Starbucks Customer Segmentation
A Clustering Analysis

Table of Contents

Executive Summary

Introduction

In this customer segmentation project, we will dive into Starbucks' customer and offer data in order to uncover valuable information that can enhance their promotional strategies. The purpose of this undertaking is to analyze the effectiveness of a number of promotional offers and perform customer segmentation to provide data-driven insights as to which customers are likely to respond to promotional offers and why. With this knowledge, Starbucks will more equipped to optimize the delivery of deals to customers, ensuring a higher rate of offer completion.

Throughout the duration of this project, we work to answer three primary questions:

- What makes an offer effective?
- What gets an offer viewed?
- How are different customers responding to offers and why?

Methodology

Data Wrangling

The data wrangling process primarily included data cleaning and joins in order to parse out information from particular columns as well as combine the three dataframes into one full dataframe. After this, the data was split into two separate dataframes which served as the "launch pads" for further analysis throughout the project. One of these dataframes contained customer transaction data, while the other focused on the customers' interactions with the promotional offers.

Data Analysis

The data analysis section began with a univariate analysis of demographic and transaction information to better understand the data. After that was a thorough examination of the characteristics of the 10 unique offers available in the data. Then began an analysis of offer effectiveness in which was defined a metric for evaluating what makes an offer effective. In that section, we uncover which features have the greatest effect on offer viewership and completion.

Feature Engineering and Transformation

This segment of the project involves creating new features to calculate the average transaction amount for each individual customer in addition to some data preprocessing to prepare the data for clustering. More dataframes were also created in this section which are used during model evaluation in order to determine the characteristics of the customers within each segment determined via data modeling.

Data Modeling

The data modeling process involved using the K-Means algorithm to find the within cluster sum of squares (WCSS) values as well as the silhouette scores for each amount of clusters ranging from 2 to 10. With this information, an optimal number of clusters was chosen, and the K-Means algorithm was used to create a model of the data containing the customer segments.

Results

Offer Viewership

The use of social media as a distribution channel was found to be the most prominent variable at determining whether or not an offer is viewed. As a result, its correlation to viewership was higher than other offer characteristics such as difficulty, duration, and reward. It is worth mentioning that, although the use of social media gets an offer viewed, these views have not been translating into completions. Because of this, offers distributed via social media had lower rates of completion.

Offer Completion

Similar to offer viewership, the most important variable at determining offer completion is also the use of social media as a distribution channel, however this time it is a negative correlation. This means that sending offers to customers via social media has led to lower rates of completion. The next most important contributor to offer completion is the offer type. Perhaps counter-intuitively, discount offers are more likely to be completed than bogo offers despite having, on average, a higher difficulty and lower reward.

Customer Segmentation

By performing K-Means clustering on the dataset of customers with two features, customer percent viewership and customer percent completion, three segments can be discovered:

- **Casual Customers - (28%)**
  - This segment has varying rates of viewership and low rates of completion. They can be identified as those who spend less per transaction than the other two segments.
- **Curious Customers - (42%)**
  - The segment boasts the highest rate of viewership, but they also have low rates of completion. They can be identified as those who are receiving more offers via social media and less discount offers which are more likely to encourage completion.
- **Committed Customers - (30%)**
  - This segment as varying rates of viewership but the highest rates of completion. These customers are receiving fewer offers via social media as well as more discount offers, and they have, on average, more time to complete offers than the other two segments. This has led to their average rate of completion being nearly twice as high as the next highest segment.

Strategic Recommendations

Based on the insights gained through data analysis and the K-Means clustering algorithm, I've curated the following list of strategic recommendations:

1. Continuously evaluate the use of social media as an offer channel with the intention of converting views into completions. This channel is outstanding when it comes to getting views, and it could become an extremely valuable asset with an increased rate of completions.

2. Consider methods to increase the average transaction amounts of the customers in the "casual" segment. With view rates similar to the "committed" segment, it is likely that they can be made to be very valuable customers.

3. Generate ways to move the customers in the "curious" segment into the "committed" segment by inspiring increased rates of offer completion. This can be achieved by following the first recommendation, or by providing them with more discount offers as opposed to bogo offers.

Data Description

- Back to Table of Contents

The datasets have been provided by Starbucks and are available on Kaggle, licensed under Community Data License Agreement - Permissive - Version 1.0. Three files are provided and can be previewed in the Import Statements section below.

- portfolio.csv
  - Offer information
  - 10 rows x 6 columns
- profile.csv
  - Demographic information
  - 17,000 rows x 5 columns
- transcript.csv
  - Customer transactions and interactions with offers
  - 306,534 rows x 4 columns

Import Statements

- Back to Table of Contents

I'll begin by importing all of the libraries that will be used in the project. Afterward, the data will be loaded and previewed. The entire dataset consists of three tables containing offer information, customer demographic data, and transaction/interaction data. They will be imported as `df0` , `df1` , and `df2` respectively for data wrangling.

In [ ]:
```
import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)

import numpy as np
import pandas as pd
pd.set_option('display.max_columns', None)
from scipy.stats import mode

import matplotlib.pyplot as plt
import matplotlib.dates as mdates
import seaborn as sns

from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import silhouette_score
from sklearn.cluster import KMeans
```

In [ ]:
```
# Import .csv files
df0 = pd.read_csv('portfolio.csv', index_col=0)
df1 = pd.read_csv('profile.csv', index_col=0)
df2 = pd.read_csv('transcript.csv', index_col=0)

# Preview portfolio.csv
print(df0.shape)
df0.head()
```

```
(10, 6)
```

Out[ ]:

	reward	channels	difficulty	duration	offer_type	ic
0	10	['email', 'mobile', 'social']	10	7	bogo	ae264e3637204a6fb9bb56bc8210ddfc
1	10	['web', 'email', 'mobile', 'social']	10	5	bogo	4d5c57ea9a6940dd891ad53e9dbe8da(
2	0	['web', 'email', 'mobile']	0	4	informational	3f207df678b143eea3cee63160fa8bec
3	5	['web', 'email', 'mobile']	5	7	bogo	9b98b8c7a33c4b65b9aebfe6a799e6d9
4	5	['web', 'email']	20	10	discount	0b1e1539f2cc45b7b9fa7c272da2e1d7

In [ ]:

```
# Preview profile.csv
print(df1.shape)
df1.head()
```

```
(17000, 5)
```

Out[ ]:

	gender	age	id	became_member_on	income
0	NaN	118	68be06ca386d4c31939f3a4f0e3dd783	20170212	NaN
1	F	55	0610b486422d4921ae7d2bf64640c50b	20170715	112000.0
2	NaN	118	38fe809add3b4fcf9315a9694bb96ff5	20180712	NaN
3	F	75	78afa995795e4d85b5d9ceeca43f5fef	20170509	100000.0
4	NaN	118	a03223e636434f42ac4c3df47e8bac43	20170804	NaN

In [ ]:

```
# Preview transcript.csv
print(df2.shape)
df2.head()
```

```
(306534, 4)
```

Out[ ]:

	person	event	value	tim
0	78afa995795e4d85b5d9ceeca43f5fef	offer received	{'offer id': '9b98b8c7a33c4b65b9aebfe6a799e6d9'}	
1	a03223e636434f42ac4c3df47e8bac43	offer received	{'offer id': '0b1e1539f2cc45b7b9fa7c272da2e1d7'}	
2	e2127556f4f64592b11af22de27a7932	offer received	{'offer id': '2906b810c7d4411798c6938adc9daaa5'}	
3	8ec6ce2a7e7949b1bf142def7d0e0586	offer received	{'offer id': 'fafdcd668e3743c1bb461111dcafc2a4'}	
4	68617ca6246f4fbc85e91a2a49552598	offer received	{'offer id': '4d5c57ea9a6940dd891ad53e9dbe8da0'}	

Data Wrangling

- Back to Table of Contents

The objective of this data cleaning and transformation process is to prepare the data to be analyzed and modeled most effectively. In order to achieve this, I will work to join all of the tables into one dataframe, after which I will separate the master dataset into two additional datasets: one specifically for transaction information and another for promotional data. The final dataframes will have each customer and their corresponding demographic information linked to the offers that they received as well as the transactions that they made. The process that follows will include data cleaning as well as some rudimentary analysis and data transformations including joins, set operations, and one-hot encoding.

Contents: Data Wrangling

Join 1

- Back to Data Wrangling Contents

To begin, I will join the two dataframes containing the unique customer IDs. However, before doing this, I will first verify that each person in `df1` is unique and that `df1` and `df2` are comprised of the same individuals. To do this, I will check for duplicated values in `df1['id']` and then confirm that the symmetric difference of a set of the two corresponding columns returns a set with zero elements.

In [ ]:
```
# Check df1['id'] for duplicates
df1.duplicated('id').any()
```

Out[ ]:
```
False
```

There are no duplicated customer ID numbers in `df1`. This means that each person in the dataframe is unique.

In [ ]:
```
# Verify matching set of customer IDs
if len(set(df1['id']) ^ set(df2['person'])) == 0:
    print("The dataframes df1 and df2 are comprised of the same individuals.
else:
    print("The people in df1 and those in df2 do not match.")
```

```
The dataframes df1 and df2 are comprised of the same individuals.
```

These two dataframes have the same people in them. Next, they will be joined on that column. I'm going to do a right join to keep all rows from `df2` and match `df1` to those rows.

In [ ]:
```
# Perform right join on column named 'id'
df3 = pd.merge(df1, df2, left_on='id', right_on='person', how='right')
```

```
# Ensure no rows were dropped during joining
if df3.shape[0] == df2.shape[0]:
    print("Right join successful.")
```

```
Right join successful.
```

In [ ]:

```
# Preview dataframe
df3.head()
```

Out[ ]:

	gender	age	id	became_member_on	income	
0	F	75	78afa995795e4d85b5d9ceeca43f5fef	20170509	100000.0	78afa
1	NaN	118	a03223e636434f42ac4c3df47e8bac43	20170804	NaN	a0322
2	M	68	e2127556f4f64592b11af22de27a7932	20180426	70000.0	e212
3	NaN	118	8ec6ce2a7e7949b1bf142def7d0e0586	20170925	NaN	8ec6c
4	NaN	118	68617ca6246f4fbc85e91a2a49552598	20171002	NaN	68617

Next, I'll confirm that the column `'id'` matches `'person'`. Then, drop the redundant column.

In [ ]:

```
if df3['id'].equals(df3['person']):
    df3.drop('person', axis=1, inplace=True)
    df3.reset_index(drop=True, inplace=True)
```

Splitting the Key-Value Pairs

- Back to Data Wrangling Contents

Now, I'll focus on the column labeled `'value'`. By examining the data on Kaggle, I've discovered that this column contains, for the most part, one key-value pair. There are observations which contain an additional element which is another key-value pair. This additional element appears when the column `'event'` is equal to `'offer completed'`. The second key-value pair's sole purpose is to display the reward for completing an offer. Because there is a `'reward'` column in `df0`, the second key-value pair will be ignored and, consequently, dropped.

In order to make the premier key-value pair in `'value'` more manageable, I will separate it into two columns. First, I'll find and print the unique keys, then complete the separation.

The process goes as follows:

1. Turn the column into a list
2. Initialize an empty list
3. While iterating over the elements in the list
   A. Convert the string element into a dictionary
   B. Append the dictionary to the new list
4. Initialize an empty set
5. Extract only the keys from each element in the list and add them to the set (the set will not keep duplicates)
6. Display the distinct keys
7. Put the keys and values into separate lists with list comprehensions
8. Use these lists to make a new dataframe
9. Concatenate the new dataframe to `df3`
10. Remove the old `'value'` column

In [ ]:

```
# Copy df3 and convert the 'value' column to a list
df3c = df3.copy()
df_value_list = df3c['value'].tolist()

# Initialize an empty list
value_list = []

# Create list of all dictionaries in 'value'
for element in df_value_list:
    value_dict = eval(element)
    value_list.append(value_dict)

# Initialize an empty set
unique_keys = set()

# Extract unique keys from the key-value pairs
for key in value_list:
    unique_keys.add(next(iter(key)))

# Print unique keys
print(f"The unique keys in column 'value' are: {unique_keys}")

# Create list for keys
value_keys = [next(iter(d)) for d in value_list]

# Create list for values
value_values = [d[key] for d, key in zip(value_list, value_keys)]

# Create new dataframe with separate columns for keys and values
df_keys_values = pd.DataFrame({'offer_id_or_amount': value_keys, 'values': v

# Preview dataframe
print(f"\n{df_keys_values.shape}")
df_keys_values.head()
```

```
The unique keys in column 'value' are: {'offer_id', 'amount', 'offer id'}
```

```
(306534, 2)
```

Out[ ]:

	offer_id_or_amount	values
0	offer id	9b98b8c7a33c4b65b9aebfe6a799e6d9
1	offer id	0b1e1539f2cc45b7b9fa7c272da2e1d7
2	offer id	2906b810c7d4411798c6938adc9daaa5
3	offer id	fafdcd668e3743c1bb461111dcafc2a4
4	offer id	4d5c57ea9a6940dd891ad53e9dbe8da0

There are two different keys which mean the same thing: `'offer_id'` and `'offer id'`. This will need to be fixed so that they are consistent.

Now, I'll examine the values for `'amount'` and how they differ from the offer ID.

In [ ]:

```python
# Preview dataframe where 'offer_id_or_amount' equals only 'amount'
df_amount = df_keys_values[df_keys_values['offer_id_or_amount'] == 'amount']
print(df_amount.shape)
df_amount.sample(3)
```

```
(138953, 2)
```

Out[ ]:

	offer_id_or_amount	values
88173	amount	14.68
243629	amount	16.75
29261	amount	0.87

It seems like the amounts correspond to transaction amounts. This will be further investigated soon.

The next step in the process is to concatenate the separated columns to the original dataframe.

In [ ]:

```python
# Concatenate df_keys_values to df3 in a new dataframe
df4 = pd.concat([df3, df_keys_values], axis=1)
df4.head()
```

Out[ ]:

	gender	age	id	became_member_on	income	eve
0	F	75	78afa995795e4d85b5d9ceeca43f5fef	20170509	100000.0	off receive
1	NaN	118	a03223e636434f42ac4c3df47e8bac43	20170804	NaN	off receive
2	M	68	e2127556f4f64592b11af22de27a7932	20180426	70000.0	off receive
3	NaN	118	8ec6ce2a7e7949b1bf142def7d0e0586	20170925	NaN	off receive
4	NaN	118	68617ca6246f4fbc85e91a2a49552598	20171002	NaN	off receive

At this point, I will fix the inconsistency in the column `'offer_id_or_amount'` by replacing the space in 'offer id' with an underscore. Then, I'll complete this section of data wrangling by dropping the redundant column.

In [ ]:

```
# Fixing the inconsistency 'offer id' and 'offer_id'
df4['offer_id_or_amount'] = df4['offer_id_or_amount'].str.replace(' ', '_')

# Confirm fix by printing unique values in the column
print(df4['offer_id_or_amount'].unique())

# Dropping the now redundant 'value' column
df4.drop(columns='value', inplace=True)
df4.tail()
```

```
['offer_id' 'amount']
```

Out[ ]:

	gender	age	id	became_member_on	income
306529	M	66	b3a1272bc9904337b331bf348c3e8c17	20180101	47000.0
306530	M	52	68213b08d99a4ae1b0dcb72aebd9aa35	20180408	62000.0
306531	F	63	a00058cf10334a308c68e7631c529907	20130922	52000.0
306532	M	57	76ddbd6576844afe811f1a3c0fbb5bec	20160709	40000.0
306533	NaN	118	c02b10e8752c4d8e9b73f918558531f7	20151211	NaN

Data Cleaning 1

- Back to Data Wrangling Contents

Now, I'll ensure that the numbers in the `'values'` columns which correspond to `'amount'` in the `'offer_id_or_amount` column are transaction amounts by confirming that `'amount'` only exists when the value in the column `'event'` is equal to `'transaction'`.

In [ ]:
```
trans_vs_amt = (df4['event'] == 'transaction') == (df4['offer_id_or_amount']
print(f"It is {trans_vs_amt.all()} that the numerical entries in 'values' ar
```

```
It is True that the numerical entries in 'values' are transaction amounts.
```

I'll need to distinguish the customer ID value from the promo ID value. I'll do this by changing the column name to `'customer_id'`.

In [ ]:
```
# Rename column 'id' to 'customer_id'
df4.rename(columns={'id': 'customer_id'}, inplace=True)

# Creating a new dataframe sorted by customer ID number
df5 = df4.sort_values(by=['customer_id'])
```

There are a number of people in the dataset without demographic information. They are recorded with the `'gender'` and `'income'` columns as `NaN` and the `'age'` column as `118`. This demographic information can provide valuable insight in the analysis and, because one of the purposes of this project is to perform customer segmentation, I will be dropping these individuals from the dataset provided that there is still sufficient data remaining for a robust analysis.

In [ ]:
```
# Display total NaN by column
df5.isna().sum()
```

Out[ ]:
```
gender                33772
age                       0
customer_id               0
became_member_on          0
income                33772
event                     0
time                      0
offer_id_or_amount        0
values                    0
dtype: int64
```

In [ ]:
```
# Display total 118-year-olds
(df5['age'] == 118).value_counts()
```

Out[ ]:
```
age
False    272762
True      33772
Name: count, dtype: int64
```

Because `'age'` is equal to `118` where the `'income'` and `'gender'` columns are `NaN`, I can simply drop the `NaN` rows, and the 118-year-olds will be dropped as well. I confirm this in the second cell below.

