DTSA 5510 Final Project: All Life Bank Customer Segmentation

Project home: https://github.com/mull0241/DTSA5510 (https://github.com/mull0241/DTSA5510)

Deadline: Dec 12, 5:50 PM EST

Part 1: Introduction

Background

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 bank poorly. Based on this, the Operations team wants to upgrade the service delivery model, to ensure that customer queries are resolved faster. Head of Marketing and Head of Delivery both decide to reach out to the Data Science team for help

Objective

The overall objective is 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. To meet these criteria, the following methodology will be used:

1. Identify Different Customer Segments:

Data Exploration:

- Explore the dataset to understand the range and distribution of variables related to spending patterns and past interactions.
- Visualize key features to identify any discernible patterns or trends.

Clustering Algorithms:

- Apply unsupervised clustering algorithms such as K-means, hierarchical clustering, or DBSCAN to group customers based on their spending behavior and interactions.
- Experiment with different numbers of clusters and evaluate their effectiveness.

Segment Characteristics:

- Analyze the characteristics of each identified segment, including average spending, frequency of transactions, and types of interactions with the bank.

2. Recommendations for Marketing:

Segment-Specific Campaigns:

- Tailor marketing campaigns to each identified segment's preferences and behaviors.
- Develop personalized promotions or offers that align with the needs and interests of each segment.

Upselling Opportunities:

- Identify upselling opportunities within each segment by understanding their past interactions and spending patterns.
- Recommend specific products or services that align with the identified needs of each segment.

Communication Channels:

- Recommend appropriate communication channels for each segment based on their preferred modes of interaction.
- Consider personalized messaging strategies to enhance the effectiveness of marketing communications.

3. Service Enhancement Recommendations:

Customer Support Analysis:

- Analyze historical customer support data to identify common issues or pain points experienced by different customer segments.
- Prioritize areas where service delivery improvements can have the most significant impact.

Efficiency Improvements:

- Propose operational changes to enhance the efficiency of customer query resolution.
- Consider the implementation of technology solutions or process optimizations to reduce resolution times.

Communication and Feedback:

- Recommend strategies for improving communication with customers during support interactions.
- Implement mechanisms for collecting feedback from customers to continuously refine and enhance support services.

4. Documentation and Communication:

Document the Process:

- Maintain a comprehensive document outlining the steps taken in data preprocessing, clustering, and analysis.
- Include details about the selected clustering algorithm, parameters chosen, and reasons behind decisions.

Presentation to Stakeholders:

- Prepare a clear and concise presentation summarizing the identified customer segments, marketing recommendations, and service enhancement strategies.
- Use visualizations and data insights to effectively communicate the findings to the

Key Questions

- Which variables are most important for clustering?
- How each cluster is different from the others?
- What are the business recommendations?

Data Description

The provided dataset encompasses diverse customer profiles from a ficticous bank, called All Bank. It includes customers' financial information, such as average credit limits, the total count of credit cards held by each customer, and the various channels they have utilized to engage with the bank for inquiries, which encompass in-person visits, online interactions, and communication through a call center.

This dataset consists of 660 rows of data spanning 7 columns of attributes describing customers. It is a public dataset, available for free and can be used without license. The dataset is available online at: https://www.kaggle.com/datasets/shivamb/bank-customer-segmentation (https://www.kaggle.com/datasets/shivamb/bank-customer-segmentation)

Data Dictionary

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

Part 2: Setting Up the Project

Importing the necessary libraries...

In [1]:
```
# 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
from numpy.polynomial.polynomial import polyfit

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

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

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

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

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

In [2]:
```
#Loading dataset
data = pd.read_excel("Credit Card Customer Data.xlsx")
```

Part 3: Initial Exploratory Data Analysis

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

Out[3]:

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

In [4]:
```
data.shape
```

Out[4]: (660, 7)

In [5]:
```
data.describe().T
```

Out[5]:

	count	mean	std	min	25%	50%	75%
Sl_No	660.0	330.500000	190.669872	1.0	165.75	330.5	495.2
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

We can see from the .shape function that there are 660 rows in the dataset. The describe function shows that all the counts = 660; which means that there are no null values in the dataset. At this point, it appears that all the data is numerical. There does not seem to be any categorical columns. The last 4 columns appear to be counts and, therefore, are discreet, numerical data. Average Credit Limit may be continuous, although it is hard to tell without more information. Sl_No and Customer Key are described as 'Primary key of the records' and 'Customer identification number', respectively. We know from the project description that these are for internal bank use and do not describe customer behaviour. These rows can be dropped.

Checking for missing values and/or duplicates...

We have a good idea that there are no missing values, as per above, however, it never hurts to double check. We can also investigate if any of the rows are duplicates, which would need to be cleaned.

In [6]:
```
data.isnull().sum() # this will tell us if there are missing values in an
```

Out[6]:
```
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 [7]:
```
dupes = data.duplicated() #This will check to see if any of the rows are d
sum(dupes)
```

Out[7]:
```
0
```

There does not appear to be any duplicate rows nor missing values. We can look at the types of data contained in each column by using the .info() function...

In [8]:
```
data.info()
```

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

In [9]:
```
data.nunique()
```

Out[9]:
```
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
```

There are 5 less unique Customer Key values compared with unique SL_No values. This means that 5 Customer Key values are duplicates. Lets look at these duplicates more closely...

In [10]:
```
data.rename(columns={'Customer Key':'Key'}, inplace=True) #personal prefer
```

In [11]:
```
data_grouped = data.groupby('Key').count()

for i in data_grouped.loc[data_grouped.Sl_No >= 2].index:
    display(data.loc[data.Key == i])
```

	Sl_No	Key	Avg_Credit_Limit	Total_Credit_Cards	Total_visits_bank	Total_visits_onli
48	49	37252	6000	4	0	
432	433	37252	59000	6	2	

	Sl_No	Key	Avg_Credit_Limit	Total_Credit_Cards	Total_visits_bank	Total_visits_onli
4	5	47437	100000	6	0	
332	333	47437	17000	7	3	

	Sl_No	Key	Avg_Credit_Limit	Total_Credit_Cards	Total_visits_bank	Total_visits_onli
411	412	50706	44000	4	5	
541	542	50706	60000	7	5	

	Sl_No	Key	Avg_Credit_Limit	Total_Credit_Cards	Total_visits_bank	Total_visits_onli
391	392	96929	13000	4	5	
398	399	96929	67000	6	2	

	Sl_No	Key	Avg_Credit_Limit	Total_Credit_Cards	Total_visits_bank	Total_visits_onli
104	105	97935	17000	2	1	
632	633	97935	187000	7	1	

We can see that there are 5 instances where there is more than one record pertaining to the same customer. The entries look unique, so its not an instance of the same customer being duplicated, it seems more likely that a unique Customer Key was assigned to two customers, in error. Since Sl_No and Key are basically redundant, we can drop the key column.

Observations

There are 7 categories of data and 660 rows. All the rows are numerical and stored as integers. This confirms our previous assumption that the last 4 columns are discrete numbers. Now, however, we also know that Average Credit Limit is a discrete number as well. For

example, the bank probably sets the limits using a formula and the limits are rounded to the nearest dollar, tens, hundreds or even thousands of dollars. We can find this out using univariate analysis later. Before we make any changes to the dataset, lets create a copy called 'BD' - short for Bank Data - this will allow us to keep the original dataset in case we need to recall it for some reason and gives us a shorthand name for easy manipulation.

In [12]:
```
BD = data.copy() # This creates a copy of the dataframe
```

Part 4: Cleaning the Data

Dropping Irrelevant Columns

The next step is to remove the columns that do not provide utility. Two such columns are Sl_No and Customer Key. Sl_No is a unique key, but considering that it is simply a numbering for the clients, in order, from 1 to 660, we can remove this column and rely on the python index instead to serve the same purpose. This way, python will not perform unuseful analysis on this column (see .describe() above, for example). Similarly, Customer Key is a unique customer identification number used internally by the bank. Keeping this column may cause python to begin clustering customers based on how similar their randomly assigned identification numbers are to one another, which would not be helpful. Both these columns should be dropped before attempting to build a model.

In [13]:
```
BD.drop('Sl_No',axis=1,inplace=True) # code to drop the S1_No and Customer
BD.drop('Key',axis=1,inplace=True)
```

Renaming Columns

Our next step is to rename some of the columns. Many of the column names are long and can be shortened for easier EDA. This is merely a preference and will not affect the model.

In [14]:
```
BD.rename(columns={'Avg_Credit_Limit':'Limit'}, inplace=True) #personal pr
BD.rename(columns={'Total_Credit_Cards':'Cards'}, inplace=True)
BD.rename(columns={'Total_visits_bank':'Visits'}, inplace=True)
BD.rename(columns={'Total_visits_online':'Online'}, inplace=True)
BD.rename(columns={'Total_calls_made':'Calls'}, inplace=True)
```

In [15]:
```
BD.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 660 entries, 0 to 659
Data columns (total 5 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   Limit   660 non-null    int64
 1   Cards   660 non-null    int64
 2   Visits  660 non-null    int64
 3   Online  660 non-null    int64
 4   Calls   660 non-null    int64
dtypes: int64(5)
memory usage: 25.9 KB
```

We now have columns names that are shorter, easier to work with, and in some cases, easier to understand. Some of the previous names would have been labourous to continually type using different analyses techniques.

Part 5: Univariate Data Analysis

We can now begin to perform Univariate Data Analysis to see if the data needs to be preprocessed before we begin to construct the models. We will create definitions to help visualize the data using the next two blocks of code:

In [16]:
```
def histogram_boxplot(data, xlabel=None, title=None, font_scale=2, figsize
    mean = np.mean(data)

    sns.set(font_scale=font_scale) # setting the font scale of the seaborr
    f2, (ax_box2, ax_hist2) = plt.subplots(2, sharex=True, gridspec_kw={"h
    sns.boxplot(data, ax=ax_box2, showmeans=True,color="violet") #boxplot
    sns.distplot(data,kde=False, ax=ax_hist2, bins=bins,palette="winter")
    ax_hist2.axvline(mean, color='g', linestyle='--') #mean will show as v
    if xlabel: ax_hist2.set(xlabel=xlabel) #xlabel
    if title: ax_box2.set(title=title) # title of the graph
    plt.show() # for plotting the graph
```

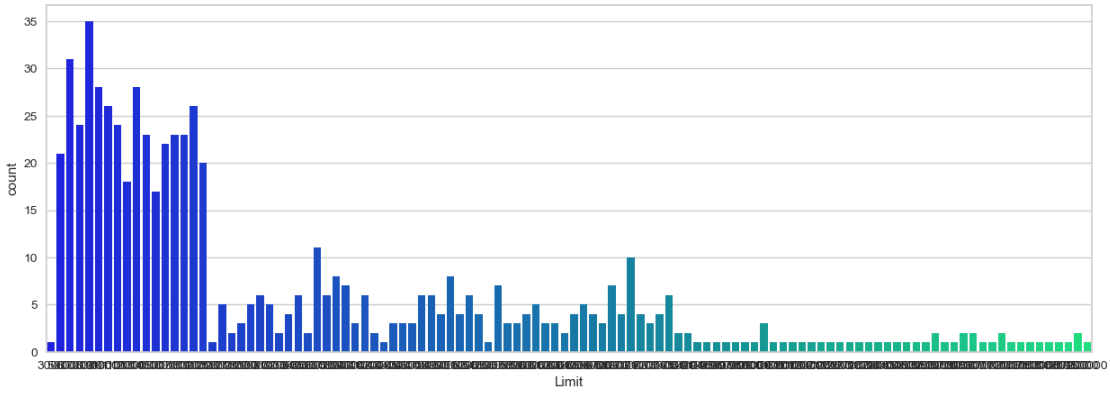
In [17]:
```
def perc_on_bar(plot, feature):
    '''
    plot
    feature: categorical feature
    the function wont work if a colume is passed in hue parameter
    '''
    total = len(feature) # length of the column
    for p in ax.patches:
        percentage = '{:.1f}%'.format(100 * p.get_height()/total) # percer
        x = p.get_x() + p.get_width() / 2 - 0.05 # width of the plot
        y = p.get_y() + p.get_height()           # height of the plot
        ax.annotate(percentage, (x,y), size = 20) # annotate the percentag
    plt.show() # shows the plot
```

Limit

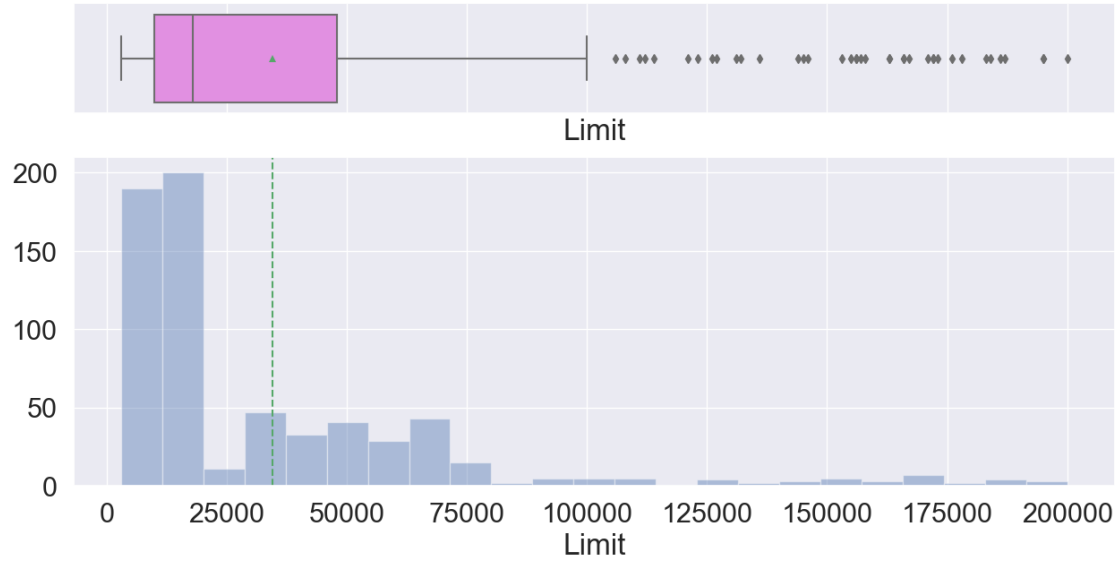
In [18]:
```
BD.Limit.describe()
```

Out[18]:
```
count       660.000000
mean      34574.242424
std       37625.487804
min        3000.000000
25%       10000.000000
50%       18000.000000
75%       48000.000000
max      200000.000000
Name: Limit, dtype: float64
```

In [19]:
```
plt.figure(figsize=(15,5))
ax = sns.countplot(BD['Limit'], palette='winter')
```



In [20]:
```
histogram_boxplot(BD['Limit'])
```



In [21]:
```
median = np.median(BD.Limit) # find the median income of all customers
mean = np.mean(BD.Limit) # find the mean income of all customers
print('The mean equals', mean)
print('The median equals', median)
```

```
The mean equals 34574.242424242424
The median equals 18000.0
```

In [22]:
```
BD.loc[BD['Limit'] > 100000] #examining outliers beyond right whisker of 1
```

Out[22]:

	Limit	Cards	Visits	Online	Calls
612	157000	9	1	14	1
614	163000	8	1	7	1
615	131000	9	1	10	1
617	136000	8	0	13	0
618	121000	7	0	13	2
619	158000	7	0	13	0
620	108000	10	0	15	1
621	166000	9	1	12	2
622	176000	10	1	15	2
623	166000	10	0	7	0
624	178000	7	0	11	0
626	156000	9	1	10	2
627	146000	10	0	12	1
629	155000	8	0	7	2
630	200000	10	0	13	0
631	195000	8	0	15	0
632	187000	7	1	7	0
633	163000	7	1	10	1
634	106000	8	0	8	1
635	114000	10	1	7	2
636	126000	10	1	8	0
637	173000	9	1	11	0
638	153000	8	1	7	1
639	184000	7	1	15	2
640	123000	8	1	15	2
641	144000	10	0	10	2
644	127000	10	1	15	1
645	171000	10	0	15	0
646	186000	7	0	8	1
647	183000	9	0	9	2
648	111000	8	1	7	0
649	112000	10	1	8	1
650	195000	10	1	12	2
651	132000	9	1	12	2
652	156000	8	1	8	0
654	172000	10	1	9	1

	Limit	Cards	Visits	Online	Calls
657	145000	8	1	9	1
658	172000	10	1	15	0
659	167000	9	0	12	2

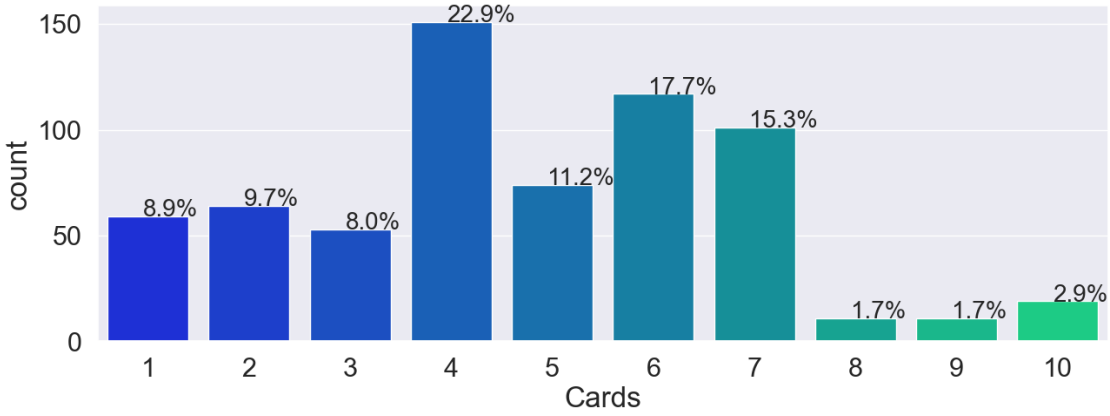
- the data is not normally distributed
- since the mean is larger than the median, there is positive skewness
- there are a large number of outliers, but this isnt necessarily problematic since there are many and they are spaced out at intervals ranging from 100,000 to 200,000
- Decision: keep outliers as they are
- Rationale: Outliers seem reasonable for the data. It makes sense different customers have different credit limits, which can explain the continuous range. Some customers would have higher earnings and therefore qualify for higher credit limits.

