cluster 2**:
  - Our *Platinum* customers prefers the online channels using it almost 11 times per year.
  - They have the highest Average Credit Limit that goes arround 141040.
  - As expected are this cluster also have the highest total number of Credit Cards, with mean around 9.
  - Customers in these cluster almost never visit the bank or call it.
  - Hence, customers on this cluster prefers the Online contact and have the highest Average Credit Limit.

---

2. Hierarchical Clustering

In [42]:
```
# scaling the original dataset before clustering
subset_h = df[num_col].copy()
subset_scaled_h = scaler.fit_transform(subset_h)
```

In [43]:
```
# creating a dataframe of the scaled columns
subset_scaled_df_h = pd.DataFrame(subset_scaled_h, columns=subset_h.columns)
subset_scaled_df_h.head()
```

Out[43]:

	Avg_Credit_Limit	Total_Credit_Cards	Total_visits_bank	Total_visits_online	Total_calls_mad
0	1.740187	-1.249225	-0.860451	-0.547490	-1.25153

	Avg_Credit_Limit	Total_Credit_Cards	Total_visits_bank	Total_visits_online	Total_calls_mad
1	0.410293	-0.787585	-1.473731	2.520519	1.89185
2	0.410293	1.058973	-0.860451	0.134290	0.14552
3	-0.121665	0.135694	-0.860451	-0.547490	0.14552
4	1.740187	0.597334	-1.473731	3.202298	-0.20373

In [44]:
```
# list of distance metrics
distance_metrics = ["euclidean", "chebyshev", "mahalanobis", "cityblock"]

# list of linkage methods
linkage_methods = ["single", "complete", "average", "weighted"]

high_cophenet_corr = 0
high_dm_lm = [0, 0]

for dm in distance_metrics:
    for lm in linkage_methods:
        Z = linkage(subset_scaled_df_h, metric=dm, method=lm)
        c, coph_dists = cophenet(Z, pdist(subset_scaled_df_h))
        print(
            f"Cophenetic correlation for {dm.capitalize()} distance and {lm}
        )

        if high_cophenet_corr < c:
            high_cophenet_corr = c
            high_dm_lm[0] = dm
            high_dm_lm[1] = lm
    print("\n")
```

```
Cophenetic correlation for Euclidean distance and single linkage is 0.7391.
Cophenetic correlation for Euclidean distance and complete linkage is 0.86.
Cophenetic correlation for Euclidean distance and average linkage is 0.8977.
Cophenetic correlation for Euclidean distance and weighted linkage is 0.8862.


Cophenetic correlation for Chebyshev distance and single linkage is 0.7382.
Cophenetic correlation for Chebyshev distance and complete linkage is 0.8533.
Cophenetic correlation for Chebyshev distance and average linkage is 0.8974.
Cophenetic correlation for Chebyshev distance and weighted linkage is 0.8914.


Cophenetic correlation for Mahalanobis distance and single linkage is 0.7058.
Cophenetic correlation for Mahalanobis distance and complete linkage is 0.542
3.
Cophenetic correlation for Mahalanobis distance and average linkage is 0.832
7.
Cophenetic correlation for Mahalanobis distance and weighted linkage is 0.780
6.


Cophenetic correlation for Cityblock distance and single linkage is 0.7252.
Cophenetic correlation for Cityblock distance and complete linkage is 0.8731.
Cophenetic correlation for Cityblock distance and average linkage is 0.8963.
Cophenetic correlation for Cityblock distance and weighted linkage is 0.8826.
```

In [45]:
```
# printing the combination of distance metric and linkage method with the hig
print(f"Highest cophenetic correlation is \33[1m{ round(high_cophenet_corr, 4
      f" which is obtained with \33[1m{high_dm_lm[0].capitalize()}\33[m dista
```

```
Highest cophenetic correlation is 0.8977, which is obtained with Euclidean di
stance and average linkage.
```

> Euclidian distance is giving the hieghst Cophenetic Correlation for almost all linkage, so we will consider Euclidean as our distance and we will explore it with different linkage methods.

