Google

Data Analytics Professional Certification Capstone

Google Data Analytics Capstone – Ask, Prepare, Process, Analyze, Share and Act
Author: Felipe Seleme Ribeiro | felipeselemeribeiro@gmail.com
Date: February 2023

Case Study: **Cyclistic bike share**

How does a bike-share navigate speedy success?

Scenario

You are a junior data analyst working in the marketing analyst team at Cyclistic, a bike-share company in Chicago. The director of marketing believes the company's future success depends on maximizing the number of annual memberships. Therefore, your team wants to understand how casual riders and annual members use Cyclistic bikes differently. From these insights, your team will design a new marketing strategy to convert casual riders into annual members. But first, Cyclistic executives must approve your recommendations, so they must be backed up with compelling data insights and professional data visualizations.

Characters and teams

● Cyclistic: A bike-share program that features more than 5,800 bicycles and 600 docking stations. Cyclistic sets itself apart by also offering reclining bikes, hand tricycles, and cargo bikes, making bike-share more inclusive to people with disabilities and riders who can't use a standard two-wheeled bike. The majority of riders opt for traditional bikes; about 8% of riders use the assistive options. Cyclistic users are more likely to ride for leisure, but about 30% use them to commute to work each day.

● Lily Moreno: The director of marketing and your manager. Moreno is responsible for the development of campaigns and initiatives to promote the bike-share program. These may include email, social media, and other channels.

● Cyclistic marketing analytics team: A team of data analysts who are responsible for collecting, analyzing, and reporting data that helps guide Cyclistic marketing strategy. You joined this team six months ago and have been busy learning about Cyclistic's mission and business goals — as well as how you, as a junior data analyst, can help Cyclistic achieve them.

● Cyclistic executive team: The notoriously detail-oriented executive team will decide whether to approve the recommended marketing program.

About the company

In 2016, Cyclistic launched a successful bike-share offering. Since then, the program has grown to a fleet of 5,824 bicycles that are geotracked and locked into a network of 692 stations across Chicago. The bikes can be unlocked from one station and returned to any other station in the system anytime.

Until now, Cyclistic's marketing strategy relied on building general awareness and appealing to broad consumer segments. One approach that helped make these things possible was the flexibility of its pricing plans: single-ride passes, full-day passes, and annual memberships. Customers who purchase single-ride or full-day passes are referred to as casual riders. Customers who purchase annual memberships are Cyclistic members.

Cyclistic's finance analysts have concluded that annual members are much more profitable than casual riders. Although the pricing flexibility helps Cyclistic attract more customers, Moreno believes that maximizing the number of annual members will be key to future growth. Rather than creating a marketing campaign that targets all-new customers, Moreno believes there is a very good chance to convert casual riders into members. She notes that casual riders are already aware of the Cyclistic program and have chosen Cyclistic for their mobility needs.

Moreno has set a clear goal: Design marketing strategies aimed at converting casual riders into annual members. In order to do that, however, the marketing analyst team needs to better understand how annual members and casual riders differ, why casual riders would buy a membership, and how digital media could affect their marketing tactics. Moreno and her team are interested in analyzing the Cyclistic historical bike trip data to identify trends.

Phase 1 - Ask

To design marketing strategies aimed at converting casual riders into annual members, the goal is to determine “**how do annual members and casual riders use Cyclistic bikes differently?**”.

Phase 2 - Prepare

To analyze and identify trends, I'm going to use the previous 12 months of Cyclistic's historical trip data made available by Motivate International Inc.

The data is reliable as it was obtained directly from the company.

The files are in CSV format and have the unique identification information for each trip (primary key) "ride_id", the type of transport used "rideable_type", the type of user (casual or member) "member_casual", date and time of start of the tour "started_at", date and time of the end of the tour "ended_at", the name of the start station "start_station_name", the identification key of the start station "start_station_id", the name of the end station "end_station_name", the identification key of the end station "end_station_id", geographic data (latitude and longitude) of the start and end stations "start_lat", "start_lng", "end_lat" and "end_lng".

Phase 3 - Process

For the data processing, we will use the Pandas library in Python (PyCharm). Python enables you to handle large volumes of data quickly and efficiently.