In [ ]:
```
# Drop NA and display new df shape
df5.dropna(axis=0, how='any', inplace=True)
df5.shape
```

Out[ ]:
```
(272762, 9)
```

In [ ]:
```
# Confirm all 118-year-olds are dropped
(df5['age'] == 118).value_counts()
```

Out[ ]:
```
age
False    272762
Name: count, dtype: int64
```

One-Hot Encoding

- Back to Data Wrangling Contents

I'm going to make some changes to `df0` before performing the final join to create the full dataset. So, turning our attention back to `df0`, the next step in the process is to parse out the `'channels'` column. I will do this by one-hot encoding. I'll define a function which creates four new columns of zeroes for the four potential values in `'channels'` and fill them with a 1 if their corresponding value exists in the `'channels'` column.

In [ ]:
```
def encode_channels(df):
    '''
    Performs one-hot encoding on the 'channels' column of the input Pandas D

    Parameters:
    The Pandas DataFrame containing the 'channels' column.

    Returns:
    A new DataFrame with the 'channels' column one-hot encoded as four
    separate columns.
    '''

    # Create four new columns of zeroes
    df['channel_web'] = 0
    df['channel_email'] = 0
    df['channel_mobile'] = 0
    df['channel_social'] = 0

    # Iterate over each element in each row, filling corresponding columns w
    for index, row in df.iterrows():
        for element in (row['channels']):
            if element == 'web':
                df.loc[index, 'channel_web'] = 1
            elif element == 'email':
                df.loc[index, 'channel_email'] = 1
            elif element == 'mobile':
                df.loc[index, 'channel_mobile'] = 1
            elif element == 'social':
                df.loc[index, 'channel_social'] = 1

    return df
```

In [ ]:
```
# Change the elements from one string to a list of strings
df0['channels'] = df0['channels'].apply(eval)
```

```
# Perform one-hot encoding on df0
df7 = encode_channels(df0)

# Preview new dataframe
df7.head()
```

Out[ ]:

	reward	channels	difficulty	duration	offer_type	ic
0	10	[email, mobile, social]	10	7	bogo	ae264e3637204a6fb9bb56bc8210ddfc
1	10	[web, email, mobile, social]	10	5	bogo	4d5c57ea9a6940dd891ad53e9dbe8da0
2	0	[web, email, mobile]	0	4	informational	3f207df678b143eea3cee63160fa8bec
3	5	[web, email, mobile]	5	7	bogo	9b98b8c7a33c4b65b9aebfe6a799e6d9
4	5	[web, email]	20	10	discount	0b1e1539f2cc45b7b9fa7c272da2e1d7

Dropping the unnecessary column.

In [ ]:
```
# Drop column 'channels'
df7.drop(columns='channels', inplace=True)
```

Join 2

- Back to Data Wrangling Contents

Here, I'll perform a left join on `df5` and `df7` to marry the information about the promos to the corresponding occurences in the previously joined table. This will complete the process of combining the original three dataframes.

In [ ]:
```
# Perform left join on df5 and df7
df_full = pd.merge(df5, df7, left_on='values', right_on='id', how='left')

# Preview new dataframe
print(df_full.shape)
df_full.head()
```

```
(272762, 18)
```

Out[ ]:

	gender	age	customer_id	became_member_on	income	ev
0	M	33	0009655768c64bdeb2e877511632db8f	20170421	72000.0	transac
1	M	33	0009655768c64bdeb2e877511632db8f	20170421	72000.0	transac
2	M	33	0009655768c64bdeb2e877511632db8f	20170421	72000.0	transac
3	M	33	0009655768c64bdeb2e877511632db8f	20170421	72000.0	c rece
4	M	33	0009655768c64bdeb2e877511632db8f	20170421	72000.0	c rece

In `df_full` , there are a large amount of observations containing `NaN` . This is because there isn't data relevant to the promotions themselves for transactional records. For simplicity and to eliminate `NaN` in the datasets, I'll split this final dataframe into two separate dataframes for promos and transactions. However, before that, I'll complete some additional data cleaning with `df_full` .

In [ ]:
```
# Drop the redundant column
df_full.drop(columns='id', inplace=True)
```

Data Cleaning 2

- Back to Data Wrangling Contents

The following tasks will be done for simplicity, consistency, and to correct formatting:

- Change column name `'became_member_on'` to `'member_since'`
- Change column name `'time'` to the more descriptive name: `'hours_passed'`
- Change spaces to underscores in column `'event'`
- Correct DateTime format for the column `'member_since'`

In [ ]:
```
# Rename the column 'became_member_on' to 'member_since'
df_full.rename(columns={'became_member_on': 'member_since'}, inplace=True)

# Rename the column 'time' to 'hours_passed'
df_full.rename(columns={'time': 'hours_passed'}, inplace=True)

# Display all column names
df_full.columns
```

Out[ ]:
```
Index(['gender', 'age', 'customer_id', 'member_since', 'income', 'event',
       'hours_passed', 'offer_id_or_amount', 'values', 'reward', 'difficult
y',
       'duration', 'offer_type', 'channel_web', 'channel_email',
       'channel_mobile', 'channel_social'],
      dtype='object')
```

In [ ]:
```
# Display all distinct values in column 'event'
df_full['event'].unique()
```

Out[ ]:
```
array(['transaction', 'offer received', 'offer viewed', 'offer completed'],
      dtype=object)
```

In [ ]:
```
# Change spaces to underscores in column 'event'
df_full['event'] = df_full['event'].str.replace(' ', '_')

# Display all distinct values in column 'event'
df_full['event'].unique()
```

Out[ ]:
```
array(['transaction', 'offer_received', 'offer_viewed', 'offer_completed'],
      dtype=object)
```

For the column `'member_since'`, I'll first confirm that it is a column of integers and that all entries begin with the year. This will be done with the `dtypes` method in the Pandas library followed by the minimum and maximum values of the `'member_since'` column.

In [ ]:
```
# Show data type of the 'member_since' column
df_full['member_since'].dtypes
```

Out[ ]:
```
dtype('int64')
```

In [ ]:
```
# Print max value in 'member_since'
df_full['member_since'].max()
```

Out[ ]:
```
20180726
```

In [ ]:
```
# Print min value in 'member_since
df_full['member_since'].min()
```

Out[ ]:
```
20130729
```

It is confirmed that the `'member_since'` column begins with the year and is populated with integer values. Next, I'll convert the integers to strings and add dashes after the 4th and 6th characters. Then, I'll convert the column to DateTime format.

In [ ]:
```
# Convert column to strings
df_full['member_since'] = df_full['member_since'].astype(str)

# Add dashes
df_full['member_since'] = df_full['member_since'].str[:4] + '-' \
                        + df_full['member_since'].str[4:6] + '-' \
                        + df_full['member_since'].str[6:]

# Convert to datetime
df_full['member_since'] = pd.to_datetime(df_full['member_since'])

# Preview dataframe
df_full.head()
```

Out[ ]:

	gender	age	customer_id	member_since	income	even
0	M	33	0009655768c64bdeb2e877511632db8f	2017-04-21	72000.0	transactior
1	M	33	0009655768c64bdeb2e877511632db8f	2017-04-21	72000.0	transactior
2	M	33	0009655768c64bdeb2e877511632db8f	2017-04-21	72000.0	transactior
3	M	33	0009655768c64bdeb2e877511632db8f	2017-04-21	72000.0	offer_receivec
4	M	33	0009655768c64bdeb2e877511632db8f	2017-04-21	72000.0	offer_receivec

I will now verify the yyyy-mm-dd format, ensuring there are no invalid months or days, by listing unique values for both months and days.

In [ ]:
```
# Display unique months in ascending order
unique_months = df_full['member_since'].dt.month.unique()
print(f'Unique months: {sorted(unique_months)}')

# Display unique days in ascending order
unique_days = df_full['member_since'].dt.day.unique()
print(f'Unique days: {sorted(unique_days)}')
```

```
Unique months: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]
Unique days: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18,
19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31]
```

Splitting the Master DataFrame

- Back to Data Wrangling Contents

That completes the data cleaning process for the full dataset. Next, I'll separate this master dataset into two separate ones. There will be one dataframe for transaction information linked to each customer called `df_trans` and another for promotion information linked to each customer called `df_promo` .

Earlier, I was able to confirm that where the column `'offer_id_or_amount` had the value `'amount'` then the column `'event'` also had the value `'transaction'` . I will use that information to split the master dataset while keeping the customer demographic information in both DataFrames.

In [ ]:
```
# Create dataframe where 'offer_id_or_amount' is 'offer_id'
df_promo = df_full[df_full['offer_id_or_amount'] == 'offer_id'].copy()

# Create dataframe where 'offer_id_or_amount' is 'amount'
df_trans = df_full[df_full['offer_id_or_amount'] == 'amount'].copy()
```

Data Cleaning 3

- Back to Data Wrangling Contents

To clean these two new tables, I'll change the necessary numeric columns back to integers and floats and remove the last 8 columns in `df_trans` .

In [ ]:
```
# Create list of column names
to_int_cols0 = ['income',
                'reward',
                'difficulty',
                'duration',
                'channel_web',
                'channel_email',
                'channel_mobile',
                'channel_social']

# Change data types to integers
df_promo[to_int_cols0] = df_promo[to_int_cols0].astype(int)
df_promo.reset_index(drop=True, inplace=True)
```

In [ ]:
```
# Remove last 8 columns in df_trans
df_trans.drop(df_trans.columns[-8:], axis=1, inplace=True)
df_trans.reset_index(drop=True, inplace=True)

# Change column data types
df_trans['income'] = df_trans['income'].astype(int)
df_trans['values'] = df_trans['values'].astype(float)
```

Preview the Final DataFrames

- Back to Data Wrangling Contents

In [ ]:
```
# Preview a sample of df_full
print(df_full.shape)
df_full.sample(10)
```

```
(272762, 17)
```

Out[ ]:

	gender	age	customer_id	member_since	income	
256110	F	19	f052f7c3f89044f9bb7097a72e62101c	2018-04-10	55000.0	offer_r
76158	M	53	46f43532df824788bee55a1d14b6b39b	2017-09-05	39000.0	trar
181424	M	66	a97051ceb2824a37bae1dbfbad9afae5	2017-11-16	31000.0	offer_
178148	F	25	a6173195564d4c7ba641bc441b28c4f0	2017-10-20	46000.0	trar
105988	M	46	63b1186a5399405087fe6a9f604bb9d4	2018-04-06	48000.0	offer_r
143067	M	55	86b493a2b7554a11aedf51eca0c6b033	2016-01-30	72000.0	offer_r
122324	F	57	7354195da3024460a43e54e3f7143726	2016-10-22	88000.0	trar
221134	M	73	cfe1dbd54f1549da8ce88b453d8d5b8b	2017-11-22	54000.0	trar
265187	M	57	f8e69750104947f0a16bfed5dab877ce	2015-08-07	91000.0	offer_r
260266	M	21	f444db6612f649c79084b88b7e1eb83f	2017-05-01	31000.0	trar

In [ ]:

```
# Preview a sample of df_promo
print(df_promo.shape)
df_promo.sample(10)
```

(148805, 17)

Out[ ]:

	gender	age	customer_id	member_since	income	
148388	F	73	ff5a8471ac1b4809b190e38ba13a678a	2017-09-16	79000	offer_
29590	M	34	338cc645ee5e42a5b61b78ab018f0445	2016-10-24	64000	offer
145114	F	62	f97ac1f446e747d491b060d9ae044df9	2018-04-11	65000	offer_
12077	M	80	153572326a8340aca794bd6ff01b24c1	2018-04-08	96000	offer
34846	F	65	3c08e4668bc34bda9f4ae4853defe255	2016-03-18	68000	offer_co
116544	F	41	c811024989454214bffab4c58bcf56a9	2018-01-28	72000	offer_
88443	M	100	97d7120a2a154a6d98b053e64b782ecf	2015-10-13	63000	offer_co
129000	F	43	de0e17b726dc49e793471ec8b6b08b3e	2015-08-14	68000	offer_
125070	M	48	d77012df51d64d7095dbbb69945e2825	2016-09-09	78000	offer_
86950	F	65	953f730903484691800d77f17750cdb0	2018-01-28	39000	offer_

In [ ]:

```
# Preview a sample of df_trans
print(df_trans.shape)
df_trans.sample(10)
```

(123957, 9)

Out[ ]:

	gender	age	customer_id	member_since	income	ev
32014	F	46	41f309195094400fa971fe39673b35c7	2018-01-09	65000	transact
34572	F	51	46cae479b56c479597ff23bbcdc95693	2017-02-24	74000	transact
60673	F	27	7e3009f88c254c89af4f99e5068f9e26	2018-07-05	38000	transact
123174	M	19	fe54dec74c99440bb4ee266e2a4c778d	2014-06-19	61000	transact
94326	F	52	c2e92293563f471784aeb176661ed097	2018-03-19	64000	transact
15966	M	51	21c16cf0c293423ca08647079f1783c7	2015-10-26	66000	transact
9796	F	24	14b0f8a0f7024e9094869bccc14150f4	2014-04-26	43000	transact
71554	M	46	940c174894734049a18b0770794d3ea3	2016-10-22	54000	transact
55128	F	63	72f6338d8b124f06be857d656019e0f8	2016-04-10	96000	transact
76109	M	63	9d1085c1daaa4838b9cf6bde5dc330bd	2016-03-23	72000	transact

At this point, it is worth noting that `df_trans` does not contain all of the customers available in `df_full` because there are 333 individuals who made no transactions.

There is one DataFrame, `df_full` which contains all of the data, including NaNs, and there are two additional DataFrames that do not have NaNs, `df_trans` and `df_promo` , which contain transaction and offer data respectively. `df_trans` and `df_promo` will serve as the "launch pads" for further analysis. The three datasets have been cleaned, formatted, and transformed so that they are much more useful to the python libraries relevant to data science than they were originally. This completes the data wrangling process.

Data Analysis

- Back to Table of Contents

Within this data analysis process, I conduct a thorough examination of the demographic and transaction information to gain a deeper understanding of the data at hand. Then, I analyze the characteristics of each unique offer available in the dataset. Afterwards, I evaluate the effectiveness of each offer, which will help to identify which features have the greatest impact on viewership and completion rates.

Contents: Data Analysis

Demographics

- Back to Data Analysis Contents

Before beginning analysis on the demographic information, I'll extract the cleaned customer demographic information from `df_full`, saving it to a new DataFrame named `df_cust`.

In [ ]:
```
# Create a copy of df_trans
df_cust = df_full.copy()

# Drop the last four columns
df_cust.drop(df_cust.columns[-12:], axis=1, inplace=True)

# Keep one row per customer by removing duplicate customer IDs
df_cust = df_cust.drop_duplicates(subset='customer_id')
df_cust.reset_index(drop=True, inplace=True)
```

In [ ]:
```
# Find the total amount of customers in the data
print(f'The total amount of unique customers: {df_cust.shape[0]}')
```

```
The total amount of unique customers: 14825
```

In [ ]:
```
# Generate summary statistics of income and age
df_cust[['income', 'age']].describe()
```

Out[ ]:

	income	age
count	14825.000000	14825.000000
mean	65404.991568	54.393524
std	21598.299410	17.383705
min	30000.000000	18.000000
25%	49000.000000	42.000000
50%	64000.000000	55.000000
75%	80000.000000	66.000000
max	120000.000000	101.000000

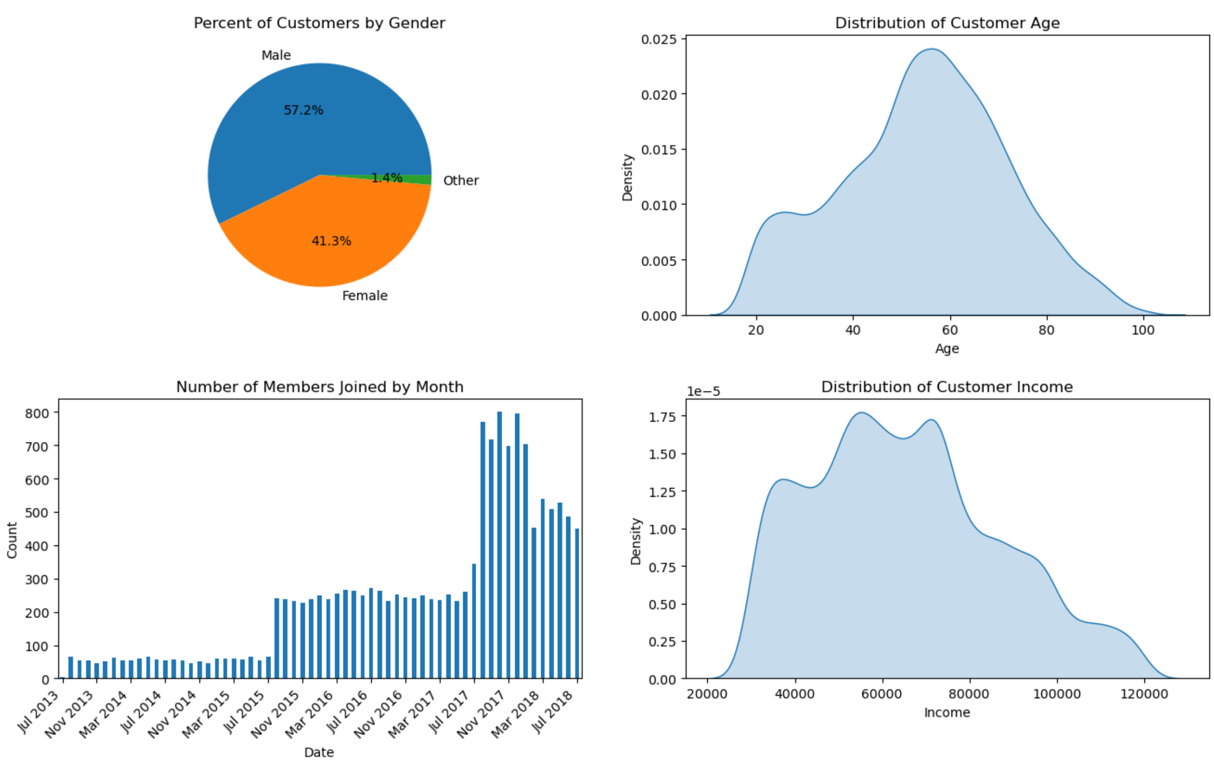
There are a total of 14,825 customers. As for the averages of the numerical columns, the average income is 65,405 USD and the average age is 54.4 years old. The distribution of these variables and more will be investigated further in this section.