Cards

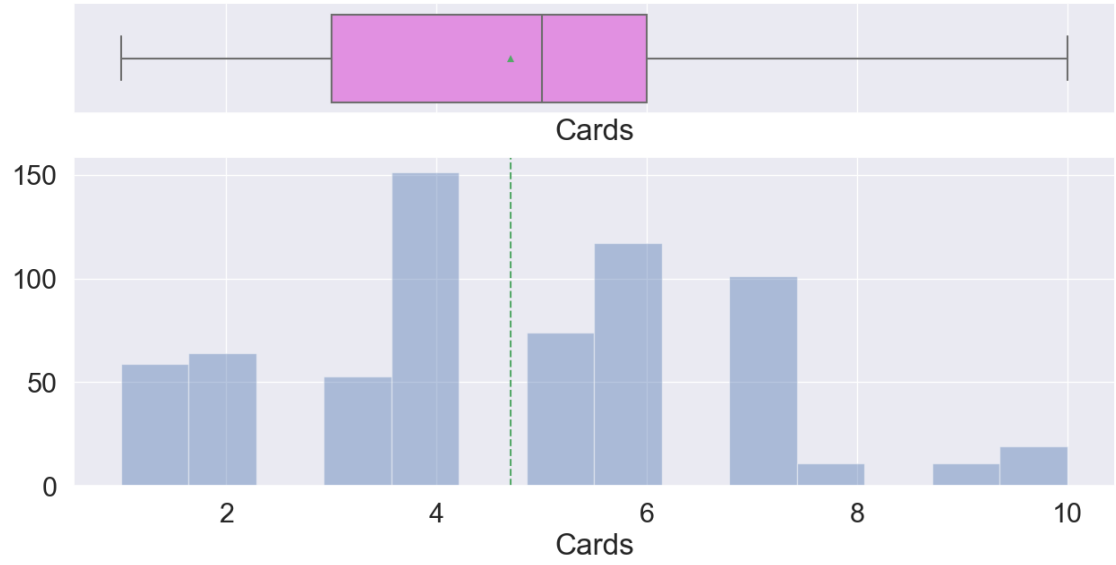
In [23]:
```
BD.Cards.describe()
```

Out[23]:
```
count    660.000000
mean       4.706061
std        2.167835
min        1.000000
25%        3.000000
50%        5.000000
75%        6.000000
max       10.000000
Name: Cards, dtype: float64
```

In [24]:
```
plt.figure(figsize=(15,5))
ax = sns.countplot(BD['Cards'], palette='winter')
perc_on_bar(ax,BD['Cards'])
```



In [25]:
```
histogram_boxplot(BD['Cards'])
```



In [26]:
```
median = np.median(BD.Cards) # find the median income of all customers
mean = np.mean(BD.Cards) # find the mean income of all customers
print('The mean equals', mean)
print('The median equals', median)
```

```
The mean equals 4.706060606060606
The median equals 5.0
```

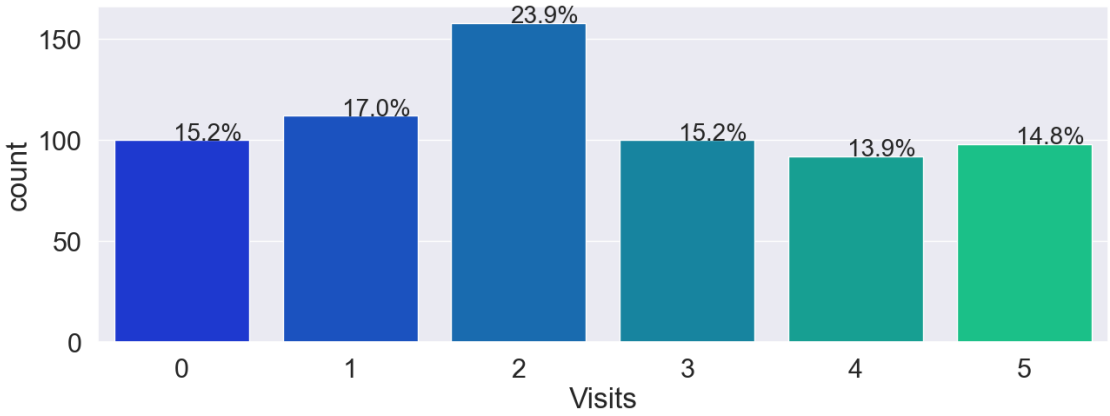
- the data is not normally distributed
- since the mean is smaller than the median, there is negative skewness
- there are no outliers
- the number of cards a customer carries ranges from 1 to 10, with 5 being the median and 4.7 being the mean

Visits

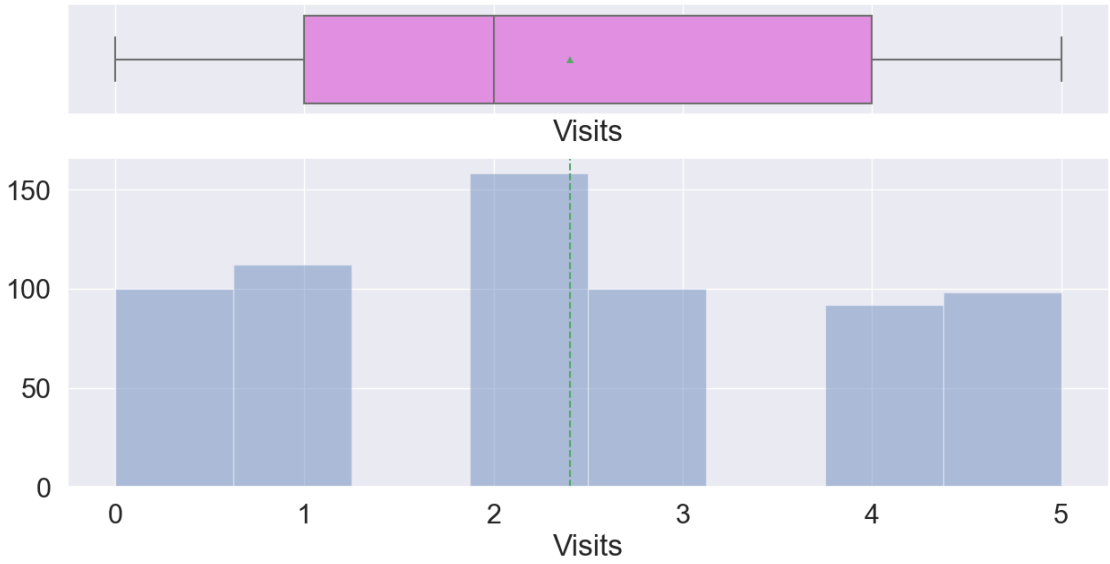
In [27]:
```
BD.Visits.describe()
```

Out[27]:
```
count    660.000000
mean       2.403030
std        1.631813
min        0.000000
25%        1.000000
50%        2.000000
75%        4.000000
max        5.000000
Name: Visits, dtype: float64
```

In [28]:
```
plt.figure(figsize=(15,5))
ax = sns.countplot(BD['Visits'], palette='winter')
perc_on_bar(ax,BD['Visits'])
```



In [29]:
```
histogram_boxplot(BD['Visits'])
```



In [30]:
```
median = np.median(BD.Visits) # find the median income of all customers
mean = np.mean(BD.Visits) # find the mean income of all customers
print('The mean equals', mean)
print('The median equals', median)
```

```
The mean equals 2.403030303030303
The median equals 2.0
```

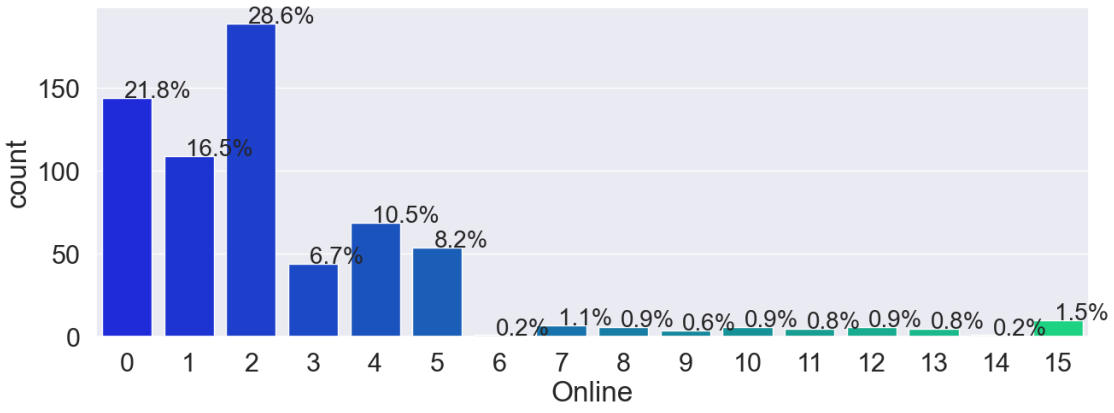
- the data is close to being normally distributed
- since the mean is larger than the median, so there is some positive skewness
- there are no outliers
- the number of visits a customer makes ranges from 1 to 5, with 2 being the median and 2.4 being the mean

Online

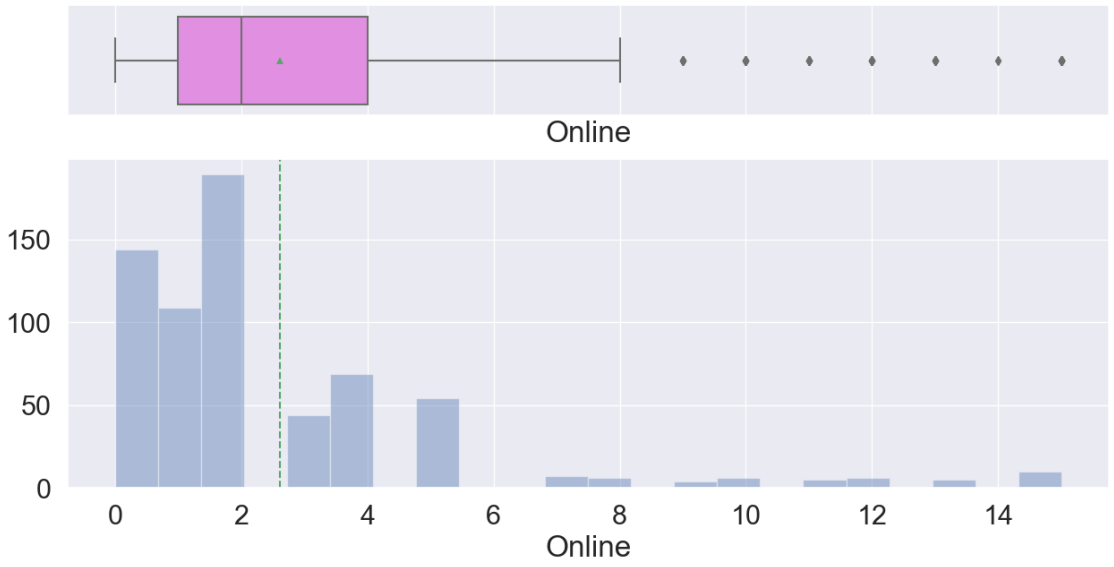
In [31]:
```
BD.Online.describe()
```

Out[31]:
```
count    660.000000
mean       2.606061
std        2.935724
min        0.000000
25%        1.000000
50%        2.000000
75%        4.000000
max       15.000000
Name: Online, dtype: float64
```

In [32]:
```
plt.figure(figsize=(15,5))
ax = sns.countplot(BD['Online'], palette='winter')
perc_on_bar(ax,BD['Online'])
```



In [33]:
```
histogram_boxplot(BD['Online'])
```



In [34]:
```
median = np.median(BD.Online) # find the median income of all customers
mean = np.mean(BD.Online) # find the mean income of all customers
print('The mean equals', mean)
print('The median equals', median)
```

```
The mean equals 2.606060606060606
The median equals 2.0
```

In [35]:
```
BD.loc[BD['Online'] > 8] #examining outliers beyond right whisker of 8 cal
```

Out[35]:

	Limit	Cards	Visits	Online	Calls
1	50000	3	0	10	9
4	100000	6	0	12	3
6	100000	5	0	11	2
612	157000	9	1	14	1
613	94000	9	1	11	0
615	131000	9	1	10	1
616	96000	10	1	11	2
617	136000	8	0	13	0
618	121000	7	0	13	2
619	158000	7	0	13	0
620	108000	10	0	15	1
621	166000	9	1	12	2
622	176000	10	1	15	2
624	178000	7	0	11	0
626	156000	9	1	10	2
627	146000	10	0	12	1
628	84000	9	1	15	0
630	200000	10	0	13	0
631	195000	8	0	15	0
633	163000	7	1	10	1
637	173000	9	1	11	0
639	184000	7	1	15	2
640	123000	8	1	15	2
641	144000	10	0	10	2
642	97000	10	1	9	2
644	127000	10	1	15	1
645	171000	10	0	15	0
647	183000	9	0	9	2
650	195000	10	1	12	2
651	132000	9	1	12	2
653	95000	10	0	15	1
654	172000	10	1	9	1
655	99000	10	1	10	0
656	84000	10	1	13	2
657	145000	8	1	9	1
658	172000	10	1	15	0

	Limit	Cards	Visits	Online	Calls
659	167000	9	0	12	2

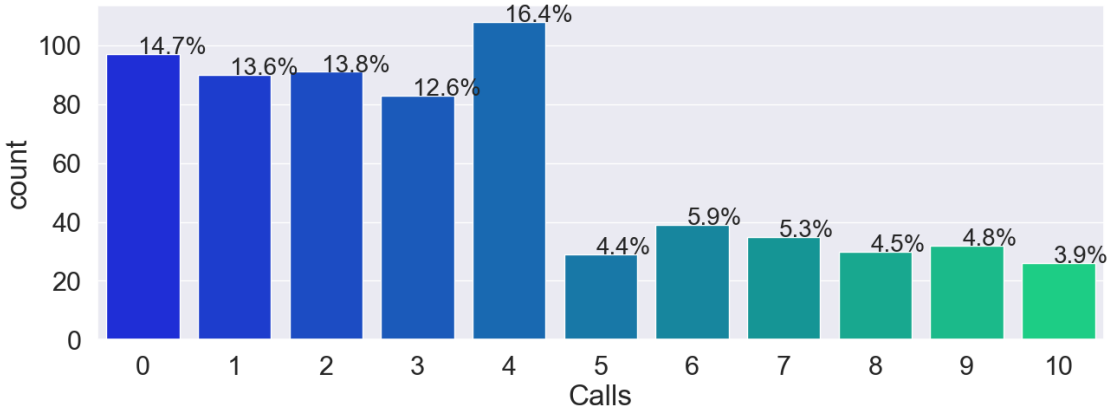
- the data is not normally distributed
- the mean is larger than the median, so there is some positive skewness
- the number of online logins ranges from 0 to 15, with 2.0 being the median and 2.6 being the mean
- there are a number of outliers, but this isnt necessarily problematic since there are many and they are spaced out at even intervals
- Decision: keep outliers as they are
- Rationale: Outliers seem reasonable for the data. It makes sense different customers use online banking more often than others. 15 total logins (max) is not unreasonable.

Calls

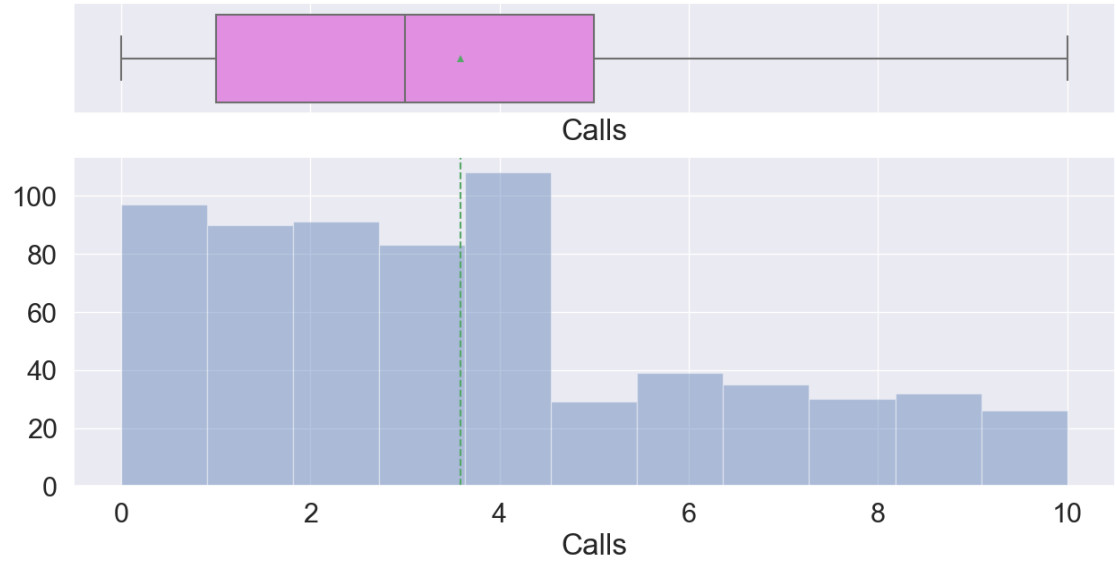
In [36]:
```
BD.Calls.describe()
```

Out[36]:
```
count    660.000000
mean       3.583333
std        2.865317
min        0.000000
25%        1.000000
50%        3.000000
75%        5.000000
max       10.000000
Name: Calls, dtype: float64
```

In [37]:
```
plt.figure(figsize=(15,5))
ax = sns.countplot(BD['Calls'], palette='winter')
perc_on_bar(ax,BD['Calls'])
```



In [38]:
```
histogram_boxplot(BD['Calls'])
```



In [39]:
```
median = np.median(BD.Calls) # find the median income of all customers
mean = np.mean(BD.Calls) # find the mean income of all customers
print('The mean equals', mean)
print('The median equals', median)
```

```
The mean equals 3.5833333333333335
The median equals 3.0
```

- the data is close to being normally distributed
- since the mean is larger than the median, so there is some positive skewness
- there are no outliers
- the number of cards a customer carries ranges from 1 to 5, with 2 being the median and 2.4 being the mean

Part 6: Bivariate/Multivariate Data Analysis

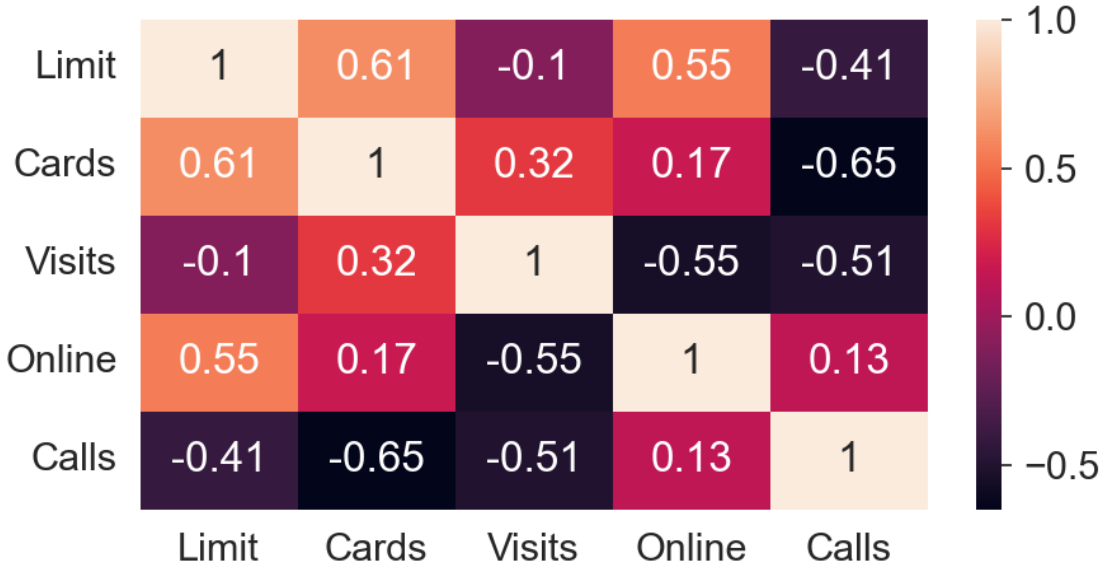
We can start by checking the correlation between the numerical data variables by using .corr and a heatmap function.