Importing the Pandas library:

```
import pandas as pd
```

With CSV files downloaded from the "Data" folder located in the same directory as the Python code development file, called "main.py", read the files and define their variables:

```
# read the CSV files and set the variables

df_2022_02 = pd.read_csv("Data/tripdata_2022_02.csv")
df_2022_03 = pd.read_csv("Data/tripdata_2022_03.csv")
df_2022_04 = pd.read_csv("Data/tripdata_2022_04.csv")
df_2022_05 = pd.read_csv("Data/tripdata_2022_05.csv")
df_2022_06 = pd.read_csv("Data/tripdata_2022_06.csv")
df_2022_07 = pd.read_csv("Data/tripdata_2022_07.csv")
df_2022_08 = pd.read_csv("Data/tripdata_2022_08.csv")
df_2022_09 = pd.read_csv("Data/tripdata_2022_09.csv")
df_2022_10 = pd.read_csv("Data/tripdata_2022_10.csv")
df_2022_11 = pd.read_csv("Data/tripdata_2022_11.csv")
df_2022_12 = pd.read_csv("Data/tripdata_2022_12.csv")
df_2023_01 = pd.read_csv("Data/tripdata_2023_01.csv")
```

Checking the structure and formatting of dataframes:

Starting with the “df_2022_02”

INPUT

```
# display information from the DataFrame

print(df_2022_02.info())
```

OUTPUT

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 115609 entries, 0 to 115608
Data columns (total 13 columns):
 #   Column              Non-Null Count   Dtype
---  ------              --------------   -----
 0   ride_id             115609 non-null  object
 1   rideable_type       115609 non-null  object
 2   started_at          115609 non-null  object
 3   ended_at            115609 non-null  object
 4   start_station_name  97029 non-null   object
 5   start_station_id    97029 non-null   object
 6   end_station_name    95254 non-null   object
 7   end_station_id      95254 non-null   object
 8   start_lat           115609 non-null  float64
 9   start_lng           115609 non-null  float64
 10  end_lat             115532 non-null  float64
 11  end_lng             115532 non-null  float64
 12  member_casual       115609 non-null  object
dtypes: float64(4), object(9)
memory usage: 11.5+ MB
```

Some inconsistencies are noted:

1 - The time attributes "started_at" and "ended_at" are in string format.

```
2   started_at          115609 non-null  object
3   ended_at            115609 non-null  object
```

2 - Null values are shown in the attributes "start_station_name", "start_station_id", "end_station_name", "end_station_id", "end_lat" and "end_lng".

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 115609 entries, 0 to 115608
Data columns (total 13 columns):
 #   Column              Non-Null Count   Dtype
---  ------              --------------   -----
 0   ride_id             115609 non-null  object
 1   rideable_type       115609 non-null  object
 2   started_at          115609 non-null  object
 3   ended_at            115609 non-null  object
 4   start_station_name  97029 non-null   object
 5   start_station_id    97029 non-null   object
 6   end_station_name    95254 non-null   object
 7   end_station_id      95254 non-null   object
 8   start_lat           115609 non-null  float64
 9   start_lng           115609 non-null  float64
 10  end_lat             115532 non-null  float64
 11  end_lng             115532 non-null  float64
 12  member_casual       115609 non-null  object
dtypes: float64(4), object(9)
memory usage: 11.5+ MB
```

Continuing the check process with the other DataFrames:

"df_2022_03"