In [46]:
```
# list of linkage methods using Euclidean as a distanc
linkage_methods = ["single", "complete", "average", "centroid", "ward", "weig

high_cophenet_corr = 0
high_dm_lm = [0, 0]

for lm in linkage_methods:
    Z = linkage(subset_scaled_df_h, metric="euclidean", method=lm)
    c, coph_dists = cophenet(Z, pdist(subset_scaled_df_h))
    print(f"Cophenetic correlation for {lm} linkage is {round(c,4)}.")
    if high_cophenet_corr < c:
        high_cophenet_corr = c
        high_dm_lm[0] = "euclidean"
        high_dm_lm[1] = lm


# printing the combination of distance metric and linkage method with the hig
print(
    "\n"
    f"Highest cophenetic correlation considering Euclidean distance is: \033[
)
```

```
Cophenetic correlation for single linkage is 0.7391.
Cophenetic correlation for complete linkage is 0.86.
Cophenetic correlation for average linkage is 0.8977.
Cophenetic correlation for centroid linkage is 0.8939.
Cophenetic correlation for ward linkage is 0.7415.
Cophenetic correlation for weighted linkage is 0.8862.

Highest cophenetic correlation considering Euclidean distance is: 0.8977, whi
ch is obtained with average linkage.
```

Observations:

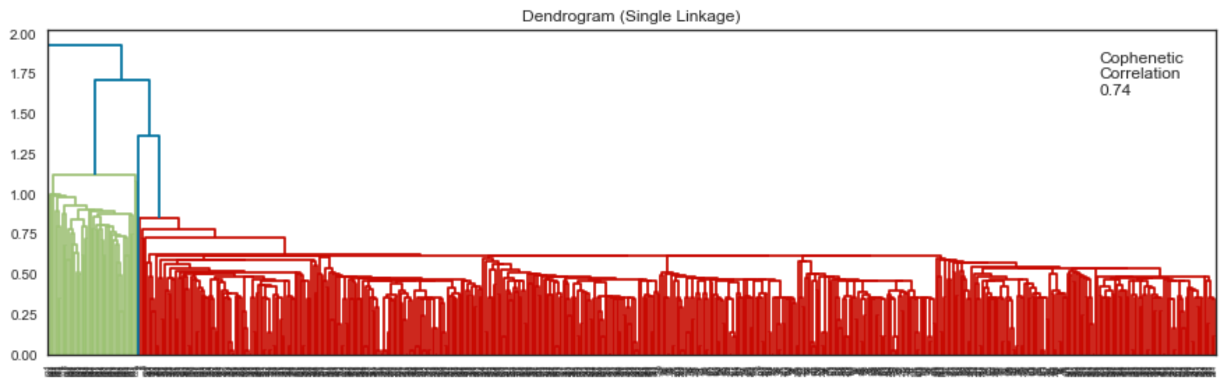
- We see that ***Euclidean*** Distance with ***Average*** linkage is giving us the maximum cophenetic correlation.

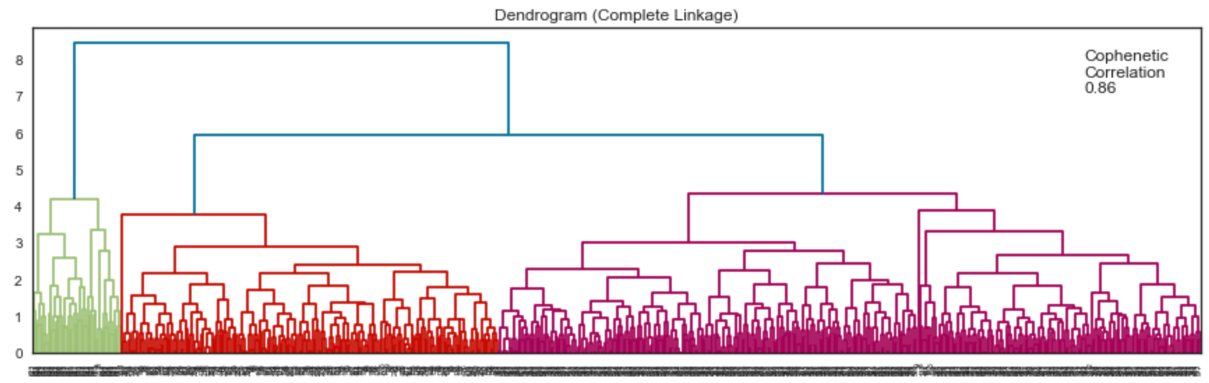
Let's see the dendograms for the different linkage methods.