In [40]:
```
BD.corr() # creates a table of how the numerical values are correlated
```

Out[40]:

	Limit	Cards	Visits	Online	Calls
Limit	1.000000	0.608860	-0.100312	0.551385	-0.414352
Cards	0.608860	1.000000	0.315796	0.167758	-0.651251
Visits	-0.100312	0.315796	1.000000	-0.551861	-0.506016
Online	0.551385	0.167758	-0.551861	1.000000	0.127299
Calls	-0.414352	-0.651251	-0.506016	0.127299	1.000000

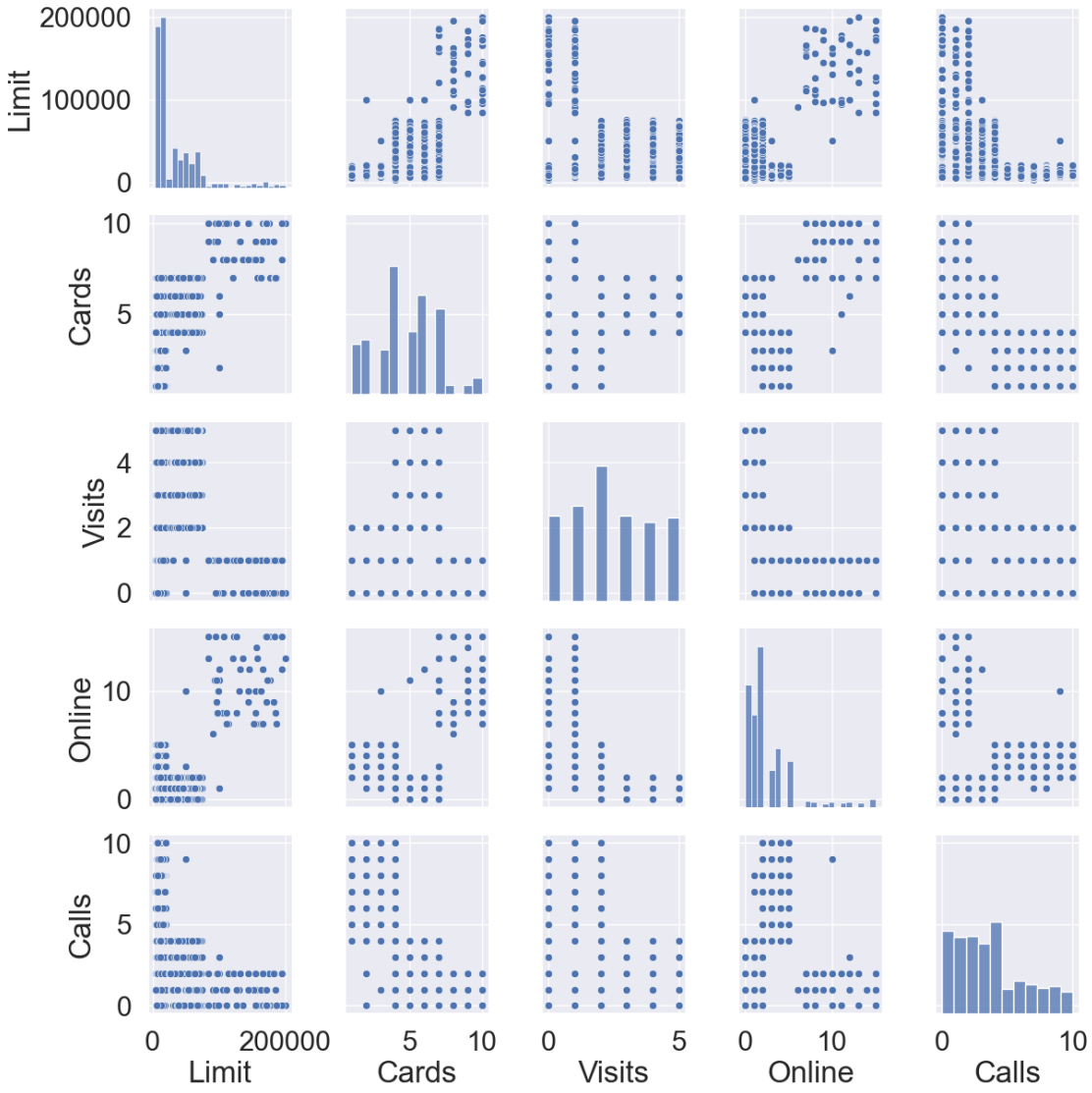
In [41]:
```
plt.figure(figsize=(10,5))
sns.heatmap(BD.corr(), annot=True)
plt.show()
```



- If we define strong correlation as +/- 0.60 to 0.99, we can see that Number of Cards and Credit Limit have strong, positive correlation; Number of Cards and Number of Calls have strong, negative correlation
- If we define moderate correlation as +/- 0.25 to 0.59, we can see that Online logins and Credit Limit, as well as Number of Visits and Number of Cards have moderate, positive correlations; whereas Online logins and Number of Visits, Number of Visits and Number of Calls, and Number of Calls and Credit Limit, all have moderate, negative correlations
- If we define weak correlation as +/- 0.10 to 0.24 then Online Logins and Number of Cards, Number of Calls and Online logins have weak, positive correlations; whereas Number of Visits and Credit Limit has a weak, negative correlation.

In [42]:
```
# Pairplot using sns

sns.pairplot(BD)
```

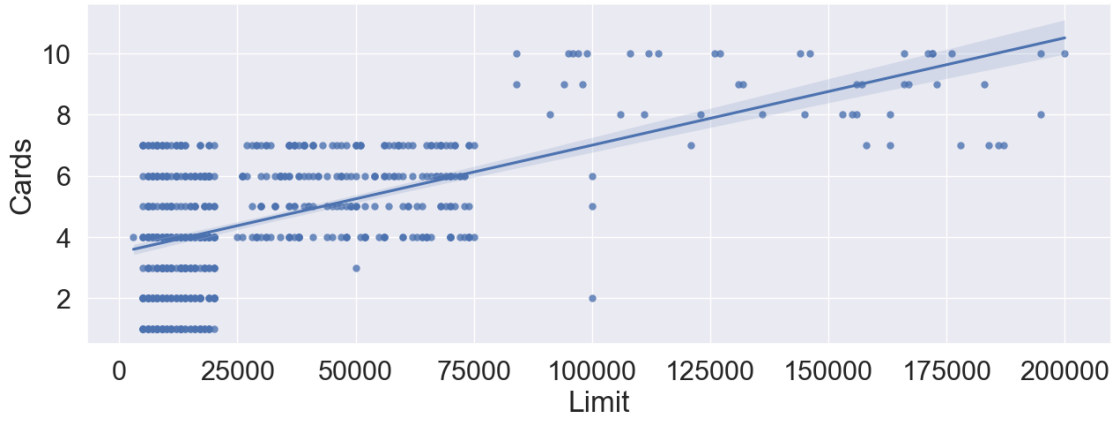
Out[42]: <seaborn.axisgrid.PairGrid at 0x1d721be1eb0>



The pairwise plots show us two things: first, we can see some of the more correlated data do have shapes that show trends either up, to the right (positive correlation) or down to the right (negative correlation); second, and more important for our study, we can see that the data has formed visible clusters in many cases. For example Online vs Limit shows a large cluster at the top right and a second cluster at the bottom left.

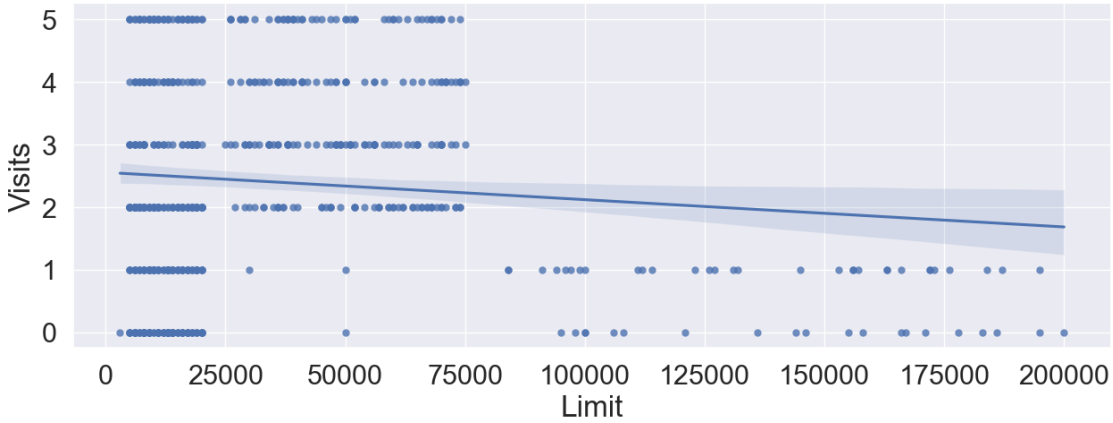
Numerical vs Numerical

In [43]:
```
#Limit vs Cards

figure = plt.figure(figsize=(15,5)) # adding a trend line will give us a b

sns.regplot(x="Limit", y="Cards", data=BD);
```



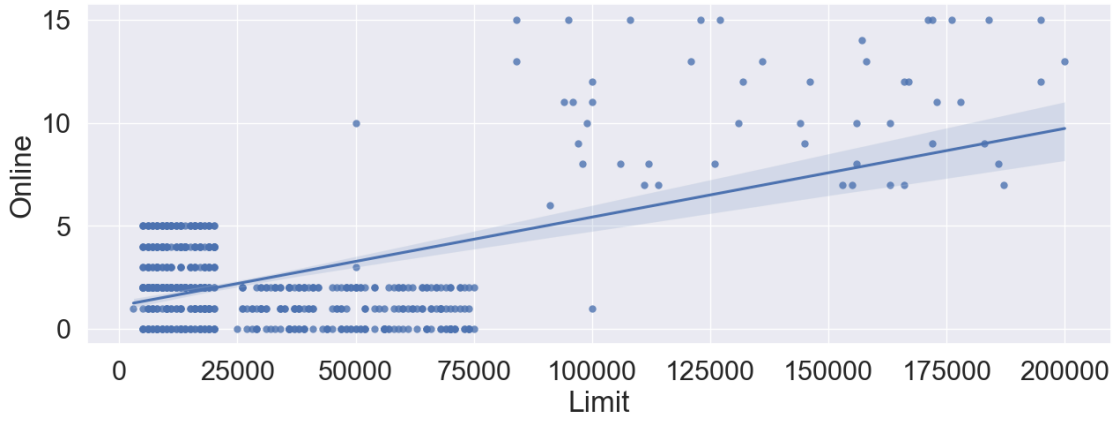
- We can see a big cluster near the bottom left
- There is positive correlation between these two variables as the trendline rises as it moves right

In [44]:
```
#Limit vs Visits

figure = plt.figure(figsize=(15,5)) # adding a trend line will give us a b

sns.regplot(x="Limit", y="Visits", data=BD);
```



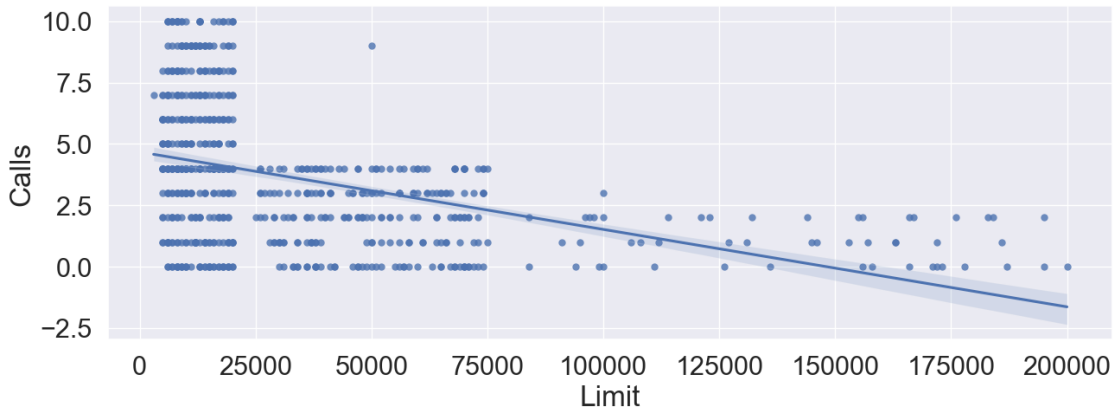
- We can see a big cluster near the left and a smaller one at the bottom right
- There is a weak negative correlation between these two variables as the trendline drops slightly as it moves right

In [45]:
```
#Limit vs Online

figure = plt.figure(figsize=(15,5)) # adding a trend line will give us a b

sns.regplot(x="Limit", y="Online", data=BD);
```



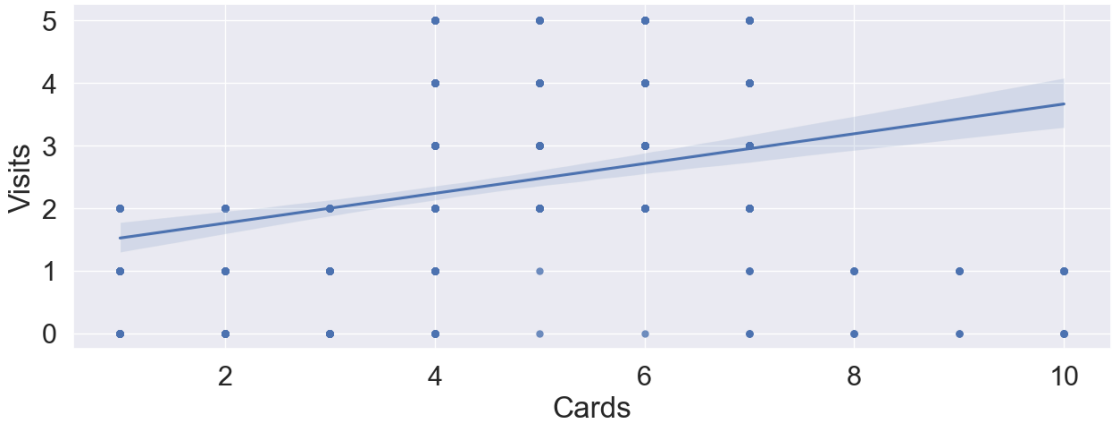
- We can see a big cluster near the bottom left
- There is positive correlation between these two variables as the trendline rises as it moves right

In [46]:
```
#Limit vs Calls

figure = plt.figure(figsize=(15,5)) # adding a trend line will give us a b

sns.regplot(x="Limit", y="Calls", data=BD);
```



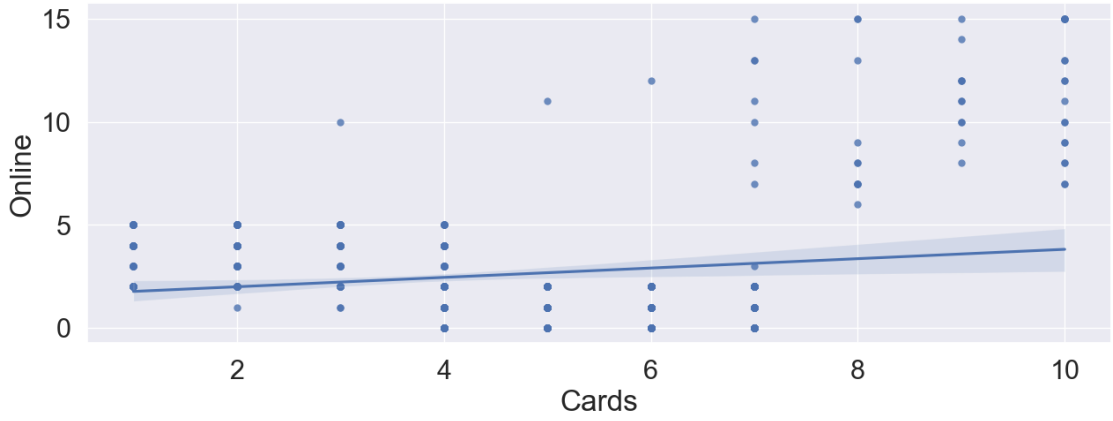
- We can see a big cluster near the left and a smaller one at the bottom right
- There is negative correlation between these two variables as the trendline drops as it moves right

In [47]:
```
#Cards vs Visits

figure = plt.figure(figsize=(15,5)) # adding a trend line will give us a b

sns.regplot(x="Cards", y="Visits", data=BD);
```



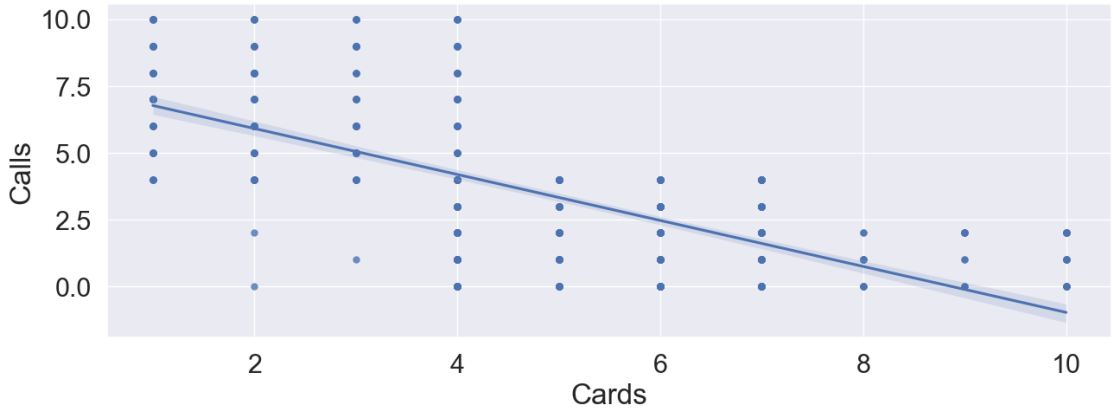
- It is difficult to see any clustering, although this does not mean it is not occuring. We do see gaps in the data at the top right and top left, meaning people who make few calls (1-3) and many calls (>7) do not make more than 2 visits
- There is positive correlation between these two variables as the trendline rises as it moves right

In [48]:
```
#Cards vs Online

figure = plt.figure(figsize=(15,5)) # adding a trend line will give us a b

sns.regplot(x="Cards", y="Online", data=BD);
```



