Credit Card Customer Data Project / Yeoman Yoon

Problem:

AllLife Bank wants to improve the marketing strategy by personalizing campaigns on new customers and upselling existing customers. At the same time, Delivery team wants to improve customer services by speeding up the customer service process.

Resolution:

Clustering Machine Learning.

In [1]:
```
import pandas as pd
import numpy as np
from sklearn import metrics
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
from sklearn.preprocessing import StandardScaler
from sklearn import metrics
# K-Mean clustering
from scipy.spatial.distance import cdist
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
# Hierarchical clustering
from scipy.spatial.distance import pdist
from sklearn.cluster import AgglomerativeClustering
from scipy.cluster.hierarchy import dendrogram, linkage, cophenet
# PCA
from sklearn.decomposition import PCA
```

Import Data

In [2]:
```
bank = pd.read_excel('CreditCardCustomerData.xlsx') # use copy to store the or
iginal data
data = bank.copy()
```

In [3]:
```
data.head(10)
```

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
5	6	58634	20000	3	0	1
6	7	48370	100000	5	0	11
7	8	37376	15000	3	0	1
8	9	82490	5000	2	0	2
9	10	44770	3000	4	0	1

In [4]:
```
print(f"The data has {data.shape[0]} rows and {data.shape[1]} columns")
```

```
The data has 660 rows and 7 columns
```