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 284042 entries, 0 to 284041
Data columns (total 13 columns):
 #   Column              Non-Null Count   Dtype  
---  ------              --------------   -----  
 0   ride_id             284042 non-null  object 
 1   rideable_type       284042 non-null  object 
 2   started_at          284042 non-null  object 
 3   ended_at            284042 non-null  object 
 4   start_station_name  236796 non-null  object 
 5   start_station_id    236796 non-null  object 
 6   end_station_name    232885 non-null  object 
 7   end_station_id      232885 non-null  object 
 8   start_lat           284042 non-null  float64
 9   start_lng           284042 non-null  float64
 10  end_lat             283776 non-null  float64
 11  end_lng             283776 non-null  float64
 12  member_casual       284042 non-null  object 
dtypes: float64(4), object(9)
memory usage: 28.2+ MB
```

"df_2022_04"

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 371249 entries, 0 to 371248
Data columns (total 13 columns):
 #   Column              Non-Null Count   Dtype  
---  ------              --------------   -----  
 0   ride_id             371249 non-null  object 
 1   rideable_type       371249 non-null  object 
 2   started_at          371249 non-null  object 
 3   ended_at            371249 non-null  object 
 4   start_station_name  300362 non-null  object 
 5   start_station_id    300362 non-null  object 
 6   end_station_name    295961 non-null  object 
 7   end_station_id      295961 non-null  object 
 8   start_lat           371249 non-null  float64
 9   start_lng           371249 non-null  float64
 10  end_lat             370932 non-null  float64
 11  end_lng             370932 non-null  float64
 12  member_casual       371249 non-null  object 
dtypes: float64(4), object(9)
memory usage: 36.8+ MB
```

“df_2022_05”

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 634858 entries, 0 to 634857
Data columns (total 13 columns):
 #   Column              Non-Null Count   Dtype
---  ------              --------------   -----
 0   ride_id             634858 non-null  object
 1   rideable_type       634858 non-null  object
 2   started_at          634858 non-null  object
 3   ended_at            634858 non-null  object
 4   start_station_name  548154 non-null  object
 5   start_station_id    548154 non-null  object
 6   end_station_name    541687 non-null  object
 7   end_station_id      541687 non-null  object
 8   start_lat           634858 non-null  float64
 9   start_lng           634858 non-null  float64
 10  end_lat             634136 non-null  float64
 11  end_lng             634136 non-null  float64
 12  member_casual       634858 non-null  object
dtypes: float64(4), object(9)
memory usage: 63.0+ MB
```

“df_2022_06”

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 769204 entries, 0 to 769203
Data columns (total 13 columns):
 #   Column              Non-Null Count   Dtype
---  ------              --------------   -----
 0   ride_id             769204 non-null  object
 1   rideable_type       769204 non-null  object
 2   started_at          769204 non-null  object
 3   ended_at            769204 non-null  object
 4   start_station_name  676260 non-null  object
 5   start_station_id    676260 non-null  object
 6   end_station_name    669052 non-null  object
 7   end_station_id      669052 non-null  object
 8   start_lat           769204 non-null  float64
 9   start_lng           769204 non-null  float64
 10  end_lat             768149 non-null  float64
 11  end_lng             768149 non-null  float64
 12  member_casual       769204 non-null  object
dtypes: float64(4), object(9)
memory usage: 76.3+ MB
```

“df_2022_07”

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 823488 entries, 0 to 823487
Data columns (total 13 columns):
 #   Column              Non-Null Count   Dtype
---  ------              --------------   -----
 0   ride_id             823488 non-null  object
 1   rideable_type       823488 non-null  object
 2   started_at          823488 non-null  object
 3   ended_at            823488 non-null  object
 4   start_station_name  711457 non-null  object
 5   start_station_id    711457 non-null  object
 6   end_station_name    702537 non-null  object
 7   end_station_id      702537 non-null  object
 8   start_lat           823488 non-null  float64
 9   start_lng           823488 non-null  float64
 10  end_lat             822541 non-null  float64
 11  end_lng             822541 non-null  float64
 12  member_casual       823488 non-null  object
dtypes: float64(4), object(9)
memory usage: 81.7+ MB
```

“df_2022_08”

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 785932 entries, 0 to 785931
Data columns (total 13 columns):
 #   Column              Non-Null Count   Dtype
---  ------              --------------   -----
 0   ride_id             785932 non-null  object
 1   rideable_type       785932 non-null  object
 2   started_at          785932 non-null  object
 3   ended_at            785932 non-null  object
 4   start_station_name  673895 non-null  object
 5   start_station_id    673895 non-null  object
 6   end_station_name    665410 non-null  object
 7   end_station_id      665410 non-null  object
 8   start_lat           785932 non-null  float64
 9   start_lng           785932 non-null  float64
 10  end_lat             785089 non-null  float64
 11  end_lng             785089 non-null  float64
 12  member_casual       785932 non-null  object
dtypes: float64(4), object(9)
memory usage: 78.0+ MB
```

"df_2022_09"