Next, I'll create a figure to display the proportion of the genders in the dataset, distribution of customer age, number of members joined by month, and distribution of customer income.

In [ ]:

```
# Create a figure with four subplots
fig, axs = plt.subplots(2, 2, figsize=(16, 9))

#####################################################
##### Subplot 1: Percent of Customers by Gender #####
#####################################################

# Get number of customers by gender
wedge_size = df_cust['gender'].value_counts()

# Get unique gender values
wedge_label = ['Male', 'Female', 'Other']

# Plot pie chart
axs[0, 0].pie(x=wedge_size, labels=wedge_label, autopct='%1.1f%%')
axs[0, 0].set_title('Percent of Customers by Gender')

###################################################
##### Subplot 2: Distribution of Customer Age #####
###################################################

# Generate histogram of customer age
sns.kdeplot(df_cust['age'], fill=True, ax=axs[0, 1])
axs[0, 1].set_title('Distribution of Customer Age')
axs[0, 1].set_xlabel('Age')

##############################################
##### Subplot 3: Members Joined by Month #####
##############################################
```

```python
# Copy dataframe and set index as 'member_since'
df_cust_0 = df_cust.copy()
df_cust_0.set_index('member_since', inplace=True)

# Plot monthly bar chart
df_cust_0.resample('M').size().plot(kind='bar', ax=axs[1, 0])
axs[1, 0].set_title('Number of Members Joined by Month')
axs[1, 0].set_ylabel('Count')
axs[1, 0].set_xlabel('Date')

# Set xticks every 4 bars
tick_positions = range(0, len(df_cust_0.resample('M').size()), 4)
axs[1, 0].set_xticks(tick_positions)

# Format date labels for xticks for each 4th month
tick_labels = [item.strftime('%b %Y') for i, item in \
               enumerate(df_cust_0.resample('M').size().index) if i % 4 == 0
axs[1, 0].set_xticklabels(tick_labels, rotation=45, ha='right')

#######################################################
##### Subplot 4: Distribution of Customer Income #####
#######################################################

# Generate histogram of customer income
sns.kdeplot(df_cust['income'], fill=True, ax=axs[1, 1])
axs[1, 1].set_title('Distribution of Customer Income')
axs[1, 1].set_xlabel('Income')

# Adjust vertical space between subplots
fig.subplots_adjust(hspace=0.3)

# Display the plot without extra output
plt.show();
```



There are approximately 16% more males than females in the dataset as well as 1.4% having their gender recorded as "other." Customer age follows a relatively normal distribution with a noticeable subset of customers in their early twenties. The number of members joining per month saw large increases both in August of 2015 and 2017. As is typical of income distributions, we can observe a right skew in this dataset.

Transactions

- Back to Data Analysis Contents

To continue the exploratory data analysis process, I'll extract two values out of the transaction data:

- Number of customers who made no transactions
- Total number of transactions

In [ ]:
```
# Print number of customers in df_full subtracted by those in df_trans
print(df_full['customer_id'].nunique() - df_trans['customer_id'].nunique())
```

```
333
```

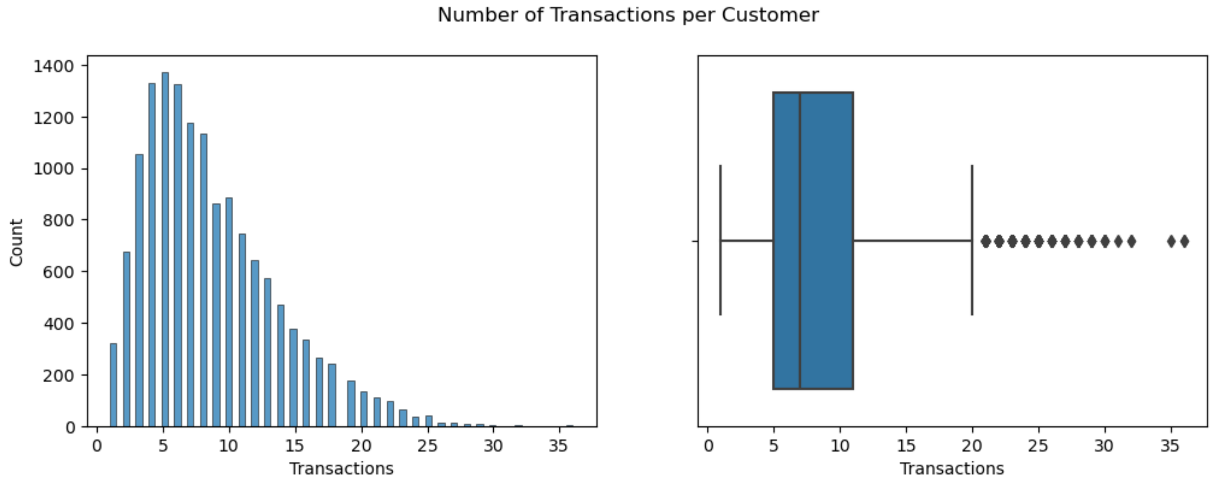
In [ ]:
```
# Print total number of transactions
print(df_trans.shape[0])
```

```
123957
```

There are 333 customers who made no transactions. Moreover, there are a total of 123,957 transactions in this dataset.

Next, I'll identify how many transactions customers are making by generating a histogram and boxplot containing the results of the `value_counts()` method used on the `'customer_id'` column.

In [ ]:
```
# Create a figure with two subplots
fig, ax = plt.subplots(1, 2, figsize=(12, 4))
fig.suptitle('Number of Transactions per Customer')

# Generate histogram of number of transactions per customer
sns.histplot(df_trans['customer_id'].value_counts(), ax=ax[0])
ax[0].set_xlabel('Transactions')

# Generate boxplot of number of transactions per customer
sns.boxplot(x=df_trans['customer_id'].value_counts(), ax=ax[1])
ax[1].set_xlabel('Transactions')

plt.show();
```

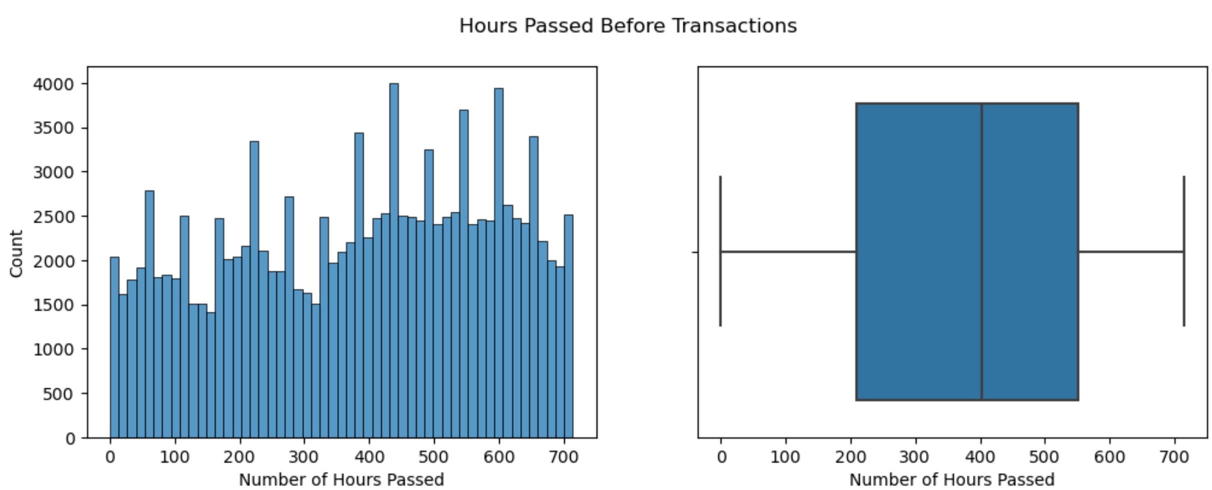


Here, we can observe another positive skew. It appears there is a large portion of customers who make less than 10 transactions, and the most commonly seen amount of transactions per customer is 5.

The upcoming plot will contain the amount of hours passed before the transaction for each transaction in the DataFrame. This will provide insight as to when customers are making transactions.

In [ ]:

```
# Create a figure with 2 subplots
fig, ax = plt.subplots(1, 2, figsize=(12, 4))
fig.suptitle('Hours Passed Before Transactions')

# Generate histogram of time of transaction
sns.histplot(df_trans['hours_passed'], ax=ax[0])
ax[0].set_xlabel('Number of Hours Passed')

# Generate boxplot of time of transaction
sns.boxplot(x=df_trans['hours_passed'], ax=ax[1])
ax[1].set_xlabel('Number of Hours Passed')

plt.show();
```



We can observe spikes in transactions at certain times. The relationship between transactions and time will be further investigated in the Offers section of the data analysis process.

Now, I'll generate some statistics for how much customers are spending in their transactions.

In [ ]:
```
trans_amt_mean = df_trans['values'].mean()
trans_amt_median = df_trans['values'].median()
trans_amt_mode = df_trans['values'].mode()[0]
trans_amt_min = df_trans['values'].min()
trans_amt_max = df_trans['values'].max()

trans_amt_stats_df = pd.DataFrame({
    'Statistic': ['Mean', 'Median', 'Mode', 'Min', 'Max'],
    'Amount': [trans_amt_mean, trans_amt_median, trans_amt_mode, trans_amt_m
})
trans_amt_stats_df['Amount'] = trans_amt_stats_df['Amount'].round(2)

trans_amt_stats_df
```

Out[ ]:

	Statistic	Amount
0	Mean	14.00
1	Median	10.80
2	Mode	0.05
3	Min	0.05
4	Max	1062.28

In [ ]:
```
# Find total number of transactions
total_num_trans = df_trans.shape[0]

# Find total number of transaction under 10 USD
total_num_trans_under_10usd = (df_trans['values'] < 10).sum()

# Find percentage of total transactions under 10 USD
perc_trans_under_10usd = ((total_num_trans_under_10usd / total_num_trans) *
print(f"Percent of transactions under 10 USD: {perc_trans_under_10usd}%")
```

```
Percent of transactions under 10 USD: 47.52%
```

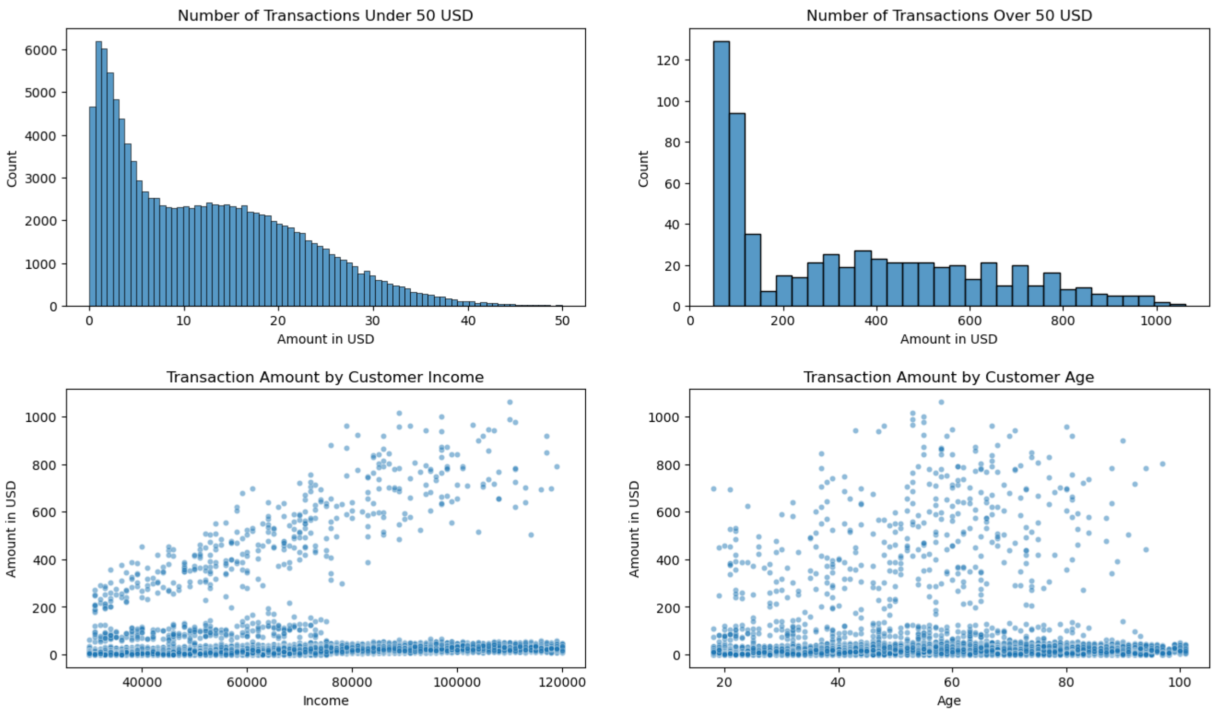
The data for transaction amount is heavily skewed to the right. It is evident that most customers choose to spend less than $20 per transaction. In fact, 47.52% of transactions are less than 10 USD.

Next, I'll create visualizations to find more information about how much customers are spending.

In [ ]:
```
# Make a column of transactions under 50 USD
df_trans_amt_filter_0 = df_trans.copy()
df_trans_amt_filter_0['values'] = df_trans_amt_filter_0['values'][df_trans_a
```

```
# Make a column of transactions over 50 USD
df_trans_amt_filter_1 = df_trans.copy()
df_trans_amt_filter_1['values'] = df_trans_amt_filter_1['values'][df_trans_a
```

In [ ]:
```
# Create a figure with 4 subplots
fig, axs = plt.subplots(2, 2, figsize=(16, 9))
fig.subplots_adjust(hspace=0.3)

# Generate histogram of transactions under 50 USD
sns.histplot(df_trans_amt_filter_0, x='values', ax=axs[0, 0])
axs[0, 0].set_title('Number of Transactions Under 50 USD')
axs[0, 0].set_xlabel('Amount in USD')

# Generate histogram of transactions over 50 USD
sns.histplot(df_trans_amt_filter_1, x='values', bins=30, ax=axs[0, 1])
axs[0, 1].set_title('Number of Transactions Over 50 USD')
axs[0, 1].set_xlabel('Amount in USD')

# Generate histogram of amount spent by income
sns.scatterplot(data=df_trans, x='income', y='values',
                s=20, alpha=0.5, ax=axs[1, 0])
axs[1, 0].set_title('Transaction Amount by Customer Income')
axs[1, 0].set_ylabel('Amount in USD')
axs[1, 0].set_xlabel('Income')

# Generate histogram of amount spent by age
sns.scatterplot(data=df_trans, x='age', y='values',
                s=20, alpha=0.5, ax=axs[1, 1])
axs[1, 1].set_title('Transaction Amount by Customer Age')
axs[1, 1].set_ylabel('Amount in USD')
axs[1, 1].set_xlabel('Age')

plt.show();
```



The histograms above confirm the findings from the statistics above: that the distribution of transactions amounts exhibits an extreme right skew, and that almost half of the total number of transactions are under 10 USD.

Offers

- Back to Data Analysis Contents

Using the table below, we can review each of the offers individually before diving deeper into the analysis.

In [ ]:
```
# Create dataframe containing each individual offer from df_promo
df_promo_offers = df_promo.copy()
df_promo_offers = df_promo_offers.drop_duplicates(subset=['values'])
df_promo_offers.reset_index(drop=True, inplace=True)
df_promo_offers.drop(columns=df_promo_offers.columns[0:8], inplace=True)
df_promo_offers
```

Out[ ]:

	values	reward	difficulty	duration	offer_type	channel_
0	5a8bc65990b245e5a138643cd4eb9837	0	0	3	informational	
1	f19421c1d4aa40978ebb69ca19b0e20d	5	5	5	bogo	
2	fafdcd668e3743c1bb461111dcafc2a4	2	10	10	discount	
3	2906b810c7d4411798c6938adc9daaa5	2	10	7	discount	
4	3f207df678b143eea3cee63160fa8bed	0	0	4	informational	
5	2298d6c36e964ae4a3e7e9706d1fb8c2	3	7	7	discount	
6	0b1e1539f2cc45b7b9fa7c272da2e1d7	5	20	10	discount	
7	9b98b8c7a33c4b65b9aebfe6a799e6d9	5	5	7	bogo	
8	4d5c57ea9a6940dd891ad53e9dbe8da0	10	10	5	bogo	
9	ae264e3637204a6fb9bb56bc8210ddfd	10	10	7	bogo	

All of the offers used email as a channel. Only one did not use mobile as a channel, and two did not use web. 6 out of 10 used social media.