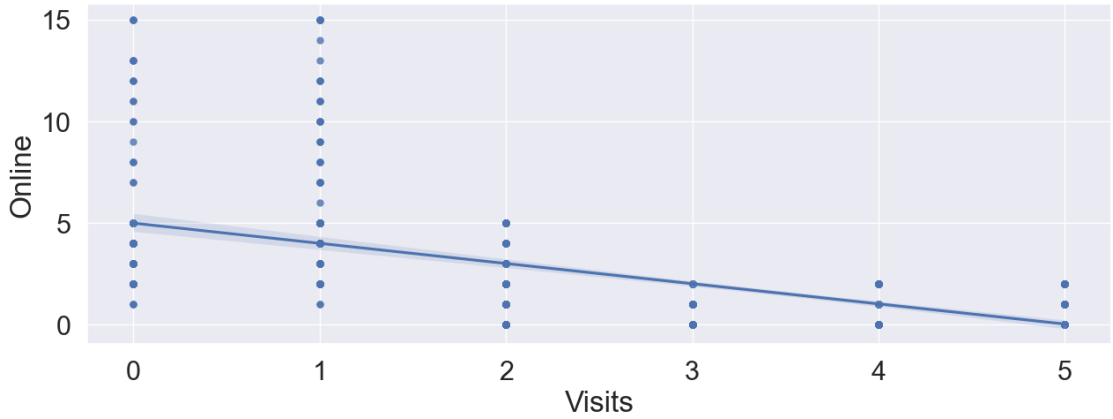
- We can see a cluster near the bottom left and one at the top right
- There is a weak positive correlation between these two variables as the trendline rises slightly as it moves right

In [49]:
```
#Cards vs Calls

figure = plt.figure(figsize=(15,5)) # adding a trend line will give us a b

sns.regplot(x="Cards", y="Calls", data=BD);
```



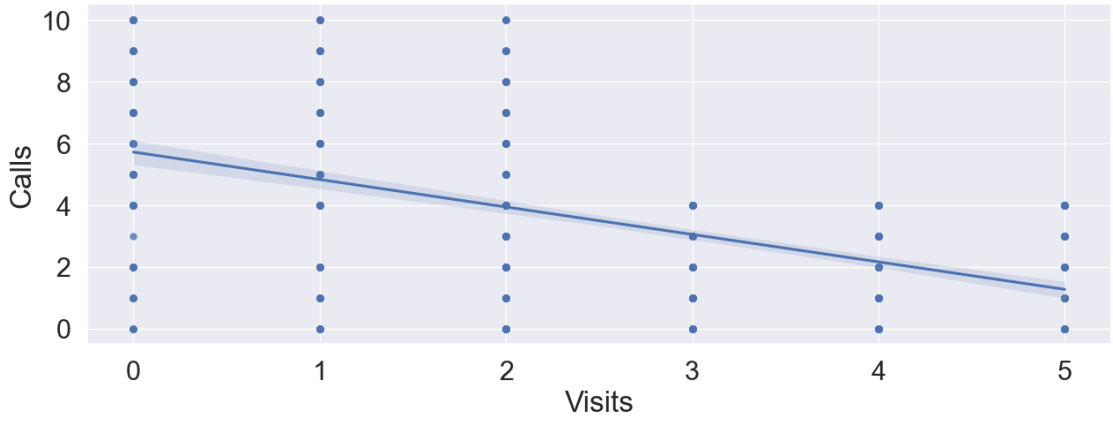
- We can see an absence of data at the top right, meaning customers with many cards dont make many calls
- There is negative correlation between these two variables as the trendline drops as it moves right

In [50]:
```
#Visits vs Online

figure = plt.figure(figsize=(15,5)) # adding a trend line will give us a b

sns.regplot(x="Visits", y="Online", data=BD);
```



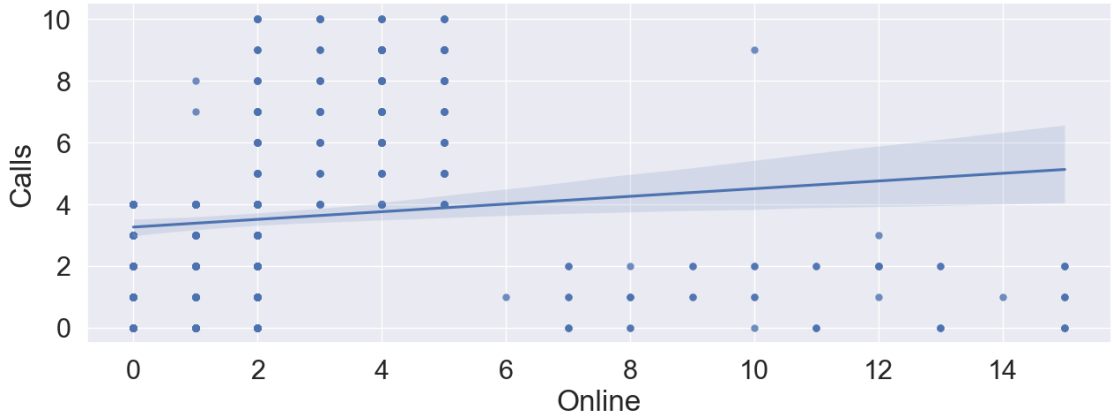
- We can see an absence of data at the top right, meaning customers who usually login online do not make many visits to the bank
- There is negative correlation between these two variables as the trendline drops as it moves right

In [51]:
```
#Visits vs Calls

figure = plt.figure(figsize=(15,5)) # adding a trend line will give us a b

sns.regplot(x="Visits", y="Calls", data=BD);
```



- We can see an absence of data at the top right, meaning customers who make many visits to the bank dont make many calls
- There is negative correlation between these two variables as the trendline drops as it moves right

In [52]:
```
#Online vs Calls

figure = plt.figure(figsize=(15,5)) # adding a trend line will give us a b

sns.regplot(x="Online", y="Calls", data=BD);
```



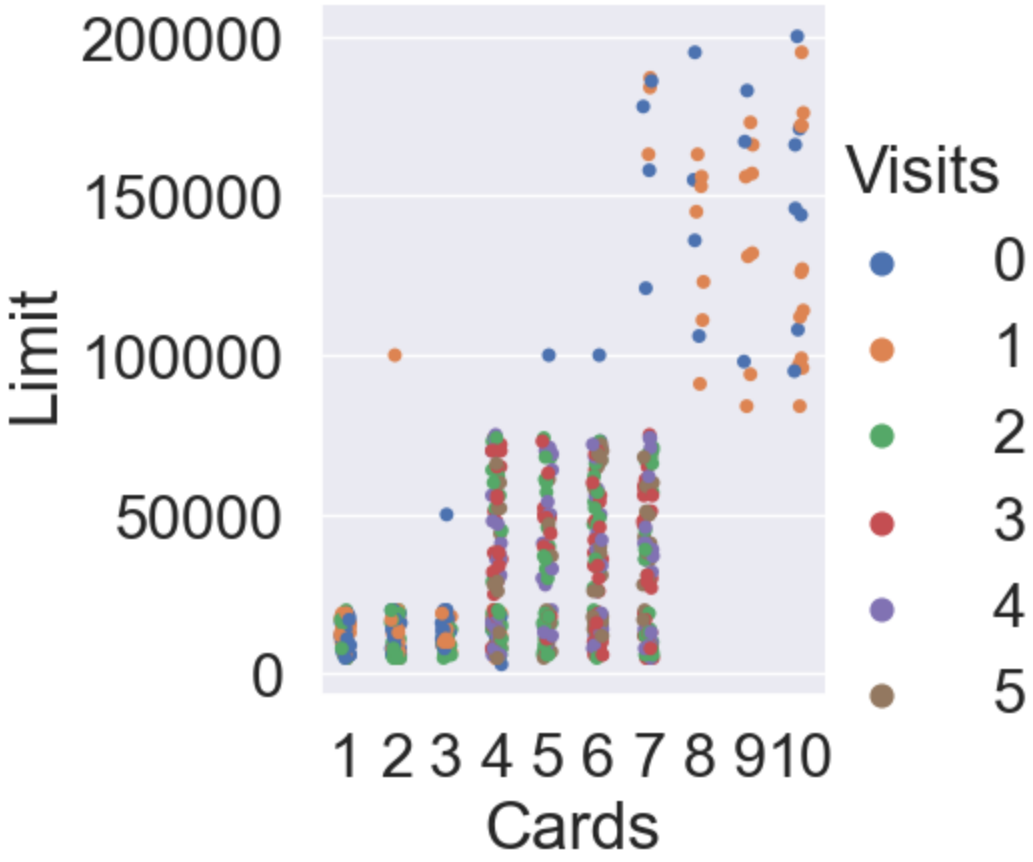
- We can see an absence of data at the top right, meaning customers who make many calls to the bank dont login online
- We can see clustering occurring on the bottom left and the top left between 4+ calls and 2-4 online logins
- We can see more clustering at the bottom right

- There is weak positive correlation between these two variables as the trendline rises slightly as it moves right

Multivariate Analysis on Strong Correlated Variables

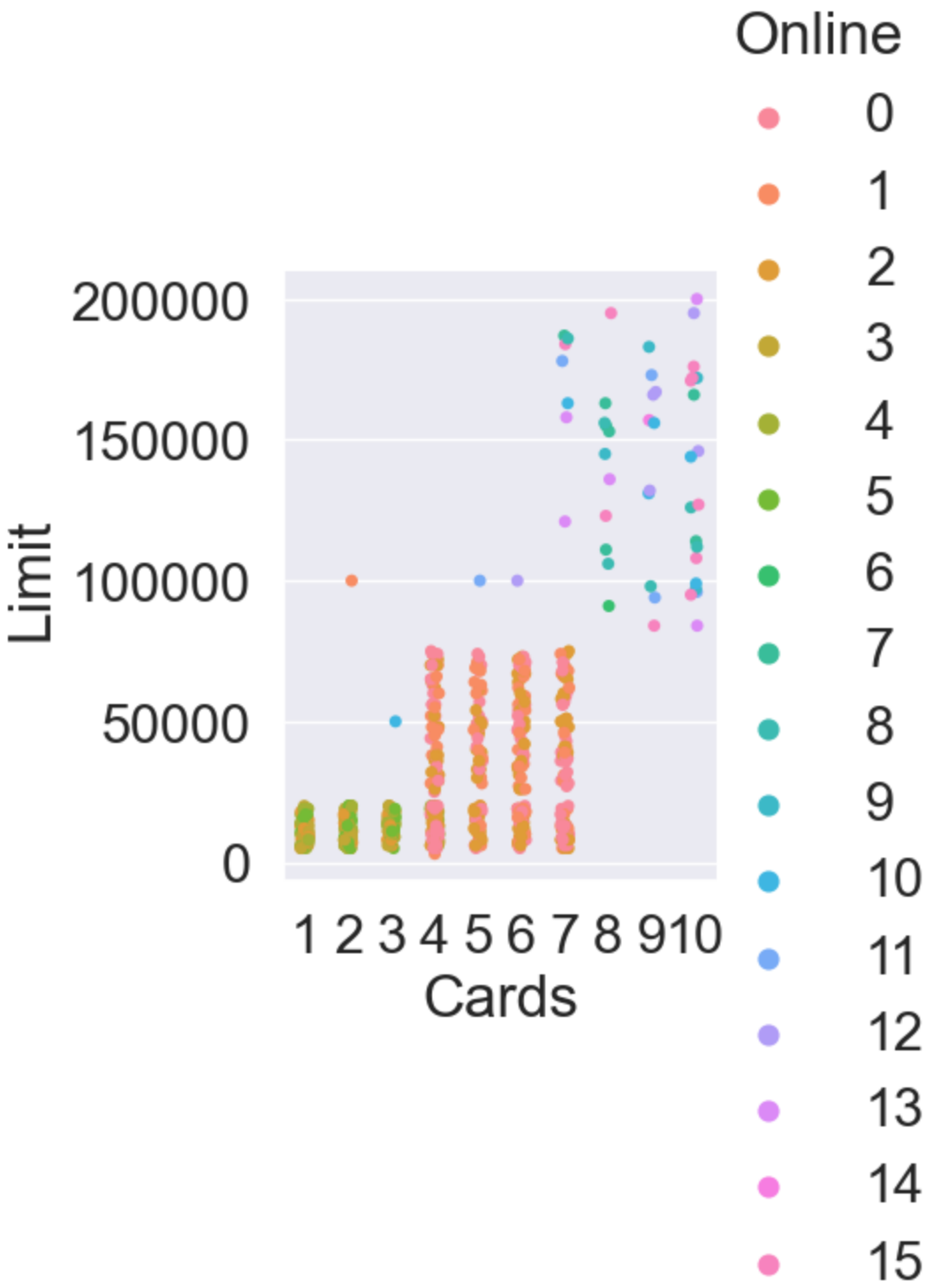
Strong Positive

In [53]:
```
#Cards vs Limit, hue = Visits

sns.catplot(x="Cards",
                y = "Limit",
                hue="Visits",
                data=BD,
                kind="strip");
```



- We saw the two distinct clusters before, one in the bottom left and the other in the top right
- Overlaying these clusters with the number of visits hue shows that 0 and 1 visits occur primarily in the top right cluster
- 2 - 5 visits occur primarily in the bottom left cluster

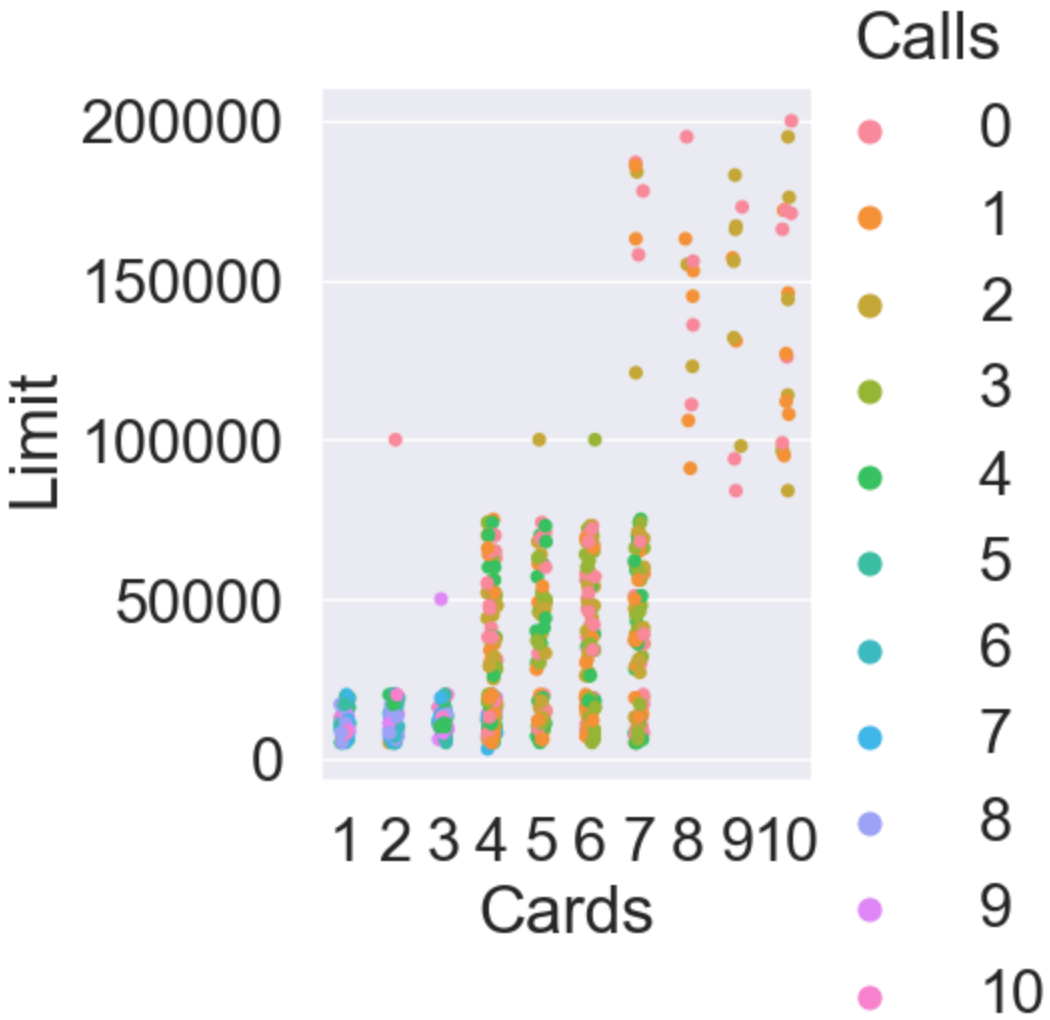
In [54]:
```
#Cards vs Limit, hue = Online

sns.catplot(x="Cards",
                y = "Limit",
                hue="Online",
                data=BD,
                kind="strip");
```



- We saw the two distinct clusters before, one in the bottom left and the other in the top right

- Overlaying these clusters with the number of online login hue shows that 10+ visits occur primarily in the top right cluster
- 0 - 9 visits occur primarily in the bottom left cluster
- It is difficult to see, but it may be the case that vists 2-5 are clustered further at the very bottom left while 0-1 online logins occurs more frequently from cards 4-7

In [55]:
```
#Cards vs Limit, hue = Calls

sns.catplot(x="Cards",
                y = "Limit",
                hue="Calls",
                data=BD,
                kind="strip");
```



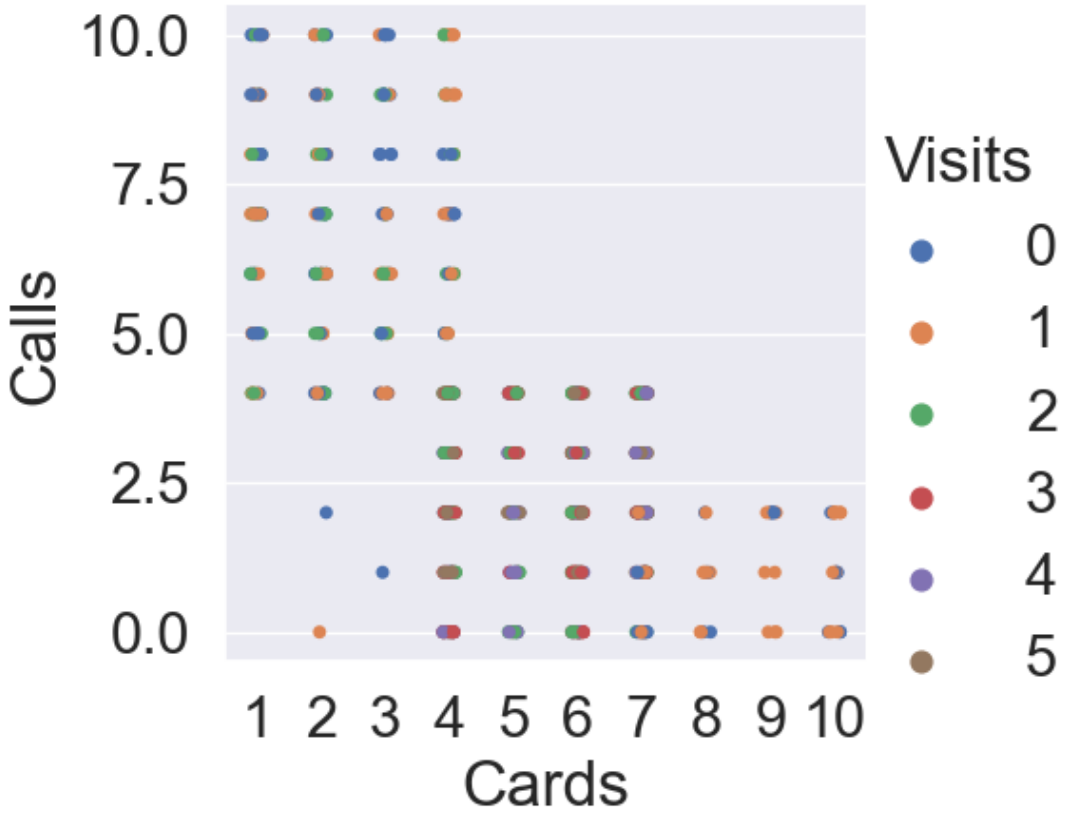
- We saw the two distinct clusters before, one in the bottom left and the other in the top right
- Overlaying these clusters with the number of calls hue shows that 6+ calls occur primarily in the bottom left cluster
- 0 - 5 calls seem to occur from 4 to 10 cards
- From this visualization, it appears there may be three distinct clusters, instead of two