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 701339 entries, 0 to 701338
Data columns (total 13 columns):
 #   Column              Non-Null Count   Dtype
---  ------              --------------   -----
 0   ride_id             701339 non-null  object
 1   rideable_type       701339 non-null  object
 2   started_at          701339 non-null  object
 3   ended_at            701339 non-null  object
 4   start_station_name  597559 non-null  object
 5   start_station_id    597559 non-null  object
 6   end_station_name    590154 non-null  object
 7   end_station_id      590154 non-null  object
 8   start_lat           701339 non-null  float64
 9   start_lng           701339 non-null  float64
 10  end_lat             700627 non-null  float64
 11  end_lng             700627 non-null  float64
 12  member_casual       701339 non-null  object
dtypes: float64(4), object(9)
memory usage: 69.6+ MB
```

"df_2022_10"

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 558685 entries, 0 to 558684
Data columns (total 13 columns):
 #   Column              Non-Null Count   Dtype
---  ------              --------------   -----
 0   ride_id             558685 non-null  object
 1   rideable_type       558685 non-null  object
 2   started_at          558685 non-null  object
 3   ended_at            558685 non-null  object
 4   start_station_name  467330 non-null  object
 5   start_station_id    467330 non-null  object
 6   end_station_name    462068 non-null  object
 7   end_station_id      462068 non-null  object
 8   start_lat           558685 non-null  float64
 9   start_lng           558685 non-null  float64
 10  end_lat             558210 non-null  float64
 11  end_lng             558210 non-null  float64
 12  member_casual       558685 non-null  object
dtypes: float64(4), object(9)
memory usage: 55.4+ MB
```

“df_2022_11”

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 337735 entries, 0 to 337734
Data columns (total 13 columns):
 #   Column              Non-Null Count   Dtype
---  ------              --------------   -----
 0   ride_id             337735 non-null  object
 1   rideable_type       337735 non-null  object
 2   started_at          337735 non-null  object
 3   ended_at            337735 non-null  object
 4   start_station_name  285778 non-null  object
 5   start_station_id    285778 non-null  object
 6   end_station_name    283476 non-null  object
 7   end_station_id      283476 non-null  object
 8   start_lat           337735 non-null  float64
 9   start_lng           337735 non-null  float64
 10  end_lat             337505 non-null  float64
 11  end_lng             337505 non-null  float64
 12  member_casual       337735 non-null  object
dtypes: float64(4), object(9)
memory usage: 33.5+ MB
```

“df_2022_12”

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 181806 entries, 0 to 181805
Data columns (total 13 columns):
 #   Column              Non-Null Count   Dtype
---  ------              --------------   -----
 0   ride_id             181806 non-null  object
 1   rideable_type       181806 non-null  object
 2   started_at          181806 non-null  object
 3   ended_at            181806 non-null  object
 4   start_station_name  152523 non-null  object
 5   start_station_id    152523 non-null  object
 6   end_station_name    150648 non-null  object
 7   end_station_id      150648 non-null  object
 8   start_lat           181806 non-null  float64
 9   start_lng           181806 non-null  float64
 10  end_lat             181678 non-null  float64
 11  end_lng             181678 non-null  float64
 12  member_casual       181806 non-null  object
dtypes: float64(4), object(9)
memory usage: 18.0+ MB
```

“df_2023_01”

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 190301 entries, 0 to 190300
Data columns (total 13 columns):
 #   Column              Non-Null Count   Dtype
---  ------              --------------   -----
 0   ride_id             190301 non-null  object
 1   rideable_type       190301 non-null  object
 2   started_at          190301 non-null  object
 3   ended_at            190301 non-null  object
 4   start_station_name  163580 non-null  object
 5   start_station_id    163580 non-null  object
 6   end_station_name    162461 non-null  object
 7   end_station_id      162461 non-null  object
 8   start_lat           190301 non-null  float64
 9   start_lng           190301 non-null  float64
 10  end_lat             190174 non-null  float64
 11  end_lng             190174 non-null  float64
 12  member_casual       190301 non-null  object
dtypes: float64(4), object(9)
memory usage: 18.9+ MB
```

All tables have the same structure and formatting standards. The inconsistencies found in the "df_2022_02" data frame are repeated in the others.

As the structure is the same, we can proceed consolidating the tables into a single DataFrame.

INPUT

