AllLife Bank Customer Segmentation

Problem Statement:

AllLife Bank wants to focus on its credit card customer base in the next financial year. The penetration in the market can be improved, therefore, the Marketing team proposes to run personalized campaigns to target new customers as well as upsell to existing customers. The customers perceive the support services of the bank poorly, so the Operations team wants to upgrade the service delivery model, to ensure that customer queries are resolved faster.

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:

- 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)

Libraries

In [1]:
```
%load_ext nb_black
# Library to suppress warnings or deprecation notes
import warnings

warnings.filterwarnings("ignore")

# Libraries to help with reading and manipulating data
import numpy as np
import pandas as pd

# Libraries to help with data visualization
import matplotlib.pyplot as plt

%matplotlib inline
import seaborn as sns
```

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

# to compute distances
from scipy.spatial.distance import cdist
from scipy.spatial.distance import pdist

# to perform k-means clustering and compute silhouette scores
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score

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

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

# 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)
```

Read Dataset

In [2]:
```
data = pd.read_excel("Credit Card Customer Data.xlsx", sheet_name="Sheet1")

df = data.copy()
```

Data Info/Details

In [3]:
```
df.head()
```

Out[3]:

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

In [4]:
```
df.tail()
```

Out[4]:

	Sl_No	Customer Key	Avg_Credit_Limit	Total_Credit_Cards	Total_visits_bank	Total_visits_online
655	656	51108	99000	10	1	1(

	Sl_No	Customer Key	Avg_Credit_Limit	Total_Credit_Cards	Total_visits_bank	Total_visits_online
656	657	60732	84000	10	1	13
657	658	53834	145000	8	1	9
658	659	80655	172000	10	1	15
659	660	80150	167000	9	0	12

In [5]:
```python
np.random.seed(2)
df.sample(10)
```

Out[5]:

	Sl_No	Customer Key	Avg_Credit_Limit	Total_Credit_Cards	Total_visits_bank	Total_visits_online
276	277	36340	15000	4	5	2
315	316	45673	19000	4	3	1
198	199	15546	19000	1	1	4
268	269	97109	17000	6	5	0
203	204	56624	9000	2	0	3
340	341	69028	7000	6	3	1
183	184	86410	16000	1	2	5
239	240	14263	16000	5	2	0
612	613	94391	157000	9	1	14
37	38	74126	17000	2	0	4

In [6]:
```python
print(f"There are {df.shape[0]} rows and {df.shape[1]} columns.")
```

```
There are 660 rows and 7 columns.
```

In [7]:
```python
df[data.duplicated()].count()
```

Out[7]:
```
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 [8]:
```python
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
```

In [9]:
```
df.isnull().sum()
```

Out[9]:
```
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 [10]:
```
df.describe().T
```

Out[10]:

	count	mean	std	min	25%	50%	75%	
Sl_No	660.0	330.500000	190.669872	1.0	165.75	330.5	495.25	
Customer Key	660.0	55141.443939	25627.772200	11265.0	33825.25	53874.5	77202.50	9
Avg_Credit_Limit	660.0	34574.242424	37625.487804	3000.0	10000.00	18000.0	48000.00	20
Total_Credit_Cards	660.0	4.706061	2.167835	1.0	3.00	5.0	6.00	
Total_visits_bank	660.0	2.403030	1.631813	0.0	1.00	2.0	4.00	
Total_visits_online	660.0	2.606061	2.935724	0.0	1.00	2.0	4.00	
Total_calls_made	660.0	3.583333	2.865317	0.0	1.00	3.0	5.00	

EDA

Univariate

In [11]:
```
# Making a list of all categorical variables
cat_col = [
    "Sl_No",
    "Customer Key",
    "Avg_Credit_Limit",
    "Total_Credit_Cards",
    "Total_visits_bank",
    "Total_visits_online",
    "Total_calls_made",
]

# Printing number of count of each unique value in each column
for column in cat_col:
    print(data[column].value_counts())
    print("-" * 40)
```

```
660    1
226    1
224    1
223    1
222    1
      ..
440    1
439    1
438    1
437    1
1      1
Name: Sl_No, Length: 660, dtype: int64
----------------------------------------
47437    2
37252    2
97935    2
96929    2
50706    2
        ..
66706    1
72339    1
69965    1
85645    1
71681    1
Name: Customer Key, Length: 655, dtype: int64
----------------------------------------
8000      35
6000      31
9000      28
13000     28
10000     26
19000     26
7000      24
11000     24
18000     23
14000     23
17000     23
16000     22
5000      21
20000     20
12000     18
15000     17
36000     11
70000     10
38000      8
50000      8
56000      7
39000      7
68000      7
52000      6
37000      6
34000      6
30000      6
74000      6
47000      6
48000      6
41000      6
60000      5
29000      5
26000      5
65000      5
31000      5
54000      4
51000      4
59000      4
```

```
73000      4
71000      4
49000      4
69000      4
64000      4
66000      4
33000      4
28000      3
67000      3
62000      3
100000     3
72000      3
61000      3
58000      3
44000      3
45000      3
46000      3
57000      3
40000      3
163000     2
84000      2
27000      2
32000      2
75000      2
42000      2
166000     2
156000     2
172000     2
195000     2
35000      2
63000      2
123000     1
171000     1
186000     1
157000     1
126000     1
121000     1
144000     1
146000     1
127000     1
200000     1
43000      1
114000     1
98000      1
178000     1
3000       1
136000     1
167000     1
132000     1
91000      1
94000      1
95000      1
187000     1
96000      1
97000      1
55000      1
99000      1
176000     1
184000     1
183000     1
145000     1
173000     1
131000     1
155000     1
108000     1
```

```
158000     1
25000      1
153000     1
111000     1
112000     1
106000     1
Name: Avg_Credit_Limit, dtype: int64
----------------------------------------
4     151
6     117
7     101
5      74
2      64
1      59
3      53
10     19
9      11
8      11
Name: Total_Credit_Cards, dtype: int64
----------------------------------------
2    158
1    112
3    100
0    100
5     98
4     92
Name: Total_visits_bank, dtype: int64
----------------------------------------
2     189
0     144
1     109
4      69
5      54
3      44
15     10
7       7
12      6
10      6
8       6
13      5
11      5
9       4
14      1
6       1
Name: Total_visits_online, dtype: int64
----------------------------------------
4     108
0      97
2      91
1      90
3      83
6      39
7      35
9      32
8      30
5      29
10     26
Name: Total_calls_made, dtype: int64
----------------------------------------
```

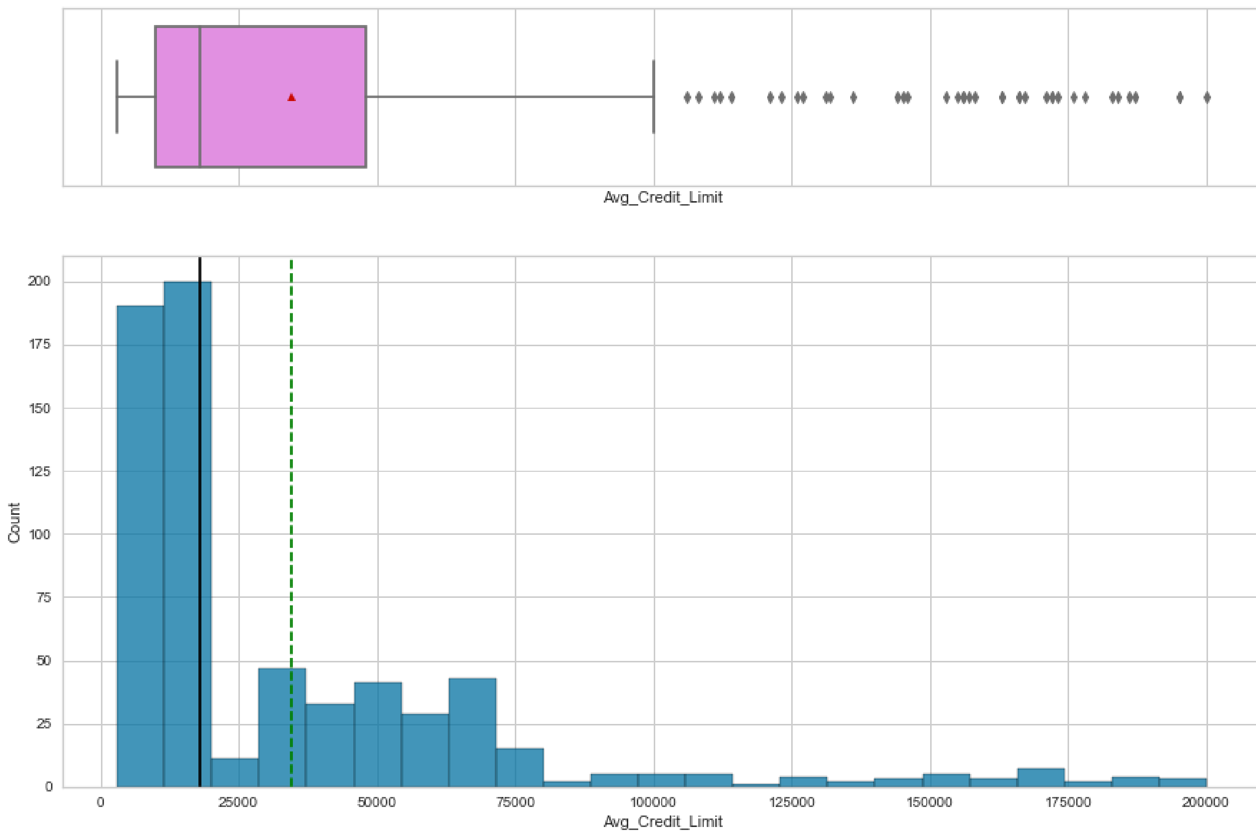
In [12]:

```
def histogram_boxplot(data, feature, figsize=(15, 10), kde=False, bins=None):
    """
    Boxplot and histogram combined
```