Strong Negative

In [56]:
```
#Cards vs Calls, hue = Limit

sns.catplot(x="Cards",
                y = "Calls",
                hue="Limit",
                data=BD,
                kind="strip");
```

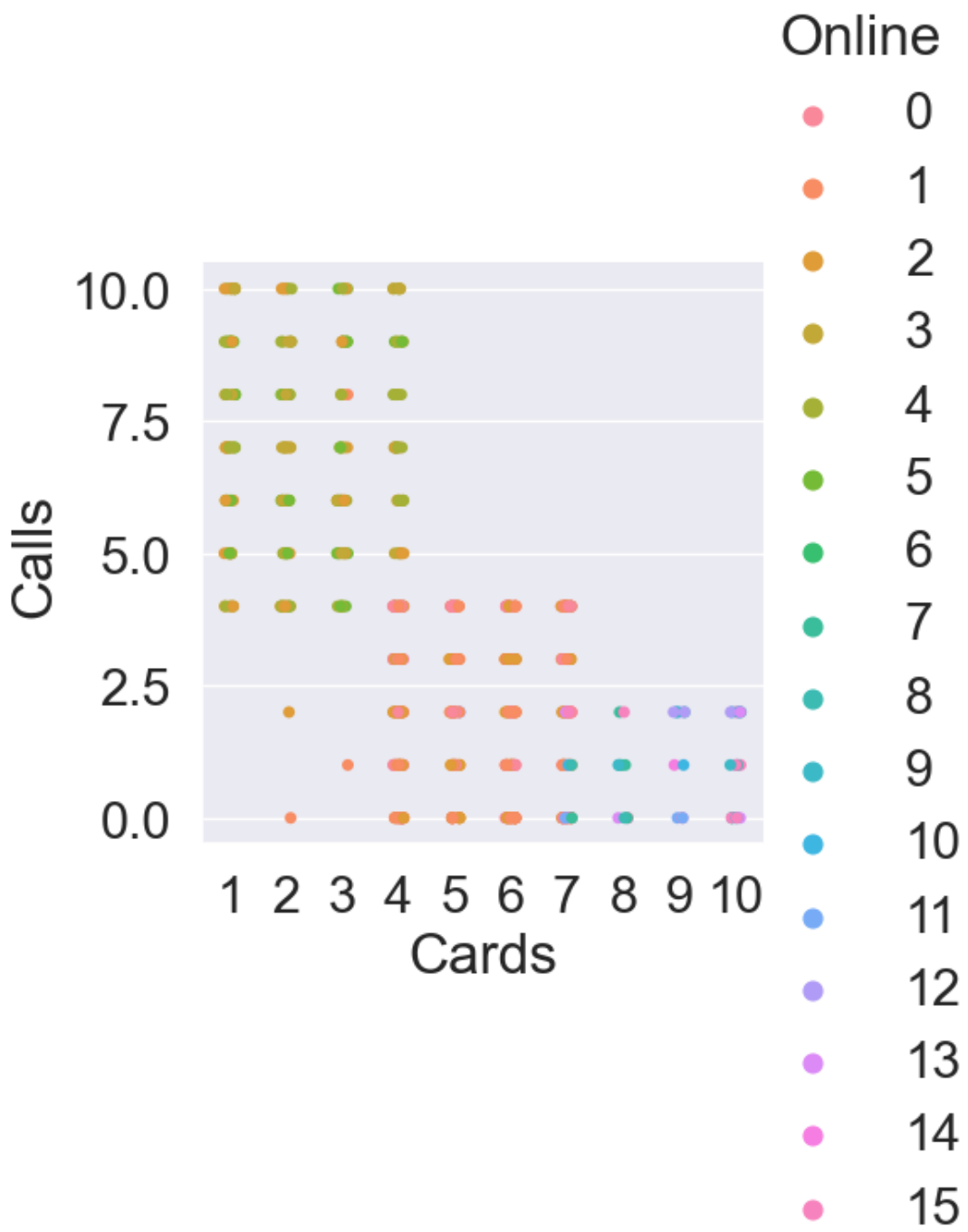
- This visualization is a bit trickier to interpret because there are so many distinct Limit values
- We can see that there are three distinct clusters: top left, middle bottom, and bottom right
- It seems the orange values occur more in the top left, greens and blues in the bottom middle and blues and purples/pink in the bottom right
- This would indicate that customers who make the most calls, have the fewest cards and the lowest credit limits
- Customers with average cards (4-7) make fewer calls (0-4) and have average credit limits
- Customers with high credit limits have many cards (8+) and make very few calls (0-2)

In [57]:
```
#Cards vs Calls, hue = Visits

sns.catplot(x="Cards",
                y = "Calls",
                hue="Visits",
                data=BD,
                kind="strip");
```



- We can see that there are three distinct clusters: top left, middle bottom, and bottom right
- Its not really clear if the number of visits correspond to the distinct clusters, because it appears the different colours bleed into all three custers
- It looks like customers with high credit and cards (bottom right) make very few visits (0-1), the bottom middle cluster makes many visits (3-5) and the top left cluster makes few visits as well (0-2)

In [58]:
```
#Cards vs Calls, hue = Online

sns.catplot(x="Cards",
                y = "Calls",
                hue="Online",
                data=BD,
                kind="strip");
```



- We can see that there are three distinct clusters: top left, middle bottom, and bottom right
- It seems the green/orange values occur more in the top left, orange/pink in the bottom middle and blues and purples in the bottom right

- This would indicate that customers who make the most calls, have the fewest cards and the login online 2-5 times
- Customers with average cards (4-7) make fewer calls (0-4) and login online 0-2 times

Part 7: Data Preprocessing

Transform the Data

One of the first steps before clustering is to scale the data. The reason for this, is because clustering takes a measure of the distances between data points to compute which cluster an individual customer belongs to. However, the data in the dataset is different orders of magnitude, measured in different units. For example, Cards range from 1 to 10 and are virtually unitless. However, credit limit ranges from 3,000 to 200,000. This variable would dominate the other variables. For this reason, all variables are brought to the same scale using scaling.

In [60]:
```
BD1 = BD.copy() # This creates a copy of the dataframe so that the origina
```

In [61]:
```
# selecting numerical columns
num_col = BD1.select_dtypes(include=np.number).columns.tolist()
num_col
```

Out[61]:
```
['Limit', 'Cards', 'Visits', 'Online', 'Calls']
```

In [62]:
```
# scaling the dataset before clustering; BD1 will be used for K-means clus
scaler = StandardScaler()
subset = BD1[num_col].copy()
subset_scaled = scaler.fit_transform(subset)
```

In [63]:
```
# creating a dataframe of the scaled columns; this will be used for K-mean
subset_scaled_BD1 = pd.DataFrame(subset_scaled, columns=subset.columns)
```

In [64]:
```
# creating a second dataframe of the scaled columns; this will be used for
subset_scaled_BD2 = pd.DataFrame(subset_scaled, columns=subset.columns)
```

In [65]:
```
BD2 = BD1.copy() # This creates a copy of the scaled dataframe, BD2 will b
```

We created two copies of a scaled version of the BD dataset. The reason for this, is because, during K-means Clustering we want to append a column to the dataset which will label to which cluster the individual customer has been assigned. If we use this altered dataframe for Hierarchical Clustering, it will include the cluster label when it attempts to measure distances between the rows of data. To prevent this, we will simply use a copy of the scaled BD dataset.

Part 8: K-means Clustering

K-means clustering is an unsupervised learning algorithm which seeks to find groups by assigning the data points to clusters on the basis of their similarity. Data points in the same cluster share many similarities. By increasing the value of k, more and more clusters are formed by dividing the data points based on their differences from one another. The objective is to optimize the number of clusters by finding the appropriate k value.

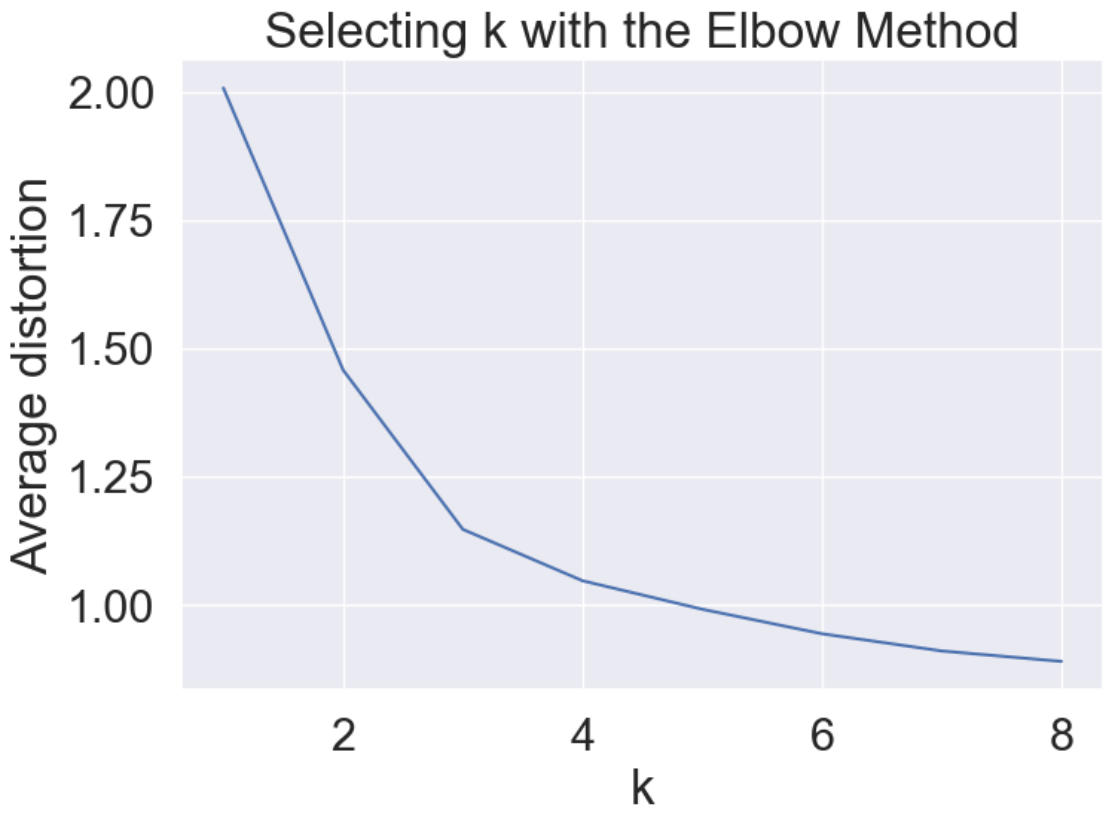
Elbow Method

The most popular and well-known method to find the optimal number of clusters is the elbow method. We shall plot the values of the cost function against different values of k and where the distortion declines most - the slope of the line becomes more horizontal than vertical - will give us a good indication of the optimal number of clusters for the dataset.

In [66]:

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

for k in clusters:
    model = KMeans(n_clusters=k)
    model.fit(subset_scaled_BD1)
    prediction = model.predict(subset_scaled_BD1)
    distortion = (
        sum(
            np.min(cdist(subset_scaled_BD1, model.cluster_centers_, "eucli
        )
        / subset_scaled_BD1.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: 0.9430843103448057
Number of Clusters: 7   Average Distortion: 0.9095680879600576
Number of Clusters: 8   Average Distortion: 0.8891818736424425
```



- We see the elbow shape of the curve as expected, with an inflection point at 3
- This would suggest that an appropriate value for k would be 3
- However, we must be careful not to make the clusters too big or else we may lose some important information
- Therefore, an appropriate value for k could be 3 or 4
- We can check the Silhouette Scores to help us choose the appropriate value for k

Silhouette Score

The Silhouette score is another metric which indicates the goodness of the clustering algorithms. The Silhouette score calculates a value between -1 and +1 based on the formula: (b-a)/max((a,b), where a is the average distance between i and all of the other points in its own cluster, b is the distance between i and its next nearest cluster centroid.

- A score of 1 indicates tight, well separated clusters
- A score of 0 indicates clusters not well separated
- A score of -1 indicates data points of a cluster is closer to the centroid of other clusters than to the centroid of its own cluster

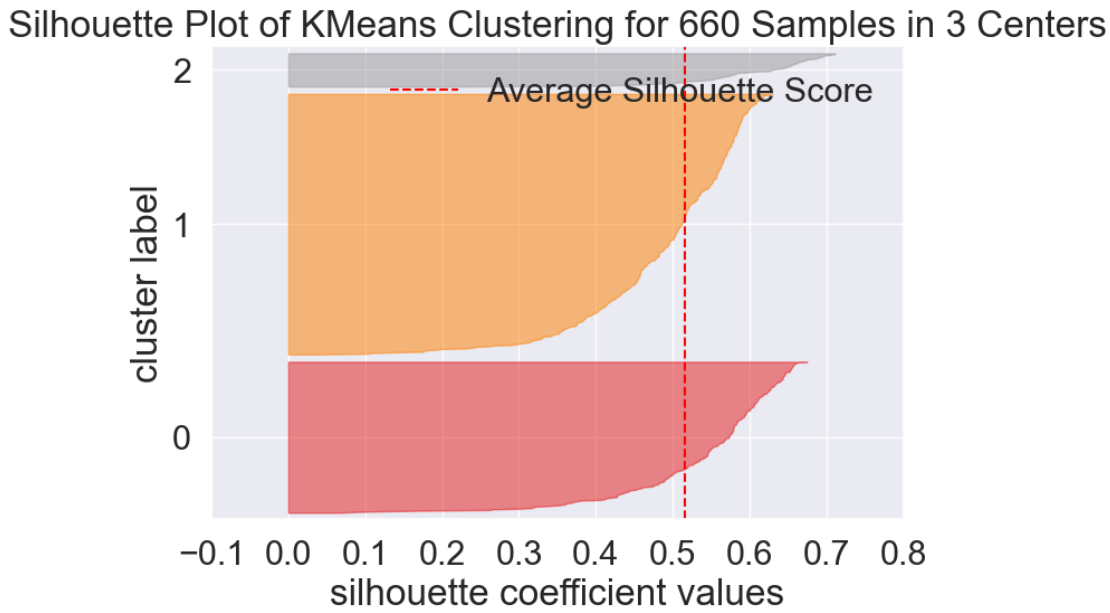
In [67]:
```
sil_score = []
cluster_list = list(range(2, 10))
for n_clusters in cluster_list:
    clusterer = KMeans(n_clusters=n_clusters)
    preds = clusterer.fit_predict((subset_scaled_BD1))
    # centers = clusterer.cluster_centers_
    score = silhouette_score(subset_scaled_BD1, preds)
    sil_score.append(score)
    print("For n_clusters = {}, the silhouette score is {})".format(n_clus

plt.plot(cluster_list, sil_score)
plt.xlabel("n-clusters")
plt.ylabel("Silhouette score")
plt.title("Selecting k with Silhouette Scores")
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 0.2717470361089752)
For n_clusters = 6, the silhouette score is 0.2560368034254374)
For n_clusters = 7, the silhouette score is 0.2486042710835614)
For n_clusters = 8, the silhouette score is 0.2265820806471936)
For n_clusters = 9, the silhouette score is 0.2180598457057277)
```

- Where n = 3, the Silhouette score is the highest at 0.52
- The above graph shows us that the Silhouette score for 3 is higher than that for 4
- There is a substantial drop between the score at 3 and the score at 4
- For this reason, we will choose 3 as value of k

- We can plot the optimal number of clusters with Silhouette coefficients where k = 3 and k = 4 to visualize the clustering...

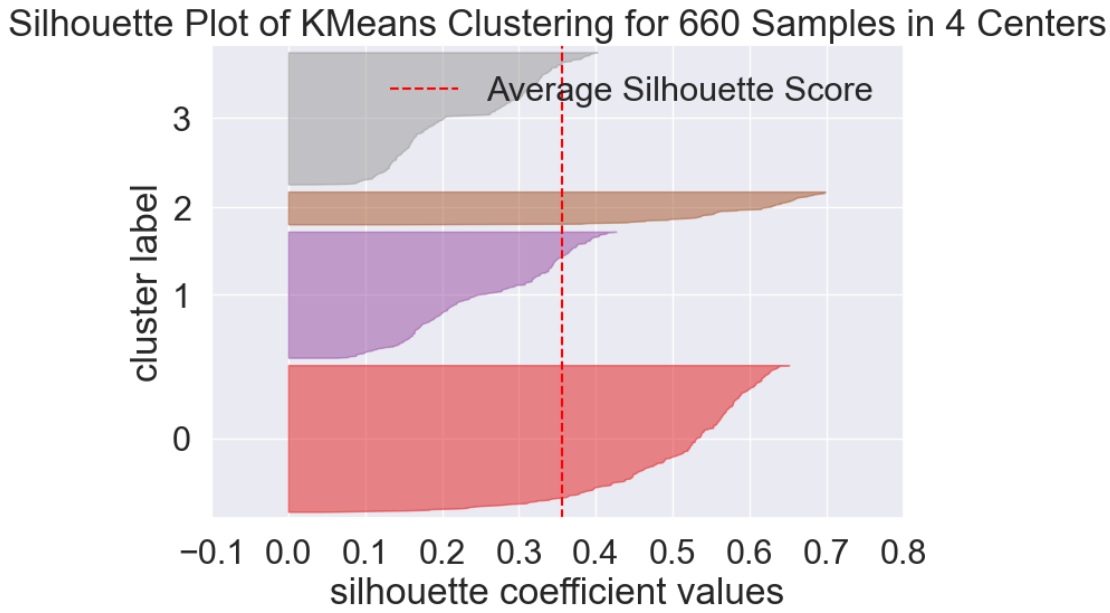
In [68]:
```
# Plotting the K-means Clustering for 660 samples using K = 3

visualizer = SilhouetteVisualizer(KMeans(3, random_state=1))
visualizer.fit(subset_scaled_BD1)
visualizer.show()
```



Out[68]:
```
<AxesSubplot:title={'center':'Silhouette Plot of KMeans Clustering for 6
60 Samples in 3 Centers'}, xlabel='silhouette coefficient values', ylabe
l='cluster label'>
```

- The average Silhouette score is indicated by the broken red line
- For k = 3, the average is about 0.5
- We can now graph k = 4 and compare the average Shilhouette score to verify that k = 3 is a better choice in this case...

In [69]:
```
# Plotting the K-means Clustering for 660 samples using K = 4

visualizer = SilhouetteVisualizer(KMeans(4, random_state=1))
visualizer.fit(subset_scaled_BD1)
visualizer.show()
```



Out[69]:
```
<AxesSubplot:title={'center':'Silhouette Plot of KMeans Clustering for 660 Samples in 4 Centers'}, xlabel='silhouette coefficient values', ylabel='cluster label'>
```

- We can that for k = 4, the average Shilhouette score is about 0.35
- This average is LOWER than the average where k = 3
- This reinforces our conclusion that we should set k = 3...

In [70]:
```
# Setting the number of clusters to 3 (k = 3)

kmeans = KMeans(n_clusters=3, random_state=0)
kmeans.fit(subset_scaled_BD1)
```

Out[70]:
```
KMeans(n_clusters=3, random_state=0)
```

In [71]:
```
# adding kmeans cluster labels to the original and scaled dataframes