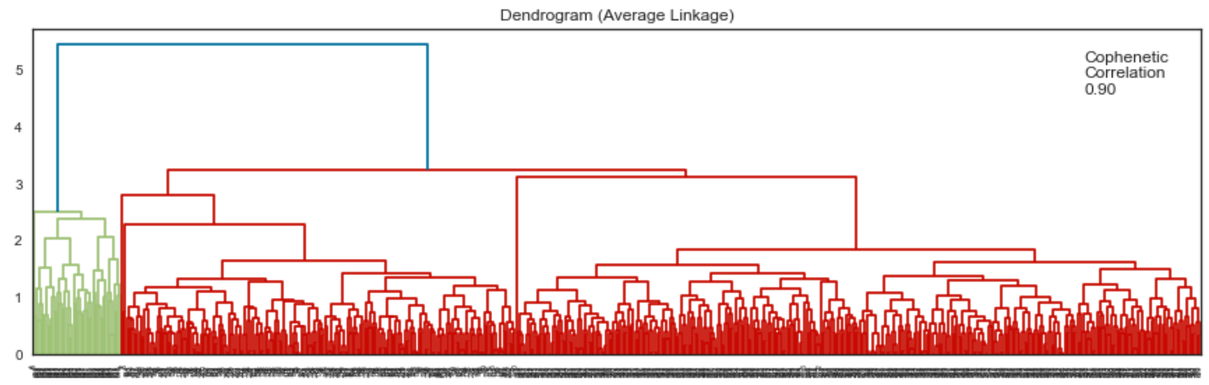
In [47]:
```
%%time
# list of linkage methods
linkage_methods = ["single", "complete", "average", "centroid", "ward", "weig

# lists to save results of cophenetic correlation calculation
compare_cols = ["Linkage", "Cophenetic Coefficient"]
compare = []

# to create a subplot image
sns.set_style("white")
fig, axs = plt.subplots(len(linkage_methods), 1, figsize=(15, 30))

# We will enumerate through the list of linkage methods above
# For each linkage method, we will plot the dendrogram and calculate the coph
```

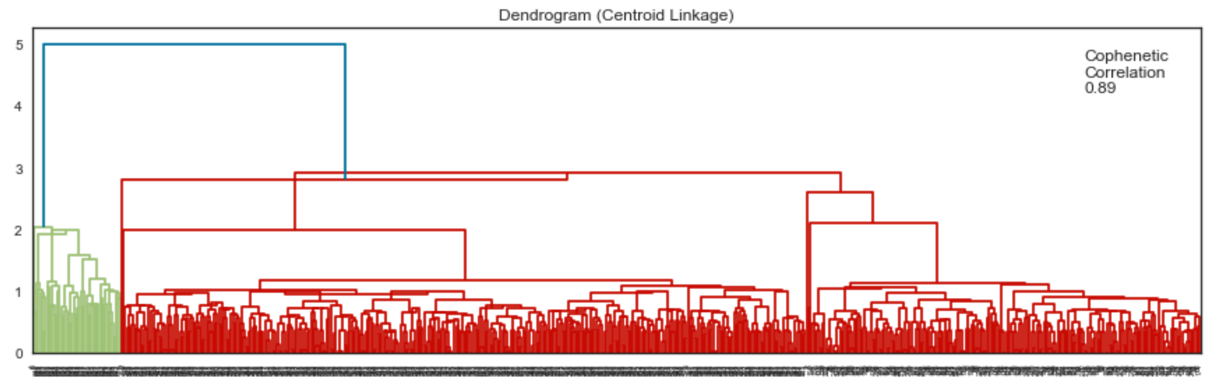
```
for i, method in enumerate(linkage_methods):
    Z = linkage(subset_scaled_df_h, metric="euclidean", method=method)

    dendrogram(Z, ax=axs[i])
    axs[i].set_title(f"Dendrogram ({method.capitalize()} Linkage)")

    coph_corr, coph_dist = cophenet(Z, pdist(subset_scaled_df_h))
    axs[i].annotate(
        f"Cophenetic\nCorrelation\n{coph_corr:0.2f}",
        (0.90, 0.80),
        xycoords="axes fraction",
    )

    compare.append([method, coph_corr])
```