```
    data: dataframe
    feature: dataframe column
    figsize: size of figure (default (15,10))
    kde: whether to show the density curve (default False)
    bins: number of bins for histogram (default None)
    """
    f2, (ax_box2, ax_hist2) = plt.subplots(
        nrows=2,
        sharex=True,
        gridspec_kw={"height_ratios": (0.25, 0.75)},
        figsize=figsize,
    )
    sns.boxplot(data=data, x=feature, ax=ax_box2, showmeans=True, color="violet"
    sns.histplot(
        data=data, x=feature, kde=kde, ax=ax_hist2, bins=bins, palette="winter"
    ) if bins else sns.histplot(data=data, x=feature, kde=kde, ax=ax_hist2)
    ax_hist2.axvline(data[feature].mean(), color="green", linestyle="--")
    ax_hist2.axvline(data[feature].median(), color="black", linestyle="-")
```

In [13]:

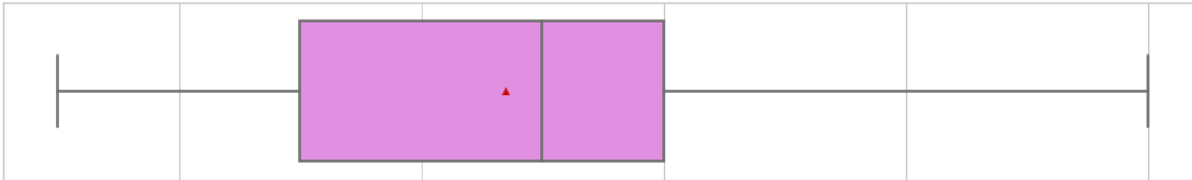
```
# Observation on Avg_Credit_Limit
histogram_boxplot(df, "Avg_Credit_Limit")
```

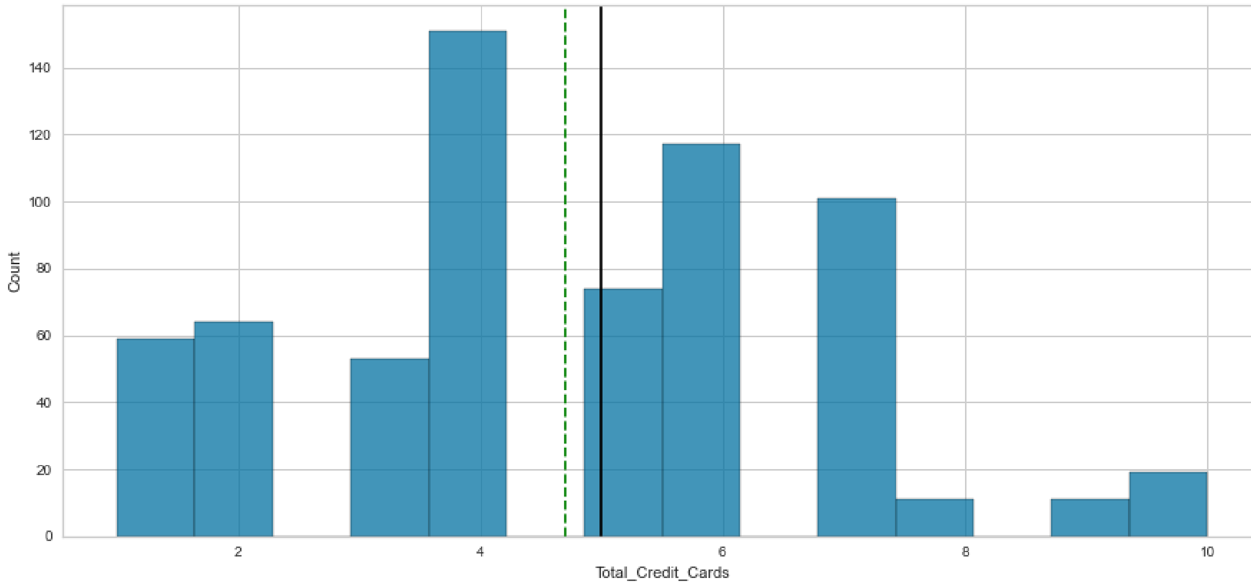


- Data is heavily right-skewed.
- Possibly 3 clusters.

In [14]:

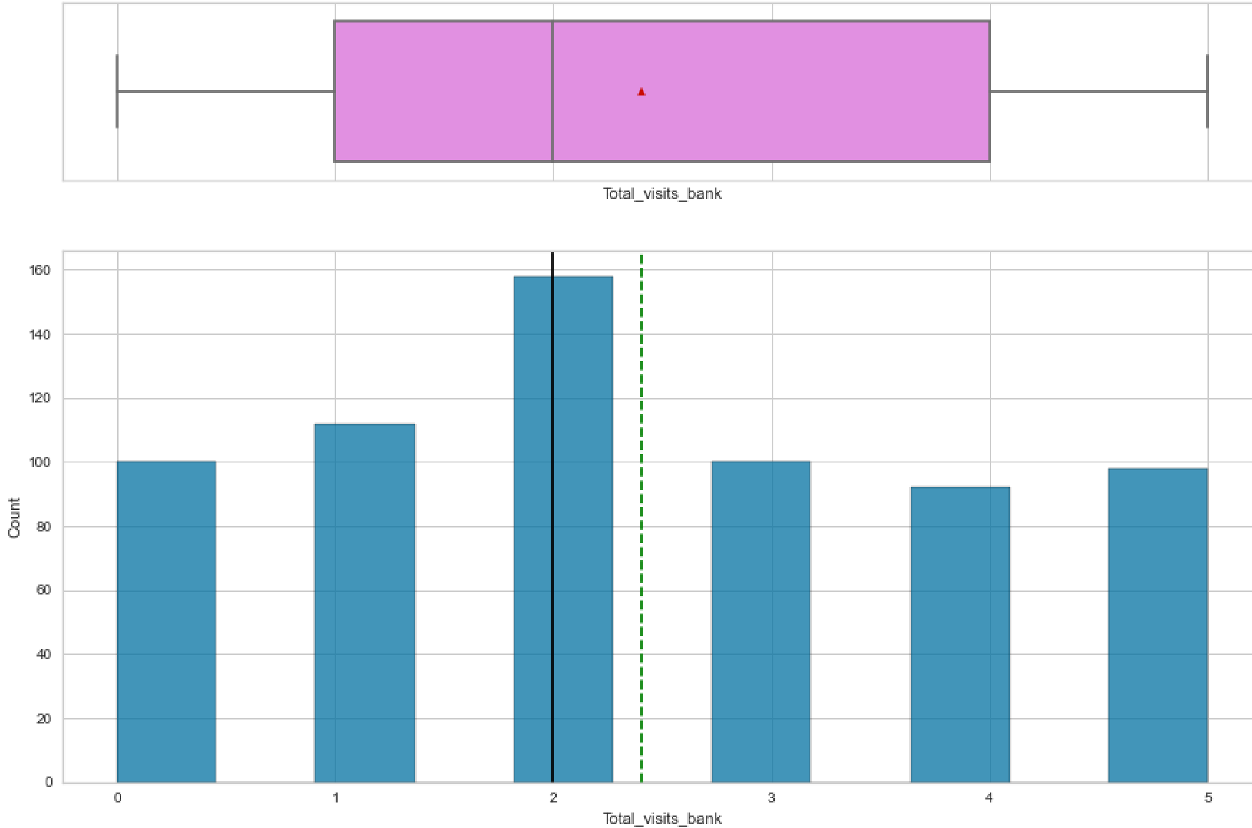
```
# Observation on Total_Credit_Cards
histogram_boxplot(df, "Total_Credit_Cards")
```





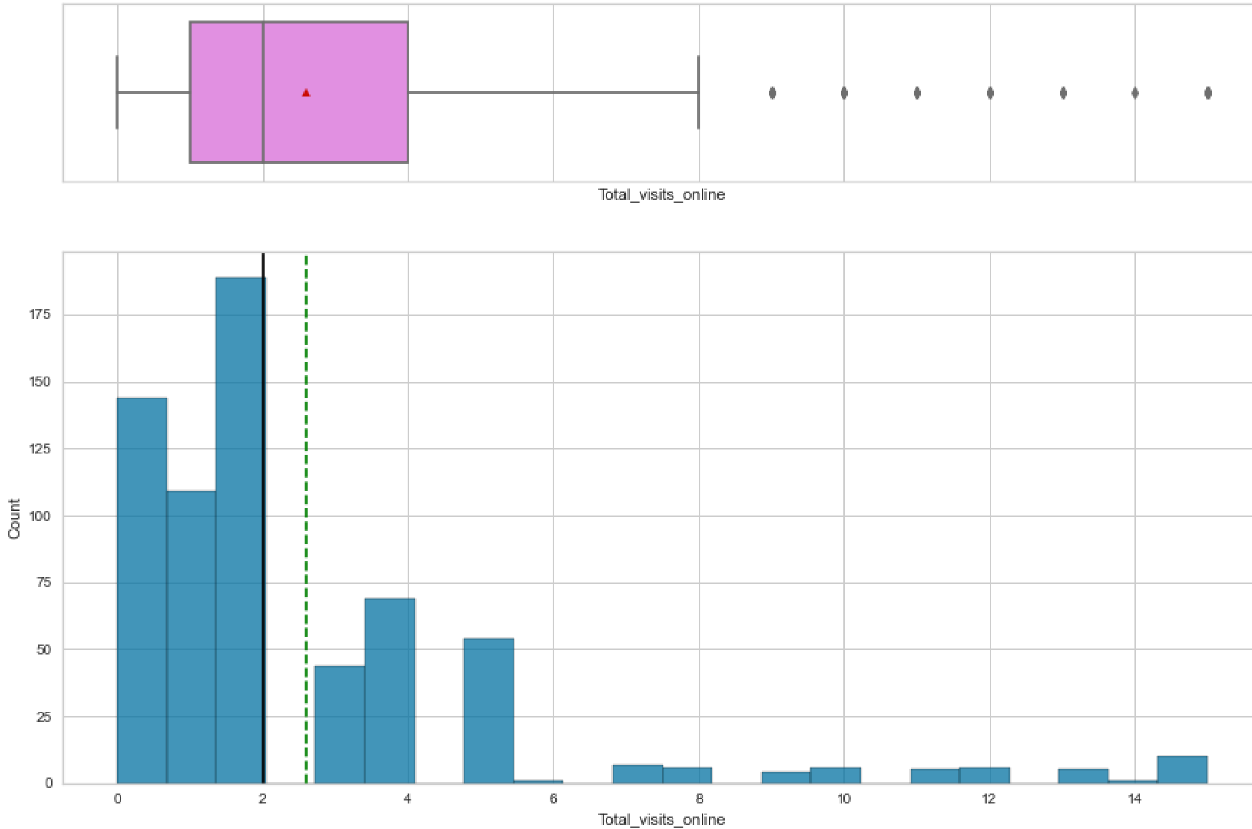
- There may be 5 clusters here.
- Data is slightly right-skewed.

In [15]:
```
# Observation on Total_visits_bank
histogram_boxplot(df, "Total_visits_bank")
```



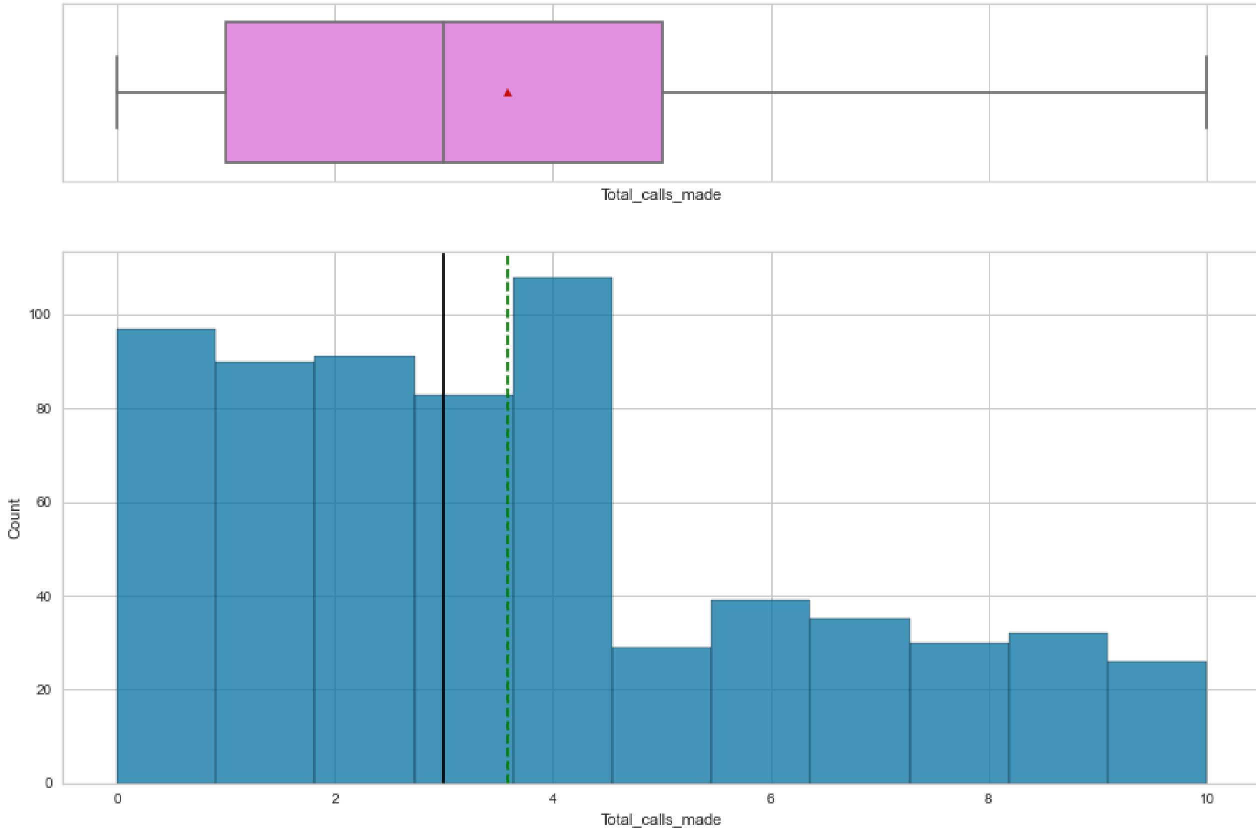
- Data is almost evenly distributed.

In [16]:
```
# Observation on Total_visits_online
histogram_boxplot(df, "Total_visits_online")
```



- Data is right-skewed

In [17]:
```
# Observation on Total_calls_made
histogram_boxplot(df, "Total_calls_made")
```

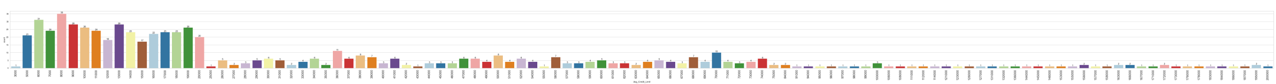


- Data is right-skewed

In [18]:
```
# 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 le
    """

    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:
```

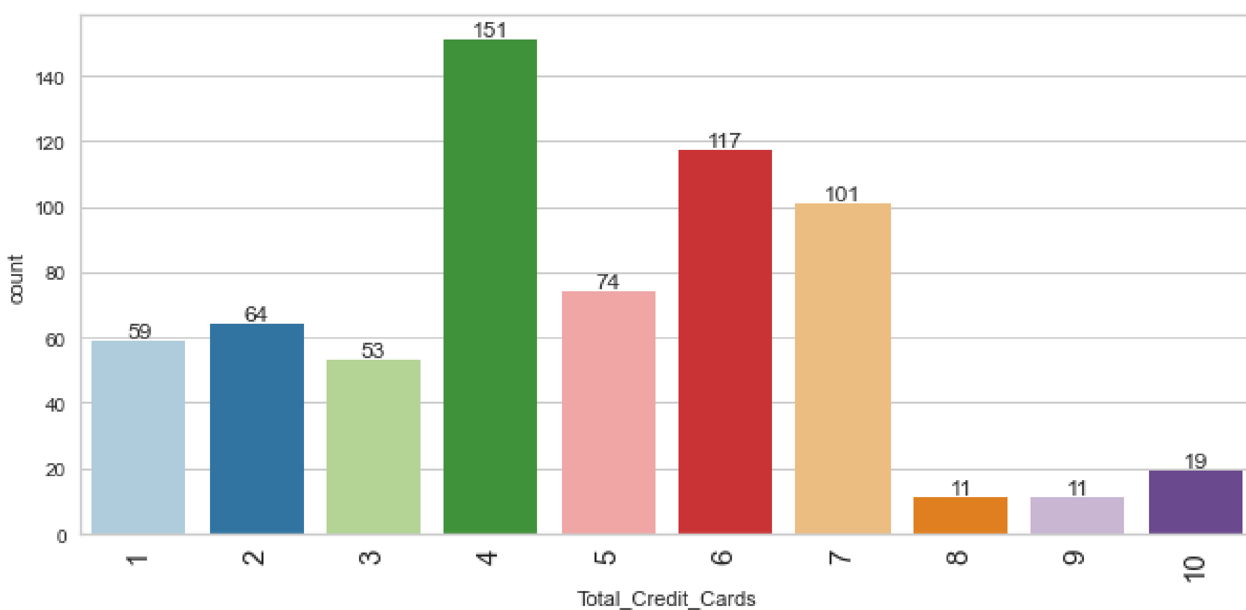
```
        if perc == True:
            label = "{:.1f}%".format(
                100 * p.get_height() / total
            )  # percentage of each class of the category
        else:
            label = p.get_height()  # count of each level of the category

        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 [19]:
```
# observations on Avg_Credit_Limit
labeled_barplot(df, "Avg_Credit_Limit")
```

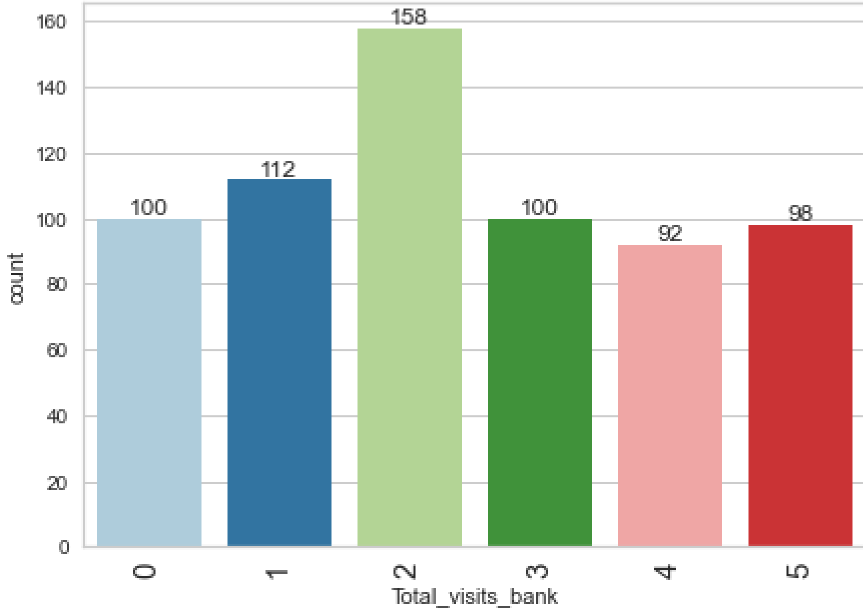


- I can see at least three clusters.