BD1["K_Clusters"] = kmeans.labels_
subset_scaled_BD1["K_Clusters"] = kmeans.labels_
```

Observations

- Using K-means clustering, we were able to divide AllBank's customers into 3 distinct clusters
- We used the Elbow method, the most popular method for finding k, and the plotting of the values of the cost function against different values of k showed that the optimal number of

clusters was 3
- We then calculated the Silhouette scores to verify our findings using the Elbow method
- Again, where n = 3, we saw the highest Silhouette score
- We then graphed the Sihouette coefficient values vs the cluster labels to give us a visualization
- We could see from the two graphs that, when n = 4, one of the three clusters from n = 3 is simply divided
- This tells us that the similarities in the cluster that was divided are not incredibly strong
- However, when the data is divided into 4 clusters, we see a drop in the Silhouette score of over 31%, indicating the clusters beginning to become not well separated
- Together, all of these factors influenced our decision to set k = 3

Part 9: Hierarchical Clustering

Hierarchical clustering can give us insight into the distribution of the bank's customers. We can understand customer behaviour, including spending patterns and past interactions with the bank. It allows us to segment the entire pool of customers into smaller groups, which may respond differently to various advertising techniques. We can formulate precise advertising, marketing and logistics mechanisms based on the individual needs of each cluster.

To begin building our Hierarchical clustering model, we can calculate the cophenetic correlations between the Euclidean distance (or other distance measure) and dendrogram distance for a particular dendrogram of all possible pair points...

In [72]:
```
# 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_BD2, metric=dm, method=lm)
        c, coph_dists = cophenet(Z, pdist(subset_scaled_BD2))
        print(
            "Cophenetic correlation for {} distance and {} linkage is {}."
                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 0.73
91220243806552.
Cophenetic correlation for Euclidean distance and complete linkage is 0.
8599730607972423.
Cophenetic correlation for Euclidean distance and average linkage is 0.8
977080867389372.
Cophenetic correlation for Euclidean distance and weighted linkage is 0.
8861746814895477.
Cophenetic correlation for Chebyshev distance and single linkage is 0.73
82354769296767.
Cophenetic correlation for Chebyshev distance and complete linkage is 0.
8533474836336782.
Cophenetic correlation for Chebyshev distance and average linkage is 0.8
974159511838106.
Cophenetic correlation for Chebyshev distance and weighted linkage is 0.
8913624010768603.
Cophenetic correlation for Mahalanobis distance and single linkage is 0.
7058064784553606.
Cophenetic correlation for Mahalanobis distance and complete linkage is
0.5422791209801747.
Cophenetic correlation for Mahalanobis distance and average linkage is
0.8326994115042134.
Cophenetic correlation for Mahalanobis distance and weighted linkage is
0.7805990615142516.
Cophenetic correlation for Cityblock distance and single linkage is 0.72
52379350252723.
Cophenetic correlation for Cityblock distance and complete linkage is 0.
8731477899179829.
Cophenetic correlation for Cityblock distance and average linkage is 0.8
96329431104133.
Cophenetic correlation for Cityblock distance and weighted linkage is 0.
8825520731498188.
```

In [73]:
```
# printing the combination of distance metric and linkage method with the
print(
    "Highest cophenetic correlation is {}, which is obtained with {} dista
        high_cophenet_corr, high_dm_lm[0].capitalize(), high_dm_lm[1]
    )
)
```

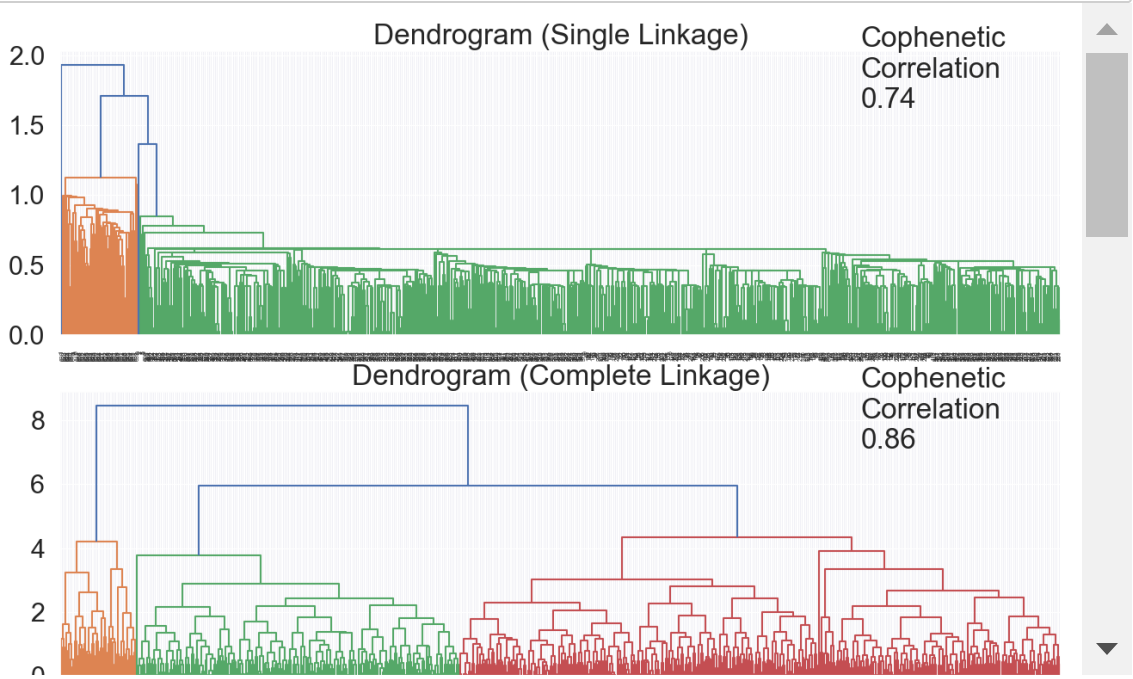
```
Highest cophenetic correlation is 0.8977080867389372, which is obtained
with Euclidean distance and average linkage.
```

Observations

- We see that the cophenetic correlation is maximum with Euclidean distance and average linkage at 0.8977
- The cophenetic correlation with average linkage is generally the maximum for each type of measurement

Dendrograms can be used to represent the distances at which the different clusters meet. They provide a visualization of the distance between various clustering and easily show which clusters are closest together. We can use the cophenetic correlation scores we calculated above to construct the dendrograms...

In [74]:
```
# list of linkage methods
linkage_methods = ["single", "complete", "average", "centroid", "ward", "w

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

# 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 c
for i, method in enumerate(linkage_methods):
    Z = linkage(subset_scaled_BD2, 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_BD2))
    axs[i].annotate(
        f"Cophenetic\nCorrelation\n{coph_corr:0.2f}",
        (0.80, 0.80),
        xycoords="axes fraction",
    )

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



Observations

- Dendrograms for Complete linkage and Ward linkage shows 3 distinct and separate clusters.
- The others show only 2

Comparing Cophenetic correlations for each linkage method

In [75]:
```
BD_cc = pd.DataFrame(compare, columns=compare_cols)
BD_cc
```

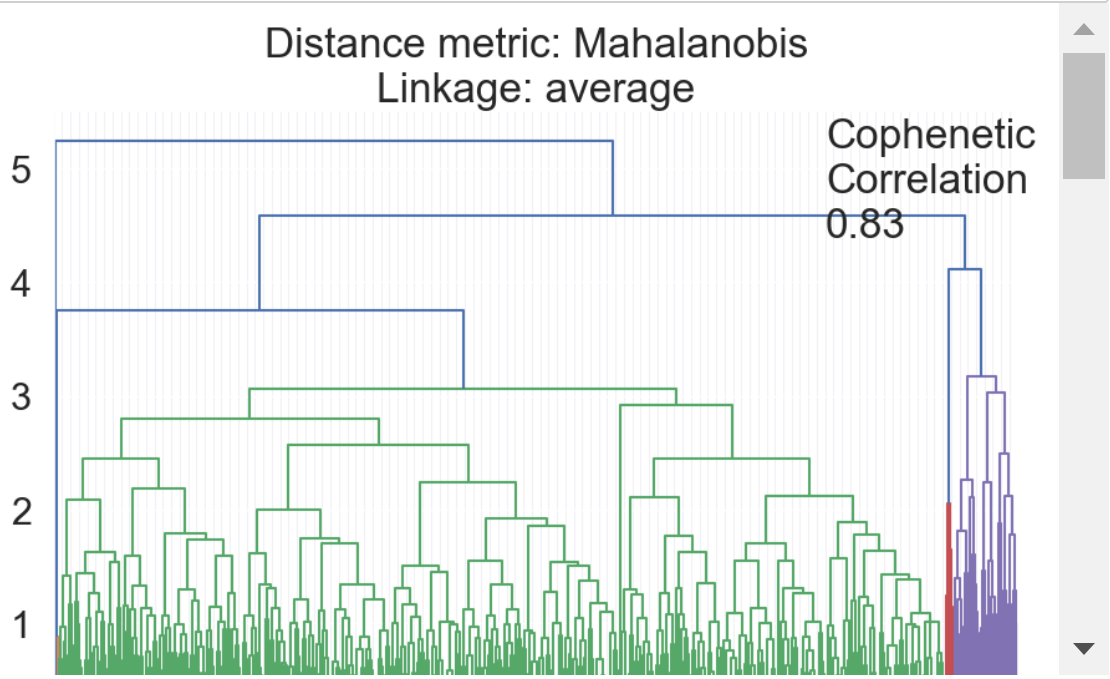
Out[75]:

	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

The above table organizes all the Cophenetic coefficients and allows us to see that Euclidean distance using average linkage has the highest score, at 0.8977. However, the dendrogram constructed using this method only has 2 clusters that are not visibly separate and distinct. We should, therefore, select a dendrogram with the highest Cophenetic coefficient AND has visibly separate and distinct clusters. The dendrogram that best matches this criteria is Euclidean distance using complete linkage. The six dendrograms above were all constructed using Euclidean distances. We can also create dendrograms for Mahalanobis and

Manhattan distances with average and weighted linkage methods to compare...

In [76]:
```
# list of distance metrics
distance_metrics = ["mahalanobis", "cityblock"]

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

# to create a subplot image
fig, axs = plt.subplots(
    len(distance_metrics) + len(distance_metrics), 1, figsize=(10, 30)
)

i = 0
for dm in distance_metrics:
    for lm in linkage_methods:
        Z = linkage(subset_scaled_BD2, metric=dm, method=lm)

        dendrogram(Z, ax=axs[i])
        axs[i].set_title("Distance metric: {}\nLinkage: {}".format(dm.capi

        coph_corr, coph_dist = cophenet(Z, pdist(subset_scaled_BD2))
        axs[i].annotate(
            f"Cophenetic\nCorrelation\n{coph_corr:0.2f}",
            (0.80, 0.80),
            xycoords="axes fraction",
        )
        i += 1
```



Observations

- We already knew that the Euclidean method gave us higher Cophenetic coefficients, so we would naturally choose the best Euclidean model unless something very unexpected happened in the above 4 dendrograms. We see that the two that give us the highest correlation scores only have two clusters, whereas the 2 that give us 3 and 4 clusters, have lower coefficients than our best Euclidean dendrogram

- We will use 3 clusters to construct our model using the Euclidean distance with complete linkage method

Building a Model with 3 Clusters.

In [77]:
```
HCmodel = AgglomerativeClustering(n_clusters=3, affinity="euclidean", link
HCmodel.fit(subset_scaled_BD2)
```

Out[77]:
```
AgglomerativeClustering(linkage='complete', n_clusters=3)
```

In [78]:
```
# adding hierarchical cluster labels to the original and scaled dataframes

subset_scaled_BD2["HC_Clusters"] = HCmodel.labels_
BD2["HC_Clusters"] = HCmodel.labels_
```

Observations

- Using Hierarchical clustering, we were able to divide AllBank's customers into 3 distinct clusters
- We calculated the Cophenetic correlations using Euclidean, Chebyshev, Mahalanobis and Cityblock distances via simple, complete, average and weighted distances and found that Euclidean distance with average linkage gave us the highest Cophenetic correlation score
- We then used these scores to construct dendrograms representing the different ways Hierarchical clustering could cluster the customer data
- Six Euclidean dendrograms were created. 4 of these 6 showed only 2 clusters, while the remaining 2 showed 3 clusters.
- 2 Cityblock dendrograms were created. 1 showed only 2 clusters while the other showed 3 clusters.
- 2 Mahalanobis dendrograms were created. 1 showed 3 clusters while another showed 4 clusters.
- The dendrograms with only 2 clusters were disregarded
- Of the dendrograms with 3 clusters only the Euclidean dendrograms showed distinct and separate clusters
- Of the dendrograms with 4 clusters, we didnt see distinct, separate clustering
- Therefore, 3 would be the appropriate number of the clusters, however, there are 2 good dendrograms giving us this information: Euclidean Dendrograms for Complete linkage and Ward linkage
- We will use the Cophenetic scores to break the tie: Complete linkage (0.86) has the higher score (Ward = 0.74)
- Therefore, our model will use 3 clusters using the dendrogram constructed with Euclidean distance with Compelete linkage method.

Part 10: Comparing K-means Clustering and Hierarchical Clustering

We constructed two different models, the first using K-means clustering and the second using Hierarchical clustering and both indicate that the optimal number of clusters to construct using AllBank's customer data is 3. At first glance, this may seem like the two models are identical, but this is not necessarily the case. Individual customers may be assigned to different clusters based on the way the two different models separate the data into segments. For example, customer 87073 may belong to cluster 1 in the K-means model but belong to cluster 2 in the Hierarchical clustering model. For this reason, we do not expect the frequencies of the clusters to match precisely between the two models.

Which model is better?

We will compare the two different clustering models to see if, in fact, one model is superior to the other. However, it may be the case that both models offer valuable insights and may actually complement one another. We will begin our comparison by profiling the models separately and then comparing our findings.

K-means Clustering Profiling

In [79]:
```
cluster_profile = BD1.groupby("K_Clusters").mean()
```

In [80]:
```
cluster_profile["Counts"] = (
    BD1.groupby("K_Clusters")["Limit"].count().values
)
```

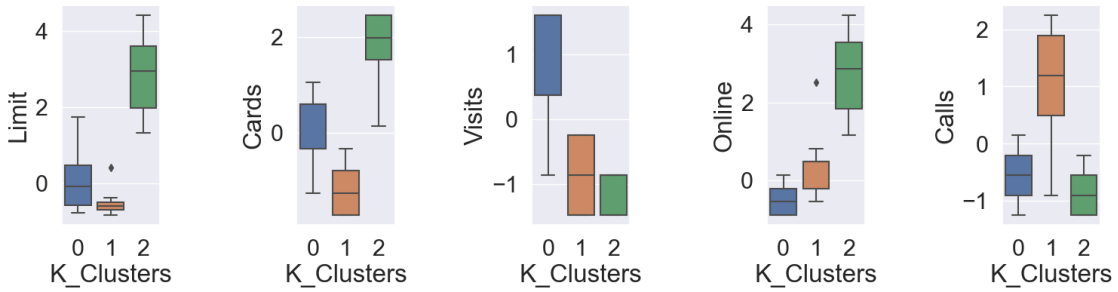
In [81]:
```
# let's display cluster profiles
cluster_profile.style.highlight_max(color="lightgreen", axis=0)
```

Out[81]:

K_Clusters	Limit	Cards	Visits	Online	Calls	Counts
0	33782.383420	5.515544	3.489637	0.981865	2.000000	386
1	12174.107143	2.410714	0.933036	3.553571	6.870536	224
2	141040.000000	8.740000	0.600000	10.900000	1.080000	50

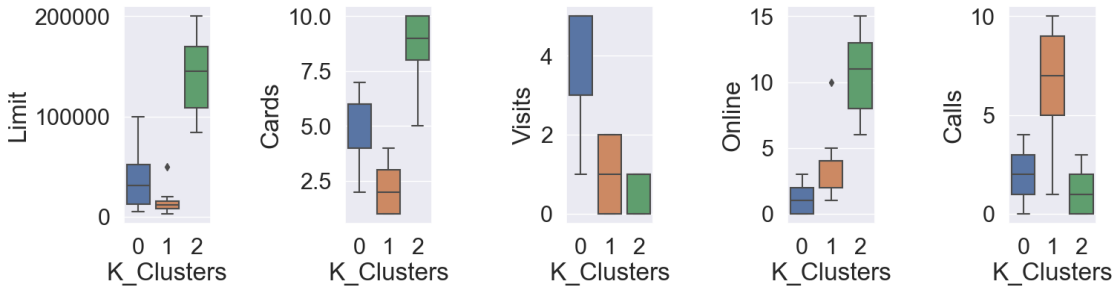
In [82]:
```
fig, axes = plt.subplots(1, 5, figsize=(16, 6))
fig.suptitle("Boxplot of scaled numerical variables for each K-means clust
counter = 0
for ii in range(5):
    sns.boxplot(
        ax=axes[ii],
        y=subset_scaled_BD1[num_col[counter]],
        x=subset_scaled_BD1["K_Clusters"],
    )
    counter = counter + 1

fig.tight_layout(pad=2.0)
```

Boxplot of scaled numerical variables for each K-means cluster



In [83]:
```
fig, axes = plt.subplots(1, 5, figsize=(16, 6))
fig.suptitle("Boxplot of original numerical variables for each K-means clu
counter = 0
for ii in range(5):
    sns.boxplot(ax=axes[ii], y=BD1[num_col[counter]], x=BD1["K_Clusters"])
    counter = counter + 1

fig.tight_layout(pad=2.0)
```