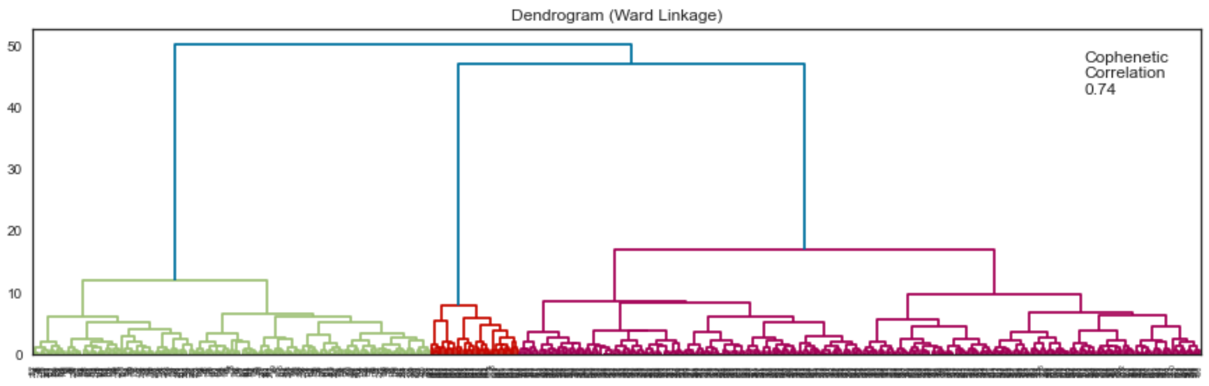
```
Wall time: 5.05 s
```

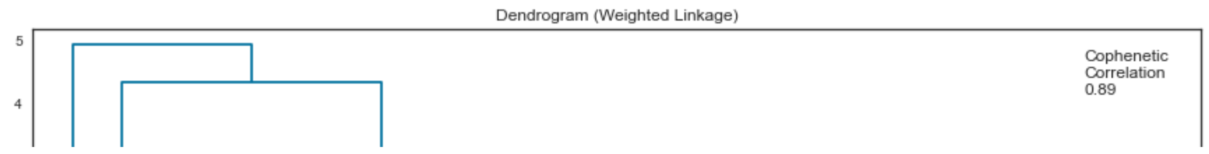

Dendrogram (Single Linkage)
Cophenetic Correlation 0.74
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00