In [20]:
```
# observations on Total_Credit_Cards
labeled_barplot(df, "Total_Credit_Cards")
```

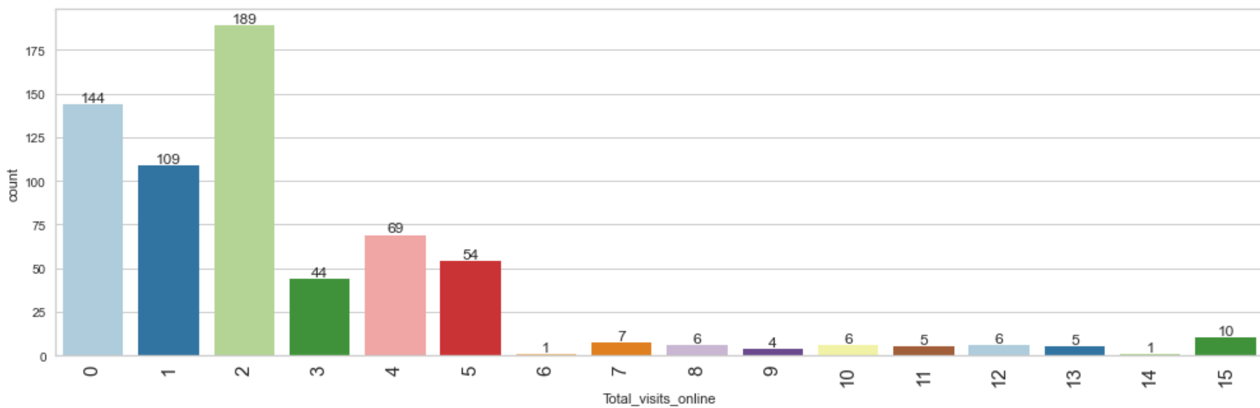


In [21]:
```
# observations on Total_visits_bank
labeled_barplot(df, "Total_visits_bank")
```



In [22]:

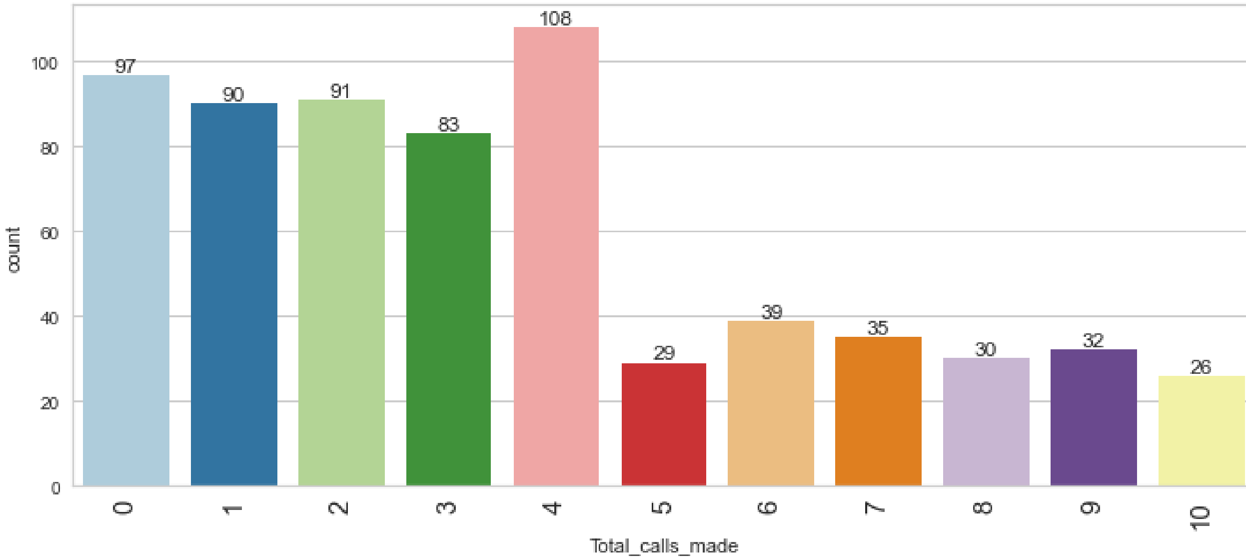
```
# observations on Total_visits_online
labeled_barplot(df, "Total_visits_online")
```



- There looks to be at least 3 clusters here.

In [23]:

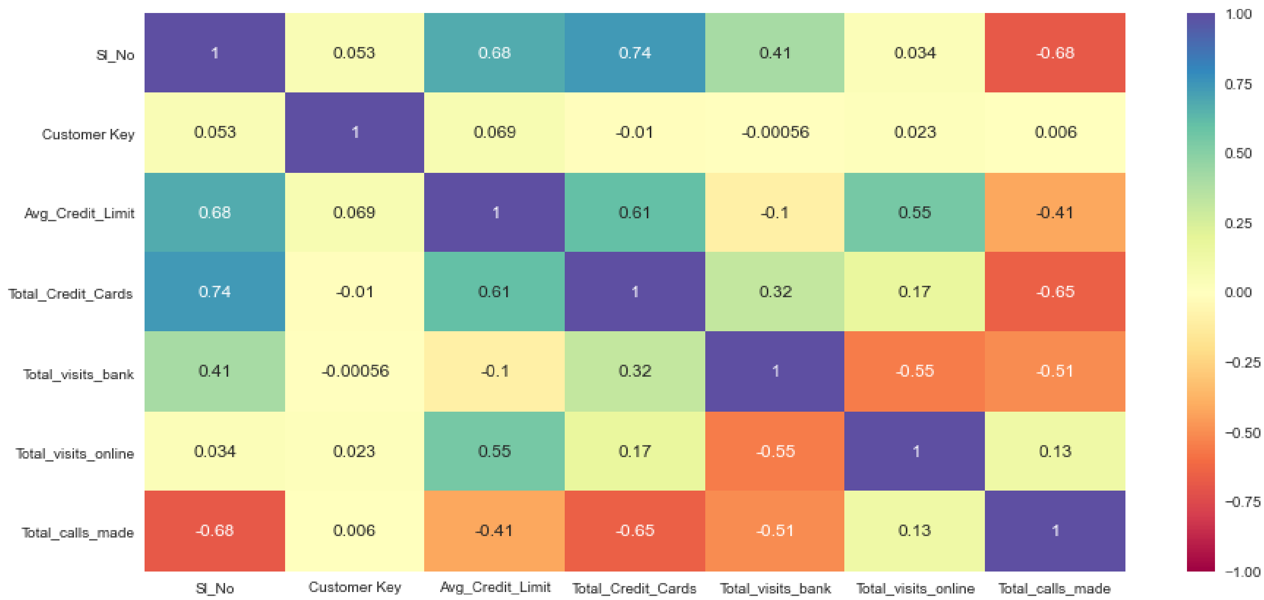
```
# observations on Total_calls_made
labeled_barplot(df, "Total_calls_made")
```



- Looks like most customer make 0-4 calls.
- At least 2 clusters.

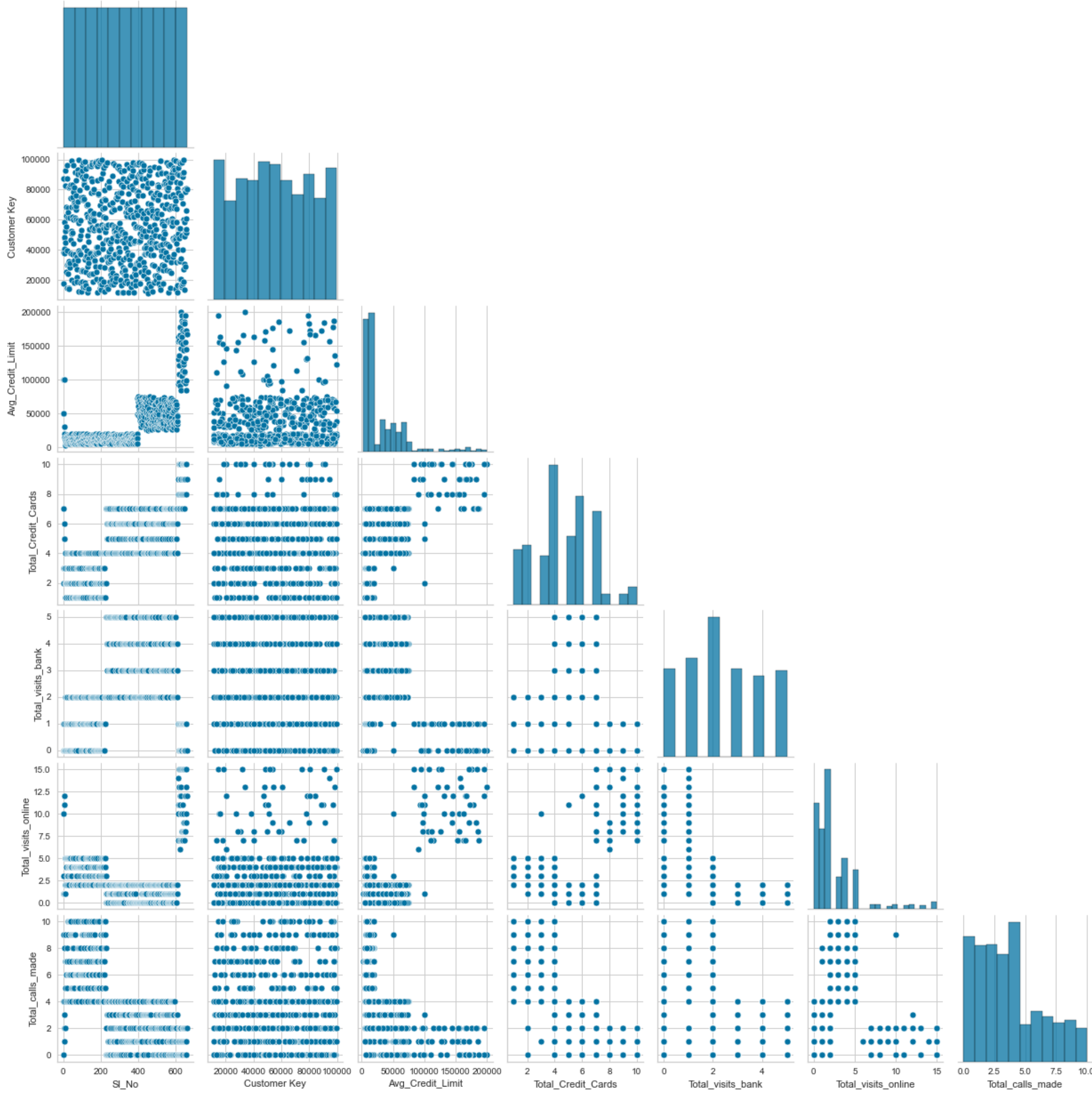
Bivariate

In [24]:
```
plt.figure(figsize=(15, 7))
sns.heatmap(df.corr(), annot=True, vmin=-1, vmax=1, cmap="Spectral")
plt.show()
```

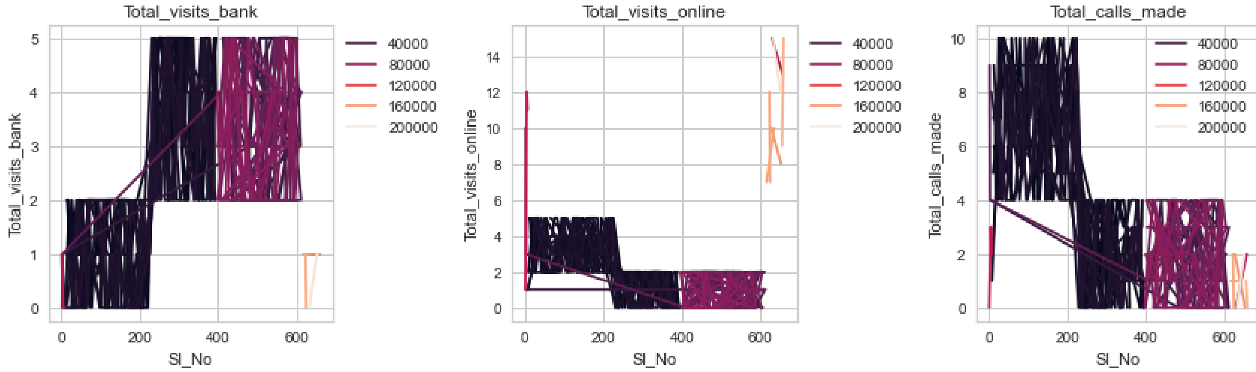


In [25]:
```
sns.pairplot(df, corner=True)
```

Out[25]:
```
<seaborn.axisgrid.PairGrid at 0x19d4a9d2430>
```



In [26]:
```
cols = df[
    ["Total_visits_bank", "Total_visits_online", "Total_calls_made"]
].columns.tolist()
plt.figure(figsize=(12, 10))
for i, variable in enumerate(cols):
    plt.subplot(3, 3, i + 1)
    palette = sns.color_palette("rocket", as_cmap=True)
    sns.lineplot(
        df["Sl_No"], df[variable], hue=data["Avg_Credit_Limit"], ci=0, palette=p
    )
    plt.tight_layout()
    plt.title(variable)
    plt.legend(bbox_to_anchor=(1, 1))
plt.show()
```