In [5]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 660 entries, 0 to 659
Data columns (total 7 columns):
Sl_No                   660 non-null int64
Customer Key            660 non-null int64
Avg_Credit_Limit        660 non-null int64
Total_Credit_Cards      660 non-null int64
Total_visits_bank       660 non-null int64
Total_visits_online     660 non-null int64
Total_calls_made        660 non-null int64
dtypes: int64(7)
memory usage: 36.2 KB
```

Data Dictionary

- Sl_No: Primary key of the records (dropped)
- Customer Key: Customer identification number (dropped)

- **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)

In [6]:
```
data.nunique()
```

Out[6]:
```
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
```

Since we know 5 of Customer Keys are repeated, find what those are and see if there are any patterns.

In [7]:
```
data['Customer Key'].value_counts()
```

Out[7]:
```
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
```

In [8]:
```
data[(data['Customer Key'] ==47437)| (data['Customer Key'] ==37252)
    |(data['Customer Key'] ==97935)| (data['Customer Key'] ==96929)
    |(data['Customer Key'] ==50706)].sort_values(by='Customer Key')
```

Out[8]:

	Sl_No	Customer Key	Avg_Credit_Limit	Total_Credit_Cards	Total_visits_bank	Total_visits_online
48	49	37252	6000	4	0	2
432	433	37252	59000	6	2	1
4	5	47437	100000	6	0	12
332	333	47437	17000	7	3	1
411	412	50706	44000	4	5	0
541	542	50706	60000	7	5	2
391	392	96929	13000	4	5	0
398	399	96929	67000	6	2	2
104	105	97935	17000	2	1	2
632	633	97935	187000	7	1	7

- Rows are not necessarily duplicates nor does Customer Key have any patterns.
- Drop Customer Key & Sl_No.
- No missing values, no outliers

In [9]:
```
data.drop(['Sl_No','Customer Key'], axis=1, inplace = True) # Drop customer id info, we are not using this
```

EDA

In [10]:
```
data.describe().T
```

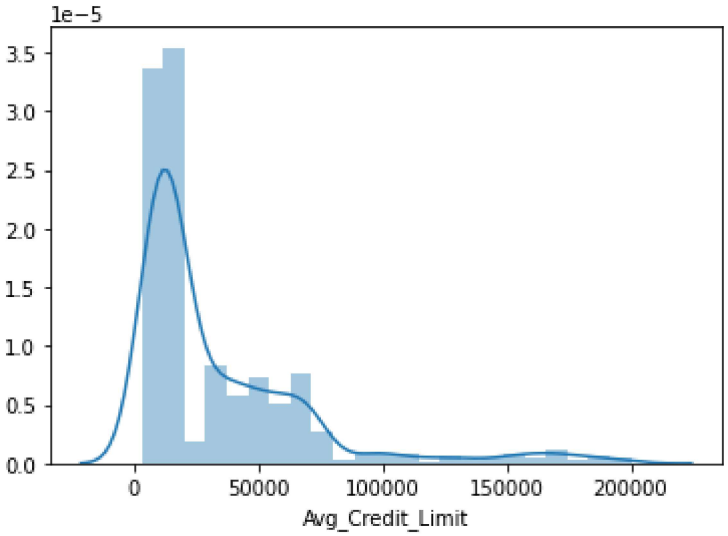
Out[10]:

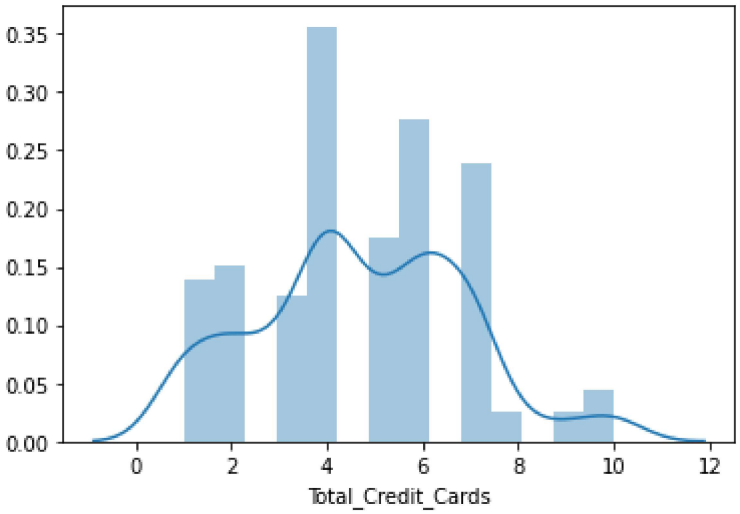
	count	mean	std	min	25%	50%	75%	max
Avg_Credit_Limit	660.0	34574.242424	37625.487804	3000.0	10000.0	18000.0	48000.0	200000.0
Total_Credit_Cards	660.0	4.706061	2.167835	1.0	3.0	5.0	6.0	10.0
Total_visits_bank	660.0	2.403030	1.631813	0.0	1.0	2.0	4.0	5.0
Total_visits_online	660.0	2.606061	2.935724	0.0	1.0	2.0	4.0	15.0
Total_calls_made	660.0	3.583333	2.865317	0.0	1.0	3.0	5.0	10.0

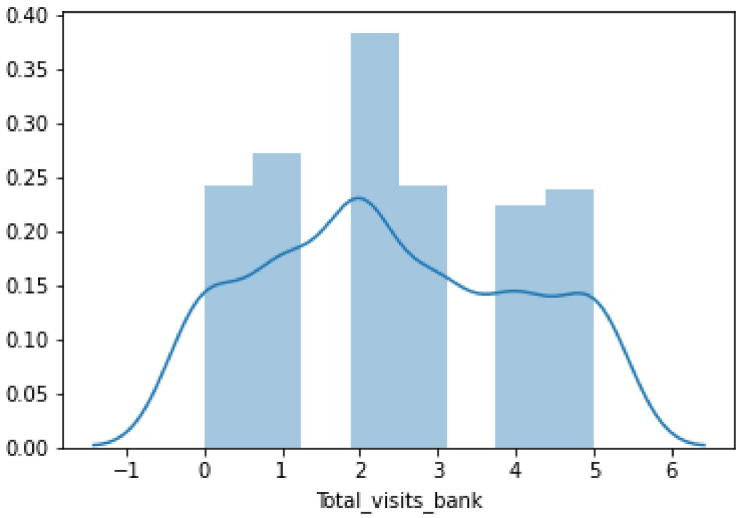
In [11]:
```
all_col = ['Avg_Credit_Limit','Total_Credit_Cards','Total_visits_bank','Total_visits_online','Total_calls_made']
```

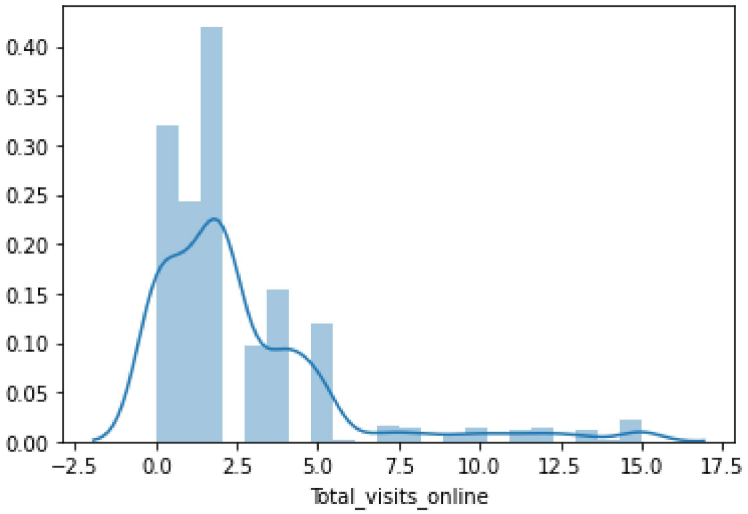
Univariate EDA

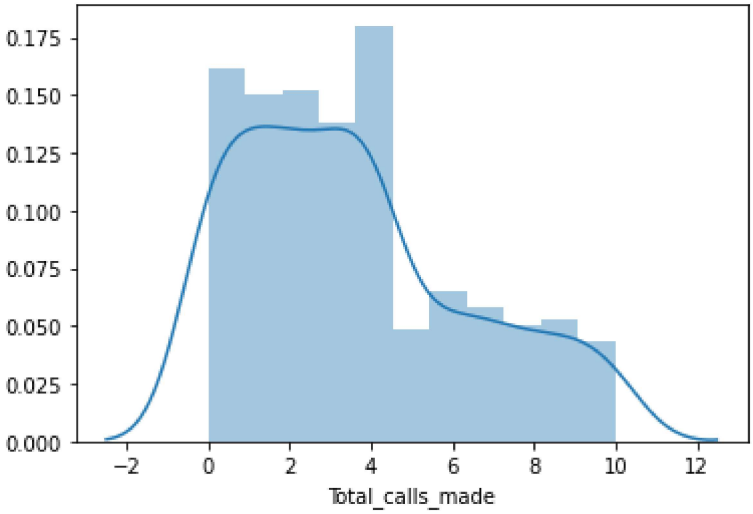
In [12]:
```
for i in all_col:
    sns.distplot(data[i])
    plt.show()
```


1e−5
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
0
50000
100000
150000
200000
Avg_Credit_Limit

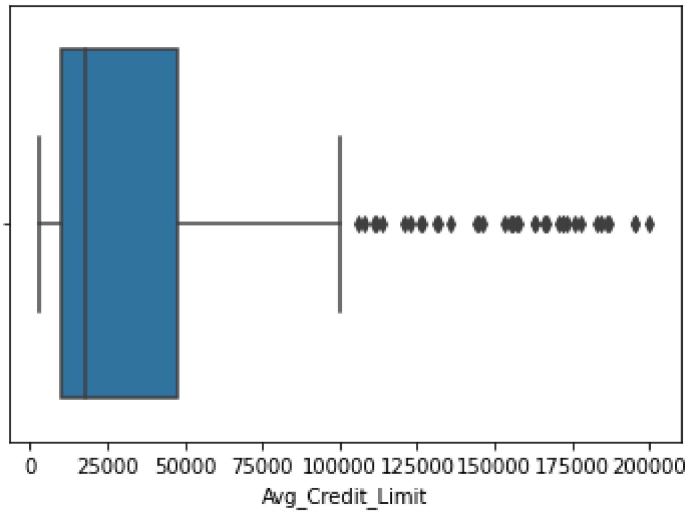

0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
0
2
4
6
8
10
12
Total_Credit_Cards

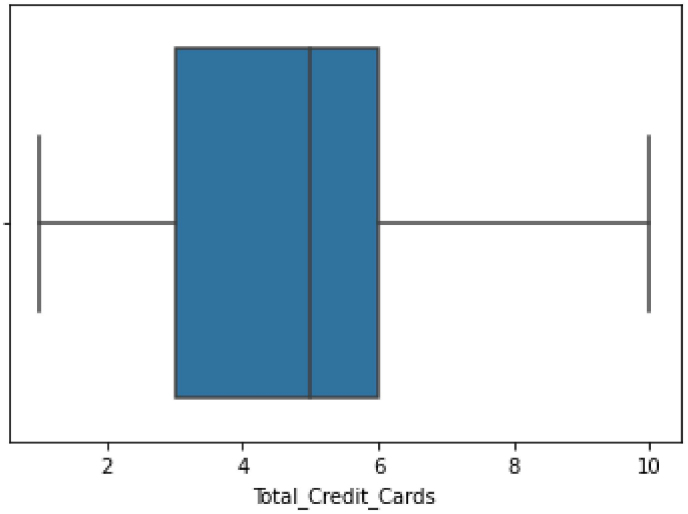

0.40
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
−1
0
1
2
3
4
5
6
Total_visits_bank

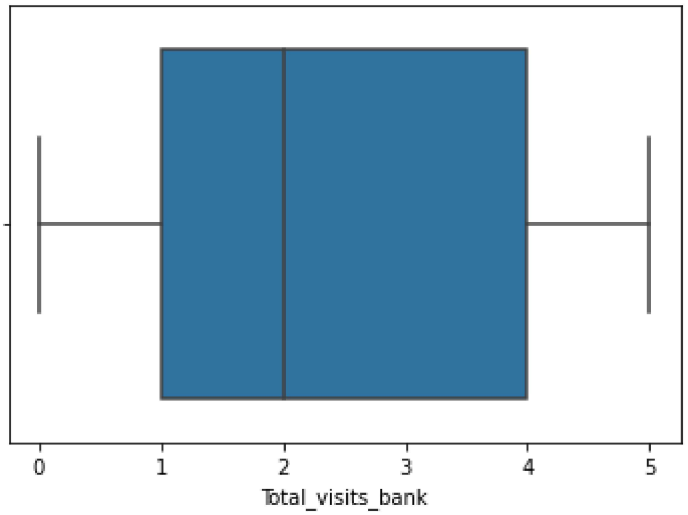

0.40
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
−2.5
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
Total_visits_online

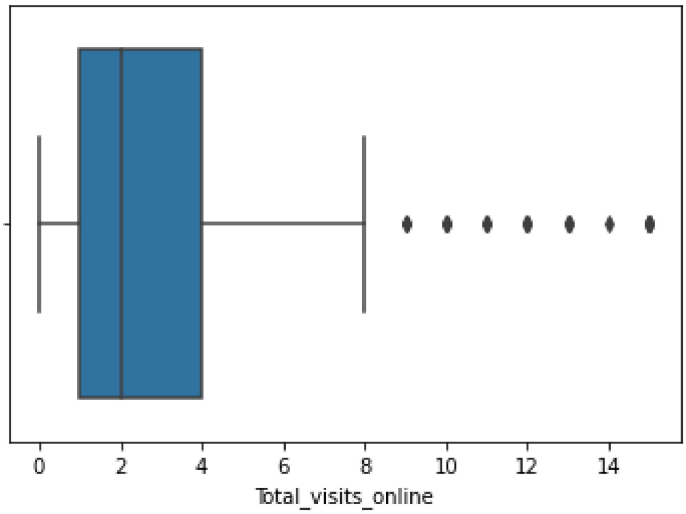

0.175
0.150
0.125
0.100
0.075
0.050
0.025
0.000
−2
0
2
4
6
8
10
12
Total_calls_made

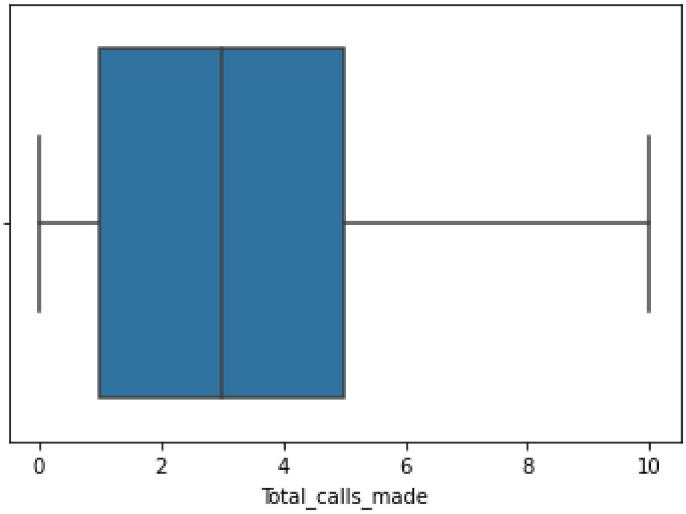