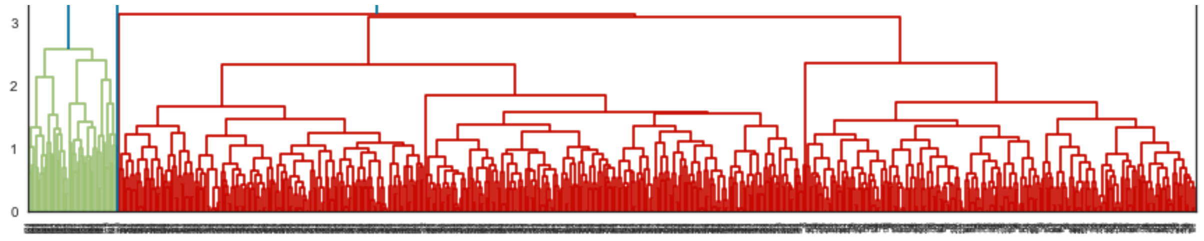

Dendrogram (Complete Linkage)
Cophenetic Correlation 0.86
8
7
6
5
4
3
2
1
0


Dendrogram (Average Linkage)
Cophenetic Correlation 0.90
5
4
3
2
1
0


Dendrogram (Centroid Linkage)
Cophenetic Correlation 0.89
5
4
3
2
1
0


Dendrogram (Ward Linkage)
Cophenetic Correlation 0.74
50
40
30
20
10
0


Dendrogram (Weighted Linkage)
Cophenetic Correlation 0.89
5
4



In [48]:
```
# let's create a dataframe to compare cophenetic correlations for each linkag
df_cc = pd.DataFrame(compare, columns=compare_cols)
df_cc
```

Out[48]:

	Linkage	Cophenetic Coefficient
0	single	0.739122
1	complete	0.859973
2	average	0.897708
3	centroid	0.893939
4	ward	0.741516
5	weighted	0.886175

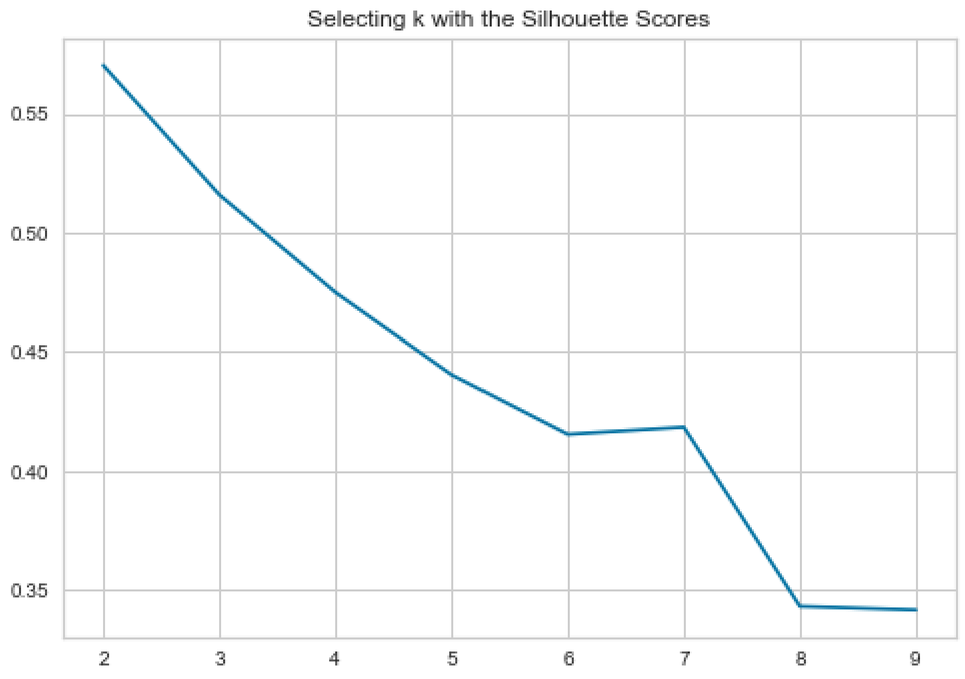
Observations:

- The cophenetic correlation is highest for average linkage methods and also is giving us distinct and separate clusters.
- 2 would be the appropriate number of clusters considering that the dendrogram shows a long distance (the y axis) for the two final groups and each of the subsequent groups has a drastically shorter distance.
- Let's also check the silhouette score and the cluster profiles.