Bank Visits

- records ~5-225 with an average credit limit of <40,000 have a range of bank visits from 0-2
- records ~225-600 with an average credit limit of <40,000 have a range of bank visits from 2-5
- records ~400-600 with an average credit limit of ~80,000 have a range of bank visits from 2-5
- records ~625-660 with an average credit limit of ~160,000 and ~200,000 have a range of bank visits from 0-1
- records ~3 with an average credit limit of ~80,000 have a range of bank visits from 0-1 #### Online Visits
- records ~5-225 with an average credit limit of <40,000 have a range of online visits from 2-5
- records ~225-600 with an average credit limit of <40,000 have a range of online visits from 0-2
- records ~400-600 with an average credit limit of ~80,000 have a range of online visits from 0-2
- records ~625-660 with an average credit limit of ~160,000 and ~200,000 have a range of online visits from 7-15
- records ~3 with an average credit limit of ~80,000 have a range of online visits from 1-12 #### Calls Made
- records ~5-225 with an average credit limit of <40,000 have a range of 4-10 calls made
- records ~225-600 with an average credit limit of <40,000 have a range of 0-4 calls made
- records ~400-600 with an average credit limit of ~80,000 have a range of 0-4 calls made
- records ~625-660 with an average credit limit of ~160,000 and ~200,000 have a range of 0-2 calls made
- records ~3 with an average credit limit of ~80,000 have a range of 0-9 calls made

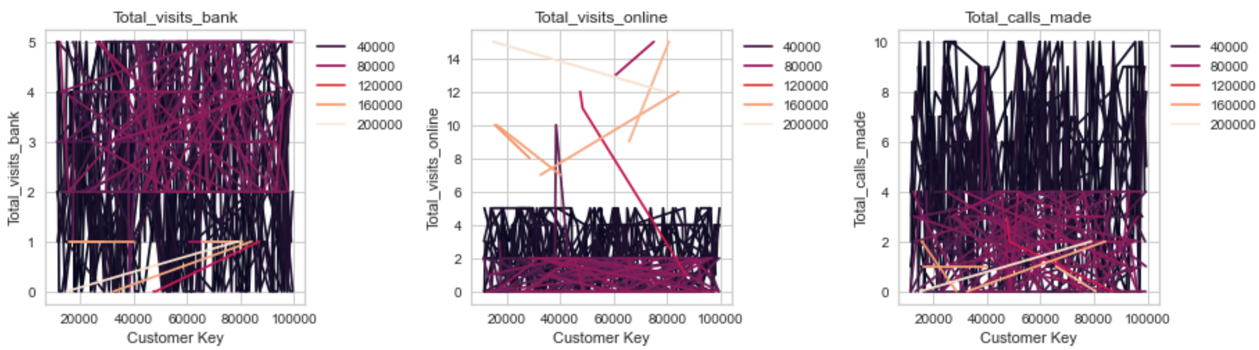
The records ~5-225 prefer (CM) making calls OVER (VB)/(VO) bank visits and online visits.

The records ~225-600 prefer (VB) visiting the bank OVER (VO)/(CM) online visits and making calls.

The records ~625-660 prefer (VO) online visits OVER (VB)/(CM) bank visits and making calls (upper credit limit).

The records ~3 prefer (VO)/(CM) online visits and making calls OVER (VB) bank visits.

In [27]:
```
cols = df[
    ["Total_visits_bank", "Total_visits_online", "Total_calls_made"]
].columns.tolist()
plt.figure(figsize=(12, 10))
for i, variable in enumerate(cols):
    plt.subplot(3, 3, i + 1)
    palette = sns.color_palette("rocket", as_cmap=True)
    sns.lineplot(
        df["Customer Key"],
        df[variable],
        hue=data["Avg_Credit_Limit"],
        ci=0,
        palette=palette,
    )
    plt.tight_layout()
    plt.title(variable)
    plt.legend(bbox_to_anchor=(1, 1))
plt.show()
```



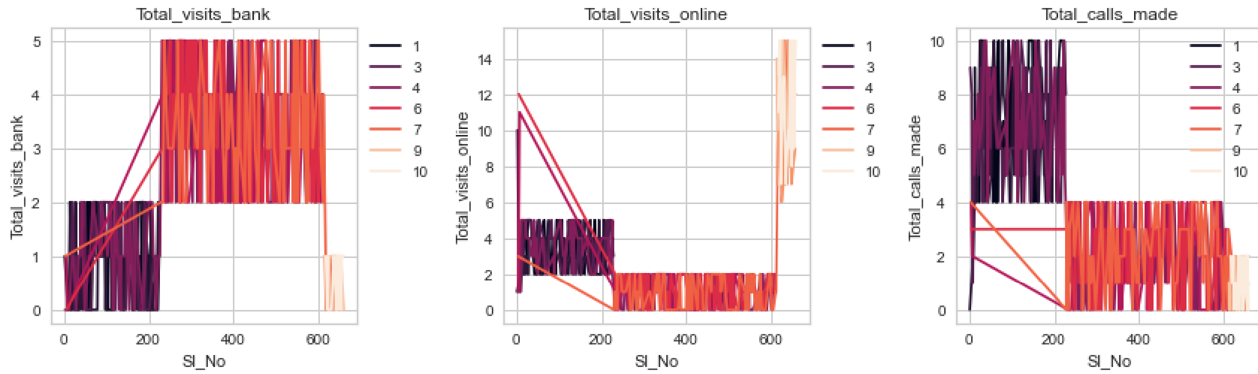
- There appears to be 3 Credit Limit groups.

<40,000 (VB) range: 0-5 (VO) range: 0-5 (CM) range: 0-10

~80,000 (VB) range: mostly 2-5 (VO) range: 0-2 (CM) range: mostly 0-4

~160,000> (VB) range: 0-1 (VO) range: 7-15 (CM) range: 0-2

In [28]:
```
cols = df[
    ["Total_visits_bank", "Total_visits_online", "Total_calls_made"]
].columns.tolist()
plt.figure(figsize=(12, 10))
for i, variable in enumerate(cols):
    plt.subplot(3, 3, i + 1)
    palette = sns.color_palette("rocket", as_cmap=True)
    sns.lineplot(
        df["Sl_No"], df[variable], hue=data["Total_Credit_Cards"], ci=0, palette
    )
    plt.tight_layout()
    plt.title(variable)
    plt.legend(bbox_to_anchor=(1, 1))
plt.show()
```



Bank Visits

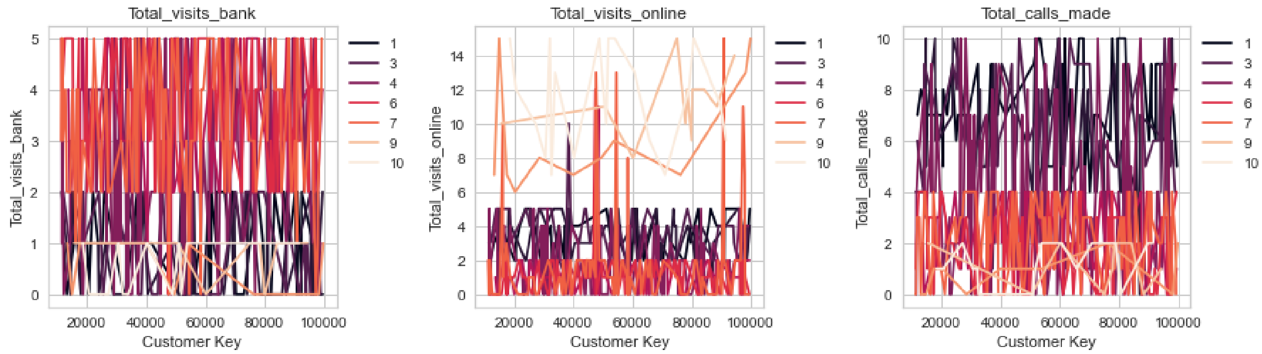
- People with 1-4 credit cards tend to have a range of bank visits from 0-2
- People with 5-7 credit cards have a range of bank visits from 2-5
- People with 8-10 credit cards have a range of bank visits from 0-1 #### Online Visits
- People with 1-4 credit cards tend to have a range of online visits from 2-5
- People with 5-7 credit cards have a range of online visits from 0-2
- People with 8-10 credit cards have a range of online visits from 6-15 #### Calls Made
- People with 1-4 credit cards tend to have a range of calls made from 4-10
- People with 5-7 credit cards have a range of calls made from 0-4
- People with 8-10 credit cards have a range of calls made from 0-2

1-4 credit cards prefer (CM) OVER (VB)/(VO)

5-7 credits cards prefer (VB) OVER (VO)/(CM)

8-10 credit cards prefer (VO) OVER (VB)/(CM)

In [29]:
```
cols = df[
    ["Total_visits_bank", "Total_visits_online", "Total_calls_made"]
].columns.tolist()
plt.figure(figsize=(12, 10))
for i, variable in enumerate(cols):
    plt.subplot(3, 3, i + 1)
    palette = sns.color_palette("rocket", as_cmap=True)
    sns.lineplot(
        df["Customer Key"],
        df[variable],
        hue=data["Total_Credit_Cards"],
        ci=0,
        palette=palette,
    )
    plt.tight_layout()
    plt.title(variable)
    plt.legend(bbox_to_anchor=(1, 1))
plt.show()
```



- This also shows that there are at least 3 Credit Card groups.

Preprocessing

In [30]:
```
df1 = df.copy()
```

In [31]:
```
df1.drop(["Customer Key"], inplace=True, axis=1)  # dropping unneeded column
```

In [32]:
```
num_col = df1.select_dtypes(
    include=np.number
).columns.tolist()  # creating a list of numerical columns
```

Scaling Data

In [33]:
```
scaler = StandardScaler()  # scaling method
subset = df1[num_col].copy()  # creating object with my num_col
subset_scaled = scaler.fit_transform(subset)  # applying transform on the subset
```

In [34]:
```
subset_scaled_df = pd.DataFrame(
    subset_scaled, columns=subset.columns
)  # creating a dataframe of the scaled columns
```

K-means Clustering

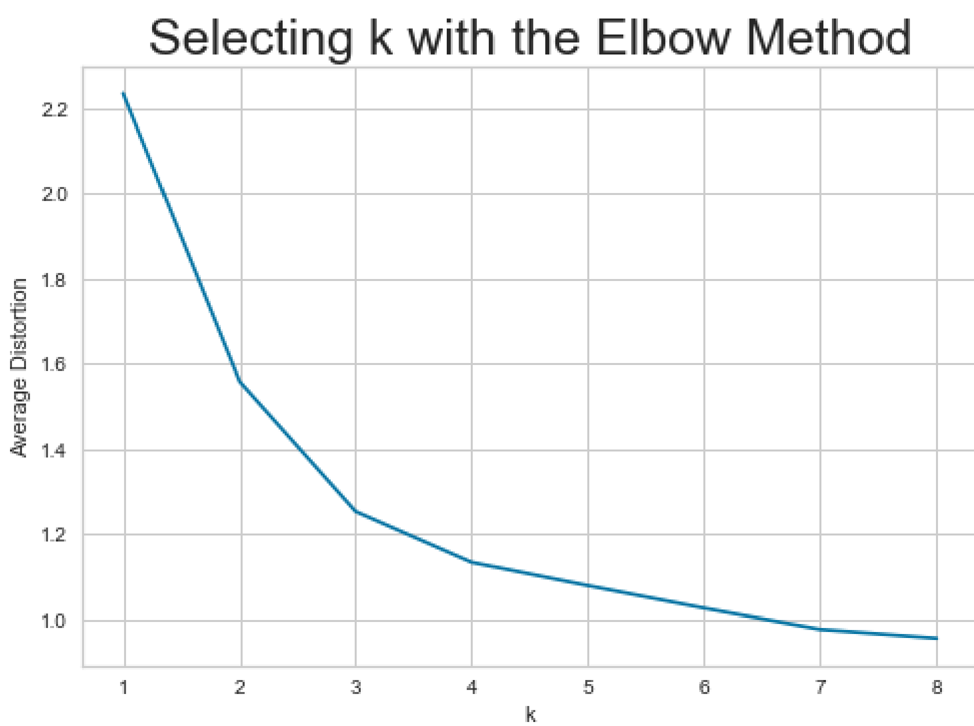
Elbow Curve

In [35]:
```
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"), ax
            )  # using euclidean distance
        )
        / 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", fontsize=25)
```

```
Number of Clusters: 1   Average Distortion: 2.2339960652784763
Number of Clusters: 2   Average Distortion: 1.5579826514081743
Number of Clusters: 3   Average Distortion: 1.2524825343997856
Number of Clusters: 4   Average Distortion: 1.1329238163414368
Number of Clusters: 5   Average Distortion: 1.078312981207808
Number of Clusters: 6   Average Distortion: 1.0257003359769887
Number of Clusters: 7   Average Distortion: 0.9757328334286154
Number of Clusters: 8   Average Distortion: 0.9550196798195167
```

Out[35]: Text(0.5, 1.0, 'Selecting k with the Elbow Method')



The best K value seems to be 3.