The level of reward ranges from 0 to 10, `'difficulty'` is typically between 0 and 10 as well with one offer being ranked at 20, and `'duration'` ranges from 3 to 10.

There are only three offer types:

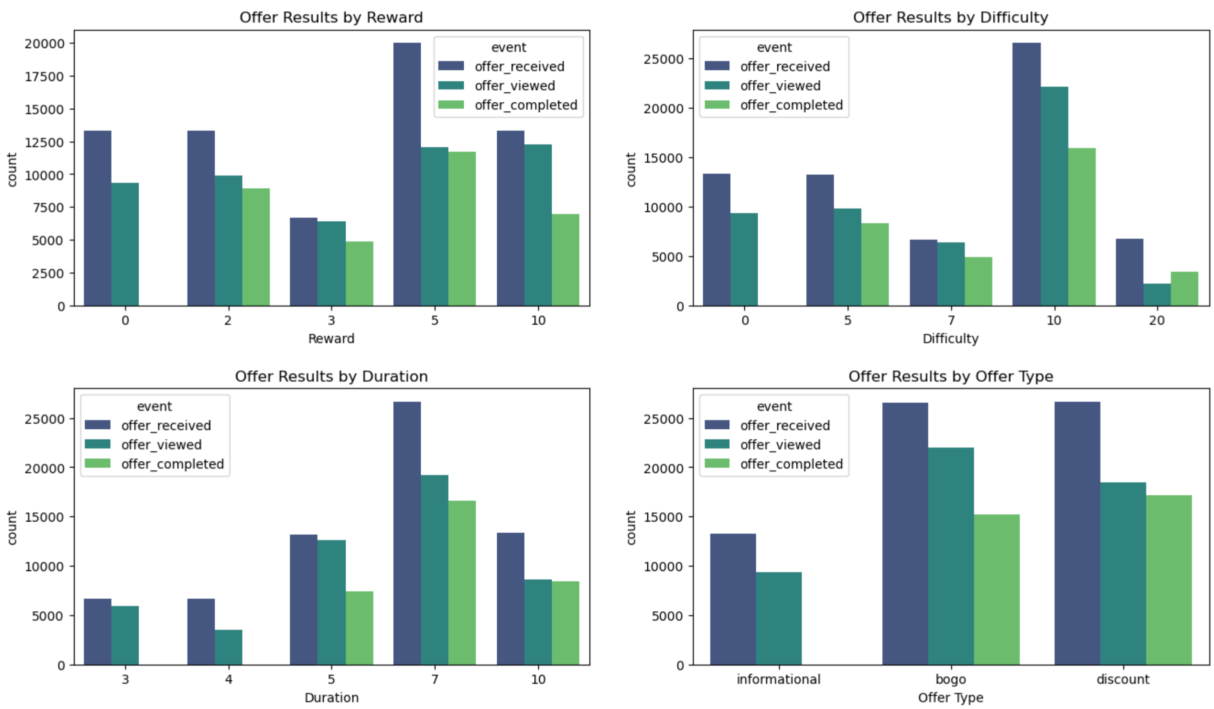
- Informational
- Bogo
- Discount

Below, I'll use more visualizations to better understand the data behind the different offers that have been given to customers.

In [ ]:

```
# Create figure with four subplots
fig, axs = plt.subplots(2, 2, figsize=(16, 9))
fig.subplots_adjust(hspace=0.3)

# Generate countplot of reward values separated by event
sns.countplot(data=df_promo, x='reward', hue='event',
              palette='viridis', ax=axs[0, 0])
axs[0, 0].set_title('Offer Results by Reward')
axs[0, 0].set_xlabel('Reward')

# Generate countplot of difficulty values separated by event
sns.countplot(data=df_promo, x='difficulty', hue='event',
              palette='viridis', ax=axs[0, 1])
axs[0, 1].set_title('Offer Results by Difficulty')
axs[0, 1].set_xlabel('Difficulty')

# Generate countplot of duration values separated by event
sns.countplot(data=df_promo, x='duration', hue='event',
```

```
                 palette='viridis', ax=axs[1, 0])
axs[1, 0].set_title('Offer Results by Duration')
axs[1, 0].set_xlabel('Duration')

# Generate countplot of offer types separated by event
sns.countplot(data=df_promo, x='offer_type', hue='event',
                 palette='viridis', ax=axs[1, 1])
axs[1, 1].set_title('Offer Results by Offer Type')
axs[1, 1].set_xlabel('Offer Type')

plt.show();
```



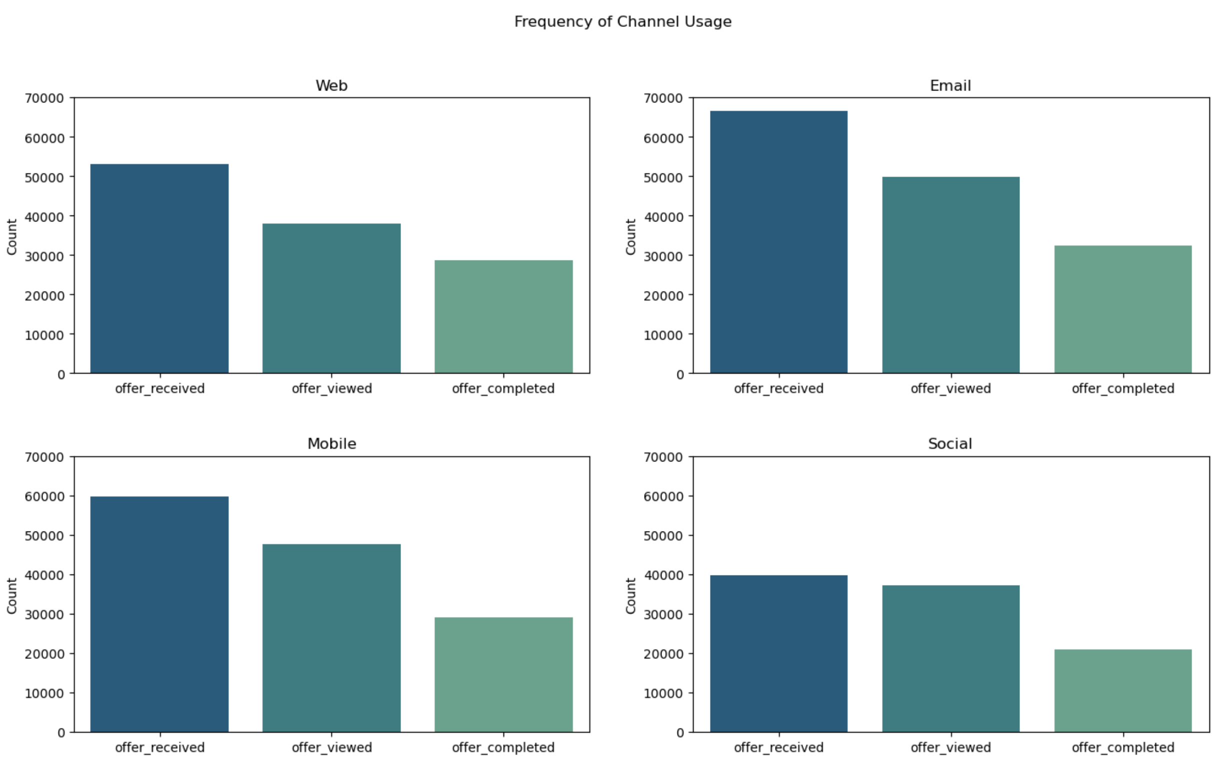
Discount offers have a higher view-to-completion ratio. In fact, the discount offer with a difficulty of 20 resulted in being completed more times than it was viewed. Additionally, the longer an offer is, the better it's view-to-completion ratio.

Even though all of the offers use some combination of web, email, mobile, and social channels, there could potentially be found valuable information about the characteristics of different channels by visualizing the frequency of events with respect to each channel.

In [ ]:

```
# Create dataframes of only rows containing each distinct channel
df_promo_web = df_promo[df_promo['channel_web'] == 1]
df_promo_email = df_promo[df_promo['channel_email'] == 1]
df_promo_mobile = df_promo[df_promo['channel_mobile'] == 1]
df_promo_social = df_promo[df_promo['channel_social'] == 1]

# Create figure with 4 subplots
fig, axs = plt.subplots(2, 2, figsize=(16, 9))
fig.suptitle('Frequency of Channel Usage')
fig.subplots_adjust(hspace=0.3)

# Generate countplot of web usage by event
sns.countplot(data=df_promo_web, x='event',
```

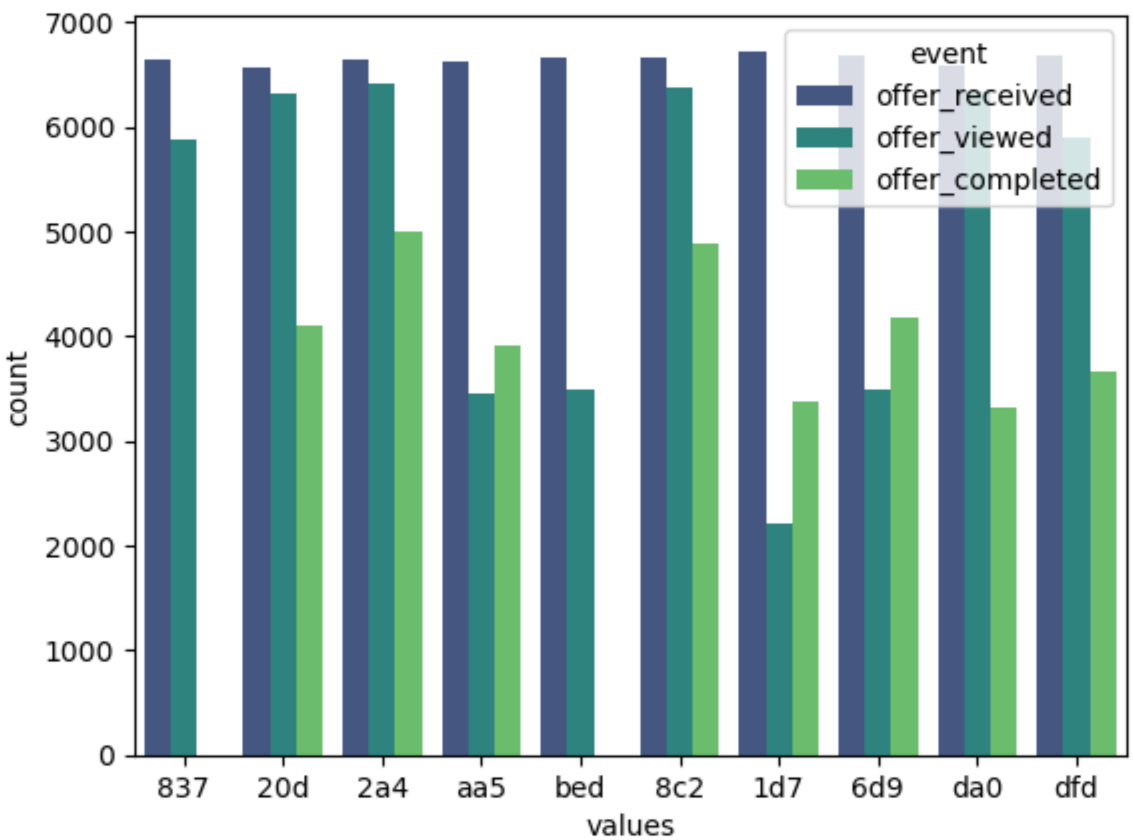
```
                 ax=axs[0, 0], palette='crest_r')
axs[0, 0].set_title('Web')
axs[0, 0].set_xlabel('')
axs[0, 0].set_ylabel('Count')
axs[0, 0].set_ylim(0, 70000)

# Generate countplot of email usage by event
sns.countplot(data=df_promo_email, x='event',
              ax=axs[0, 1], palette='crest_r')
axs[0, 1].set_title('Email')
axs[0, 1].set_xlabel('')
axs[0, 1].set_ylabel('Count')
axs[0, 1].set_ylim(0, 70000)

# Generate countplot of mobile usage by event
sns.countplot(data=df_promo_mobile, x='event',
              ax=axs[1, 0], palette='crest_r')
axs[1, 0].set_title('Mobile')
axs[1, 0].set_xlabel('')
axs[1, 0].set_ylabel('Count')
axs[1, 0].set_ylim(0, 70000)

# Generate countplot of social usage by event
sns.countplot(data=df_promo_social, x='event',
              ax=axs[1, 1], palette='crest_r')
axs[1, 1].set_title('Social')
axs[1, 1].set_xlabel('')
axs[1, 1].set_ylabel('Count')
axs[1, 1].set_ylim(0, 70000)

plt.show();
```



Social media is very effective at getting an offer viewed. However those views don't necessarily translate into completions any more so than the other channels. The relationship between the usage of social media as a promotional channel and its effectiveness will be explored in more detail throughout this project.

In the next plot, I'll get the offer results by offer ID. The offer ID will be shortened to the final 3 characters for better readability.

In [ ]:
```
# Generate countplot of offers by event
sns.countplot(data=df_promo, x=df_promo['values'].str[-3:],
              hue='event', palette='viridis')
plt.show();
```



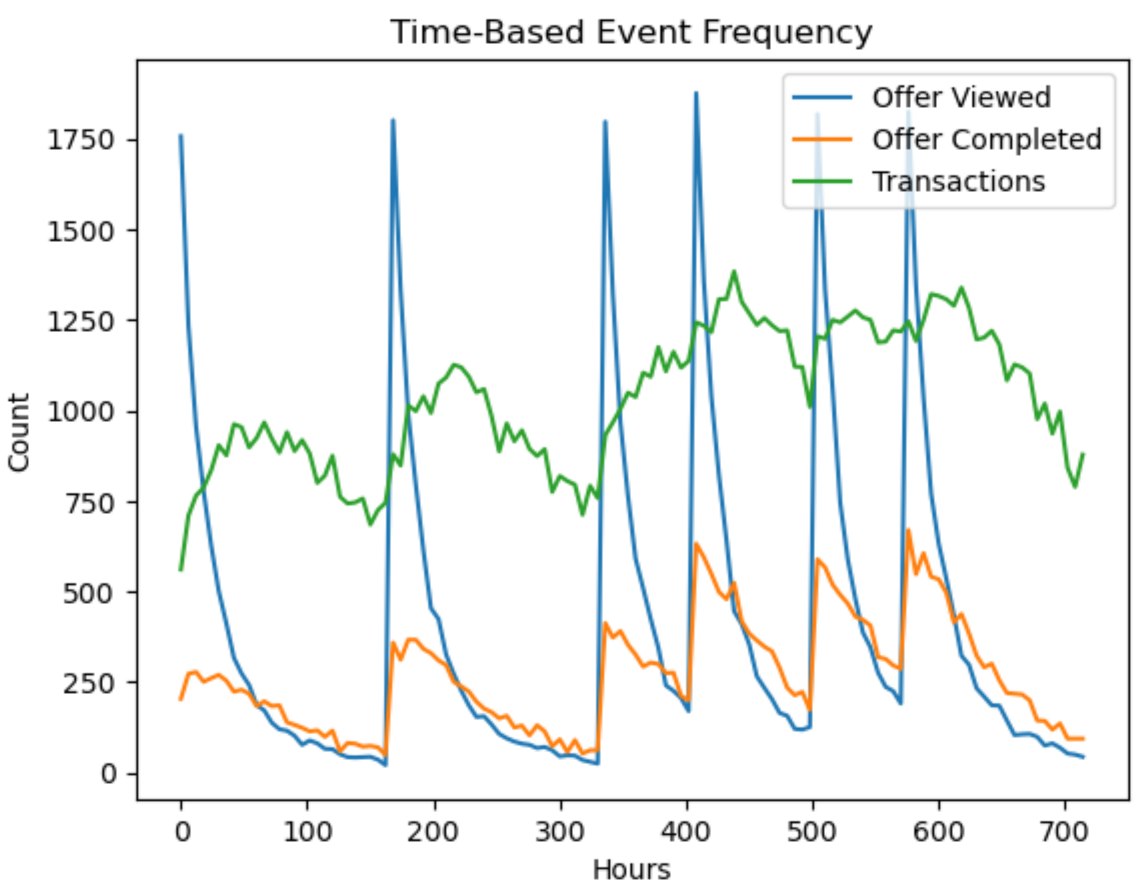
The offers in this DataFrame have been received by customers at nearly the same amount, but they differ on how much they are viewed and completed. Also, the informational offers do not have the event `'offer_completed'`.

Next, I'll evaluate the offer events with respect to time.