Checking Silhouette scores

In [49]:
```
sns.set_style("whitegrid")
sil_score = []
cluster_list = list(range(2, 10))
for n_clusters in cluster_list:
    clusterer = AgglomerativeClustering(
        n_clusters=n_clusters, affinity="euclidean", linkage="average"
    )
    preds = clusterer.fit_predict((subset_scaled_df_h))
    # centers = clusterer.cluster_centers_
    score = silhouette_score(subset_scaled_df_h, preds)
    sil_score.append(score)
    print("For n_clusters = {}, the silhouette score is {})".format(n_cluster

plt.title("Selecting k with the Silhouette Scores")
plt.plot(cluster_list, sil_score)
plt.show()
```

```
For n_clusters = 2, the silhouette score is 0.5703183487340514)
For n_clusters = 3, the silhouette score is 0.515922432650965)
For n_clusters = 4, the silhouette score is 0.47495143595793504)
For n_clusters = 5, the silhouette score is 0.44039753024783956)
For n_clusters = 6, the silhouette score is 0.4153547954831452)
For n_clusters = 7, the silhouette score is 0.41837756746720256)
For n_clusters = 8, the silhouette score is 0.34306710358280806)
For n_clusters = 9, the silhouette score is 0.34154869328908927)
```



Observation:

- Silhouette score for k=2 is higher (0.57). So, we will choose 2 as value of k.

In [50]:
```
%%time

HCmodel = AgglomerativeClustering(n_clusters=2, affinity="euclidean", linkage
HCmodel.fit(subset_scaled_df_h)
```

```
Wall time: 15.9 ms
```

Out[50]:
```
AgglomerativeClustering(linkage='average')
```

In [51]:
```
# adding hierarchical cluster labels to the original and scaled dataframes

subset_scaled_df_h["HC_Clusters"] = HCmodel.labels_
subset_h["HC_Clusters"] = HCmodel.labels_
```

2.1 Hierarchical Cluster Profiling

In [52]:
```
cluster_profile_h = subset_h.groupby("HC_Clusters").mean()
```

In [53]:
```
cluster_profile_h["count_in_each_segments"] = (
    subset_h.groupby("HC_Clusters")["Avg_Credit_Limit"].count().values
)
```

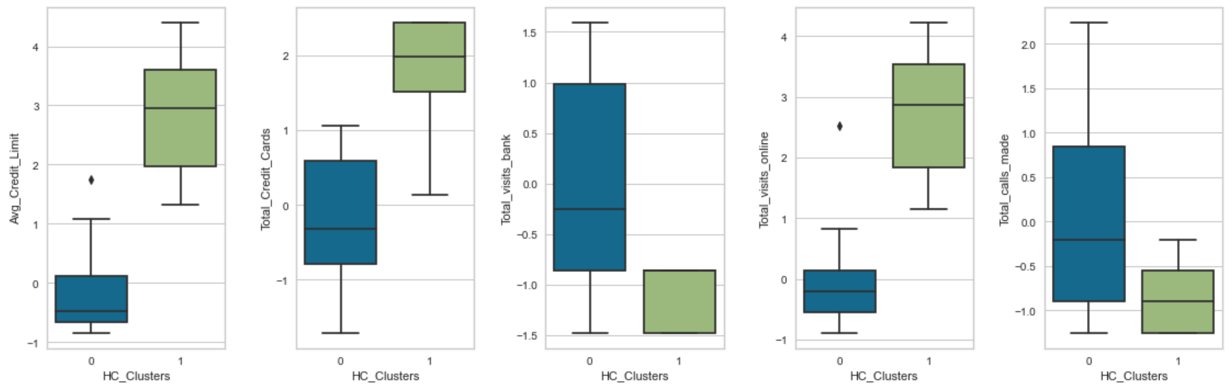
In [54]:
```
cluster_profile_h.style.highlight_max(color="lightgreen", axis=0)
```

Out[54]:

	Avg_Credit_Limit	Total_Credit_Cards	Total_visits_bank	Total_visits_online	Total
HC_Clusters					
0	25847.540984	4.375410	2.550820	1.926230	
1	141040.000000	8.740000	0.600000	10.900000	

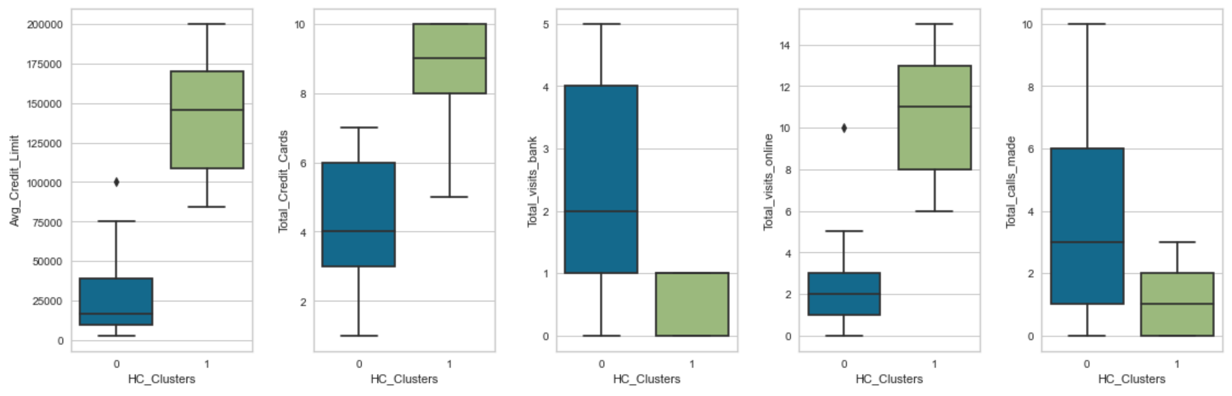
In [55]:
```python
sns.set_style("whitegrid")
fig, axes = plt.subplots(1, 5, figsize=(16, 6))
fig.suptitle("Boxplot of scaled numerical variables for each cluster", fontsi
counter = 0
for ii in range(5):
    sns.boxplot(
        ax=axes[ii],
        y=subset_scaled_df_h[num_col[counter]],
        x=subset_scaled_df_h["HC_Clusters"],
    )
    counter = counter + 1

fig.tight_layout(pad=2.0)
```

Boxplot of scaled numerical variables for each cluster



In [56]:
```python
fig, axes = plt.subplots(1, 5, figsize=(16, 6))
fig.suptitle("Boxplot of original numerical variables for each cluster", font
counter = 0
for ii in range(5):
    sns.boxplot(ax=axes[ii], y=subset_h[num_col[counter]], x=subset_h["HC_Clu
    counter = counter + 1

fig.tight_layout(pad=2.0)
```

Boxplot of original numerical variables for each cluster



Insights

- **Cluster 0**:
  - We can say these are our *Silver* customers, with moderate Average Credit Limit and Total Credit Cards.
  - With an average of approximately 26 thousands Credit Limit and total of 5 Credit Cards.
  - These are the customer that visit the bank the most (3 times yearly), on the other hands are our customers that almost do not use our online chanells.
  - Customer in these cluster have a mean of 4 calls made per customer yearly.
  - Therefore this cluster is composed of moderate Average Limit and preferable contact is to call customer service department.

- **Cluster 2**:
  - Our *Platinum* customers prefers the online channels using it almost 11 times per year.
  - They have the highest Average Credit Limit that goes arround 141040.
  - As expected are this cluster also have the highest total number of Credit Cards, with mean around 9.
  - Customers in these cluster almost never visit the bank or call it.
  - Hence, customers on this cluster prefers the Online contact and have the highest Average Credit Limit.

Comparing the clusters obtained from K-means clusters and Hierarchical clusters:

- Both technique took almost same time for execution, considering is a small dataset, but Hierarchical clustering use a high space as we need to store the similarity matrix in the RAM.