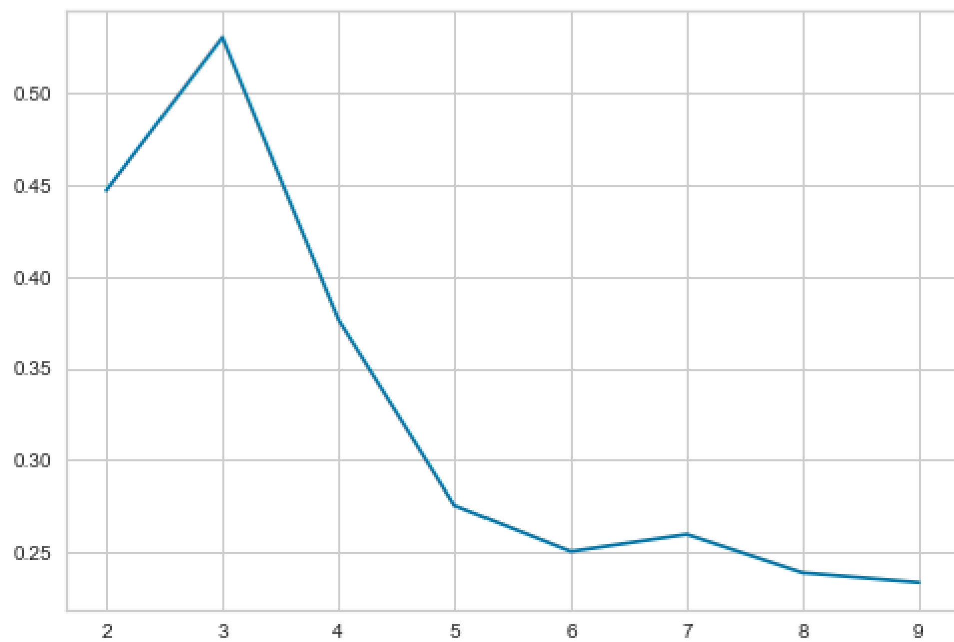
Silhouette Score

In [36]:
```
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))
```

```
    score = silhouette_score(subset_scaled_df, preds)
    sil_score.append(score)
    print("For n_clusters = {}, silhouette score is {}".format(n_clusters, score

plt.plot(cluster_list, sil_score)
```

```
For n_clusters = 2, silhouette score is 0.4470141077035807
For n_clusters = 3, silhouette score is 0.5304536180389302
For n_clusters = 4, silhouette score is 0.37663553573943503
For n_clusters = 5, silhouette score is 0.2754399426440398
For n_clusters = 6, silhouette score is 0.2503796759039485
For n_clusters = 7, silhouette score is 0.25971170197507565
For n_clusters = 8, silhouette score is 0.23870362057720315
For n_clusters = 9, silhouette score is 0.23339313591742275
```

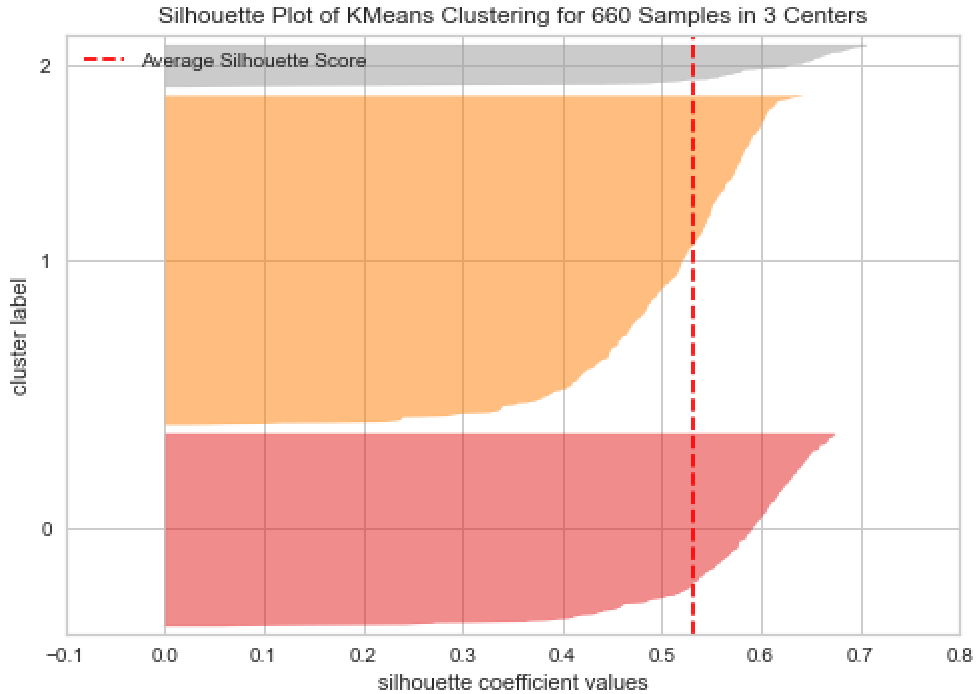
Out[36]: [<matplotlib.lines.Line2D at 0x19d4a9d2400>]



3 clusters have the highest silhouette score of 0.53

Looking at average silhouette

In [37]:
```
visualizer = SilhouetteVisualizer(KMeans(3, random_state=1))
visualizer.fit(subset_scaled_df)
visualizer.show()
```



Out[37]:
```
<AxesSubplot:title={'center':'Silhouette Plot of KMeans Clustering for 660 Sampl
es in 3 Centers'}, xlabel='silhouette coefficient values', ylabel='cluster labe
l'>
```

All 3 cluster reach the average silhouette score

Going to use 3 as the correct number of clusters as it is the elbow and has the highest silhouette score of 0.53

In [38]:
```
kmeans = KMeans(n_clusters=3, random_state=0)
kmeans.fit(subset_scaled_df)  # fitting 3 clusters to the dataframe
```

Out[38]:
```
KMeans(n_clusters=3, random_state=0)
```

In [39]:
```
df1["K_means_segments"] = kmeans.labels_  # adding cluster labels on df1
```

Cluster Profiling

In [40]:
```
cluster_profile = df1.groupby("K_means_segments").mean()
```

In [41]:
```
cluster_profile["count_in_each_segment"] = (
    df1.groupby("K_means_segments")["Avg_Credit_Limit"].count().values
)
```

In [42]:
```
cluster_profile.style.highlight_max(color="lightgreen", axis=0)
```

Out[42]:

K_means_segments	Sl_No	Avg_Credit_Limit	Total_Credit_Cards	Total_visits_bank	Total_vi
0	420.500000	33507.812500	5.518229	3.505208	
1	115.460177	12831.858407	2.433628	0.929204	
2	611.280000	141040.000000	8.740000	0.600000	

In [43]:
```
cluster_profile2 = df1.groupby("K_means_segments").min()
```

In [44]:
```
cluster_profile2["count_in_each_segment"] = (
    df1.groupby("K_means_segments")["Avg_Credit_Limit"].count().values
)
```

In [45]:
```
cluster_profile2.style.highlight_max(color="lightgreen", axis=0)
```

Out[45]:

K_means_segments	Sl_No	Avg_Credit_Limit	Total_Credit_Cards	Total_visits_bank	Total_visits_o
0	229	5000	4	2	
1	1	3000	1	0	
2	5	84000	5	0	

In [46]:
```
cluster_profile3 = df1.groupby("K_means_segments").max()
```

In [47]:
```
cluster_profile3["count_in_each_segment"] = (
    df1.groupby("K_means_segments")["Avg_Credit_Limit"].count().values
)
```

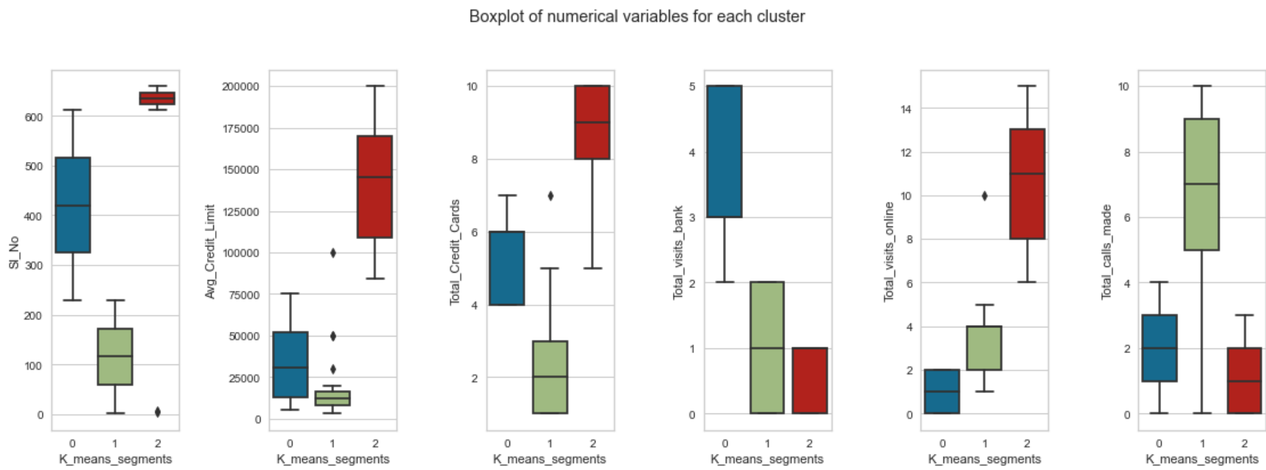
In [48]:
```
cluster_profile3.style.highlight_max(color="lightgreen", axis=0)
```

Out[48]:

K_means_segments	Sl_No	Avg_Credit_Limit	Total_Credit_Cards	Total_visits_bank	Total_visits_o
0	612	75000	7	5	
1	228	100000	7	2	
2	660	200000	10	1	

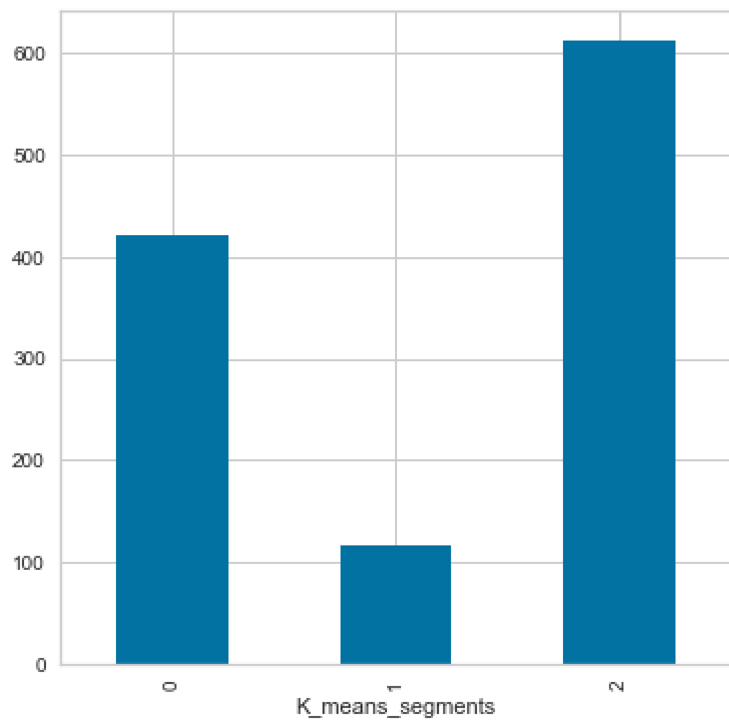
In [49]:
```
fig, axes = plt.subplots(1, 6, figsize=(16, 6))
fig.suptitle("Boxplot of numerical variables for each cluster")
counter = 0
for ii in range(6):
    sns.boxplot(ax=axes[ii], y=df1[num_col[counter]], x=df1["K_means_segments"])
    counter = counter + 1
```

```
fig.tight_layout(pad=2.0)
```



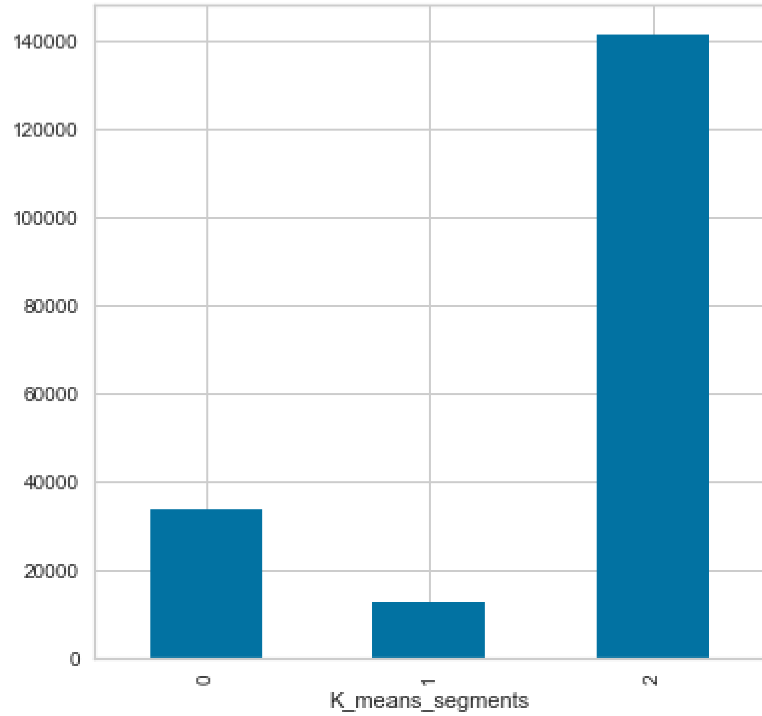
In [50]:
```
df1.groupby("K_means_segments")["Sl_No"].mean().plot.bar(figsize=(6, 6))
```