In [ ]:
```
# Create series of number of offers viewed by hour in order
hourly_viewed = df_promo[df_promo['event'] == 'offer_viewed'].value_counts('

# Create series of number of offers completed by hour in order
hourly_completed = df_promo[df_promo['event'] == 'offer_completed'].value_co

# Create series of number of transactions by hour in order
hourly_trans = df_trans[df_trans['event'] == 'transaction'].value_counts('ho
```

```python
# Create dataframe containing offer events by time
df_hourly_events = pd.DataFrame({'hourly_viewed': hourly_viewed,
                                 'hourly_completed': hourly_completed,
                                 'hourly_trans': hourly_trans})
```

In [ ]:
```python
# Generate lineplot of offers viewed per hour
sns.lineplot(data=df_hourly_events, x=df_hourly_events.index,
             y=hourly_viewed, label='Offer Viewed')

# Generate lineplot of offers completed per hour
sns.lineplot(data=df_hourly_events, x=df_hourly_events.index,
             y=hourly_completed, label='Offer Completed')

# Generate lineplot of transactions per hour
sns.lineplot(data=df_hourly_events, x=df_hourly_events.index,
             y=hourly_trans, label='Transactions')

plt.title('Time-Based Event Frequency')
plt.xlabel('Hours')
plt.ylabel('Count')

plt.plot();
```



The relationship between the time of completion and the offer events is simply that offers have a spike in completions immediately after they're viewed. It is worth mentioning that the completions decrease at a rate slower than the views. This indicates that there exists a portion of customers who need extra time to complete offers. This will be explored in more detail as the relationship between the variable `'duration'` and completion rate.

Analysis of Offer Effectiveness

- Back to Data Analysis Contents

In order to evaluate the effectiveness of an offer, I must first define what it is that makes an offer *effective*. From a business perspective, an effective offer would be one that produces the results that we want without requiring excess resources. This can be evaluated in the context of the available data by turning our attention toward the `'event'` column in `'df_promo'`. In this column, there are three potential categories:

- Offer Received
- Offer Viewed
- Offer Completed

From this perspective, an effective offer is one that converts reception into views but primarily turns views into completions. To assess these two outcomes, I will create two new features for each one of the offers:

- Percent Viewed
- Percent Completed

With offer effectiveness being defined as "percent completed," I will strive to answer the following questions:

1. Which offers are the most effective?
2. What makes an offer effective?
3. What gets an offer viewed?

With the conclusions gained by answering these questions, there will be sufficient information to begin making strategic, data-driven recommendations.

To begin this section, I will create a dataframe that includes the count for each event category with respect to each of the ten offers. Then, I'll find the values for `'perc_view'` and `'perc_comp'` where `'perc_view'` will be equal to the total amount of times the offer is viewed divided by the total amount of receptions times 100, and `'perc_comp'` will be equal to the total amount of times the offer is completed divided by the total amount of views times 100.

In [ ]:
```
# Create a new dataframe with the offer IDs, events, and one extra column
df_count_results_0 = df_promo.copy()
df_count_results_0.drop(df_count_results_0.iloc[:, :5], axis=1, inplace=True
```

```
df_count_results_0.drop(df_count_results_0.iloc[:, 1:3], axis=1, inplace=Tru
df_count_results_0.drop(df_count_results_0.iloc[:, -7:], axis=1, inplace=Tru

# Rename the extra column, 'reward', to 'count'
df_count_results_0.rename(columns={'reward': 'count'}, inplace=True)

# Rename 'values' to 'offer_id'
df_count_results_0.rename(columns={'values': 'offer_id'}, inplace=True)

# Find the count of each event for each offer ID
df_count_results_grouped = df_count_results_0.groupby(['offer_id', 'event'])

# Unstack grouped dataframe by the 'event' column
df_count_results_unstacked = df_count_results_grouped.unstack('event')

# Keep only the last 3 characters in the offer IDs for readability
df_count_results_unstacked.index = df_count_results_unstacked.index.str[-3:]

# View dataframe
df_count_results_unstacked
```

Out[ ]:

		count	
event	**offer_completed**	**offer_received**	**offer_viewed**
offer_id			
1d7	3386.0	6726.0	2215.0
8c2	4886.0	6655.0	6379.0
aa5	3911.0	6631.0	3460.0
bed	NaN	6657.0	3487.0
da0	3310.0	6593.0	6329.0
837	NaN	6643.0	5873.0
6d9	4188.0	6685.0	3499.0
dfd	3657.0	6683.0	5901.0
20d	4103.0	6576.0	6310.0
2a4	5003.0	6652.0	6407.0

Despite there being no decimals, the values will need to stay as floats because there are NaNs. Also this dataframe seems to agree with its corresponding visualization in the table here.

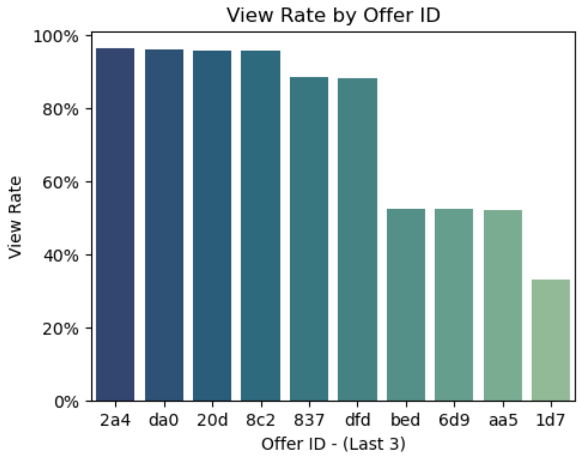
Now that I have a dataframe of the results by offer ID, I can find the view rate and completion rate for each offer. Because informational offers don't have any results for completion, they will be removed from the completion rate calculation.

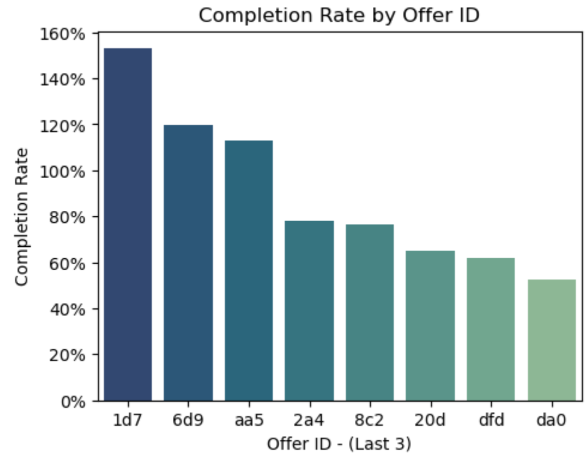
In [ ]:
```
# Initialize empty lists for offer_id and perc_view
```

```
offer_id_list_0 = []
view_rate_list_0 = []

# Generate values for offer percent viewed
for index, row in df_count_results_unstacked.iterrows():
    view_rate = round(row['count']['offer_viewed'] / \
                row['count']['offer_received'] * 100, 2)
    offer_id_list_0.append(index)
    view_rate_list_0.append(view_rate)

# Create dataframe of percent viewed by offer ID
df_view_rate = pd.DataFrame({'offer_id': offer_id_list_0,
                             'perc_view': view_rate_list_0})
```

In [ ]:
```
# Initialize empty lists for offer_id and perc_comp
offer_id_list_1 = []
comp_rate_list_0 = []

# Generate values for offer percent completed
for index, row in df_count_results_unstacked.iterrows():
    if index == 'bed' or index == '837':
        continue
    comp_rate = round(row['count']['offer_completed'] / \
                row['count']['offer_viewed'] * 100, 2)
    offer_id_list_1.append(index)
    comp_rate_list_0.append(comp_rate)

# Create dataframe of percent completed by offer ID
df_comp_rate = pd.DataFrame({'offer_id': offer_id_list_1,
                             'perc_comp': comp_rate_list_0})
```

In [ ]:
```
# Create figure with 2 subplots
fig, ax = plt.subplots(1, 2, figsize=(12, 4))
fig.subplots_adjust(wspace=0.3)

# Generate barplot showing percent viewed by offer id
sns.barplot(data=df_view_rate, x='offer_id', y='perc_view',
            order=df_view_rate.sort_values('perc_view', ascending=False)['of
            ax=ax[0], palette='crest_r')
ax[0].set_title('View Rate by Offer ID')
ax[0].set_yticks(range(0, 120, 20), ['0%', '20%', '40%', '60%', '80%', '100%
ax[0].set_ylabel('View Rate')
ax[0].set_xlabel('Offer ID - (Last 3)')

# Generate barplot showing percent completed by offer id
sns.barplot(data=df_comp_rate, x='offer_id', y='perc_comp',
            order=df_comp_rate.sort_values('perc_comp', ascending=False)['of
            ax=ax[1], palette='crest_r')
ax[1].set_title('Completion Rate by Offer ID')
ax[1].set_yticks(range(0, 180, 20), ['0%', '20%', '40%', '60%', '80%', '100%
                                      '120%', '140%', '160%'])
ax[1].set_ylabel('Completion Rate')
ax[1].set_xlabel('Offer ID - (Last 3)')

plt.show();
```





By using the plot above, we can see which offers were the most effective and which ones get viewed the most. These offers will be covered in more detail throughout this project.

The next question to answer is "what makes an offer effective?" First, I'll find the average values for a few features in order to determine if there are any significant differences between these averages by offer ID.

In [ ]:

```
# Create dataframe containing only observations of offer reception
df_offer_received = df_promo[df_promo['event'] == 'offer_received']

# Average time offer received grouped by offer IDs
df_received_avg_hrs = df_offer_received.groupby('values', as_index=False).ag
df_received_avg_hrs
```

Out[ ]:

	values	avg_hrs
0	0b1e1539f2cc45b7b9fa7c272da2e1d7	330.576271
1	2298d6c36e964ae4a3e7e9706d1fb8c2	336.108189
2	2906b810c7d4411798c6938adc9daaa5	332.163475
3	3f207df678b143eea3cee63160fa8bed	332.488508
4	4d5c57ea9a6940dd891ad53e9dbe8da0	334.620355
5	5a8bc65990b245e5a138643cd4eb9837	333.239801
6	9b98b8c7a33c4b65b9aebfe6a799e6d9	334.919372
7	ae264e3637204a6fb9bb56bc8210ddfd	329.826725
8	f19421c1d4aa40978ebb69ca19b0e20d	331.686131
9	fafdcd668e3743c1bb461111dcafc2a4	330.508719

In [ ]:

```
# Average income grouped by offer IDs
df_received_avg_inc = df_offer_received.groupby('values', as_index=False).ag
df_received_avg_inc
```

Out[ ]:

	values	avg_inc
0	0b1e1539f2cc45b7b9fa7c272da2e1d7	65179.601546
1	2298d6c36e964ae4a3e7e9706d1fb8c2	65228.099174
2	2906b810c7d4411798c6938adc9daaa5	65467.953552
3	3f207df678b143eea3cee63160fa8bed	65181.613339
4	4d5c57ea9a6940dd891ad53e9dbe8da0	65483.998180
5	5a8bc65990b245e5a138643cd4eb9837	65562.245973
6	9b98b8c7a33c4b65b9aebfe6a799e6d9	65155.422588
7	ae264e3637204a6fb9bb56bc8210ddfd	65567.409846
8	f19421c1d4aa40978ebb69ca19b0e20d	65493.461071
9	fafdcd668e3743c1bb461111dcafc2a4	65401.834035

In [ ]:

```
# Average age grouped by offer IDs
df_received_avg_age = df_offer_received.groupby('values', as_index=False).ag
df_received_avg_age
```

Out[ ]:

	values	avg_age
0	0b1e1539f2cc45b7b9fa7c272da2e1d7	54.381653
1	2298d6c36e964ae4a3e7e9706d1fb8c2	54.269872
2	2906b810c7d4411798c6938adc9daaa5	54.180365
3	3f207df678b143eea3cee63160fa8bed	54.561064
4	4d5c57ea9a6940dd891ad53e9dbe8da0	54.281207
5	5a8bc65990b245e5a138643cd4eb9837	54.558633
6	9b98b8c7a33c4b65b9aebfe6a799e6d9	54.376215
7	ae264e3637204a6fb9bb56bc8210ddfd	54.199461
8	f19421c1d4aa40978ebb69ca19b0e20d	54.474909
9	fafdcd668e3743c1bb461111dcafc2a4	54.409802

Looking at the three tables above, it is evident that there are no significant differences between the 10 offers when looking at features that aren't relevant to the offers themselves. This suggests that the variables which contribute the most to offer effectiveness can be found in the characteristics of the offers. For the sake of brevity, there will be no further statistical analyses on the demographic data as it relates to the completion and view rates at this moment.

In this next step, I'll add the offer results to the DataFrame `df_promo_offers` .

In [ ]:
```python
# Display completion rates for offer IDs
df_comp_rate
```

Out[ ]:

	offer_id	perc_comp
0	1d7	152.87
1	8c2	76.60
2	aa5	113.03
3	da0	52.30
4	6d9	119.69
5	dfd	61.97
6	20d	65.02
7	2a4	78.09

In [ ]:
```python
# Create dataframe for completion rate that includes offer information
df_comp_rate_full = df_promo_offers.copy()

# Remove two offers with no completions
df_comp_rate_full.drop([0, 4], axis=0, inplace=True)

# Add perc_comp column
df_comp_rate_full['perc_comp'] = [65.02, 78.09, 113.03, 76.60, 152.87, 119.6

# Sort by completion rate
df_comp_rate_full.sort_values('perc_comp', ascending=False, inplace=True)
df_comp_rate_full.reset_index(drop=True, inplace=True)

# Display dataframe
df_comp_rate_full
```

Out[ ]:

	values	reward	difficulty	duration	offer_type	channel_w
0	0b1e1539f2cc45b7b9fa7c272da2e1d7	5	20	10	discount	
1	9b98b8c7a33c4b65b9aebfe6a799e6d9	5	5	7	bogo	
2	2906b810c7d4411798c6938adc9daaa5	2	10	7	discount	
3	fafdcd668e3743c1bb461111dcafc2a4	2	10	10	discount	
4	2298d6c36e964ae4a3e7e9706d1fb8c2	3	7	7	discount	
5	f19421c1d4aa40978ebb69ca19b0e20d	5	5	5	bogo	
6	ae264e3637204a6fb9bb56bc8210ddfd	10	10	7	bogo	
7	4d5c57ea9a6940dd891ad53e9dbe8da0	10	10	5	bogo	

In [ ]:
```python
df_comp_rate_full.groupby('offer_type').agg({'perc_comp': 'mean'})
```

Out[ ]:

	perc_comp
offer_type	
bogo	74.7450
discount	105.1475

Discount offers have, on average, a completion rate 30% higher than bogo offers. The offer ending in '1d7' is a large contributor to this, having a completion rate greater than 150%! This means that, for each offer view, the offer was completed 1.5 times. Additionally, the three worst-performing offers were bogo, and this is despite the reward for two of them being the highest available.

In order to answer the question "what makes an offer effective," I will need to discover which features have the greatest impact on completion rate. This will be done by creating a correlation matrix and heatmap for the features in `df_comp_rate_full`. Note that `'channel_email'` is removed from this process because every offer includes email as a channel.

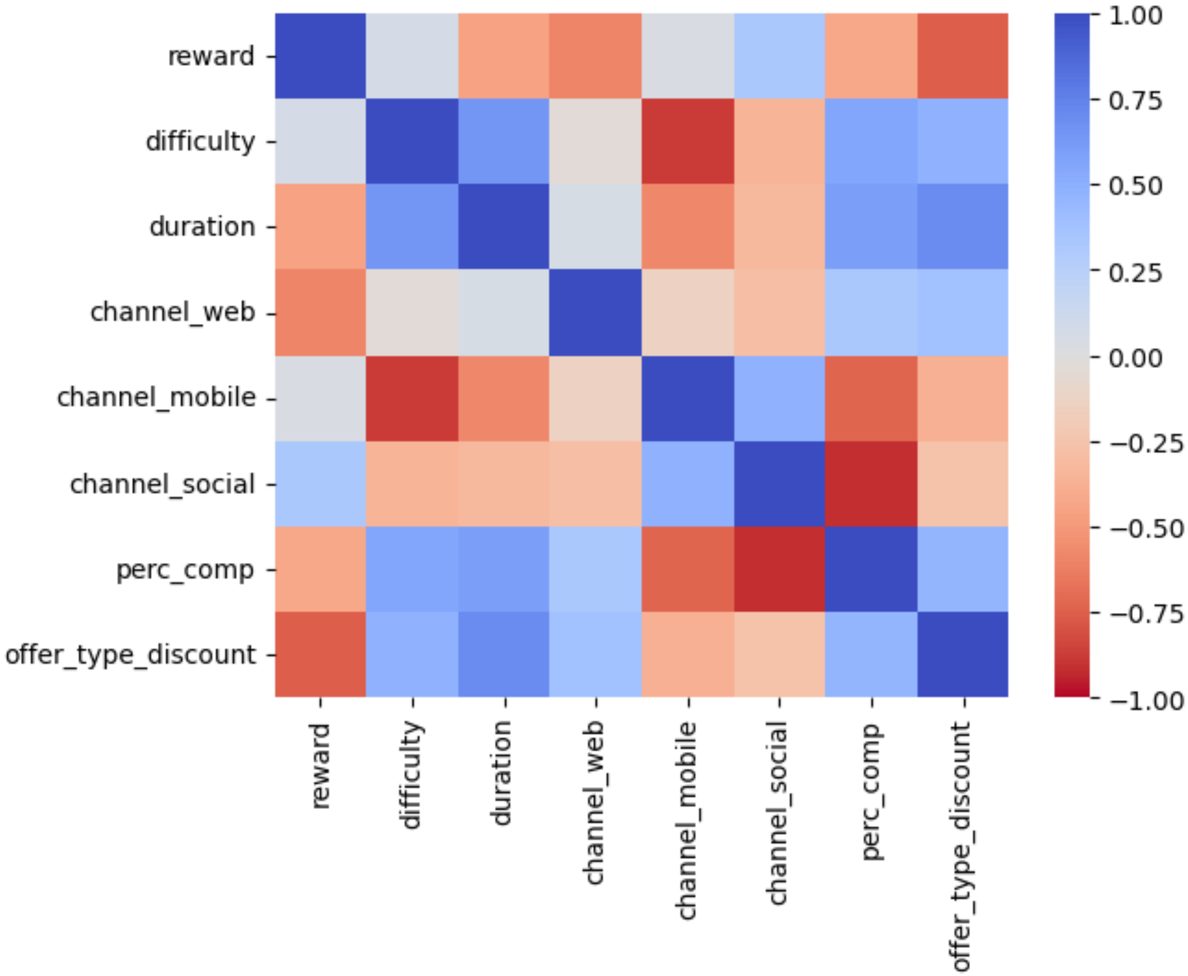
For better readability, the correlation matrix will be displayed below the heatmap rather than on top of it.