```
# create a single DataFrame with all information grouped together

df_tripdata = pd.concat([df_2022_02, df_2022_03, df_2022_04,
df_2022_05, df_2022_06, df_2022_07, df_2022_08, df_2022_09,
df_2022_10, df_2022_11, df_2022_12, df_2023_01], axis=0)
```

Let's adjust the date/time attributes:

```
2   started_at          object
3   ended_at            object
```

INPUT

```
# modifies string format to datetime in "started_at" and "ended_at"
attributes

df_tripdata['started_at'] = pd.to_datetime(df_tripdata['started_at'],
format='%Y-%m-%d %H:%M:%S')

df_tripdata['ended_at'] = pd.to_datetime(df_tripdata['ended_at'],
format='%Y-%m-%d %H:%M:%S')

print(df_tripdata.info())
```

OUTPUT

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 5754248 entries, 0 to 190300
Data columns (total 13 columns):
 #   Column              Dtype
---  ------              -----
 0   ride_id             object
 1   rideable_type       object
 2   started_at          datetime64[ns]
 3   ended_at            datetime64[ns]
 4   start_station_name  object
 5   start_station_id    object
 6   end_station_name    object
 7   end_station_id      object
 8   start_lat           float64
 9   start_lng           float64
 10  end_lat             float64
 11  end_lng             float64
 12  member_casual       object
dtypes: datetime64[ns](2), float64(4), object(7)
memory usage: 614.6+ MB
```

It seems all right.

Because it has not yet been defined whether attributes that have null values will be relevant to all analyses or not, we will deal with them later if necessary.

For now, let's add the attribute by calculating the duration of each trip.

INPUT

```
# add a column for duration

df_tripdata['duration'] = df_tripdata['ended_at'] -
df_tripdata['started_at']
```

We'll also add a column that brings the day of the week corresponding to the trip date column.

INPUT

```
# add a column for the day of the week

df_tripdata['day_of_week'] = df_tripdata['started_at'].dt.day_name()
```

Finally, taking advantage of the geographic coordinates of the collection and delivery points, we will create a column with the result of the distance between those points.

Let's use the "geopy" library

```
from geopy import distance
```

We will define a function that seeks the latitude and longitude information of the source and end points and ignores if there are null values. Returning the distance between the points in kilometers.

INPUT

```
# defines a function to calculate the distance between two geographic points

def calc_distance(row):
    if pd.isnull(row[['start_lat', 'start_lng', 'end_lat',
                      'end_lng']]).any():
        return float('nan')
    lat1, long1, lat2, long2 = row[['start_lat', 'start_lng',
                                    'end_lat', 'end_lng']]
    coords_1 = (lat1, long1)
    coords_2 = (lat2, long2)
    return distance.distance(coords_1, coords_2).km
```

And enter the code to perform the calculations in the DataBase.

INPUT

```
# adds a column with the calculation of the distance between the geographic coordinates of the collection and delivery points

df_tripdata['distance_km'] = df_tripdata.apply(calc_distance, axis=1)
```

Finally, let's reorder the columns for a more intuitive view.

INPUT

```
# reorder the columns for better visualization of the data

df_tripdata = df_tripdata[['ride_id', 'rideable_type',
'member_casual', 'started_at', 'ended_at', 'duration',
'day_of_week','start_station_name', 'start_station_id',
'end_station_name', 'end_station_id', 'start_lat', 'start_lng',
'end_lat', 'end_lng', 'distance_km']]
```

Because of the size of the DataBase, the processing time of the distance calculation has increased significantly. As we have not yet defined the relevance of this information, let’s leave aside for now.

Latest checks:

Search for duplicate information by the "ride_id" key.

INPUT

```
# check for duplicate records

duplicates = df_tripdata['ride_id'].duplicated()
if duplicates.any():
    print('There are duplicate values in the ride_id column')
else:
    print('There are no duplicate values in the ride_id column')
```

OUTPUT

```
There are no duplicate values in the ride_id column
```

There are no duplicate records.

Let's then check for anomalies in the "duration" attribute.