Boxplot of original numerical variables for each K-means cluster



Business Insights

- **Cluster 0**:
  - This cluster appears to contain more of the 'middle of the pack' customers in terms of spending patterns

    - Their credit limit is close to the mean of 34,574.24 dollars and the number of cards in their possession is slightly higher than both the mean (4.7 cards) and the median (5 cards)
    - We can see that this segment prefers to deal with the bank in face-to-face interactions since the number of bank visits is much higher than the mean (2.4) or median (2.0)
    - Customers in this cluster are the least likely to use online banking, falling quite a bit below the mean (2.61) and the median (2.0)
    - Although this group isnt the least likely to call the customer service department, it still falls below both the mean (3.58) and the median (3.0)
    - This cluster contains the largest number of customers, at 58% of the total number of customers in the dataset
- **Cluster 1**:
    - Cluster 1 appears to contain the bottom rung of customers in terms of spending patterns
    - These customers are well below the average credit limit, and possess almost half the number of cards as the total customer average (5 cards)
    - This cluster is the biggest user of the bank's call center for customer support.
    - These customers call the bank, on average, nearly twice as much as the total customer mean (3.58)
    - They are almost 3.5x more likely to call the bank than segment 0 and 6.8x more likely to call the bank than segment 2
    - Customers in cluster 1 are using online banking slightly more than average (2.61) but they tend to visit the bank less often than segment 1
    - This cluster contains the second most number of customers at roughly 34% of the total number of customers in the dataset
- **Cluster 2**:
    - Cluster 2 appears to contain the top tier of customers in terms of spending patterns
    - These customers are well above the average credit limit, and possess almost double the number of cards as the total customer average (5 cards)
    - This cluster is the biggest user of the bank's online customer support service, using it almost 4x the total average (2.61)
    - These customers rarely call the bank, falling substantially below total customer mean (3.58)
    - Customers in cluster 2 are the least likely to visit the bank
    - This cluster contains the smallest number of customers at roughly 8% of the total number of customers in the dataset

K-means Clustering vs Spending Patters

We can further examine spending patterns of customers by examining how Credit Limit and Number of Cards were grouped by the model

Credit Limit by Cluster

In [84]:
```
pd.crosstab(BD1.K_Clusters, BD1.Limit).style.highlight_max(
    color="lightgreen", axis=0
)
```

Out[84]:

Limit	3000	5000	6000	7000	8000	9000	10000	11000	12000	13000	14000	1500
K_Clusters												
0	0	8	14	11	15	10	12	9	10	10	9	
1	1	13	17	13	20	18	14	15	8	18	14	1
2	0	0	0	0	0	0	0	0	0	0	0	

Observations

- For the most part, the division by credit limit follows the break we noticed above
- Most customers with credit limits below 25,000 belong to Cluster 1
- Most customers with credit limits between 25,000 and 75,000 belong to cluster 0
- Almost all customers with credit limits above 75,000 belong to cluster 2

Number of Cards by Cluster

In [85]:
```
pd.crosstab(BD1.K_Clusters, BD1.Cards).style.highlight_max(
    color="lightgreen", axis=0
)
```

Out[85]:

Cards	1	2	3	4	5	6	7	8	9	10
K_Clusters										
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

Observations

- We can see a nice, clean break based on the number of cards a customer has
- For segment 0, almost all the customers have between 4 and 7 cards
- For segment 1, we can see that no customer has more than 4 cards
- We see that there is overlap between cluster 0 and cluster 1 at 4 cards; 2/3 of customers with 4 cards belong to cluster 0 whereas 1/3 belong to cluster 1. We can infer that there were other reasons customers with 4 cards were divided this way
- Cluster 2 contains all the customers with 8+ cards. Again, we can see there is some overlap at 7 cards, but almost 95% of the customers with 7 cards belong to cluster 0

K-Means Clustering vs Interaction with Bank

We can further examine how customers interact with the bank by examining how Total Visits, Total Online Visits and Total Calls were grouped by the model

Total Visits by Cluster

In [86]:
```
pd.crosstab(BD1.K_Clusters, BD1.Visits).style.highlight_max(
    color="lightgreen", axis=0
)
```

Out[86]:

Visits	0	1	2	3	4	5
K_Clusters						
0	0	3	93	100	92	98
1	80	79	65	0	0	0
2	20	30	0	0	0	0

Observations

- We can see from how the clusters are broken down by Total Visits that this variable probably wasnt the most important feature when dividing the total group into segments
- We can wee that all customers who made more than 2 visits to the bank were placed in segment 0
- Customers who made two visits were divided between segment 0 and 1 in about a 60:40 ratio
- We can see that customers who made between 0 and 1 visit were divided between segments 1 and 2

Total Online Visits by Cluster

In [87]:
```
pd.crosstab(BD1.K_Clusters, BD1.Online).style.highlight_max(
    color="lightgreen", axis=0
)
```

Out[87]:

Online	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
K_Clusters																
0	144	106	135	1	0	0	0	0	0	0	0	0	0	0	0	0
1	0	3	54	43	69	54	0	0	0	0	1	0	0	0	0	0
2	0	0	0	0	0	0	1	7	6	4	5	5	6	5	1	10

Observations

- We can see some clean breaks in how the data was grouped
- Almost all customers who made 2 or fewer online logins belong to segment 0
- Most moderate users of online banking (3-5 logins) were placed in segment 1
- Almost all users who logged in 6 or more times belong to segment 2

Total Calls by Cluster

In [88]:
```
pd.crosstab(BD1.K_Clusters, BD1.Calls).style.highlight_max(
    color="lightgreen", axis=0
)
```

Out[88]:

Calls	0	1	2	3	4	5	6	7	8	9	10
K_Clusters											
0	81	74	72	82	77	0	0	0	0	0	0
1	0	1	1	0	31	29	39	35	30	32	26
2	16	15	18	1	0	0	0	0	0	0	0

Observations

- We can see from how the clusters are broken down by Total Callss that this variable probably wasnt the most important feature when dividing the total group into segments
- Only segments 0 and 1 contain a majority for each frequency of calls
- We see that all customers who make more than 4 calls belong to Segment 1
- However, we see overlap between different segments for 4 or fewer calls
- We know that segment 2 has the fewest number of customers, so its difficult to say that people who make 0-2 calls are more likely to be in segment 0
- Clearly, there are other factors considered when the segments were created

Hierarchical Clustering Profiling

In [89]:
```
cluster_profile2 = BD2.groupby("HC_Clusters").mean()
```

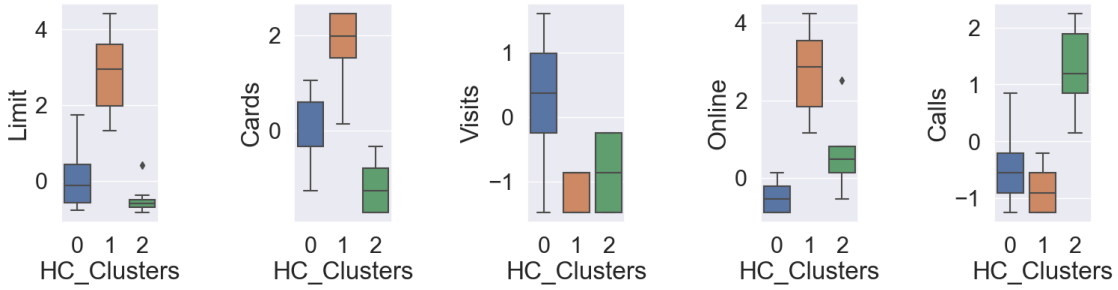
In [90]:
```
cluster_profile2["Counts"] = (
    BD2.groupby("HC_Clusters")["Limit"].count().values
)
```

In [91]:
```
# let's display cluster profiles
cluster_profile2.style.highlight_max(color="lightgreen", axis=0)
```

Out[91]:

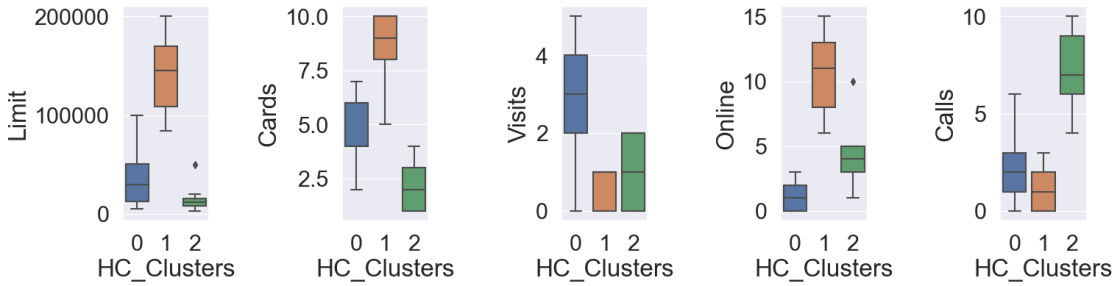
HC_Clusters	Limit	Cards	Visits	Online	Calls	Counts
0	33151.133501	5.460957	3.405542	1.010076	2.060453	397
1	141040.000000	8.740000	0.600000	10.900000	1.080000	50
2	12234.741784	2.352113	0.957746	3.633803	7.009390	213

In [92]:
```
fig, axes = plt.subplots(1, 5, figsize=(16, 6))
fig.suptitle("Boxplot of scaled numerical variables for each cluster", for
counter = 0
for ii in range(5):
    sns.boxplot(
        ax=axes[ii],
        y=subset_scaled_BD2[num_col[counter]],
        x=subset_scaled_BD2["HC_Clusters"],
    )
    counter = counter + 1

fig.tight_layout(pad=2.0)
```



In [93]:
```
fig, axes = plt.subplots(1, 5, figsize=(16, 6))
fig.suptitle("Boxplot of original numerical variables for each cluster", f
counter = 0
for ii in range(5):
    sns.boxplot(ax=axes[ii], y=BD2[num_col[counter]], x=BD2["HC_Clusters"]
    counter = counter + 1

fig.tight_layout(pad=2.0)
```

Boxplot of original numerical variables for each cluster



Business Insights

- **Cluster 0**:
  - This cluster appears to contain more of the 'middle of the pack' customers in terms of spending patterns
  - Their credit limit is close to the mean of 34,574.24 dollars and the number of cards in their possession is slightly higher than both the mean (4.7 cards) and the median (5 cards)
  - We can see that this segment prefers to deal with the bank in face-to-face interactions since the number of bank visits is much higher than the mean (2.4) or median (2.0)
  - Customers in this cluster are the least likely to use online banking, falling quite a bit below the mean (2.61) and the median (2.0)
  - Although this group is not the least likely to call the customer service department, it still falls below both the mean (3.58) and the median (3.0)
  - This cluster contains the largest number of customers, at 60% of the total number of customers in the dataset
- **Cluster 1**:
  - Cluster 1 appears to contain the top tier of customers in terms of spending patterns
  - These customers are well above the average credit limit, and possess almost double the number of cards as the total customer average (5 cards)
  - This cluster is the biggest user of the bank's online customer support service, using it almost 4x the total average (2.61)
  - These customers rarely call the bank, falling substantially below total customer mean (3.58)
  - Customers in cluster 1 are the least likely to visit the bank
  - This cluster contains the smallest number of customers at roughly 8% of the total number of customers in the dataset
- **Cluster 2**:

- Cluster 2 appears to contain the bottom rung of customers in terms of spending patterns
- These customers are well below the average credit limit, and possess almost half the number of cards as the total customer average (5 cards)
- This cluster is the biggest user of the bank's call center for customer support.
- These customers call the bank, on average, nearly twice as much as the total customer mean (3.58)
- They are almost 3.5x more likely to call the bank than segment 0 and 6.8x more likely to call the bank than segment 2
- Customers in cluster 2 are using online banking slightly more than average (2.61) but they tend to visit the bank less often than segment 0
- This cluster contains the second most number of customers at roughly 32% of the total number of customers in the dataset

Hierarchical Clustering vs Spending Patters

We can further examine spending patterns of customers by examining how Credit Limit and Number of Cards were grouped by the model

Credit Limit by Cluster

In [94]:
```
pd.crosstab(BD2.HC_Clusters, BD2.Limit).style.highlight_max(
    color="lightgreen", axis=0
)
```

Out[94]:

Limit	3000	5000	6000	7000	8000	9000	10000	11000	12000	13000	14000	150
HC_Clusters												
0	0	10	14	13	15	11	12	10	11	10	10	
1	0	0	0	0	0	0	0	0	0	0	0	
2	1	11	17	11	20	17	14	14	7	18	13	

Observations

- For the most part, the division by credit limit follows the break we noticed above
- Most customers with credit limits below 25,000 belong to Cluster 2
- No customer from cluster 2 has a credit limit above 25,000
- Most customers with credit limits between 25,000 and 75,000 belong to cluster 0, however, this cluster ranges from 5000 to 100,000
- Almost all customers with credit limits above 75,000 belong to cluster 1

Number of Cards by Cluster

In [95]:
```
pd.crosstab(BD2.HC_Clusters, BD2.Cards).style.highlight_max(
    color="lightgreen", axis=0
)
```

Out[95]:

HC_Clusters \ Cards	1	2	3	4	5	6	7	8	9	10
0	0	2	3	109	73	116	94	0	0	0
1	0	0	0	0	1	1	7	11	11	19
2	59	62	50	42	0	0	0	0	0	0

Observations

- We can see a nice, clean break based on the number of cards a customer has
- For segment 0, almost all the customers have between 4 and 7 cards
- For segment 2, we can see that no customer has more than 4 cards
- We see that there is overlap between cluster 0 and cluster 2 at 4 cards; 3/4 of customers with 4 cards belong to cluster 0 whereas 1/4 belong to cluster 2. We can infer that there were other reasons customers with 4 cards were divided this way
- Cluster 1 contains all the customers with 8+ cards. Again, we can see there is some overlap at 7 cards, but almost 95% of the customers with 7 cards belong to cluster 0

Hierarchical Clustering vs Interaction with Bank

We can further examine how customers interact with the bank by examining how Total Visits, Total Online Visits and Total Calls were grouped by the model

Total Visits by Cluster

In [96]:
```
pd.crosstab(BD2.HC_Clusters, BD2.Visits).style.highlight_max(
    color="lightgreen", axis=0
)
```

Out[96]:

HC_Clusters \ Visits	0	1	2	3	4	5
0	7	6	94	100	92	98
1	20	30	0	0	0	0
2	73	76	64	0	0	0

Observations

- We can see from how the clusters are broken down by Total Visits that this variable probably wasnt the most important feature when dividing the total group into segments

- We can wee that all customers who made more than 2 visits to the bank were placed in segment 0
- Customers who made two visits were divided between segment 0 and 2 in about a 60:40 ratio
- We can see that customers who made between 0 and 1 visit were divided between segments 1 and 2

Total Online Visits by Cluster

In [97]:
```
pd.crosstab(BD2.HC_Clusters, BD2.Online).style.highlight_max(
    color="lightgreen", axis=0
)
```

Out[97]:

Online	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
HC_Clusters																
0	144	107	144	2	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	1	7	6	4	5	5	6	5	1	10
2	0	2	45	42	69	54	0	0	0	0	1	0	0	0	0	0

Observations

- We can see some clean breaks in how the data was grouped
- Almost all customers who made 2 or fewer online logins belong to segment 0
- Most moderate users of online banking (3-5 logins) were placed in segment 2
- Almost all users who logged in 6 or more times belong to segment 1

Total Calls by Cluster

In [98]:
```
pd.crosstab(BD2.HC_Clusters, BD2.Calls).style.highlight_max(
    color="lightgreen", axis=0
)
```

Out[98]:

Calls	0	1	2	3	4	5	6	7	8	9	10
HC_Clusters											
0	81	75	73	82	80	5	1	0	0	0	0
1	16	15	18	1	0	0	0	0	0	0	0
2	0	0	0	0	28	24	38	35	30	32	26

Observations

- We can see from how the clusters are broken down by Total Callss that this variable probably wasnt the most important feature when dividing the total group into segments
- Only segments 0 and 2 contain a majority for each frequency of calls
- We see that all customers who make more than 4 calls belong to Segment 2

- However, we see overlap between different segments for 4 or fewer calls
- We know that segment 1 has the fewest number of customers, so its difficult to say that people who make 0-2 calls are more likely to be in segment 0
- Clearly, there are other factors considered when the segments were created

Comparing the Two Clustering Models

One of the good things about the way we constructed the models above is that we added labels to the dataframes. For the K-means clustering model, once an individual customer was assigned to a specific cluster, this assignment was recorded and appended to the dataframe in the column 'Segments'. The values 0, 1 or 2 were recorded in the dataframe to indicate which segment they were assigned to. Similarly, for the Hierarchical clustering model, once an individual customer was assigned to a specific cluster, this assignment was recorded and appended to the dataframe in the column 'HC_Clusters'. This will allow us to compare how individual customers were assigned to different segments. Let's see how the first 5 customers were assigned to segments in both the K-means and the Hierarchical clustering models...

In [99]:
```
BD.head()
```

Out[99]:

	Limit	Cards	Visits	Online	Calls
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 [100]:
```
BD1.head()
```

Out[100]:

	Limit	Cards	Visits	Online	Calls	K_Clusters
0	100000	2	1	1	0	0
1	50000	3	0	10	9	1
2	50000	7	1	3	4	0
3	30000	5	1	1	4	0
4	100000	6	0	12	3	2

In [101]:
```
BD2.head()
```

Out[101]:

	Limit	Cards	Visits	Online	Calls	HC_Clusters
0	100000	2	1	1	0	0
1	50000	3	0	10	9	2
2	50000	7	1	3	4	0
3	30000	5	1	1	4	0
4	100000	6	0	12	3	1

We can immediately see that, for the first 5 customers, all the segments labeled 0 line up. However, customers assigned to Segment 1 and 2 where assigned to HC_Cluster 2 and 1, respectively. We must remember that each model chose the label indiscriminately. We noticed that, in K-means clustering, in terms of spending patterns, cluster 2 was the "top" tier, cluster 0 was the "middle" tier and cluster 1 was the "bottom" tier in terms of both credit limits and number of cards held by the customer. In Hierarchical clustering, however, these labels were slightly inverted. Cluster 1 was the "top" tier, cluster 0 was the "middle" tier and cluster 2 was the "bottom" tier. For simplification purposes, we can change the names of the cluster labels in each dataset to correspond to the level of spending habits: high, medium, low. Afterwards, it will be much easier to compare the two models...