Out[50]: <AxesSubplot:xlabel='K_means_segments'>



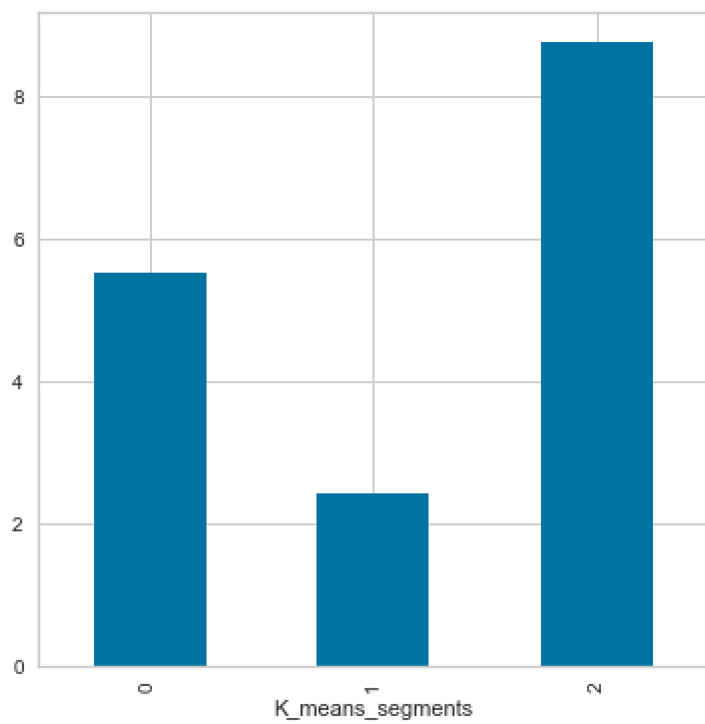
In [51]:
```
df1.groupby("K_means_segments")["Avg_Credit_Limit"].mean().plot.bar(figsize=(6,
```

Out[51]: <AxesSubplot:xlabel='K_means_segments'>



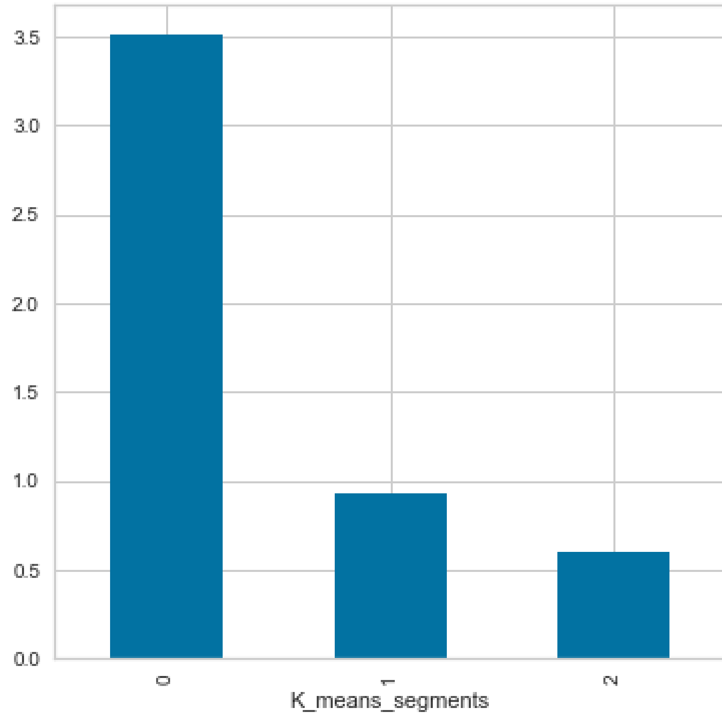
In [52]:
```
df1.groupby("K_means_segments")["Total_Credit_Cards"].mean().plot.bar(figsize=(6
```

Out[52]: <AxesSubplot:xlabel='K_means_segments'>



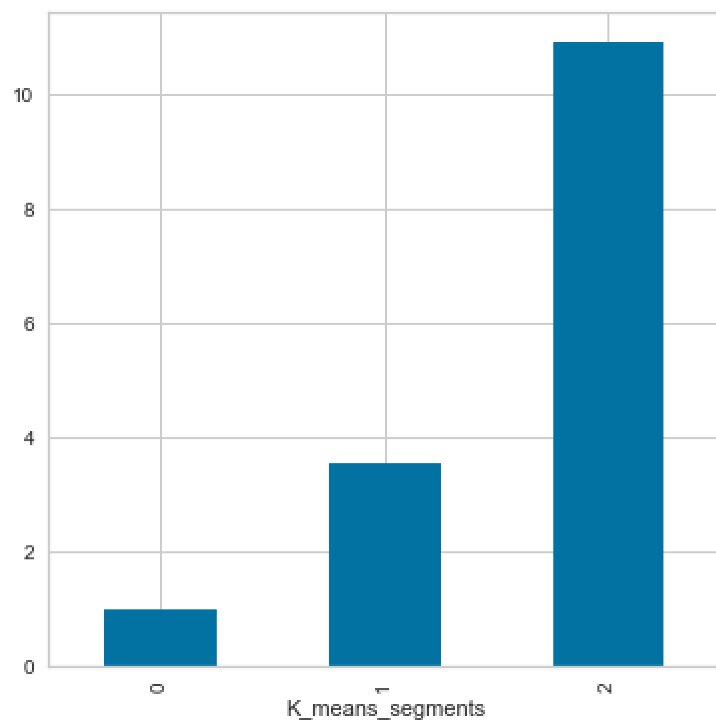
In [53]:
```
df1.groupby("K_means_segments")["Total_visits_bank"].mean().plot.bar(figsize=(6,
```

Out[53]: <AxesSubplot:xlabel='K_means_segments'>



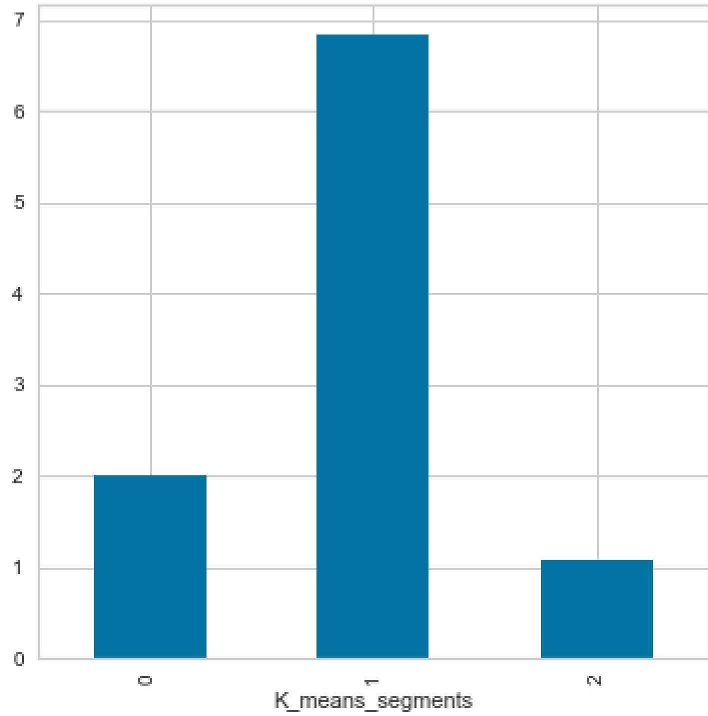
In [54]:
```
df1.groupby("K_means_segments")["Total_visits_online"].mean().plot.bar(figsize=(
```

Out[54]: <AxesSubplot:xlabel='K_means_segments'>



In [55]:
```
df1.groupby("K_means_segments")["Total_calls_made"].mean().plot.bar(figsize=(6,
```

Out[55]: <AxesSubplot:xlabel='K_means_segments'>



Insight (K-means clusters)

- **Cluster 0**: count of 384
  - Sl_No: range of 229-612. Average value of 420
  - Avg_Credit_Limit: range of 5,000-75,000. Average value of 33,508
  - Total_Credit_Cards: range of 4-7. Average value of 5
  - Total_visits_bank: range of 2-5. Average value of 3
  - Total_visits_online: range of 0-2. Average value of 1
  - Total_calls_made: range of 0-4. Average value of 2
- **Cluster 1**: count of 226
  - Sl_No: range of 1-228. Average value of 115
  - Avg_Credit_Limit: range of 3,000-100,000. Average value of 12,832
  - Total_Credit_Cards: range of 1-7. Average value of 2
  - Total_visits_bank: range of 0-2. Average value of 1
  - Total_visits_online: range of 1-10. Average value of 3
  - Total_calls_made: range of 0-10. Average value of 7
- **Cluster 2**: count of 50
  - Sl_No: range of 5-660. Average value of 611
  - Avg_Credit_Limit: range of 84,000-200,000. Average value of 141,040
  - Total_Credit_Cards: range of 5-10. Average value of 9
  - Total_visits_bank: range of 0-1. Average value of 1
  - Total_visits_online: range of 6-15. Average value of 11
  - Total_calls_made: range of 0-3. Average value of 1

Low credit limit = low number of credit cards = high number of calls made

Medium credit limit = medium number of credit cards = higher bank visits

High credit limit = high number of credit cards = high online visits

Hierarchical Clustering

In [56]:
```
distance_metrics = [
    "euclidean",
    "chebyshev",
    "mahalanobis",
    "cityblock",
]  # metrics to try

linkage_methods = [
    "single",
    "complete",
    "average",
    "weighted",
]  # linkage methods to try

high_cophenet_corr = 0
high_dm_lm = [0, 0]

for dm in distance_metrics:
    for lm in linkage_methods:
        Z = linkage(subset_scaled_df, metric=dm, method=lm)
        c, coph_dists = cophenet(Z, pdist(subset_scaled_df))
        print(
            "Cophenetic correlation for {} distance and {} linkage is {}.".forma
                dm.capitalize(), lm, c
            )
        )
        if high_cophenet_corr < c:
            high_cophenet_corr = c
            high_dm_lm[0] = dm
            high_dm_lm[1] = lm
```

```
Cophenetic correlation for Euclidean distance and single linkage is 0.7868636180
208888.
Cophenetic correlation for Euclidean distance and complete linkage is 0.88649539
66816567.
Cophenetic correlation for Euclidean distance and average linkage is 0.905290055
540969.
Cophenetic correlation for Euclidean distance and weighted linkage is 0.90336810
64765489.
Cophenetic correlation for Chebyshev distance and single linkage is 0.6924235963
979357.
Cophenetic correlation for Chebyshev distance and complete linkage is 0.89831272
55410409.
Cophenetic correlation for Chebyshev distance and average linkage is 0.904026592
243805.
Cophenetic correlation for Chebyshev distance and weighted linkage is 0.88591295
88415952.
Cophenetic correlation for Mahalanobis distance and single linkage is 0.52268900
39044828.
Cophenetic correlation for Mahalanobis distance and complete linkage is 0.584627
```

```
0884844046.
Cophenetic correlation for Mahalanobis distance and average linkage is 0.7633009
676646201.
Cophenetic correlation for Mahalanobis distance and weighted linkage is 0.813359
8885006322.
Cophenetic correlation for Cityblock distance and single linkage is 0.8155401698
293172.
Cophenetic correlation for Cityblock distance and complete linkage is 0.87874436
70247781.
Cophenetic correlation for Cityblock distance and average linkage is 0.902846279
5173166.
Cophenetic correlation for Cityblock distance and weighted linkage is 0.86895135
32895893.
```

The highest cophenetic correlation is 0.905 with the Euclidean distance and average linkage

In [57]:
```
linkage_methods = [
    "single",
    "complete",
    "average",
    "centroid",
    "ward",
    "weighted",
]  # more linkage methods

high_cophenet_corr = 0
high_dm_lm = [0, 0]

for lm in linkage_methods:
    Z = linkage(subset_scaled_df, metric="euclidean", method=lm)  # just on Eucl
    c, coph_dists = cophenet(Z, pdist(subset_scaled_df))
    print("Cophenetic correlation for {} linkage is {}.".format(lm, c))
    if high_cophenet_corr < c:
        high_cophenet_corr = c
        high_dm_lm[0] = "euclidean"
        high_dm_lm[1] = lm
```

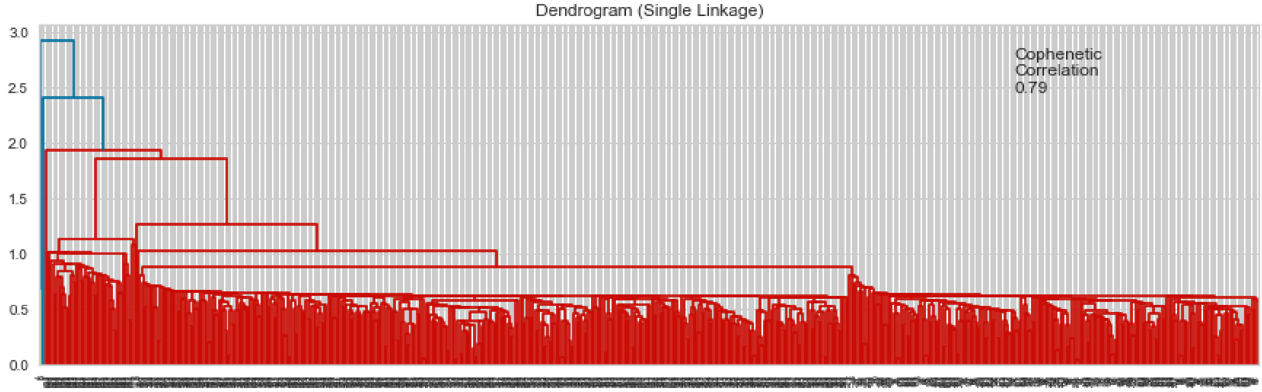
```
Cophenetic correlation for single linkage is 0.7868636180208888.
Cophenetic correlation for complete linkage is 0.8864953966816567.
Cophenetic correlation for average linkage is 0.905290055540969.
Cophenetic correlation for centroid linkage is 0.9020249185838797.
Cophenetic correlation for ward linkage is 0.7571185018058815.
Cophenetic correlation for weighted linkage is 0.9033681064765489.
```