INPUT

```
# look for anomalies in the duration of the rides

print(df_tripdata['duration'].describe())
```

OUTPUT

```
count                        5754248
mean      0 days 00:19:18.334898148
std       0 days 02:55:19.871287872
min               -8 days +19:26:39
25%                 0 days 00:05:46
50%                 0 days 00:10:12
75%                 0 days 00:18:20
max                28 days 17:47:15
Name: duration, dtype: object
```

Negative time values are anomalies. Let's eliminate these records.

The column values are in Timedelta format:

```
5   duration            timedelta64[ns]
```

Counting negative or zero values to determine whether the absence of such information may impact future analysis results:

INPUT

```
# count negative or zero values in the "duration" column

count_neg_duration = (df_tripdata['duration'] <=
pd.Timedelta(0)).sum()

print("There are", count_neg_duration, "negative or equal to zero
values in the duration column.")
```

OUTPUT

```
There are 534 negative or equal to zero values in the duration column.
```

A total of 534 records were found with negative Timedelta. As the DataFrame has 5,754,248 records, the representativeness is approximately 0.000093%. Let's proceed with the deletion of the records.

INPUT

```
# eliminate rows with negative or zero values in the "duration" column

df_tripdata = df_tripdata.loc[df_tripdata['duration'] >=
pd.Timedelta(0)]

print(df_tripdata['duration'].describe())
```

OUTPUT

```
count                        5753714
mean      0 days 00:19:18.579606146
std       0 days 02:55:17.143808701
min                  0 days 00:00:01
25%                  0 days 00:05:46
50%                  0 days 00:10:12
75%                  0 days 00:18:20
max                 28 days 17:47:15
Name: duration, dtype: object
```

Then we need to understand and treat the values with duration above normal.

Let's look at the amount of trips equal to or greater than a day:

INPUT

```
# count values greater than or equal to 1 day in the "duration" column

count_duration_greater_than_1day = (df_tripdata['duration'] >=
pd.Timedelta(days=1)).sum()

print("There are", count_duration_greater_than_1day, "values greater
than or equal to 1 day in the duration column.")
```

OUTPUT

```
There are 5390 values greater than 1 day in the duration column.
```

The representativeness of these values is approximately 0.00094%. So let's proceed with the elimination of these records for further future analysis.

INPUT

```
# delete values greater than or equal to 1 day in the "duration"
column

df_tripdata = df_tripdata.loc[df_tripdata['duration'] <
pd.Timedelta(days=1)]

print(df_tripdata['duration'].describe())
```

OUTPUT

```
count                      5748324
mean      0 days 00:16:09.175244819
std       0 days 00:29:10.949300718
min                 0 days 00:00:01
25%                 0 days 00:05:46
50%                 0 days 00:10:12
75%                 0 days 00:18:18
max                 0 days 23:59:56
Name: duration, dtype: object
```

Finally, let's search for null values in all columns and evaluate their representations:

INPUT

```
# count the null values in each column of the DataFrame

for attribute in df_tripdata:
    nulls = df_tripdata[attribute].isnull().sum()
    print(f"The column {attribute} has {nulls} null values.")
```

OUTPUT

```
The column ride_id has 0 null values.
The column rideable_type has 0 null values.
The column member_casual has 0 null values.
The column started_at has 0 null values.
The column ended_at has 0 null values.
The column duration has 0 null values.
The column day_of_week has 0 null values.
The column start_station_name has 843499 null values.
The column start_station_id has 843499 null values.
The column end_station_name has 897204 null values.
The column end_station_id has 897204 null values.
The column start_lat has 0 null values.
The column start_lng has 0 null values.
The column end_lat has 703 null values.
The column end_lng has 703 null values.
```

Only attributes with start and end stations IDs have significant null values. Deleting these records would represent approximately 15.62% of the total partially cleaned DataFrame.

We will choose to do the first analyses with the full DataFrame and clean up the null values for the identifications of the stations at the time this information is for analysis.

With everything organized, we are ready to start the analysis.

Phase 4 - Analyze

Let's divide the analyses into five steps starting with a more global view of proportion by user type and then deepening the correlation of each attribute.