In [13]:
```
for i in all_col:
    sns.boxplot(data[i])
    plt.show()
```


0
25000
50000
75000
100000
125000
150000
175000
200000
Avg_Credit_Limit


2
4
6
8
10
Total_Credit_Cards


0
1
2
3
4
5
Total_visits_bank


0
2
4
6
8
10
12
14
Total_visits_online


0
2
4
6
8
10
Total_calls_made

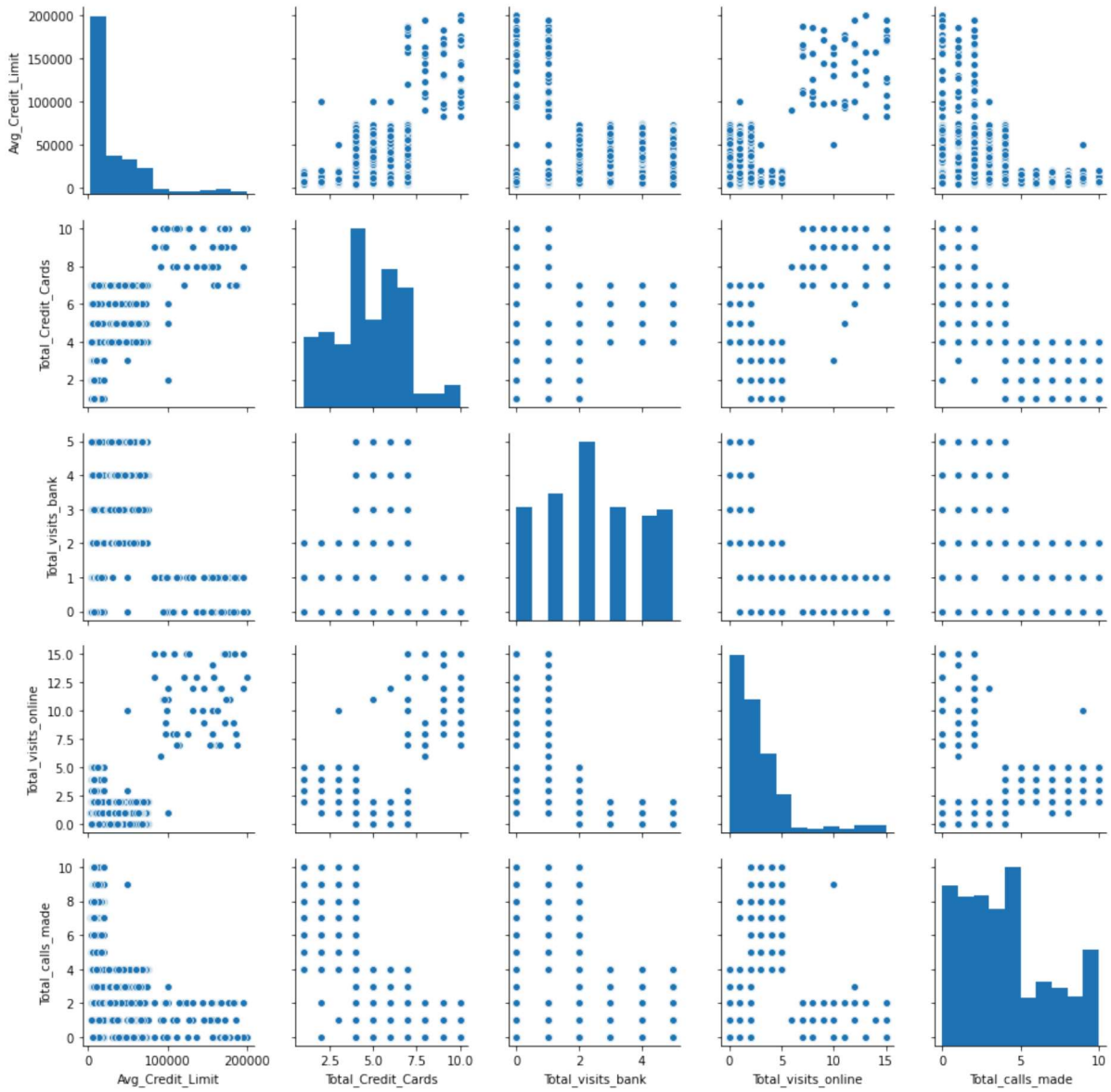
Bivariabte EDA

In [14]:
```
plt.subplots(figsize=(10,10))
sns.heatmap(data.corr(), annot =True, linewidth=0.5)
```

Out[14]: <matplotlib.axes._subplots.AxesSubplot at 0x1b065b4c048>

In [15]:
```
sns.pairplot(data)
```

Out[15]:
```
<seaborn.axisgrid.PairGrid at 0x1b06632edc8>
```



EDA insights

- 0 credit limit is another topic to deal in the later date.
- The credit limits, online visits, and calls made to bank are right skewed.
- Most of the variables are correlated, this is not optimal in clustering method.
- Not much of outliers.

Clustering

Scale the data before clustering to reduce unit problem

In [16]:
```
scaler = StandardScaler()
copy_data = data[all_col].copy() # we need to use data again later, so don't c
hange it
copy_data_scaled = scaler.fit_transform(copy_data)
scaled_data = pd.DataFrame(copy_data_scaled, columns=copy_data.columns)
scaled_data.head()
```

Out[16]:

	Avg_Credit_Limit	Total_Credit_Cards	Total_visits_bank	Total_visits_online	Total_calls_made
0	1.740187	-1.249225	-0.860451	-0.547490	-1.251537
1	0.410293	-0.787585	-1.473731	2.520519	1.891859
2	0.410293	1.058973	-0.860451	0.134290	0.145528
3	-0.121665	0.135694	-0.860451	-0.547490	0.145528
4	1.740187	0.597334	-1.473731	3.202298	-0.203739

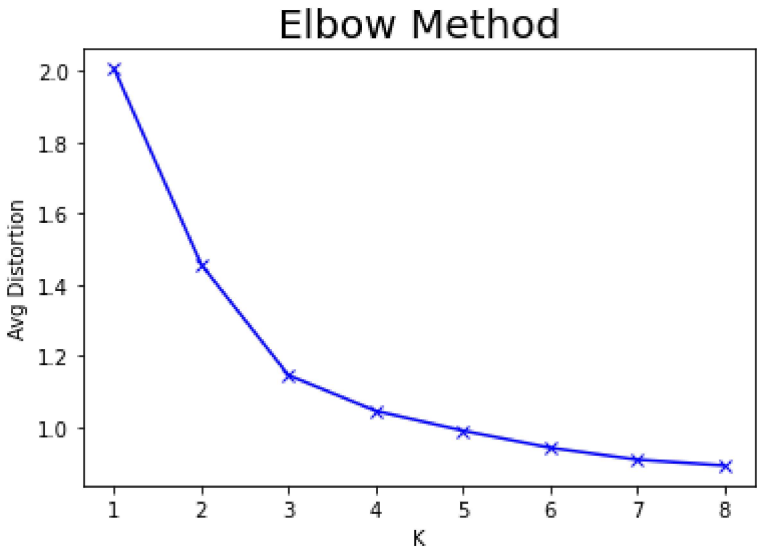
K-Mean Clustering

Elbow Method

In [17]:
```
elbow = range(1, 9)
avgDistortions = []