In [ ]:

```
# Dummy encode 'offer_type'
df_comp_rate_full_2 = pd.get_dummies(df_comp_rate_full, columns=['offer_type
                                     dtype=int, drop_first=True)
```

In [ ]:

```
# Create dataframe of the offer results and its numerical columns
df_comp_rate_heat = df_comp_rate_full_2.drop(columns={'values',
                                                      'channel_email'})

# Generate heatmap using the correlation matrix
sns.heatmap(df_comp_rate_heat.corr(), annot=False, vmin=-1, vmax=1, cmap='co

plt.show();
```



In [ ]:
```
# Display correlation matrix for percent completion
df_comp_rate_heat.corr()
```

Out[ ]:

	reward	difficulty	duration	channel_web	channel_mobile	chann
reward	1.000000	0.063532	-0.456679	-0.600533	0.031607	(
difficulty	0.063532	1.000000	0.642245	-0.031906	-0.882729	-(
duration	-0.456679	0.642245	1.000000	0.052926	-0.582182	-(
channel_web	-0.600533	-0.031906	0.052926	1.000000	-0.142857	-(
channel_mobile	0.031607	-0.882729	-0.582182	-0.142857	1.000000	(
channel_social	0.323875	-0.355999	-0.325396	-0.292770	0.487950	1
perc_comp	-0.429422	0.556404	0.597279	0.324075	-0.728904	-(
offer_type_discount	-0.752618	0.478352	0.700140	0.377964	-0.377964	-(

What makes an offer effective? It should be mentioned that this analysis uses a sample size of 8, so it is likely that the conclusions that can be made here won't be as robust as they would with additional offers on which to perform analysis. Because of this, a few assumptions will need to be made. One of these being that there isn't enough variation to draw conclusions about the effectiveness of `'channel_web'` and `'channel_mobile'`. The reason is that there is only one observation in which an offer does not use web as a channel and another one that doesn't use mobile. And so, despite the high correlation between `'channel_mobile'` and `'perc_comp'`, `'channel_web'` and `'channel_mobile'` can not be considered as contributors to percent completion.

Other interesting relationships in the matrix above are the correlations between `'offer_type_discount'` and the three columns of `'reward'`, `'difficulty'`, and `'duration'` as well as these three columns' correlations to `'perc_comp'`. It is the case that discount offers indeed have higher difficulties, lower rewards, and longer durations than their bogo counterparts. Additionally, the most effective offer by far had a difficulty of 20, twice that of the next highest difficulty level, and this is having a large effect on the perceived correlation between `'perc_comp'` and `'difficulty'`. It can also be seen that `'reward'` has a negative correlation with the completion rate. This is most likely due to the fact that the average reward for bogo offers is 7.5 and the average reward for discount offers was 3. All the while, discount offers performed better than bogo offers, and the correlation between `'perc_comp'` and `'offer_type_discount'` is not insignificant.

The most significant observation regarding the correlation matrix is the strong negative correlation between `'perc_comp'` and `'channel_social'`. As was suspected from the data visualizations in this section, offers that use social media as a channel are less likely to be completed than if they had not. Unfortunately, due to the nature of the available data, it is not possible to determine through which channel each individual customer received their offer.

With that being said, we can conclude that the most important factor in determining the completion rate of a promotional offer is `'channel_social'` while the second most important factor is `'offer_type'`; namely, offers distributed via social media are the most ineffective and discount offers are more effective than bogo offers.

With the question "what makes an offer effective" being answered, I will now strive to find out what it is that gets an offer viewed. For better readability, the correlation matrix will again be displayed below the heatmap rather than on top of it.

In [ ]:

```
# Display view rates for offer IDs
df_view_rate
```

Out[ ]:

	offer_id	perc_view
0	1d7	32.93
1	8c2	95.85
2	aa5	52.18
3	bed	52.38
4	da0	96.00
5	837	88.41
6	6d9	52.34
7	dfd	88.30
8	20d	95.95
9	2a4	96.32

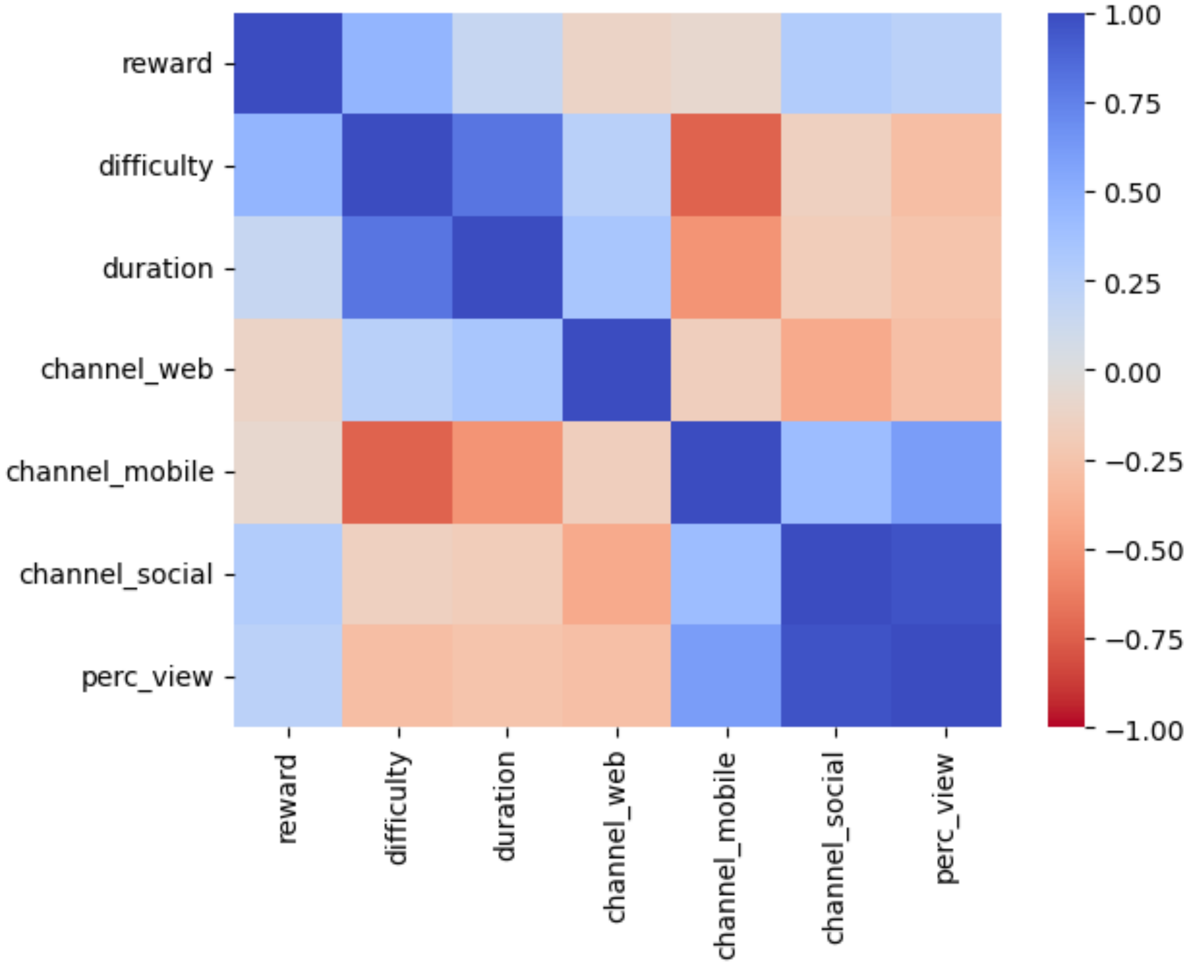
In [ ]:

```
# Create dataframe for completion rate that includes offer information
df_view_rate_full = df_promo_offers.copy()

# Add perc_view column
df_view_rate_full['perc_view'] = [88.41, 95.95, 96.32, 52.18, 52.38, 95.85,
                                   32.93, 52.34, 96.00, 88.30]

# Sort by view rate
df_view_rate_full.sort_values('perc_view', ascending=False, inplace=True)
df_view_rate_full.reset_index(drop=True, inplace=True)

# Display dataframe
df_view_rate_full
```

Out[ ]:

	values	reward	difficulty	duration	offer_type	channel_
0	fafdcd668e3743c1bb461111dcafc2a4	2	10	10	discount	
1	4d5c57ea9a6940dd891ad53e9dbe8da0	10	10	5	bogo	
2	f19421c1d4aa40978ebb69ca19b0e20d	5	5	5	bogo	
3	2298d6c36e964ae4a3e7e9706d1fb8c2	3	7	7	discount	
4	5a8bc65990b245e5a138643cd4eb9837	0	0	3	informational	
5	ae264e3637204a6fb9bb56bc8210ddfd	10	10	7	bogo	
6	3f207df678b143eea3cee63160fa8bed	0	0	4	informational	
7	9b98b8c7a33c4b65b9aebfe6a799e6d9	5	5	7	bogo	
8	2906b810c7d4411798c6938adc9daaa5	2	10	7	discount	
9	0b1e1539f2cc45b7b9fa7c272da2e1d7	5	20	10	discount	

In [ ]:
```
# Create dataframe of the offer results and its numerical columns
df_view_rate_heat = df_view_rate_full.drop(columns={'values',
                                                     'offer_type',
                                                     'channel_email'})

# Generate heatmap using the correlation matrix
sns.heatmap(df_view_rate_heat.corr(), annot=False, vmin=-1, vmax=1, cmap='co

plt.show();
```



In [ ]:
```
# Display view rate correlation matrix
df_view_rate_heat.corr()
```

Out[ ]:

	reward	difficulty	duration	channel_web	channel_mobile	channel_so
reward	1.000000	0.465686	0.160262	-0.117647	-0.078431	0.288
difficulty	0.465686	1.000000	0.808414	0.244007	-0.741058	-0.154
duration	0.160262	0.808414	1.000000	0.340557	-0.529756	-0.185
channel_web	-0.117647	0.244007	0.340557	1.000000	-0.166667	-0.408
channel_mobile	-0.078431	-0.741058	-0.529756	-0.166667	1.000000	0.408
channel_social	0.288175	-0.154957	-0.185376	-0.408248	0.408248	1.000
perc_view	0.228057	-0.295272	-0.256166	-0.284838	0.602099	0.966

Just like with `'perc_comp'`, there isn't enough information to consider the correlations of `'perc_view'` to `'channel_web'` and `'channel_mobile'`. Moreover, when it comes to percent views, there are low correlations of the views to `'reward'`, `'difficulty'`, and `'duration'`. However, similary to `'perc_comp'`, there is a large correlation between `'perc_view'` and `'channel_social'`.

From this information, it can be concluded that `'channel_social'` is the most important feature in determining whether or not an offer is viewed. However, as can be seen above, this particular feature strongly negatively correlates with percent completion.

Feature Engineering and Transformation

- Back to Table of Contents

In this section, the data will be prepared for data modeling using the K-Means unsupervised learning algorithm. In addition to this, I will work to prepare dataframes to be used in order to analyze the differences between the customers in the segments determined by K-Means.

Contents: Feature Engineering and Transformation

Feature Engineering I: Completion and View Rates

- Back to Feature Engineering and Transformation Contents

Earlier, I calculated the completion and view rates for each offer. In order to perform customer segmentation, I will need to calculate the completion and view rates for each customer. These features will be engineered in the same way that they were before, except this time the data will be grouped by customer ID number.

In [ ]:

```
# Create a new dataframe from df_promo
df_cust_results = df_promo.copy()

# Drop 'gender' and 'age' columns
df_cust_results.drop(df_cust_results.iloc[:, :2], axis=1, inplace=True)

# Drop 'member_since' and 'income' columns
df_cust_results.drop(df_cust_results.iloc[:, 1:3], axis=1, inplace=True)

# Drop remaining keeping only 'customer_id', 'event', and 'channel_social'
df_cust_results.drop(df_cust_results.iloc[:, 2:12], axis=1, inplace=True)

# Rename 'channel_social' column to 'count'
df_cust_results.rename(columns={'channel_social': 'count'}, inplace=True)

# Find the count of each event for each customer ID
df_cust_results_grouped = df_cust_results.groupby(['customer_id', 'event'],

# Unstack grouped dataframe by the 'event' column
df_cust_results_unstacked = df_cust_results_grouped.unstack('event')

# Preview dataframe
df_cust_results_unstacked.head()
```

Out[ ]:

	count		
event	offer_completed	offer_received	offer_viewed
customer_id			
0009655768c64bdeb2e877511632db8f	3.0	5.0	4.0
0011e0d4e6b944f998e987f904e8c1e5	3.0	5.0	5.0
0020c2b971eb4e9188eac86d93036a77	3.0	5.0	3.0
0020ccbbb6d84e358d3414a3ff76cffd	3.0	4.0	4.0
003d66b6608740288d6cc97a6903f4f0	3.0	5.0	4.0

In [ ]:

```
# Initialize empty list for customer ID and view rate
cust_id_list_0 = []
view_rate_list_1 = []

# Generate values for customer percent viewed
for index, row in df_cust_results_unstacked.iterrows():
    view_rate = round(row['count']['offer_viewed'] / \
                row['count']['offer_received'] * 100, 2)
    cust_id_list_0.append(index)
    view_rate_list_1.append(view_rate)

# Create dataframe of percent viewed by customer ID
df_cust_view = pd.DataFrame({'customer_id': cust_id_list_0,
                             'cust_perc_view': view_rate_list_1})

# Preview dataframe
df_cust_view.head()
```

Out[ ]:

	customer_id	cust_perc_view
0	0009655768c64bdeb2e877511632db8f	80.0
1	0011e0d4e6b944f998e987f904e8c1e5	100.0
2	0020c2b971eb4e9188eac86d93036a77	60.0
3	0020ccbbb6d84e358d3414a3ff76cffd	100.0
4	003d66b6608740288d6cc97a6903f4f0	80.0

In [ ]:

```
# Initialize empty list for customer ID and comp rate
cust_id_list_1 = []
comp_rate_list_1 = []

# Generate values for customer percent completed
for index, row in df_cust_results_unstacked.iterrows():
    comp_rate = round(row['count']['offer_completed'] / \
                row['count']['offer_viewed'] * 100, 2)
    cust_id_list_1.append(index)
    comp_rate_list_1.append(comp_rate)
```

```python
# Create dataframe of percent completed by customer ID
df_cust_comp = pd.DataFrame({'customer_id': cust_id_list_1,
                             'cust_perc_comp': comp_rate_list_1})

# Preview dataframe
df_cust_comp.head()
```

Out[ ]:

	customer_id	cust_perc_comp
0	0009655768c64bdeb2e877511632db8f	75.0
1	0011e0d4e6b944f998e987f904e8c1e5	60.0
2	0020c2b971eb4e9188eac86d93036a77	100.0
3	0020ccbbb6d84e358d3414a3ff76cffd	75.0
4	003d66b6608740288d6cc97a6903f4f0	75.0

Joining the customer viewership dataframe to the customer completion dataframe.

In [ ]:

```python
# Join customer percent view and completions in one dataframe
df_cust_results_full = pd.merge(df_cust_view, df_cust_comp, on='customer_id'

# Preview dataframe
print(df_cust_results_full.shape)
df_cust_results_full.sample(6)
```

```
(14820, 3)
```

Out[ ]:

	customer_id	cust_perc_view	cust_perc_comp
3855	429e00f2242445c4b34e612ec99e85e5	100.00	20.00
6074	68eee228713c429ebe1300155ad1fb33	100.00	60.00
12761	dc4906f1e1a0416cb955689ba144a59b	66.67	100.00
11742	ca715b3f17e24692b18f3cd90b9bf232	50.00	200.00
7635	83f56199b07249b69029948015cdf146	100.00	66.67
5613	612b51c917404bd4a62d46e5b2fedfca	100.00	50.00

In [ ]:

```python
# Print sum of NA rows
df_cust_results_full.isna().sum()
```

Out[ ]:

```
customer_id          0
cust_perc_view     145
cust_perc_comp    2904
dtype: int64
```

Out of a total of 14,820 rows, there are a potential maximum of 3,049 NaNs. Most of these are in the `'cust_perc_comp'` column. This means that there is a considerable number of customers in the dataset who are completing offers, but there is no data regarding when they viewed them if at all. Because of this, it may be misleading or incorrect to replace the NaNs with zeros or averages. In order to deal with this problem, I will be removing the NaN values from `df_cust_results_full` .