For data visualization, let's use the "matplotlib" library:

INPUT

```
import matplotlib.pyplot as plt
```

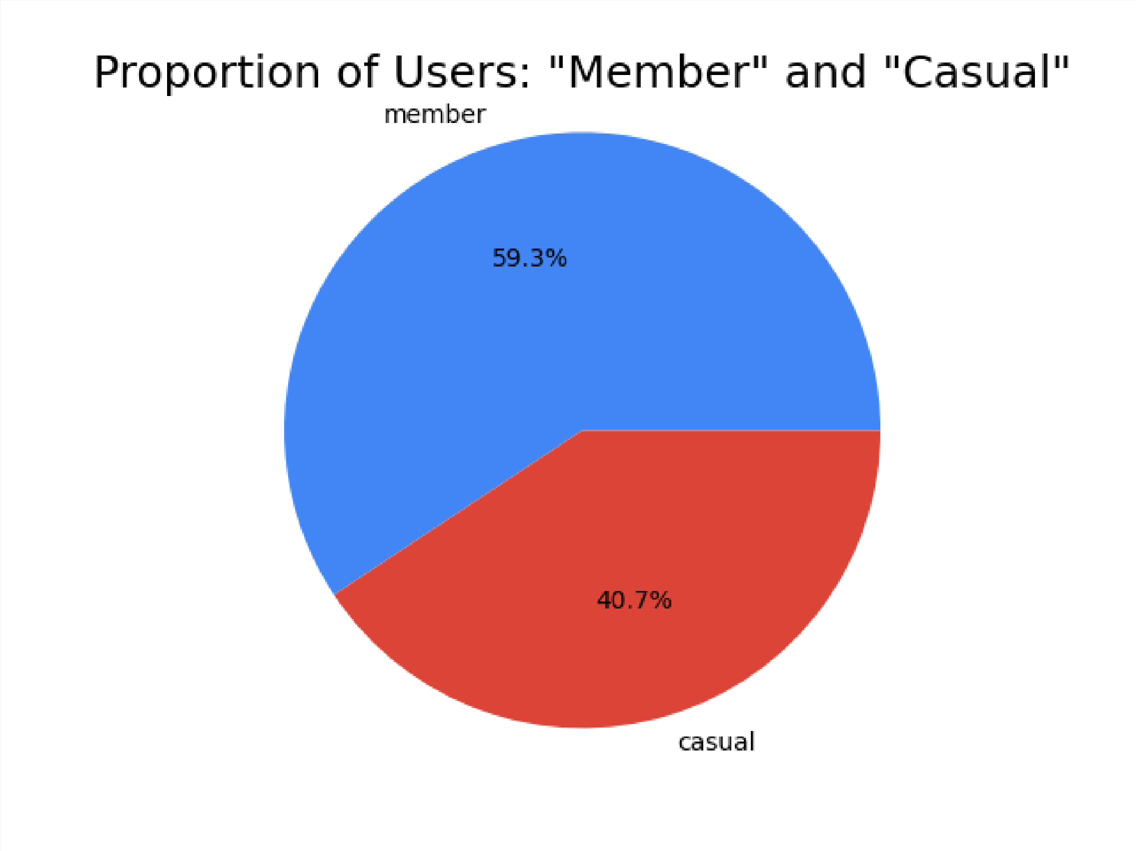
FIRST STEP

1) The ratio of casual users to members in the year gives an overview of the amount of potential new members. (pie chart)

INPUT

```
''' Proportion of Users'''  # pie chart

# count values in each category of the 'member_casual' column
count_users = df_tripdata['member_casual'].value_counts()

# create the pie chart
colors = ['#4285F4', '#DB4437']
count_users.plot(kind='pie', autopct='%1.1f%%', colors=colors)

# set title and display chart
plt.title('Proportion of Users: "Member" and "Casual"',
fontdict={'fontname': 'Roboto', 'fontsize': 18})
plt.axis('equal')
plt.axis('off')
plt.show()
```

OUTPUT



59.3% of users are already members. 40.7% of users remain for a possible conversion. We need to better understand the behavior of these users to learn how we can revert them into members.