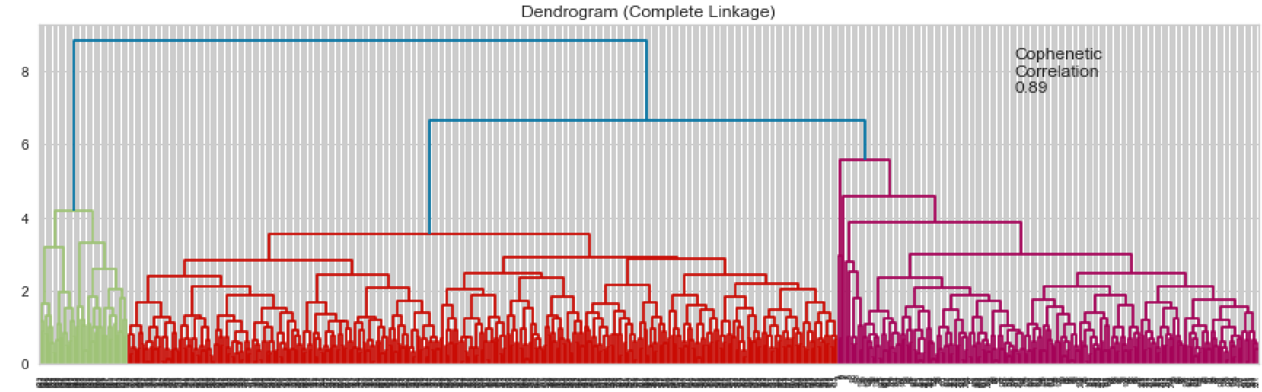
The highest is still with the average linkage method

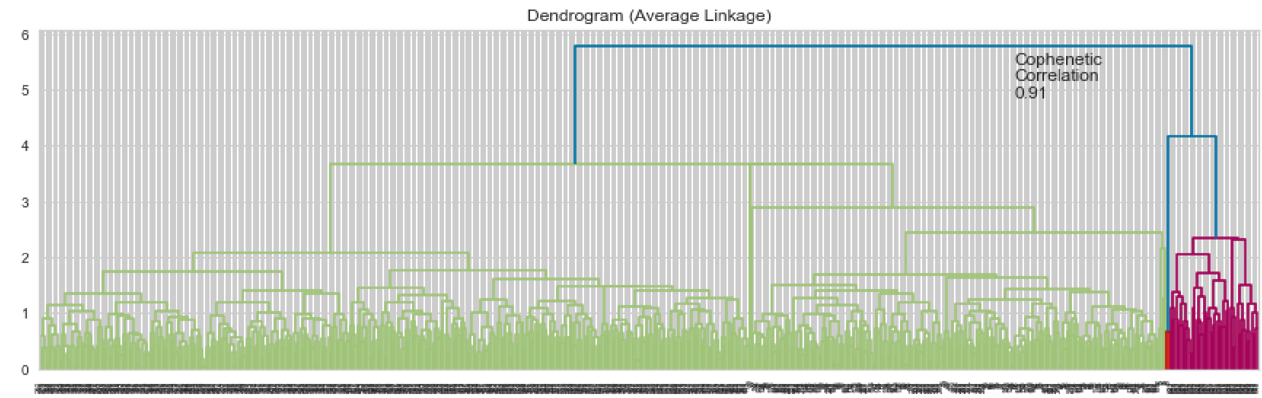
Dendrograms for the different linkage methods

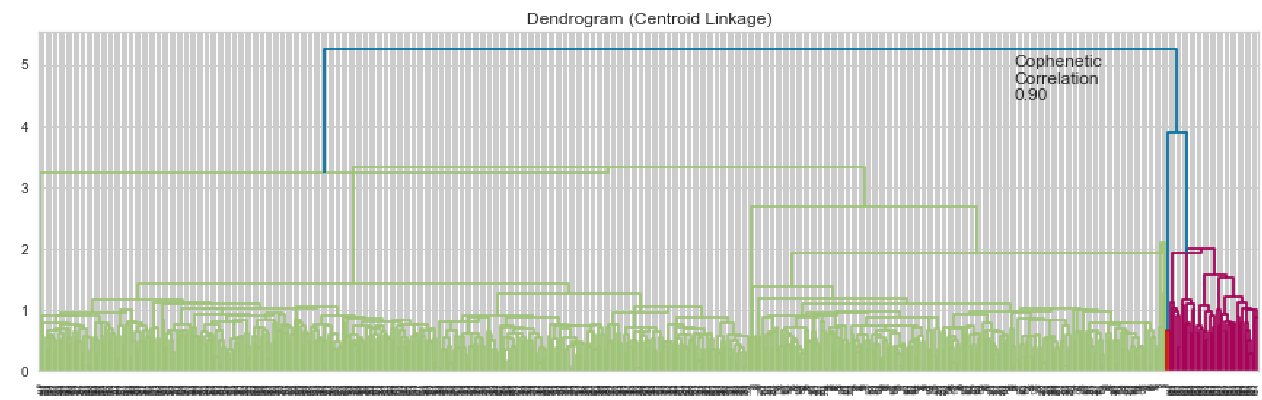
In [58]:
```
# list of linkage methods
linkage_methods = ["single", "complete", "average", "centroid", "ward", "weighte

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

# subplot image
fig, axs = plt.subplots(len(linkage_methods), 1, figsize=(15, 30))

# will pass through the list of linkage methods
# will plot the dendrogram and calculate the cophenetic correlation for each lin
```

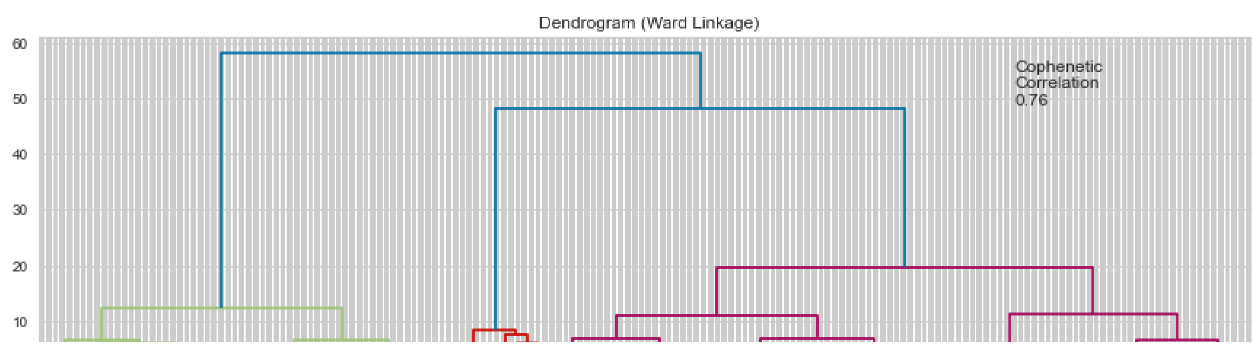
```
for i, method in enumerate(linkage_methods):
    Z = linkage(subset_scaled_df, metric="euclidean", method=method)

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

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

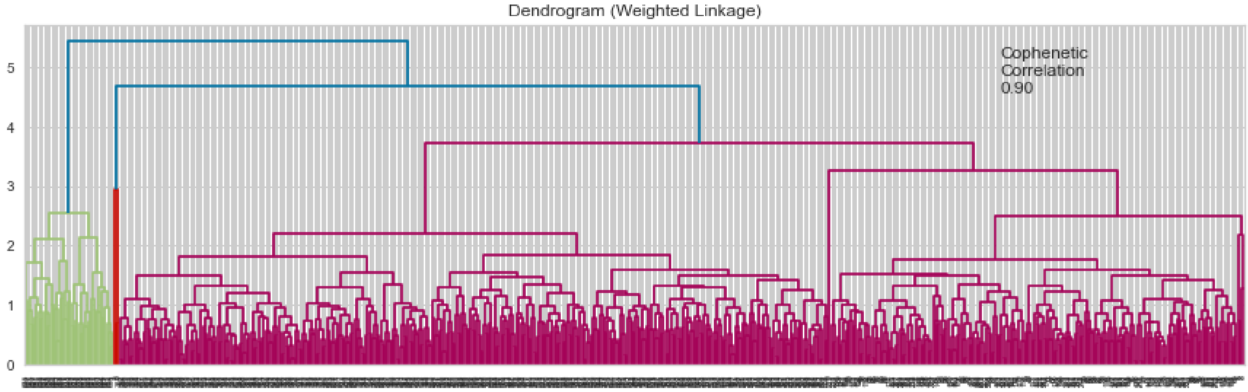

Dendrogram (Single Linkage)
Cophenetic
Correlation
0.79


Dendrogram (Complete Linkage)
Cophenetic
Correlation
0.89


Dendrogram (Average Linkage)
Cophenetic
Correlation
0.91


Dendrogram (Centroid Linkage)
Cophenetic
Correlation
0.90


Dendrogram (Ward Linkage)
Cophenetic
Correlation
0.76



Almost all dendrograms have 3 distinct clusters

From the dendrograms, I like how the Complete linkage graph looks.

- The 3 main cluster look to have a distance of 6 (higher than almost all other dengrograms).
- The cophenetic correlation is still pretty high at 0.89
- Each cluster has a decent amount of samples

Average linkage

- Distance is lower at 4
- One cluster has very few samples
- Cophenetic correlation is high though at 0.91

Ward linkage

- High distance of ~48
- lower correlation at 0.76
- Would like one cluster to have more samples

In [59]:
```
HCmodel = AgglomerativeClustering(
    n_clusters=3, affinity="euclidean", linkage="complete"
)
HCmodel.fit(subset_scaled_df)
```

Out[59]:
```
AgglomerativeClustering(linkage='complete', n_clusters=3)
```

In [60]:
```
subset_scaled_df["HC_Clusters"] = HCmodel.labels_
df1["HC_Clusters"] = HCmodel.labels_
```

Cluster Profiling

In [61]:
```
cluster_profile4 = df1.groupby("HC_Clusters").mean()
cluster_profile4.drop(["K_means_segments"], inplace=True, axis=1)
```

In [62]:
```
cluster_profile4["count_in_each_segments"] = (
    df1.groupby("HC_Clusters")["Avg_Credit_Limit"].count().values
)
```

In [63]:
```
cluster_profile4.style.highlight_max(color="lightgreen", axis=0)
```

Out[63]:

HC_Clusters	Sl_No	Avg_Credit_Limit	Total_Credit_Cards	Total_visits_bank	Total_visits_onl
0	114.500000	13596.491228	2.460526	0.921053	3.6052
1	636.500000	142750.000000	8.875000	0.625000	10.8750
2	420.500000	33507.812500	5.518229	3.505208	0.979

In [64]:
```
cluster_profile5 = df1.groupby("HC_Clusters").min()
cluster_profile5.drop(["K_means_segments"], inplace=True, axis=1)
```

In [65]:
```
cluster_profile5["count_in_each_segments"] = (
    df1.groupby("HC_Clusters")["Avg_Credit_Limit"].count().values
)
```

In [66]:
```
cluster_profile5.style.highlight_max(color="lightgreen", axis=0)
```

Out[66]:

HC_Clusters	Sl_No	Avg_Credit_Limit	Total_Credit_Cards	Total_visits_bank	Total_visits_online	T
0	1	3000	1	0	1	
1	613	84000	7	0	6	
2	229	5000	4	2	0	

In [67]:
```
cluster_profile6 = df1.groupby("HC_Clusters").max()
cluster_profile6.drop(["K_means_segments"], inplace=True, axis=1)
```

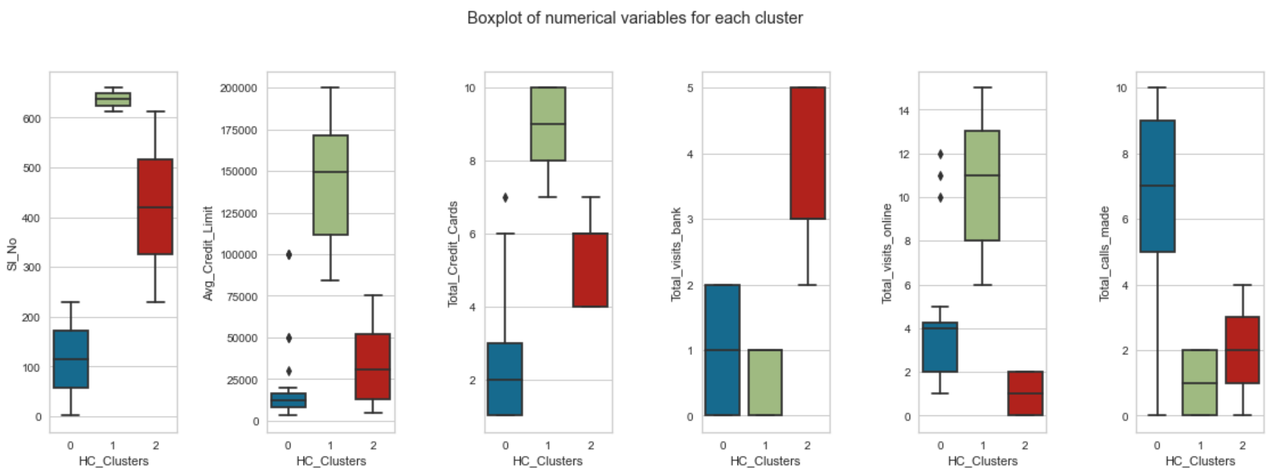
In [68]:
```
cluster_profile6["count_in_each_segments"] = (
    df1.groupby("HC_Clusters")["Avg_Credit_Limit"].count().values
)
```

In [69]:
```
cluster_profile6.style.highlight_max(color="lightgreen", axis=0)
```

Out[69]:

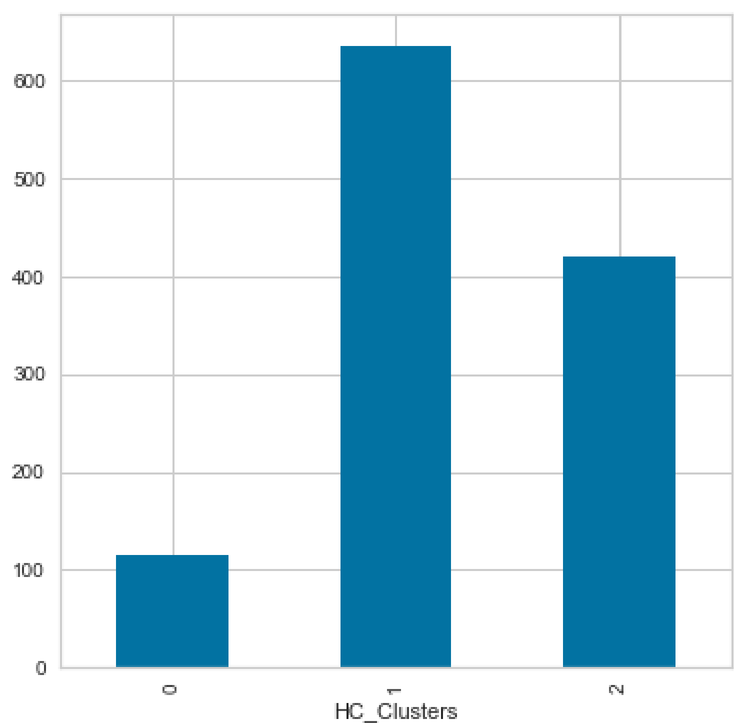
HC_Clusters	Sl_No	Avg_Credit_Limit	Total_Credit_Cards	Total_visits_bank	Total_visits_online	T
0	228	100000	7	2	12	
1	660	200000	10	1	15	
2	612	75000	7	5	2	

In [70]:
```
fig, axes = plt.subplots(1, 6, figsize=(16, 6))
fig.suptitle("Boxplot of numerical variables for each cluster")
counter = 0
for ii in range(6):
    sns.boxplot(ax=axes[ii], y=df1[num_col[counter]], x=df1["HC_Clusters"])
    counter = counter + 1