In [102]:
```
BD1['K_Clusters'] = BD1['K_Clusters'].map({0: 'Medium', 1: 'Low', 2: 'High
                                          })

#Changes the indiscriminate labels to categories
```

In [103]:
```
BD2['HC_Clusters'] = BD2['HC_Clusters'].map({0: 'Medium', 1: 'High', 2: 'L
                                          })

#Changes the indiscriminate labels to categories
```

In [104]:
```
BD3 = BD1.copy() #makes a copy of the dataset with K-means labels, to whic
```

In [105]:
```
HC_Labels = BD2[["HC_Clusters"]] #separates just the HC_labels from the B2
```

In [106]:
```
BD3["HC_Clusters"] = HC_Labels #adds the HC_Labels to the comparative data
```

In [107]:
```
BD3.head()
```

Out[107]:

	Limit	Cards	Visits	Online	Calls	K_Clusters	HC_Clusters
0	100000	2	1	1	0	Medium	Medium
1	50000	3	0	10	9	Low	Low
2	50000	7	1	3	4	Medium	Medium
3	30000	5	1	1	4	Medium	Medium
4	100000	6	0	12	3	High	High

What should we expect?

Now that we have created a comparative table, what exactly should we expect to find before we run any analysis? Knowing what to expect is a good idea because any unexpected findings will immediately stand out and be flagged for further examination. We already know the counts of both models. For K_means clustering, Low = 224, Medium = 386, and High = 50. For Hierarchical Clustering, Low = 213, Medium = 397, and High = 50. We can expect that both models classified High customers the same (however, this is not a given). It seems that K_means clustering classifiedn 11 more customers as Low and 11 less customers as Medium. Therefore, we should expect to find 11 customers who were classified differently out of 660, or about 1.7%. Let's see if this is true...

In [108]:
```
counterL = 0
counterM = 0
counterH = 0

for index in range(0, len(BD3)):
    if BD3['K_Clusters'].loc[index] == 'Low' and BD3['HC_Clusters'].loc[ir
        counterL = counterL + 1
    if BD3['K_Clusters'].loc[index] == 'Medium' and BD3['HC_Clusters'].loc
        counterM = counterM + 1
    if BD3['K_Clusters'].loc[index] == 'High' and BD3['HC_Clusters'].loc[i
        counterH = counterH + 1

counterDelta = 660 - (counterL+counterM+counterH)

print('There are', counterL, 'customers classified as Low by both Models')
print('There are', counterM, 'customers classified as Medium by both Model
print('There are', counterH, 'customers classified as High by both Models'
print('')
print('There are', counterDelta, 'customers classified differently by the
```

```
There are 213 customers classified as Low by both Models
There are 386 customers classified as Medium by both Models
There are 50 customers classified as High by both Models

There are 11 customers classified differently by the two Models
```

Let's examine the 11 customers who were classified differently by the two models and see

In [109]:
```
BD4 = BD3[(BD3['K_Clusters'] != BD3['HC_Clusters'])] #Creates a dataframe
```

In [110]:
```
BD4.shape # let's check the shape, we know from above that there should be
```

Out[110]: (11, 7)

In [111]:
```
BD4.head(11) # the dataframe is small enough to examine all 11 rows...
```

Out[111]:

	Limit	Cards	Visits	Online	Calls	K_Clusters	HC_Clusters
7	15000	3	0	1	1	Low	Medium
8	5000	2	0	2	2	Low	Medium
87	5000	4	1	2	5	Low	Medium
113	7000	4	1	2	4	Low	Medium
131	9000	3	0	2	5	Low	Medium
132	12000	4	0	2	6	Low	Medium
184	14000	4	0	2	4	Low	Medium
192	20000	3	0	3	5	Low	Medium
217	11000	4	0	2	5	Low	Medium
225	16000	4	1	2	5	Low	Medium
313	7000	4	2	2	4	Low	Medium

What we see is basically what we expected. First, there are no differently classified customers from the high clusters. We also see that there are only 11 differently classified customers, as we expected. K_means clustering placed ALL 11 in the low category while Hierarchical clustering placed all 11 in the medium category. We can infer that the dividing line between low and medium categories is slightly different. What we now want to know, is why. As we did above, let's break down the analysis by Spending Patterns and Interactions with Bank...

Clustering vs Spending Patter

Credit Limit by Cluster

In [112]:
```
pd.crosstab(BD1.K_Clusters, BD1.Limit).style.highlight_max(
    color="lightgreen", axis=0
)
```

Out[112]:

Limit	3000	5000	6000	7000	8000	9000	10000	11000	12000	13000	14000	1500
K_Clusters												
High	0	0	0	0	0	0	0	0	0	0	0	
Low	1	13	17	13	20	18	14	15	8	18	14	1
Medium	0	8	14	11	15	10	12	9	10	10	9	

In [113]:
```
pd.crosstab(BD2.HC_Clusters, BD2.Limit).style.highlight_max(
    color="lightgreen", axis=0
)
```

Out[113]:

Limit	3000	5000	6000	7000	8000	9000	10000	11000	12000	13000	14000	150
HC_Clusters												
High	0	0	0	0	0	0	0	0	0	0	0	
Low	1	11	17	11	20	17	14	14	7	18	13	
Medium	0	10	14	13	15	11	12	10	11	10	10	

Observations

- We know from previous analysis that the crosstabs will be identical above Credit Limit = 20,000
- We can see slight differences from Credit Limit = 5000 right up to Credit Limit = 20,000
- The 11 different classifications seem to be distributed rather evenly across the range, ie. there is no single group at a specific Credit Limit amount
- In all cases, the 11 differently classified values were moved from the Low category in K_means clustering to the medium category in Hierarchical clustering
- At present, there is no strong indication that Credit Limit was the major factor for the difference in classification

Cards by Cluster

In [114]:
```
pd.crosstab(BD1.K_Clusters, BD1.Cards).style.highlight_max(
    color="lightgreen", axis=0
)
```

Out[114]:

Cards	1	2	3	4	5	6	7	8	9	10
K_Clusters										
High	0	0	0	0	1	1	7	11	11	19
Low	59	63	53	49	0	0	0	0	0	0
Medium	0	1	0	102	73	116	94	0	0	0

In [115]:
```
pd.crosstab(BD2.HC_Clusters, BD2.Cards).style.highlight_max(
    color="lightgreen", axis=0
)
```

Out[115]:

Cards	1	2	3	4	5	6	7	8	9	10
HC_Clusters										
High	0	0	0	0	1	1	7	11	11	19
Low	59	62	50	42	0	0	0	0	0	0
Medium	0	2	3	109	73	116	94	0	0	0

Observations

- We know from previous analysis that the crosstabs will be identical above Cards = 4
- We can see slight differences from Cards = 2 up to Cards = 4
- The 11 different classifications seem skewed towards the higher number of cards. At Cards = 1, there is 1 difference; at Cards = 2 there are 3 differences; but at Cards = 4 there are 7 differences
- In all cases, the 11 differently classified values were moved from the Low category in K_means clustering to the medium category in Hierarchical clustering
- Cards may be a contributing factor. It seems that Hierarchical clustering is more likely to place customers with 4 cards in the medium category. However, 42 customers out of 151 customers with 4 cards were still placed in the low category by Hierarchical clustering. This difference of 7 only represents 5% of all customers with 4 cards.

Clustering vs Interactions with Bank

Total Visits by Cluster

In [116]:
```
pd.crosstab(BD1.K_Clusters, BD1.Visits).style.highlight_max(
    color="lightgreen", axis=0
)
```

Out[116]:

Visits	0	1	2	3	4	5
K_Clusters						
High	20	30	0	0	0	0
Low	80	79	65	0	0	0
Medium	0	3	93	100	92	98

In [117]:
```
pd.crosstab(BD2.HC_Clusters, BD2.Visits).style.highlight_max(
    color="lightgreen", axis=0
)
```

Out[117]:

Visits	0	1	2	3	4	5
HC_Clusters						
High	20	30	0	0	0	0
Low	73	76	64	0	0	0
Medium	7	6	94	100	92	98

Observations

- We know from previous analysis that the crosstabs will be identical above Visits = 2
- We can see differences only where Calls = 0, 1 or 2
- The 11 different classifications seem to be skewed towards the lower end: there are 7 differences where Calls = 0; 4 where Calls = 1; but only 1 where Calls = 2
- In all cases, the 11 differently classified values were moved from the Low category in K_means clustering to the medium category in Hierarchical clustering
- Calls can be a contributing factor since most customers who make 0 to 1 calls are predominately in the Low category

Total Online Visits by Cluster

In [118]:
```
pd.crosstab(BD1.K_Clusters, BD1.Online).style.highlight_max(
    color="lightgreen", axis=0
)
```

Out[118]:

Online	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
K_Clusters																
High	0	0	0	0	0	0	1	7	6	4	5	5	6	5	1	10
Low	0	3	54	43	69	54	0	0	0	0	1	0	0	0	0	0
Medium	144	106	135	1	0	0	0	0	0	0	0	0	0	0	0	0

In [119]:
```
pd.crosstab(BD2.HC_Clusters, BD2.Online).style.highlight_max(
    color="lightgreen", axis=0
)
```

Out[119]:

Online	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
HC_Clusters																
High	0	0	0	0	0	0	1	7	6	4	5	5	6	5	1	10
Low	0	2	45	42	69	54	0	0	0	0	1	0	0	0	0	0
Medium	144	107	144	2	0	0	0	0	0	0	0	0	0	0	0	0

Observations

- We know from previous analysis that the crosstabs will be identical where Online Visits = 0 OR > 3
- A majority of the differences occur where Online Visits = 2 (9/11 = 82%)
- The 11 different classifications seem to be skewked
- In all cases, the 11 differently classified values were moved from the Low category in K_means clustering to the medium category in Hierarchical clustering
- Online Visits can be a contributing factor, since 9/11 misclassified customers made 2 online visits where a majority of customers who made 2 visits belong to the medium category

Total Calls by Cluster

In [120]:
```
pd.crosstab(BD1.K_Clusters, BD1.Calls).style.highlight_max(
    color="lightgreen", axis=0
)
```

Out[120]:

Calls	0	1	2	3	4	5	6	7	8	9	10
K_Clusters											
High	16	15	18	1	0	0	0	0	0	0	0
Low	0	1	1	0	31	29	39	35	30	32	26
Medium	81	74	72	82	77	0	0	0	0	0	0

In [121]:
```
pd.crosstab(BD2.HC_Clusters, BD2.Calls).style.highlight_max(
    color="lightgreen", axis=0
)
```

Out[121]:

Calls	0	1	2	3	4	5	6	7	8	9	10
HC_Clusters											
High	16	15	18	1	0	0	0	0	0	0	0
Low	0	0	0	0	28	24	38	35	30	32	26
Medium	81	75	73	82	80	5	1	0	0	0	0

Observations

- We know from previous analysis that the crosstabs will be identical where Calls = 0 OR > 6
- The 11 different classifications seem to be skewed towards the higher number of calls, since Calls = 1 and 2 have just one difference, Calls = 3 are identical, but Calls = 4 and 5 have 3 and 5 differences, respectively. Calls = 6 has one difference.
- In all cases, the 11 differently classified values were moved from the Low category in K_means clustering to the medium category in Hierarchical clustering
- Calls may be a factor since 8 of 11 occur where calls = 4 and 5; at 4 most customers are Medium, at 5 most customers are low. This seems to be an inflection point

Additional Factors

- We can further compare the two models by looking at factors such as the time it took to execute the model
- K-means clustering was more efficient from a time perspective; it ran more quickly than Hierarchical clustering
- Hierarchical clustering performed well, however, there were only 660 customers in our dataset
- It is known that Hierarchical clustering is very computationally expensive, so as All Bank's customer database grows, it is probably recommended that the Bank continue evolving the K-means clustering model

- Comparing the silhouette scores would normally be another way to compare the models, however, since both models used n = 3 for clustering, the silhouette scores would be the same
-

Business Insights

- **K-means Clustering**:
  - This model MAY do a better job categorizing customers based off their spending patterns. It seems like, for both models, the distribution of the differences in customer classification based on Credit Limit was evenly distributed; however, K_means clustering was more strict when it came to the number of cards held by a customer.
  - K_means clustering only classified one single customer outside of the low cluster who had 3 cards or less; whereas Hierarchical clustering placed 5 customers with 3 cards or less in the low cluster
  - K_means clustering was also more strict at the 4 card point, with only 49 vs 42 customers in the low category compared to Hierarchical clustering
  - This could indicate that more weight was placed on Cards in K_means clustering, and the fewer the cards a customer holds, the more pressure the model put on placing the customer into the low category
- **Hierarchical Clustering**:
  - This model MAY do a better job categorizing customers based off their interactions with the bank.
  - For Online logins and call center contacts, Hierarchical clustering is creating 'cleaner' separations. For example, we see that any customers who call the call centre between 0 and 3 times are completely removed from the Low category. We see a similar effect in Online logins, where 0-2 logins are more favoured by people in the medium category. K_means clustering allows for some 'bleeding' of one category into the other
  - However, the opposite seems to be true for bank visits. K_means clustering created cleaner breaks and Hierarchical clustering is allowing some 'bleeding', where some customers are classified as medium, despite 0-1 visits to the bank
- **Which model is performing better?**:
  - It may be true that both models are performing well, overall, but each can be used for different purposes
  - For example, if the bank wants to conduct a marketing campaign and target customers with specific numbers of credit cards, the K-means clustering model would probably be more appropriate
  - However, if the bank is looking to improve its customer service in areas such as online banking and call centre contacts, the Hierarchical clustering model might be better
  - Finally, if the bank wishes to take a holistic approach, and target customers based on how they spend and how they interact with the bank, using the two models as complementary tools may be the best option
  - This holistic approach can be done by focusing on the 11 customers who were classified differently, since aside from these customers, the models are identical
  - The bank can examine each customer individually, since there are so few, and adapt staffing models or marketing models accordingly.

    - For example, K_means places more customers in the low category, so when the bank begins offering additional cards to specific groups, it may wish to use the K_means clusters since it would decrease the risk. The 11 differently classified customers would be offered a maximum of 4 cards, not 7
    - Similarly, there are more low categorized customers calling the bank's call center according to the K_means model. If call center employees are trained to specialize based on the 3 cluster categories in an effort to improve customer service, there would need to be more low level specialists on hand.
- **What is causing the disagreements between models?**:
    - When we look closely at the 11 customers who were classified differently, we can see that, for more than one variable, the values are at the dividing line between the two groups.
    - For example, 7 out of the 11 differently classified customers hold 4 cards, the data shows that roughly 1/3 of all customer with 4 cards are Low while 2/3 are high
    - Similarly, we see overlap at 2 Total Online Visits. Roughly 75% of all customers who log on twice per year fall into the Medium category, while the other 25% are in the Low category. 9 of the 11 customers who were differently classified logged in twice last year.

Part 11: Actionable Insights and Business Recommendations

Business Recommendations by Cluster

- **Cluster: Low**:
    - The Low category has the second largest group of customers, between 32% and 34%, depending on the model
    - This cluster has the least number of cards and the lowest credit limit, overall
    - This cluster is moderately tech savvy, taking advantage of online banking several times a year
    - This cluster is the overwhelming consumer of calling the call center for customer service
    - Call center staffing models could be built on this group, since almost all members of this group call the call center 4+ times per year
    - If self-service via online banking is substantially cheaper than using the call center, the bank needs to take steps to encourage these customers to migrate to online banking, since we have seen they are capable of using this service
    - These customers have between 1-4 cards, although a majority of customers with 4 cards belong to a different cluster
    - The bank could market to these individuals, touting the benefits of having 2-4 cards in an attempt to increase their spending activities
- **Cluster: Medium**:
    - This cluster makes up between 58% to 60% of all customers, and so special attention should be paid to this group because it makes up a majority of all the bank's customers

    - Credit Limits range from 5,000 to 75,000, however, there is one outlier at 100,000 dollars
    - Despite the wide range, almost all customers with credit limits between 25,000 and 75,000 belong to this cluster
    - This cluster overwhelmingly prefers to visit the bank over online or call center interacts
    - As such, the bank can build staffing models based on the number of customers in this group and the frequency they visit the bank. For example, this group alone, represents 1,347 total annual visits to the bank. If this group perceives poor service, more tellers should be hired, hours could be extended, the bank may consider opening on days it normally does not (ie Sundays, holidays, etc)
    - Other strategies could be to market additional cards to customers with 4, 5 or 6 cards, in an attempt to get them to spend more
    - It should be cautioned that these customers should not be encouraged to take more than 7 cards, or this risks moving them into the High cluster. This could cause segmentation problems since, we know the Medium cluster prefers NOT to use online banking and takes advantage of visits to the bank, whereas the High cluster is the opposite.
- **Cluster: High**:
    - The High cluster has the fewest customers, at 8%, however, they represent the richest group or the biggest spenders
    - This cluster cannot be ignored, despite its small size, because it probably generates a disproportionate amount of interest and other related credit card fees due to the high number of cards and the high credit limits these customers enjoy
    - This cluster prefers to use online banking, which is probably the cheapest of the three service options
    - This behaviour could be encouraged further via targeted marketing, since this group represents 30 visits to the bank per year and an additional 54 calls to the call center - reducing these interactions would further reduce costs
    - There are customers in this segment with only 5,6 and 7 cards; these customers should be targeted and convinced to take on more cards, to entrench them even further within this cluster. Customers with 8 and 9 cards can also be encouraged to

Overall Business Recommendations

- **Overall Recommendations**:
    - Extreme caution needs to be taken by the bank, because this data does not tell the whole story
    - At no point do we recommend increasing customers' credit limits based on this data alone
    - While there is correlation between higher credit limits and online banking, which is presumably the cheapest delivery mode of customer service, this correlation does NOT mean causation. Factors such as age, education, etc could be the real reason certain customers are using online banking and the real reason their credit limits are so high. For example, a younger person with a university education may be more comfortable using online banking and, due to their education, have a higher salary, which made them a lower risk for a high credit limit. Simply increasing someone's credit limit because you want to move them from cluster 1 to cluster 0 or cluster 2,

would be problematic, since it might not change their behavior and could expose the bank to an increase in risk.
    - Similarly, this is why we recommend that customers in each cluster are encouraged to take out more credit cards up to their cluster max, without exceeding this maximum, since more cards would presumably encourage spending, but more cards will not necessarily change a customer's behaviour. This recommendation would have to align with the bank's risk tolerance, customers' credit scores, etc
    - Costs associated with customer service delivery need to be thoroughly researched before additional recommendations can be given. For example, online delivery is probably the cheapest, followed by customer service centers, followed by brick and mortar stores. Encouraging customers to change their preferred delivery service makes sense in an effort to cut costs, but only if this improves customer satisfaction and factors such as educating people on how online banking works and incentivizing them to make the switch is less costly than the current model

Important Variables by Model

- **K-means clustering**:
    - Number of Cards and Total Bank Visits may be more important in the K-means clustering model
    - We see that K-means is making a stronger effort to categorize customers with 1-3 cards and 0-1 visits in the Low category.
    - The Hierarchical clustering model does not seem to be as strict at the lower values of these variables
- **Hierarchical clustering**:
    - Total Online Visits and Total Calls Made may be more important in the Hierarchical clustering model
    - We see that Hierarchical clustering is making a stronger effort to categorize customers with 0-2 online logins and customers who make 0-4 calls in the Medium category.
    - The K-means clustering model does not seem to be as strict at the lower values of these variables