In [ ]:
```
# Drop NaNs
df_cust_results_full.dropna(axis=0, inplace=True)
df_cust_results_full.reset_index(drop=True, inplace=True)
```

This DataFrame will receive more work before it is ready to be used in K-Means.

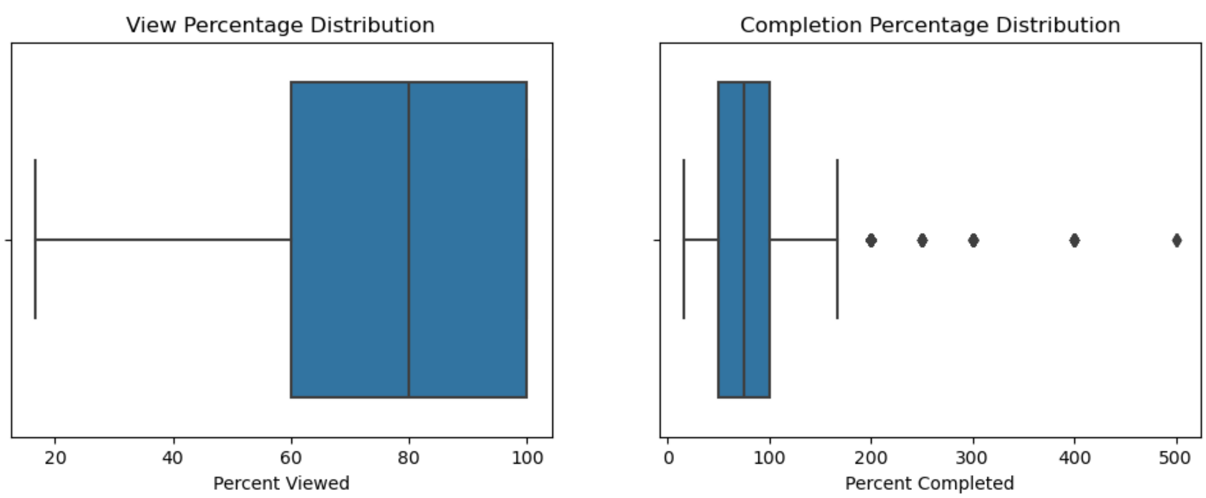
Feature Transformation I: Feature Scaling

- Back to Feature Engineering and Transformation Contents

I will start this section by checking the distribution of `'cust_perc_view'` and `'cust_perc_comp'` to ensure that there are no outliers that will negatively affect the clustering process.

In [ ]:
```
# Create figure of 4 subplots
fig, ax = plt.subplots(1, 2, figsize=(12 , 4))

# Generate boxplot of customer percent viewership
sns.boxplot(data=df_cust_results_full, x='cust_perc_view', ax=ax[0])
ax[0].set_title('View Percentage Distribution')
ax[0].set_xlabel('Percent Viewed')

# Generate boxplot of customer percent completion
sns.boxplot(data=df_cust_results_full, x='cust_perc_comp', ax=ax[1])
ax[1].set_title('Completion Percentage Distribution')
ax[1].set_xlabel('Percent Completed')
```

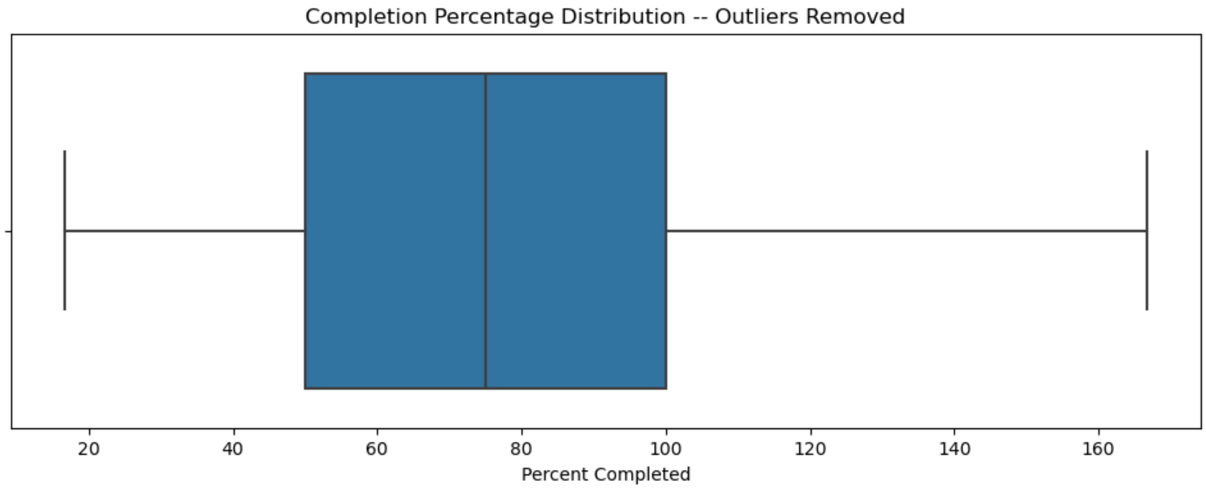
Out[ ]:
```
Text(0.5, 0, 'Percent Completed')
```



The completion percentage distribution is skewed to the right. So much so that it is likely that this will affect the K-Means clustering algorithm, making it more difficult to create reliable segments. To deal with this, I will be removing the values which lie outside of 1.5 times the interquartile range (IQR) from Q1 and Q3.

In [ ]:
```
def iqr_rm_outliers(df, column):
    """
    Removes outliers from a column using the IQR method of identification

    Args:
        df: A Pandas DataFrame
        column: (string) The column with the outliers to be removed

    Returns:
        Pandas DataFrame with the outliers removed from the specified column
    """

    # Calculate the first and third quartiles
    q1 = df[column].quantile(0.25)
    q3 = df[column].quantile(0.75)

    # Find interquartile range
    iqr = q3 - q1

    # Define the upper and lower bounds
    lower_bound = q1 - (1.5 * iqr)
    upper_bound = q3 + (1.5 * iqr)

    # Remove observations below the lower bound and above the upper bound
    df_rm_outliers = df[(df[column] >= lower_bound) & (df[column] <= upper_b

    return df_rm_outliers
```

In [ ]:
```
# Remove outliers
df_cust_results_rm_outliers = iqr_rm_outliers(df_cust_results_full, 'cust_pe

# Plot the new distribution
plt.figure(figsize=(12, 4))
sns.boxplot(data=df_cust_results_rm_outliers, x='cust_perc_comp')
plt.title('Completion Percentage Distribution -- Outliers Removed')
plt.xlabel('Percent Completed')
plt.show();
```



The next step in preparing the data for clustering is to scale `'cust_perc_view'` and `'cust_perc_comp'` using scikit-learn's `MinMaxScaler`.

In [ ]:
```
# Instantiate the scaler
scaler = MinMaxScaler()

# Create new dataframe for the scaled data
df_cust_seg = df_cust_results_rm_outliers.copy()

# Define columns to scale
scale_cols = ['cust_perc_view', 'cust_perc_comp']

# Fit the scaler to the columns to scale
df_cust_seg[scale_cols] = scaler.fit_transform(df_cust_seg[scale_cols])

# Preview dataframe
print(df_cust_seg.shape)
df_cust_seg.head()
```

```
(11395, 3)
```

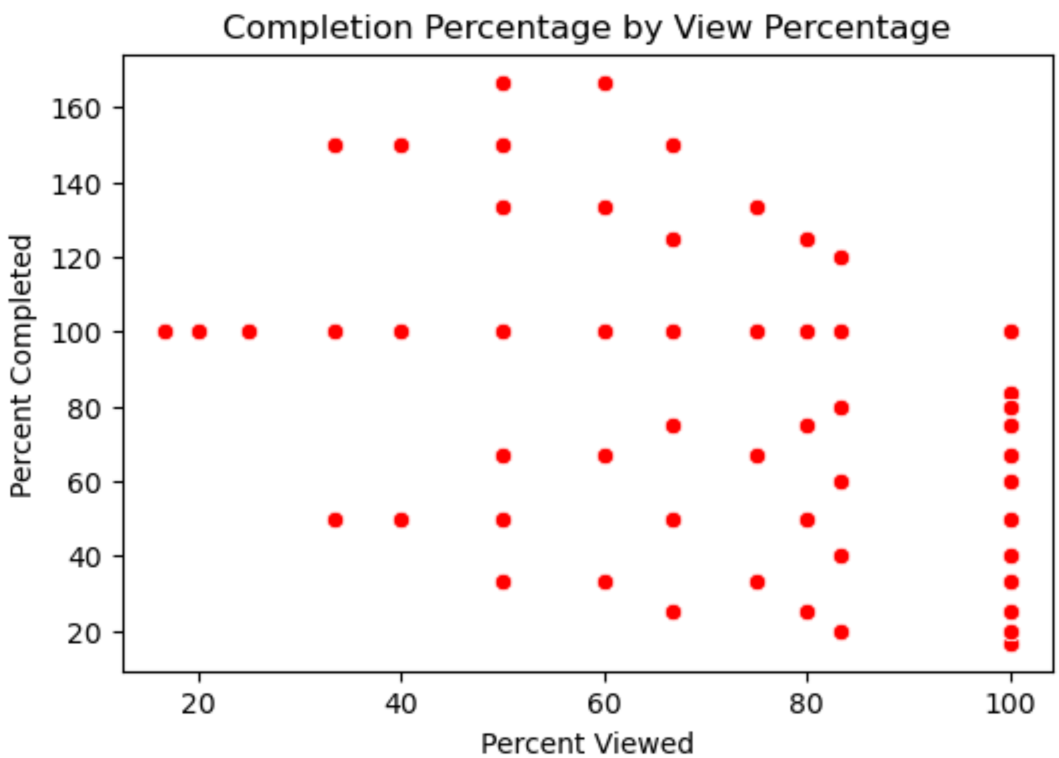
Out[ ]:

	customer_id	cust_perc_view	cust_perc_comp
0	0009655768c64bdeb2e877511632db8f	0.759990	0.388867
1	0011e0d4e6b944f998e987f904e8c1e5	1.000000	0.288867
2	0020c2b971eb4e9188eac86d93036a77	0.519981	0.555533
3	0020ccbbb6d84e358d3414a3ff76cffd	1.000000	0.388867
4	003d66b6608740288d6cc97a6903f4f0	0.759990	0.388867

Before moving on, I will display a scatterplot of the two variables that will be used in the clustering algorithm. This is so that it can be viewed before the segments are decided. It will be discussed later in order to make more sense of the evaluation metrics.

In [ ]:
```
# Create figure
plt.figure(figsize=(6, 4))
```

```
# Generate scatterplot of perc_comp by perc_view
sns.scatterplot(data=df_cust_results_rm_outliers, x='cust_perc_view',
                y='cust_perc_comp', color='red')
plt.title('Completion Percentage by View Percentage')
plt.xlabel('Percent Viewed')
plt.ylabel('Percent Completed')

plt.show();
```



Even though there are more than 14,000 observations in this dataframe, this scatterplot shows that there are less than 60 potential x and y values for each observation.

Feature Engineering II: Demographic and Transaction Data

- Back to Feature Engineering and Transformation Contents

In order to evaluate the data model and understand the results of the customer segmentation process, I will need to understand who are the customers in each cluster. For this purpose, I will be creating another dataframe that contains additional customer-specific information including the total amount spent, total number of transactions, and average spend per transaction.

In [ ]:
```
# Find total transaction amount for each individual customer
df_engr_0 = df_trans.groupby('customer_id').agg({'values': 'sum'}).reset_ind

# Rename total transaction amount column to total_trans_amt
df_engr_0.rename(columns={'values': 'total_trans_amt'}, inplace=True)
```

```
# Find total transaction count for each individual customer
df_engr_1 = df_trans.groupby('customer_id').agg({'values': 'count'}).reset_i

# Rename total transaction count column to count_trans
df_engr_1.rename(columns={'values': 'count_trans'}, inplace=True)

# Perform join on customer_id to get total_trans_amt & count_trans in one da
df_engr_2 = pd.merge(df_engr_0, df_engr_1, on='customer_id')

# Calculate average transaction amount for each individual customer
df_engr_2['avg_trans_amt'] = df_engr_2['total_trans_amt'] / df_engr_2['count
df_engr_2['avg_trans_amt'] = df_engr_2['avg_trans_amt'].round(2)

# Join customer view and completion rates to total transaction information
df_engr_3 = pd.merge(df_cust_results_full, df_engr_2, on='customer_id', how=

# Join customer behavioral information to the demographic data
df_engr_4 = pd.merge(df_engr_3, df_cust, on='customer_id', how='inner')

# Preview dataframe
print(df_engr_4.shape)
df_engr_4.head()
```

(11916, 10)

Out[ ]:

	customer_id	cust_perc_view	cust_perc_comp	total_trans_amt
0	0009655768c64bdeb2e877511632db8f	80.0	75.0	127.60
1	0011e0d4e6b944f998e987f904e8c1e5	100.0	60.0	79.46
2	0020c2b971eb4e9188eac86d93036a77	60.0	100.0	196.86
3	0020ccbbb6d84e358d3414a3ff76cffd	100.0	75.0	154.05
4	003d66b6608740288d6cc97a6903f4f0	80.0	75.0	48.34

Feature Transformation II: Customer Offer Reception

- Back to Feature Engineering and Transformation Contents

For the model evaluation section of this project, not only will I be using demographic and transaction data regarding each individual customer in the segmentation dataset, but also information about which offers they received. Ultimately, the offers which they received will provide further insight as to which features account for the differences between the customer segments.

In [ ]:

```
df_form_1 = df_promo.copy()

# Dummy encode offer_type column
df_form_2 = pd.get_dummies(df_form_1, columns=['offer_type'], dtype=int)

# Keep only offer_received observations
```

```
df_form_3 = df_form_2[df_form_2['event'] == 'offer_received']

# Sum offer information columns
df_form_4 = df_form_3.groupby('customer_id', as_index=False).agg({'channel_w
                                                                   'channel_e
                                                                   'channel_m
                                                                   'channel_s
                                                                   'offer_typ
                                                                   'offer_typ
                                                                   'offer_typ
                                                                   'reward':
                                                                   'difficult
                                                                   'duration'


# Preview dataframe
df_form_4.head()
```

Out[ ]:

	customer_id	channel_web	channel_email	channel_mobile	chan
0	0009655768c64bdeb2e877511632db8f	4	5	5	
1	0011e0d4e6b944f998e987f904e8c1e5	4	5	4	
2	0020c2b971eb4e9188eac86d93036a77	3	5	5	
3	0020ccbbb6d84e358d3414a3ff76cffd	3	4	4	
4	003d66b6608740288d6cc97a6903f4f0	4	5	4	

In [ ]:

```
# Join customer offer information with customer demographic and transaction
df_eval_0 = pd.merge(df_engr_4, df_form_4, on='customer_id')

# Preview dataframe
print(df_eval_0.shape)
df_eval_0.sample(6)
```

(11916, 20)

Out[ ]:

	customer_id	cust_perc_view	cust_perc_comp	total_trans_ar
11213	f105b8f61dda45739cd5b0d64807ec0a	33.33	200.00	196.(
4398	5e9e648ac1924ed19cc9665f000c0309	33.33	300.00	113.:
6742	90e3006dafc4487aa3edadcaa8cc36de	100.00	80.00	88.
10884	e95e8899562f4cad8d53ed81367af82a	60.00	133.33	421.
7595	a216df544a944c2f9796f44e20d0de78	40.00	150.00	156.:
3018	40c068d24b4c48e995bbbc6351429157	50.00	33.33	19.!

In [ ]:

```
df_eval_0.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 11916 entries, 0 to 11915
Data columns (total 20 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   customer_id               11916 non-null  object
 1   cust_perc_view            11916 non-null  float64
 2   cust_perc_comp            11916 non-null  float64
 3   total_trans_amt           11916 non-null  float64
 4   count_trans               11916 non-null  int64
 5   avg_trans_amt             11916 non-null  float64
 6   gender                    11916 non-null  object
 7   age                       11916 non-null  int64
 8   member_since              11916 non-null  datetime64[ns]
 9   income                    11916 non-null  float64
 10  channel_web               11916 non-null  int64
 11  channel_email             11916 non-null  int64
 12  channel_mobile            11916 non-null  int64
 13  channel_social            11916 non-null  int64
 14  offer_type_informational  11916 non-null  int64
 15  offer_type_discount       11916 non-null  int64
 16  offer_type_bogo           11916 non-null  int64
 17  reward                    11916 non-null  int64
 18  difficulty                11916 non-null  int64
 19  duration                  11916 non-null  int64
dtypes: datetime64[ns](1), float64(5), int64(12), object(2)
memory usage: 1.8+ MB
```

There are a bit more customer IDs in this dataframe than in the one which will be used for clustering `df_cust_seg`. Later on, I will perform an inner join to add the customer cluster assignment to the corresponding customer ID. This will remove the additional customer IDs from `df_eval_0`.