for k in elbow:
    model = KMeans(n_clusters=k)
    model.fit(scaled_data)
    prediction = model.predict(scaled_data)
    distortion = (
        sum(np.min(cdist(scaled_data, model.cluster_centers_, 'euclidean'), ax
is=1))
        /scaled_data.shape[0])
    avgDistortions.append(distortion)

plt.plot(elbow, avgDistortions, 'bx-')
plt.xlabel('K')
plt.ylabel('Avg Distortion')
plt.title('Elbow Method', fontsize=20)
```

```
C:\Users\yeoma\Anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:882: Us
erWarning: KMeans is known to have a memory leak on Windows with MKL, when th
ere are less chunks than available threads. You can avoid it by setting the e
nvironment variable OMP_NUM_THREADS=3.
  f"KMeans is known to have a memory leak on Windows "
```

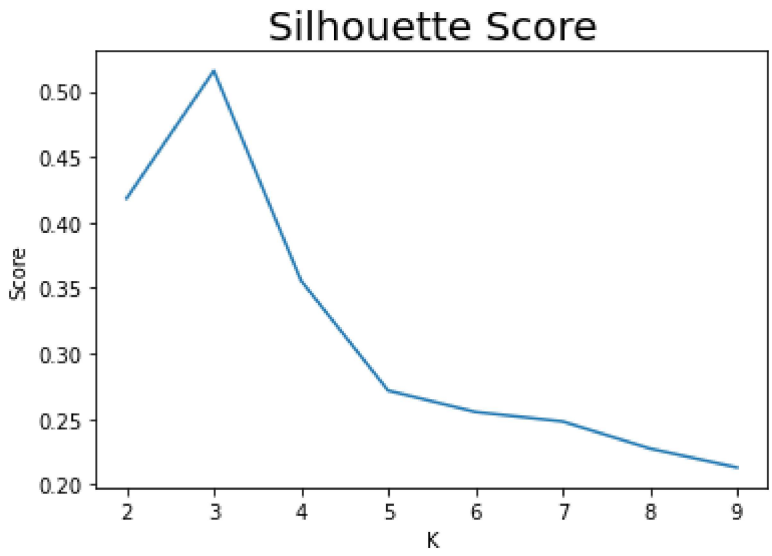
Out[17]: Text(0.5, 1.0, 'Elbow Method')



Silhouette Score

In [18]:
```
cluster_list = range(2, 10)
sil_score = []
for n_clusters in cluster_list:
    clusterer = KMeans(n_clusters=n_clusters)
    preds = clusterer.fit_predict((scaled_data))
    score = silhouette_score(scaled_data, preds)
    sil_score.append(score)


plt.plot(cluster_list, sil_score)
plt.xlabel('K')
plt.ylabel('Score')
plt.title('Silhouette Score', fontsize=20)
```

Out[18]:
```
Text(0.5, 1.0, 'Silhouette Score')
```



Pretty Clear that 3 is good number for K.

In [19]:
```
kmeans = KMeans(n_clusters=3, random_state=0)
kmeans.fit(scaled_data)
```

Out[19]:
```
KMeans(n_clusters=3, random_state=0)
```

In [20]:
```
#Add Which Clusters they belong
scaled_data['K_means_segments'] = kmeans.labels_
data['K_means_segments'] = kmeans.labels_
data.head()
```

Out[20]:

	Avg_Credit_Limit	Total_Credit_Cards	Total_visits_bank	Total_visits_online	Total_calls_made	K
0	100000	2	1	1	0	
1	50000	3	0	10	9	
2	50000	7	1	3	4	
3	30000	5	1	1	4	
4	100000	6	0	12	3	

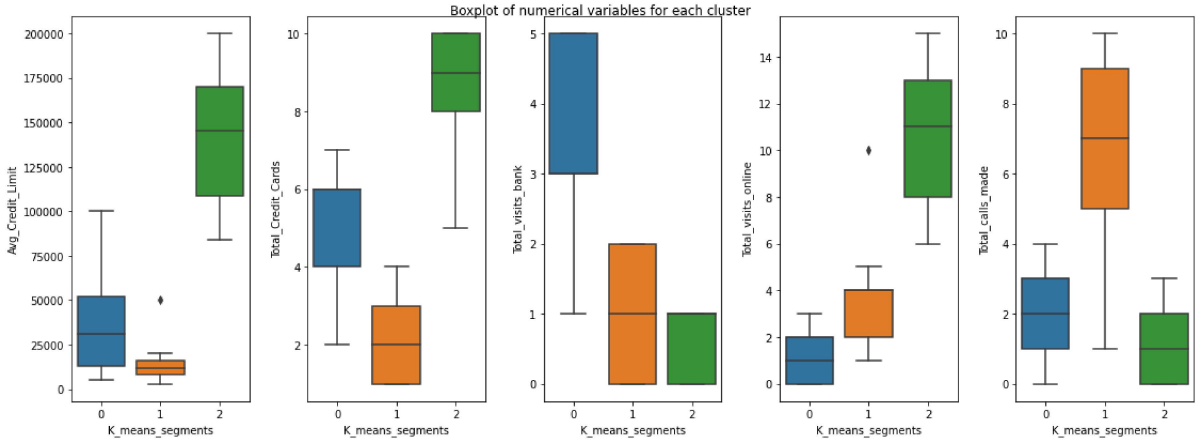
In [21]:
```
cluster_profile = data.groupby('K_means_segments').mean()
cluster_profile['count_in_each_segment'] = (
    data.groupby('K_means_segments')['Total_Credit_Cards'].count().values
)
```

In [22]:
```
# let's display cluster profiles
cluster_profile.style.highlight_max(color='lightgreen', axis=0)
```

Out[22]:

K_means_segments	Avg_Credit_Limit	Total_Credit_Cards	Total_visits_bank	Total_visits_online	T
0	33782.4	5.51554	3.48964	0.981865	
1	12174.1	2.41071	0.933036	3.55357	
2	141040	8.74	0.6	10.9	

In [23]:
```
fig, axes = plt.subplots(1, 5, figsize=(16, 6))
fig.suptitle('Boxplot of numerical variables for each cluster')
counter = 0
for i in range(5):
    sns.boxplot(ax=axes[i], y=data[all_col[counter]], x=data['K_means_segments'])
    counter = counter + 1