- Considering Elbow method and Silhouette score, 3 is the appropriate number of clusters for K-mean technique.

- Hierarchical clustering suggest 2 clusters considering the Cophenetic Coefficient and Silhouette score for euclidean distance and average linkage.

- K-means gave us more distinct clusters, grouping the customers in 3 distinct clusters:

  - *Cluster 0* : 386 customers with moderate Average Limit and in person visit preferable behavier.
  - *Cluster 1* : This cluster is composed of 224 customers with low Average Limit and the preferable contact is to call customer service department.
  - *Cluster 2* : Online chanell are preferable and have the highest Average Credit Limit, composed of 50 customers.

- Hierarchical gruped them in only 2:

  - *Cluster 0* : Composed of 610 customers with low and moderate Average Limit Credit, with almost same prefference between in person or call contact.
  - *Cluster 1* : Online chanell are preferable and have the highest Average Credit Limit, composed of 50 customers. Just like K-means cluster 2.

> As we could see, ***K-means*** identify 3 different segments in the existing customer, based on their spending patterns as well as past interaction with the bank better than the 2 clusters given by Hierarchical clusters. This 3 clusters can help us with better support services, considering their prefferable interaction with the bank.

Conclusion

- We have been able to build a clustering model that:

  a) The Bank can deploy this model to do customers segmentations.

  b) The bank can use it to find the key behaviers that differs one cluster from another.

  c) Based on it, the bank can take appropriate actions to build personalized campaingns and optimize support service for customers.

- Behaviors that differ clusters:

  a) Low, Moderate or High Average Credit Limit;

  b) How customers prefers to handle bank transactions (In person, By call or Online Channels).

- Armed with this segmentation, All Bank Life can not only craft better value propositions but also identify groups that are not well served by current offers.

Business Recommendations

- **Cluster 2**: Our Platinum customers have high total credit cards; they look for easy of use through online channels and for better benefits and fees.

  - We can use more Behavioral Segmentation like Spending habits, Usage of Rewards and track what they look on the online channels to understand more they needs.
  - Considering their high Credit Limited, they are looking for more sophisticate tools in our online channel like Card usage (day, week, month), Spending by category, Rewards points, Promotions etc. That will help them track they spending and manage their credit cards.
  - Also we can set online as the preferable communication channel, using e-mail text and app pop up to talk with our customers offering new products, campaigns and for satisfaction survey.
- **Cluster 1**: These are our customers that use online channel moderate and a high call contact and have a low Credit Limit.

- We need to tabulate the calls to understand what they are looking for, are the calls related to how to use the online channel?
- We should use our call center to tabulate the data capturing customers concerns, monitor complains and develop dashboards to detect addressable patterns that can help us to improve our support service, personalize campaigns , keep our customers and increase our customer base.
- As phone is the preferable communication channel, we can text and call this customers for personalized campaigns and satisfaction survey.

- **Cluster 0**: 59% of our customer base falls into this cluster, with moderate Credit Card Limit and in person contact preferable followed by calls.

  - We should use Demographic Segmentation as age, to understand their preferable contact model, Are they young or middle age customers?
  - We also should map their visits, what are they looking for? Deep understand our customers needs will help us to improve on our service and better target customers with our campaigns.
  - These are the king of customers we should reach by mail.
  - We can do a campaign to incentive them use our online channels, start with a tour through the online channel and interactive menus that helps them understand easily how it works.
- Therefore, redesign our online channels with more intuition bottoms, adding the most used functions on a more visibly way, like balance, due date and payments methods.

- Adding more sophisticate tools like Card usage (day, week, month), Spending by category, Rewards points
- Run periodic satisfaction survey in all different contact channels to understand what customers wants and what others Credit Card are offering can help us develop the best marketing campaign.
- Understand how satisfied are the customers and what we can improve to provide the best experience for them is the most important takeaway to differs our company.