Data Modeling

- Back to Table of Contents

In this section, I will use the K-Means clustering algorithm to segment the customers based on the data available. The process that follows will include 5 steps:

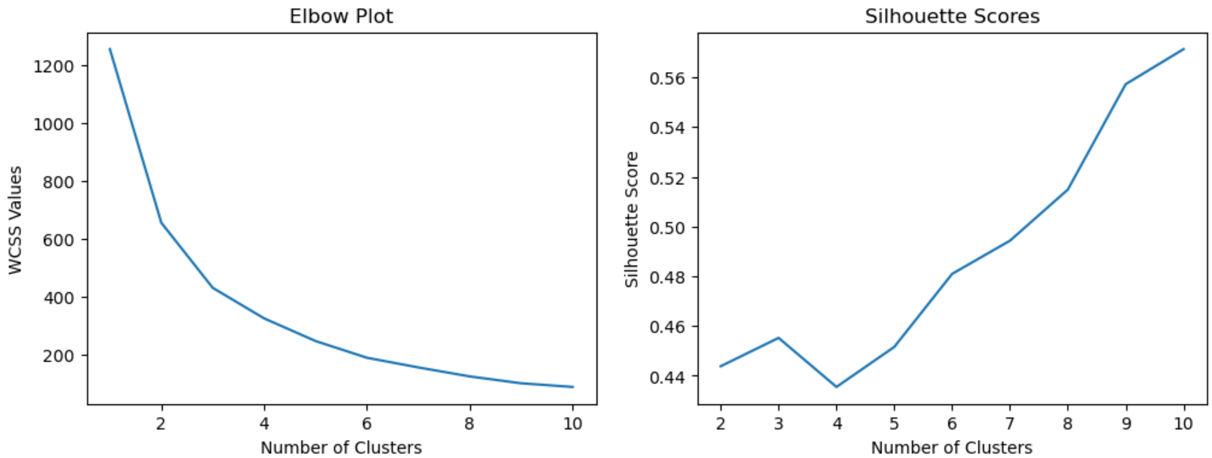
- Create the dataframe for segmentation
- Find the within cluster sum of squares (WCSS) and silhoutte scores for clusters ranging from 2 to 10
- Examine plots for the elbow method and silhouette method to determine the optimal amount of clusters
- Train a model using K-Means
- Plot the clusters

In [ ]:
```
# Assign df_cust_seg to the variable X
X = df_cust_seg.iloc[:, [1, 2]]

# Initialize list for within cluster sum of squares values
wcss = []

# Generate wcss for 0-10 clusters
for i in range(1, 11):
    kmeans = KMeans(n_clusters=i, init='k-means++', random_state=42)
    kmeans.fit(X)
    wcss.append(kmeans.inertia_)

# Create dataframe with the cluster and wcss values
df_wcss = pd.DataFrame({'cluster': range(1, 11), 'wcss': wcss})
```

In [ ]:
```
# Initialize list for silhouette score values
sil_score = []

# Generate silhouette scores for 0 - 10 custers
for i in range(2, 11):
    kmeans = KMeans(n_clusters=i, init='k-means++', random_state=42)
    kmeans.fit(X)
    sil_score.append(silhouette_score(X, kmeans.labels_))

# Create dataframe with the cluster and silhouette score values
df_sil = pd.DataFrame({'cluster': range(2, 11), 'sil_score': sil_score})
```

In [ ]:
```
# Create figure with 2 subplots
fig, ax = plt.subplots(1, 2, figsize=(12, 4))

# Generate elbow plot
sns.lineplot(data=df_wcss, x='cluster', y='wcss', ax=ax[0])
ax[0].set_title('Elbow Plot')
ax[0].set_xlabel('Number of Clusters')
```

```
ax[0].set_ylabel('WCSS Values')

# Generate plot of silhouette scores
sns.lineplot(data=df_sil, x='cluster', y='sil_score', ax=ax[1])
ax[1].set_title('Silhouette Scores')
ax[1].set_xlabel('Number of Clusters')
ax[1].set_ylabel('Silhouette Score')

plt.show();
```



From the scatterplot at the end of the Feature Transformation section titled "Feature Transformation I: Feature Scaling," it is evident that there are less than 60 potential values for more than 14,000 observations. The silhouette score evaluates both the cohesion within clusters and the separation between them. Due to the unique nature of this data, the silhouette score continues to increase when k > 4.

Moving on to the elbow plot, we can see that there isn't an obviously identifiable elbow. Although it is difficult to see, the elbows at k > 3 do not produce as much of an angle as the one at k = 3.

With this information at hand, I will be choosing to fit the model using 3 clusters. This amount of clusters produces the local maximum silhouette score as well as the smallest angle (if ever so slightly) in the elbow plot.

In [ ]:
```
# Fit a 3-cluster model to the data
kmeans3 = KMeans(n_clusters=3, random_state=42)
kmeans3.fit(X)
```

Out[ ]:
```
▾ KMeans
KMeans(n_clusters=3, random_state=42)
```

In [ ]:
```
# Print unique cluster labels
np.unique(kmeans3.labels_)
```

Out[ ]:
```
array([0, 1, 2], dtype=int32)
```

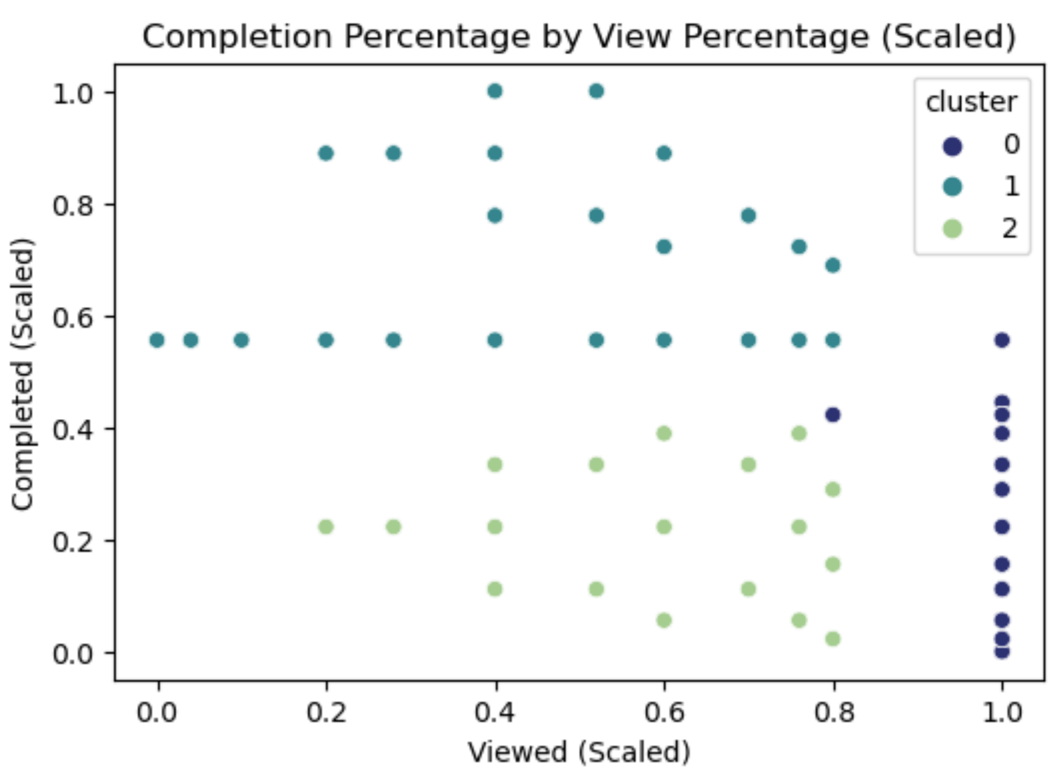
Creating a column in `df_cust_seg` to specify which cluster each customer belongs to.

In [ ]:
```
# Create column of cluster labels
df_cust_seg['cluster'] = kmeans3.labels_

# Preview a sample of the dataframe
df_cust_seg.sample(5)
```

Out[ ]:

	customer_id	cust_perc_view	cust_perc_comp	cluster
9878	d42ea4439175405fa8f010b27483e7c2	0.600024	0.555533	1
1329	1d832bb5ffc6420ea7cea524ccdd45e1	1.000000	0.333333	0
3435	496ac7798e9d49e68468823dad0f43df	0.399976	0.888867	1
8662	b8dc729c581045a59f2186c210c5a06c	0.199928	0.555533	1
10669	e4ebf2c6fb654838be483f43c6e31b82	1.000000	0.444400	0

In [ ]:
```
# Create a figure
plt.figure(figsize=(6, 4))

# Generate a scatterplot to display the results of clustering
sns.scatterplot(data=df_cust_seg, x='cust_perc_view', y='cust_perc_comp',
                hue='cluster', palette='crest_r')
plt.title('Completion Percentage by View Percentage (Scaled)')
plt.xlabel('Viewed (Scaled)')
plt.ylabel('Completed (Scaled)')

plt.show();
```



It appears that the algorithm has chosen to place the customers who view nearly all of the offers that they receive into their own cluster. Moreover, for much of the range of `'cust_perc_view'`, it has chosen a completion rate of ~0.5 (scaled value, ~90% actual) as the cutoff between clusters 1 and 2. In the next section, I'll evaluate what the characteristics are that separate these three clusters.

Moving forward, for simplicity, I will be using three more human-friendly names for the clusters. They are:

- Cluster 0
  - *"Curious"*
  - This group will be referred to as "curious customers" or the "curious segment"
- Cluster 1
  - *"Committed"*
  - This group will be referred to as "committed customers" or the "committed segment"
- Cluster 2
  - *"Casual"*
  - This group will be referred to as "casual customers" or the "casual segment"

Model Evaluation and Results

- Back to Table of Contents

The goal of this section is to discover what are these characteristics that define the customers in each segment. For additional information regarding these segments, I'll add the cluster integers to the dataframe made in "Feature Transformation II: Customer Offer Reception." Then, I'll print how many observations are in each one along with the percent of customers that they have.

In [ ]:
```
# Select clusters and customer IDs from df_cust_seg
df_engr_6 = df_cust_seg.iloc[:, [0, 3]]

# Use inner join to add clusters to df_eval_0
df_eval_1 = pd.merge(df_engr_6, df_eval_0, on='customer_id')
```

In [ ]:
```
# Print the number of observations in each cluster
df_eval_1['cluster'].value_counts()
```

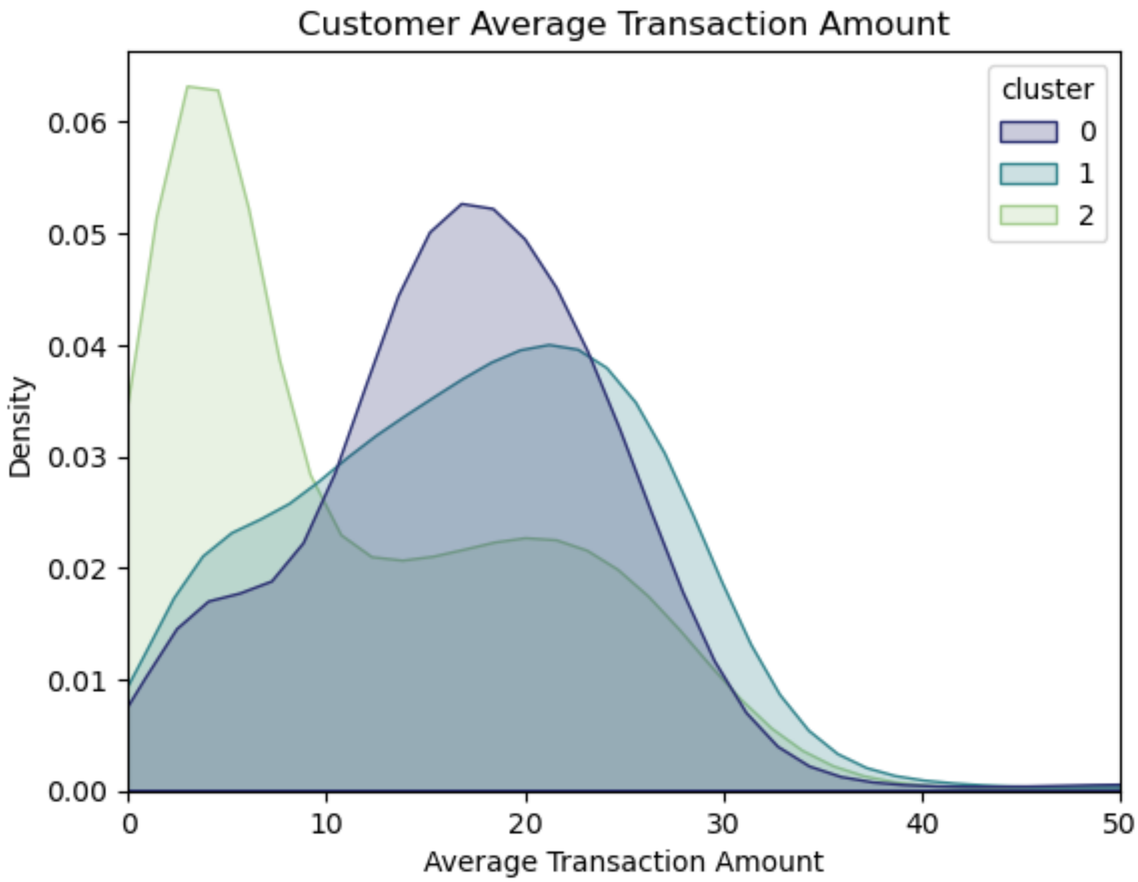
Out[ ]:
```
cluster
0    4773
1    3417
2    3205
Name: count, dtype: int64
```

In [ ]:
```
# Print the percent of observations in each cluster
df_eval_1['cluster'].value_counts(normalize=True)
```

Out[ ]:
```
cluster
0    0.418868
1    0.299868
2    0.281264
Name: proportion, dtype: float64
```

From the plot below we can visually identify that the casual segment is comprised of those who spend less per transaction.

In [ ]:
```
# Generate density plot of customer average transaction amounts by cluster
sns.kdeplot(data=df_eval_1, x='avg_trans_amt', hue=df_eval_1['cluster'],
            fill=True, common_norm=False, palette='crest_r')
plt.xlim(0, 50)
plt.xlabel('Average Transaction Amount')
plt.title('Customer Average Transaction Amount')

plt.show();
```



In [ ]:
```
# Display measures of central tendency by cluster for avg_trans_amt
df_eval_1.groupby('cluster').agg({'avg_trans_amt': ['mean', 'median', mode]}
```

Out[ ]:

	avg_trans_amt		
	mean	**median**	**mode**
cluster			
0	18.708508	17.33	(14.54, 8)
1	19.059198	18.19	(18.94, 7)
2	12.225011	6.98	(2.46, 17)

Among the casual segment, a right skew can be observed in the distribution of average transaction amount. It is the casual customer segment that is largely responsible for the observation during data analysis that 47.52% of all transactions are under 10 USD. The transaction amount distribution without the separation by cluster can be seen on the first histogram in this figure.

Even though the defining characteristic of the casual customer segment has been found, the average transaction amount isn't displaying much of a difference between the committed customers and the curious customers. In order to uncover the differences between these two, I'll group `df_eval_1` by cluster and display the average for all of the numeric columns.

In [ ]:

```
df_eval_1.groupby('cluster', as_index=False).agg({'channel_web': 'mean',
                                                   'channel_email': 'mean',
                                                   'channel_mobile': 'mean',
                                                   'channel_social': 'mean',
                                                   'offer_type_informational'
                                                   'offer_type_discount': 'me
                                                   'offer_type_bogo': 'mean',
                                                   'reward': 'mean',
                                                   'duration': 'mean',
                                                   'difficulty': 'mean',
                                                   'income': 'mean',
                                                   'age': 'mean',
                                                   'count_trans': 'mean',
                                                   'total_trans_amt': 'mean',
                                                   'avg_trans_amt': 'mean',
                                                   'cust_perc_view': 'mean',
                                                   'cust_perc_comp': 'mean'})
```

Out[ ]:

	cluster	channel_web	channel_email	channel_mobile	channel_social	offer_type_informat
0	0	3.412948	4.314268	4.029332	2.917033	0.8(
1	1	3.904302	4.647059	4.055604	2.564823	0.6;
2	2	3.740094	4.816849	4.364119	2.892668	1.2

During the Analysis of Offer Effectiveness, we were able to learn that using social media as a promotional medium is a driving force behind increased viewership but decreased completions. In addition to this, we also found out that discount offers are more effective than bogo offers, meaning that they have higher completion rates, despite having lower rewards and higher difficulties on average.

With this information in mind, we can conclude that what separates the committed customers from the curious customers is that the committed customers received, on average, 12% fewer offers via social media and 26% more discount offers. Even though the difference in social media offers received is relatively small, the correlations of `'channel_social'` to the view and completion rates has been found to be very high.

Moreover, though `'duration'` is not as much of a predictor as `'channel_social'` and `'offer_type'` , the committed segment had, on average, 14% more time to complete the offers that they received than the curious segment. The combination of these variables is what led to a lower view rate among the committed customers than the curious segment but almost twice the completion rate.

This marks the end of this project. A presentation of the results as well as data-driven recommendations are in the Executive Summary at the beginning of this document.