fig.tight_layout(pad=2.0)
```



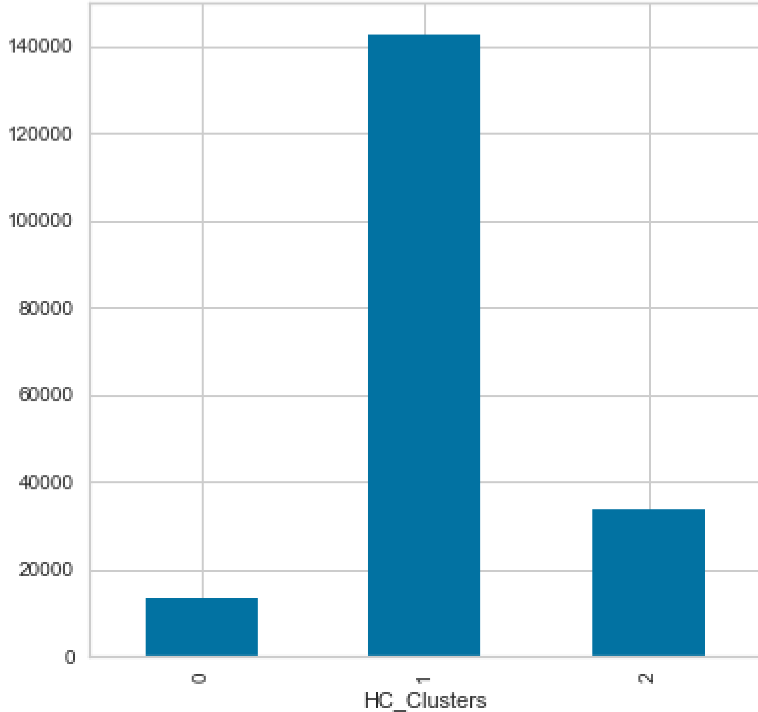
In [71]:
```
df1.groupby("HC_Clusters")["Sl_No"].mean().plot.bar(figsize=(6, 6))
```

Out[71]: <AxesSubplot:xlabel='HC_Clusters'>



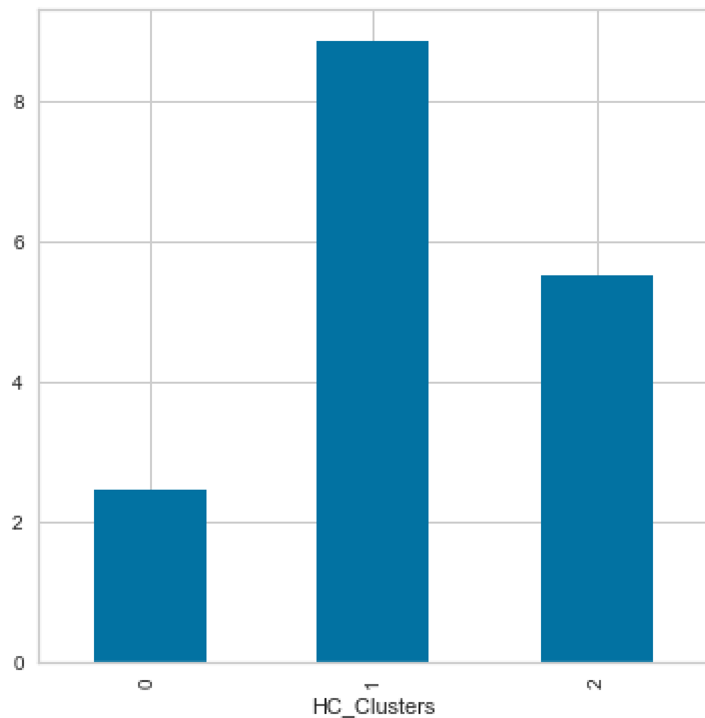
In [72]:
```
df1.groupby("HC_Clusters")["Avg_Credit_Limit"].mean().plot.bar(figsize=(6, 6))
```

Out[72]: <AxesSubplot:xlabel='HC_Clusters'>



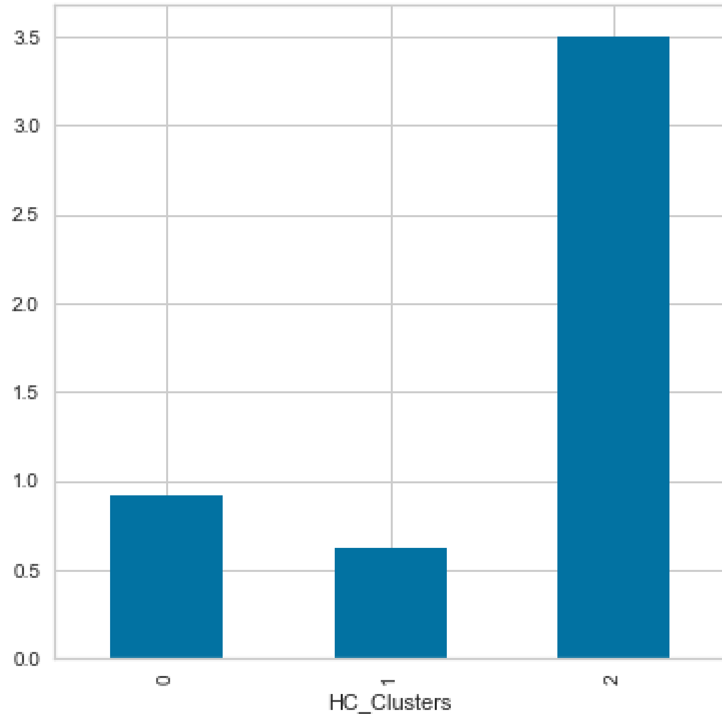
In [73]:
```
df1.groupby("HC_Clusters")["Total_Credit_Cards"].mean().plot.bar(figsize=(6, 6))
```

Out[73]: <AxesSubplot:xlabel='HC_Clusters'>



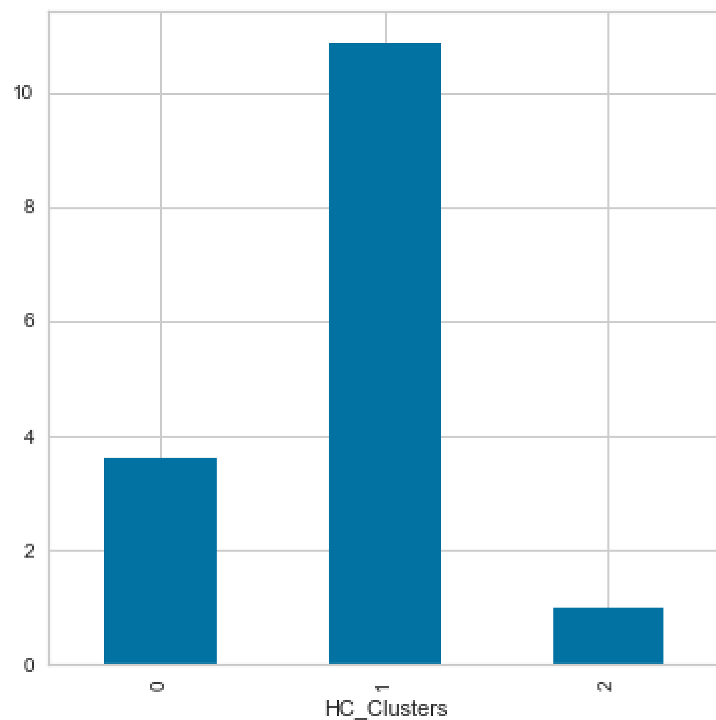
In [74]:
```
df1.groupby("HC_Clusters")["Total_visits_bank"].mean().plot.bar(figsize=(6, 6))
```

Out[74]: <AxesSubplot:xlabel='HC_Clusters'>



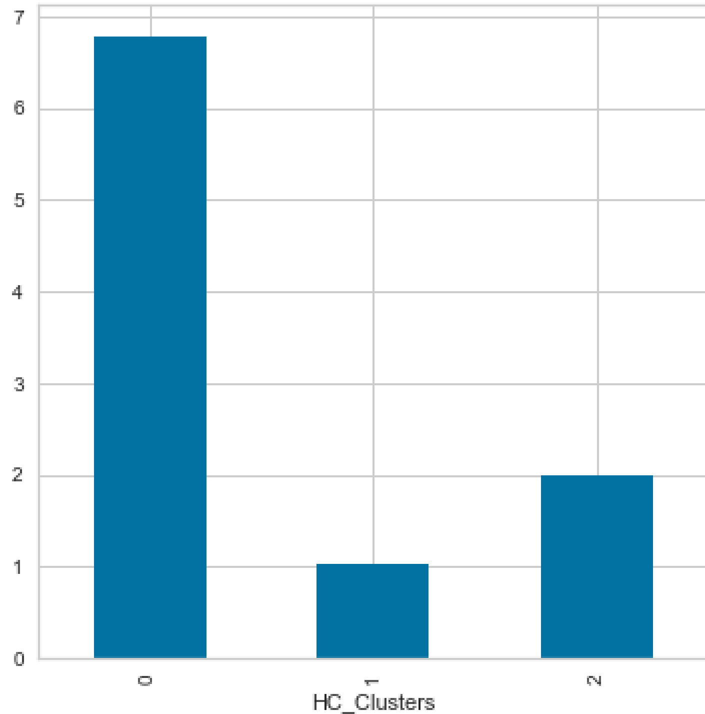
In [75]:
```
df1.groupby("HC_Clusters")["Total_visits_online"].mean().plot.bar(figsize=(6, 6)
```

Out[75]: <AxesSubplot:xlabel='HC_Clusters'>



In [76]:
```
df1.groupby("HC_Clusters")["Total_calls_made"].mean().plot.bar(figsize=(6, 6))
```

Out[76]: <AxesSubplot:xlabel='HC_Clusters'>



Insight (Hierarchical clusters)

- **Cluster 0**: count of 228
  - Sl_No: range of 1-228. Average value of 114
  - Avg_Credit_Limit: range of 3,000-100,000. Average value of 13,596
  - Total_Credit_Cards: range of 1-7. Average value of 2
  - Total_visits_bank: range of 0-2. Average value of 1
  - Total_visits_online: range of 1-12. Average value of 4
  - Total_calls_made: range of 0-10. Average value of 7
- **Cluster 1**: count of 48
  - Sl_No: range of 613-660. Average value of 635
  - Avg_Credit_Limit: range of 84,000-200,000. Average value of 142,750
  - Total_Credit_Cards: range of 7-10. Average value of 9
  - Total_visits_bank: range of 0-1. Average value of 1
  - Total_visits_online: range of 6-15. Average value of 11
  - Total_calls_made: range of 0-2. Average value of 1
- **Cluster 2**: count of 384
  - Sl_No: range of 229-612. Average value of 420
  - Avg_Credit_Limit: range of 5,000-75,000. Average value of 33,508
  - Total_Credit_Cards: range of 4-7. Average value of 5
  - Total_visits_bank: range of 2-5. Average value of 4
  - Total_visits_online: range of 0-2. Average value of 1
  - Total_calls_made: range of 0-4. Average value of 2

Low credit limit = low number of credit cards = high number of calls made

Medium credit limit = medium number of credit cards = higher bank visits

High credit limit = high number of credit cards = high online visits

Compairing K-means and Hierarchical Clustering

Both have the same correlation:

```
- Low credit limit = low number of credit cards = high number of
calls made
- Medium credit limit = medium number of credit cards = higher bank
visits
- High credit limit = high number of credit cards = high online
visits
```

Both have a similar amount of samples in each cluster

Hierarchical clustering to me looks more organized and easier to visualize

```
- For this reason, I prefer Hierarchical clustering.
```

Actionable Insights & Recommendations

If you want to decrease the number of calls, increase average credit limit to ~30,000 or offer more credit card deals to incentivize current or future customers to possess more credit cards. Custumers with 5 or more cards tend to call less.

If you want to minize bank vistits, increase the average credit limit to ~80,000 or again, offer more credit card deals, especially to current customers.

I imagine you don't want to decrease online visits, as the more people visiting the website, the more opportunities you will have to advertise a product or service towards a specific customer or potential customer.

I'm not sure how to recommend improvements on support services, as I don't have any rating data.