fig.tight_layout(pad=2.0)
```



In [24]:
```
pd.crosstab(data.K_means_segments, data.Total_Credit_Cards).style.highlight_max(color="lightgreen", axis=0)
```

Out[24]:

Total_Credit_Cards	1	2	3	4	5	6	7	8	9	10
K_means_segments										
0	0	1	0	102	73	116	94	0	0	0
1	59	63	53	49	0	0	0	0	0	0
2	0	0	0	0	1	1	7	11	11	19

Insights

These insights are written based on majorities.

- **Clusters 0**:
  - Credit Limit between: 15000-50000
  - Number of credit cards: 4-6
  - Total Number of bank visit: 3-5
  - Visiting online: 0-2
  - Calling bank: 1-3
- **Clusters 1**:
  - Credit Limit between: 0-15000
  - Number of credit cards: 0-3
  - Total Number of bank visit: 0-2
  - Visiting online: 2-4
  - Calling bank: 5-9
- **Clusters 2**:
  - Credit Limit between: 110000-170000
  - Number of credit cards: 8-10
  - Total Number of bank visit: 0-1
  - Visiting online: 8-13
  - Calling bank: 0-2

Seems pretty clear the customers have different ways of contacting/interacting with the bank.

Hierarchical clustering

In [54]:
```
# list of distance metrics
distance_metrics = ["euclidean", "chebyshev", "mahalanobis", "cityblock"]

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

high_dm_lm = [0, 0]

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

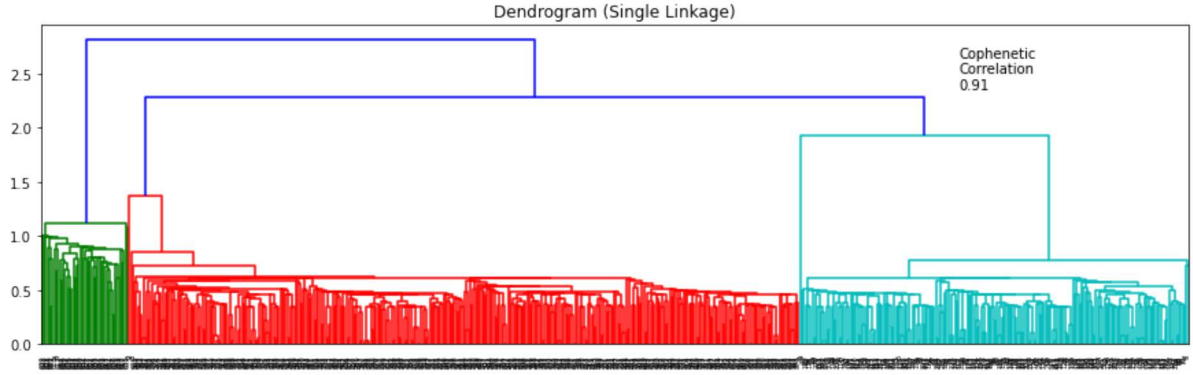
```
Cophenetic correlation for Euclidean distance and single linkage is 0.9125311
786920032.
Cophenetic correlation for Euclidean distance and complete linkage is 0.92331
20605315953.
Cophenetic correlation for Euclidean distance and average linkage is 0.937728
9776168706.
Cophenetic correlation for Euclidean distance and weighted linkage is 0.91898
5765628112.
Cophenetic correlation for Chebyshev distance and single linkage is 0.8560343
558461828.
Cophenetic correlation for Chebyshev distance and complete linkage is 0.86671
50729267128.
Cophenetic correlation for Chebyshev distance and average linkage is 0.930194
4455491864.
Cophenetic correlation for Chebyshev distance and weighted linkage is 0.92268
51235364995.
Cophenetic correlation for Mahalanobis distance and single linkage is 0.91309
83781874598.
Cophenetic correlation for Mahalanobis distance and complete linkage is 0.735
5006687005069.
Cophenetic correlation for Mahalanobis distance and average linkage is 0.8175
601213759511.
Cophenetic correlation for Mahalanobis distance and weighted linkage is 0.752
008952000544.
Cophenetic correlation for Cityblock distance and single linkage is 0.9272888
822057859.
Cophenetic correlation for Cityblock distance and complete linkage is 0.91906
56310817734.
Cophenetic correlation for Cityblock distance and average linkage is 0.934532
9585453204.
Cophenetic correlation for Cityblock distance and weighted linkage is 0.91222
01247969678.
```

In [55]:
```
# list of linkage methods only works with euclidean.
linkage_methods = ["centroid", "ward"]

high_dm_lm = [0, 0]

for lm in linkage_methods:
    Z = linkage(scaled_data, metric="euclidean", method=lm)
    c, coph_dists = cophenet(Z, pdist(scaled_data))
    print("Cophenetic correlation for {} linkage is {}.".format(lm, c))
```

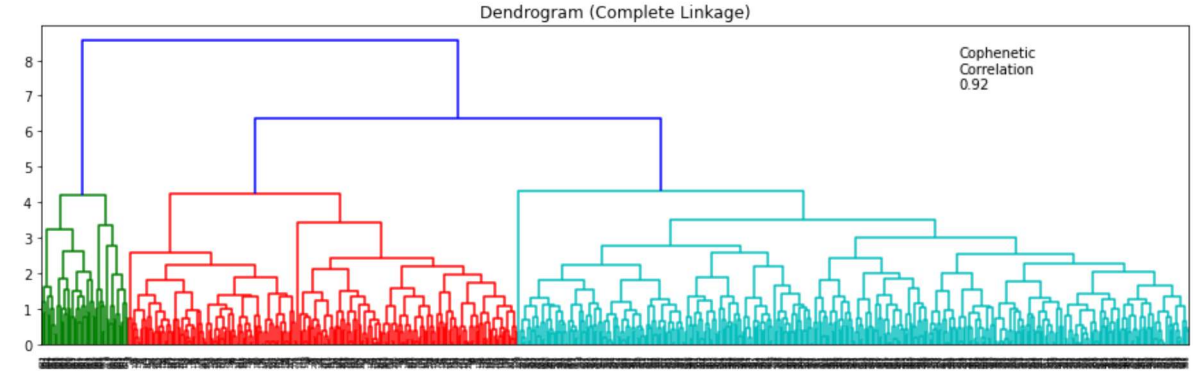
```
Cophenetic correlation for centroid linkage is 0.933924274498229.
Cophenetic correlation for ward linkage is 0.8321686614948933.
```

In [48]:
```
# list of linkage methods
linkage_methods = ["single", "complete", "average", "centroid", "ward", "weighted"]

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

# to create a subplot image
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 cophenetic correlation
for i, method in enumerate(linkage_methods):
    Z = linkage(scaled_data, 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(scaled_data))
    axs[i].annotate(
        f"Cophenetic\nCorrelation\n{coph_corr:0.2f}",
        (0.80, 0.80),
        xycoords="axes fraction",
    )
```

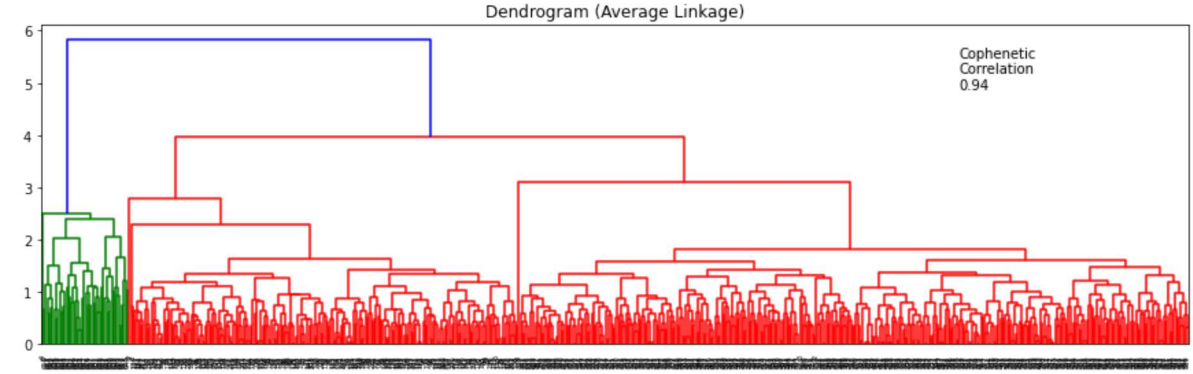
Dendrogram (Single Linkage)



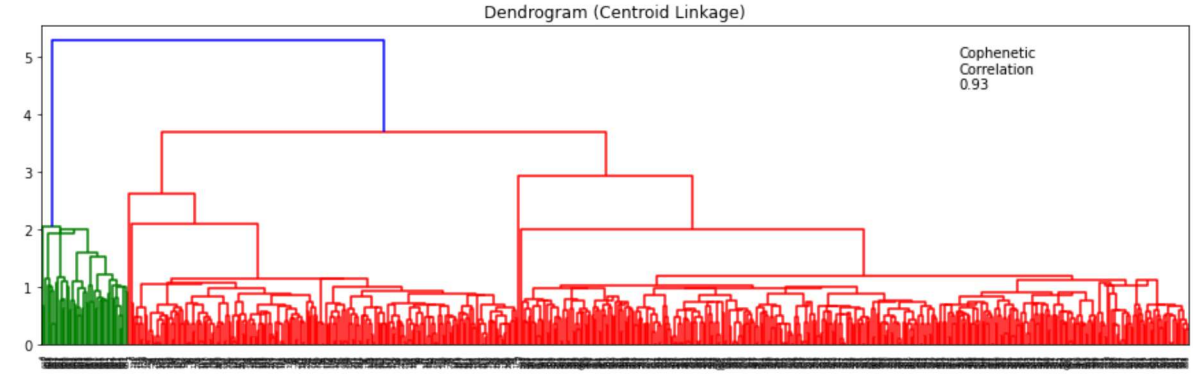
Dendrogram (Complete Linkage)



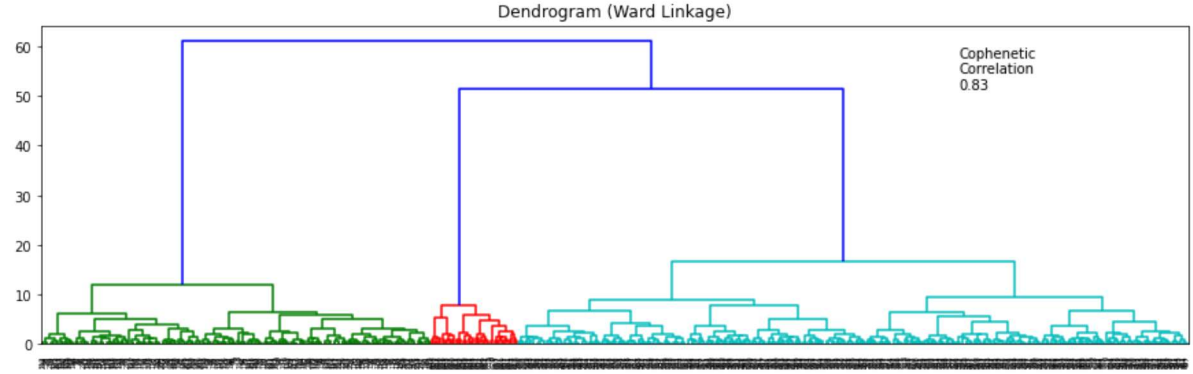
Dendrogram (Average Linkage)



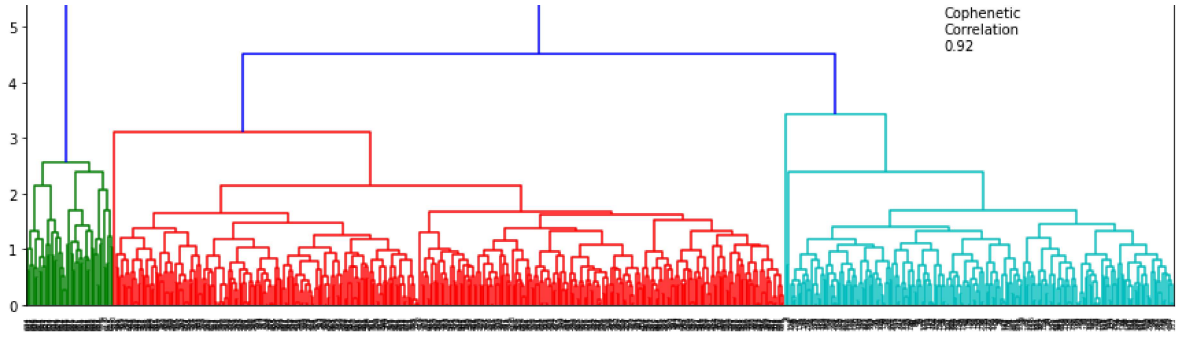
Dendrogram (Centroid Linkage)



Dendrogram (Ward Linkage)



Dendrogram (Weighted Linkage)



- Average Linkage has highest Cophenetic Correlation
- 3 seems to be good number of clusters

In [30]:
```
HCmodel = AgglomerativeClustering(n_clusters=3, affinity="euclidean", linkage
="average")
HCmodel.fit(scaled_data)
```

Out[30]:
```
AgglomerativeClustering(linkage='average', n_clusters=3)
```

In [31]:
```
#Add Which Clusters they belong
scaled_data["HC_Clusters"] = HCmodel.labels_
data["HC_Clusters"] = HCmodel.labels_
data.head()
```

Out[31]:

	Avg_Credit_Limit	Total_Credit_Cards	Total_visits_bank	Total_visits_online	Total_calls_made	H
0	100000	2	1	1	0	
1	50000	3	0	10	9	
2	50000	7	1	3	4	
3	30000	5	1	1	4	
4	100000	6	0	12	3	

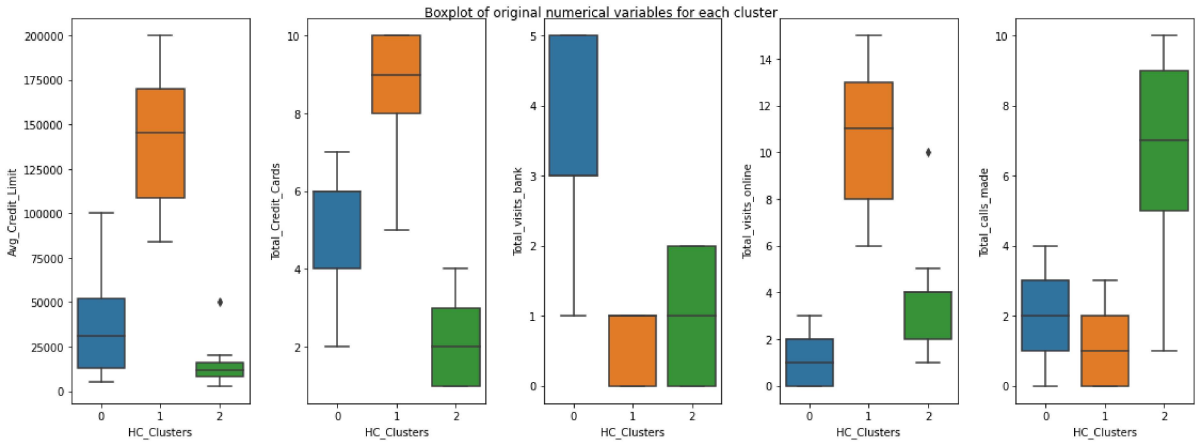
In [32]:
```
cluster_profile = data.groupby("HC_Clusters").mean()
cluster_profile["count_in_each_segments"] = (
    data.groupby("HC_Clusters")["Total_visits_bank"].count().values
)
```

In [33]:
```
# let's display cluster profiles
cluster_profile.style.highlight_max(color="lightgreen", axis=0)
```

Out[33]:

HC_Clusters	Avg_Credit_Limit	Total_Credit_Cards	Total_visits_bank	Total_visits_online	Total_cal
0	33782.4	5.51554	3.48964	0.981865	
1	141040	8.74	0.6	10.9	
2	12174.1	2.41071	0.933036	3.55357	

In [34]:
```
fig, axes = plt.subplots(1, 5, figsize=(16, 6))
fig.suptitle("Boxplot of original numerical variables for each cluster")
counter = 0
for i in range(5):
    sns.boxplot(ax=axes[i], y=data[all_col[counter]], x=data["HC_Clusters"])
    counter = counter + 1

fig.tight_layout(pad=2.0)
```



Insights

These insights are written based on majorities.

- **Clusters 0**:
  - Credit Limit between: 15000-50000
  - Number of credit cards: 4-6
  - Total Number of bank visit: 3-5
  - Visiting online: 0-2
  - Calling bank: 1-3
- **Clusters 1**:
  - Credit Limit between: 110000-170000
  - Number of credit cards: 8-10
  - Total Number of bank visit: 0-1
  - Visiting online: 8-13
  - Calling bank: 0-2
- **Clusters 2**:
  - Credit Limit between: 0-15000
  - Number of credit cards: 0-3
  - Total Number of bank visit: 0-2
  - Visiting online: 2-4
  - Calling bank: 5-9

Same as K-Mean Cluster except cluster 1&2 are switched.

Compare Clusterings

In [49]:
```
pd.crosstab(data.HC_Clusters, data.Total_Credit_Cards).style.highlight_max(color="lightgreen", axis=0)
```

Out[49]:

Total_Credit_Cards	1	2	3	4	5	6	7	8	9	10
HC_Clusters										
0	0	1	0	102	73	116	94	0	0	0
1	0	0	0	0	1	1	7	11	11	19
2	59	63	53	49	0	0	0	0	0	0

In [51]:
```
pd.crosstab(data.K_means_segments, data.Total_Credit_Cards).style.highlight_max(color="lightgreen", axis=0)
```

Out[51]:

Total_Credit_Cards	1	2	3	4	5	6	7	8	9	10
K_means_segments										
0	0	1	0	102	73	116	94	0	0	0
1	59	63	53	49	0	0	0	0	0	0
2	0	0	0	0	1	1	7	11	11	19

Concluson:

- It is more preferred to have more variables to use PCA and then initiate the clustering. More detailed information on customer could be helpful. The data is too correlated to each other.
- The customers have their own preferred methods of contacting the bank. They seems to have patterns according to the credit limit and number of credit cards owned.
- When advertising, it is important to target right group(cluster). Call service seems to have more numbers of contact. It is necessary to note, the more calls and visits occured could represent the customer service is not done correctly.
- However, it is also noticeable that customers with credit limit of 15000-50000 need less assistance from bank in general. Consider pushing out the similar information provided with groups of customers accordingly.

In [ ]:
```
```