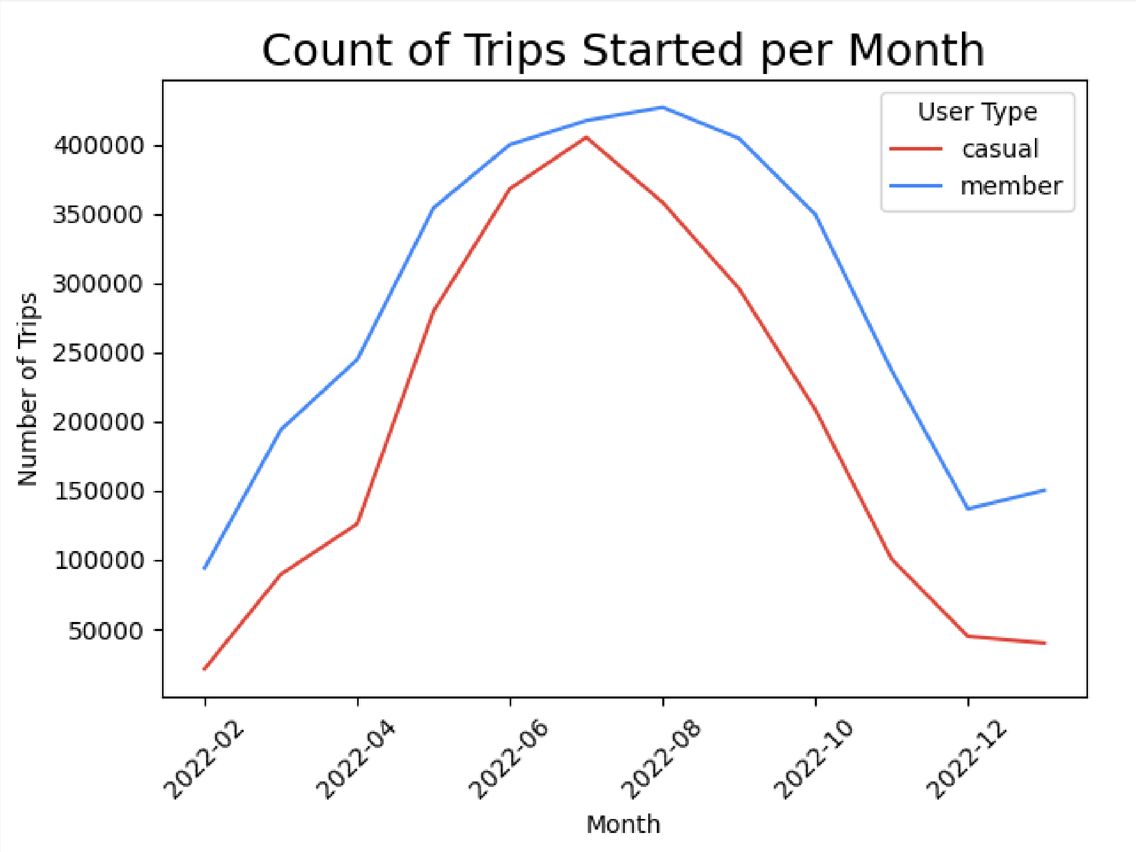
SECOND

2) The variations in use among the year (seasonality) and the correlations between casual users and members can bring relevant information regarding the behavior of users in relation to the different times of the year. (line chart)

INPUT

```
'''Count of trips started per month'''  # line chart

# group by month and count occurrences for each user category
count_tsm =
df_tripdata.groupby([df_tripdata['started_at'].dt.strftime('%Y-%m'),
'member_casual'])['ride_id'].count()

# create the line chart
colors = ['#DB4437', '#4285F4']
count_tsm.unstack().plot(kind='line', color=colors)

# configure axis title and labels
plt.title('Count of Trips Started per Month', fontdict={'fontname':
'Roboto', 'fontsize': 18})
plt.xlabel('Month')
plt.ylabel('Number of Trips')
plt.legend(title='User Type', loc='upper right')
plt.xticks(rotation=45)
plt.show()
```

OUTPUT



There is a big difference in the number of users during the year. In the summer months both casual users and members use the service much more. The number of members remains higher throughout the year, however, in July there was a significant spike in the number of casual users almost reaching the same number of members. This can point to a profile of users who are on vacation and/or tourists.

A marketing action to convert casual users into members should be more effective in the summer, especially in June and July.

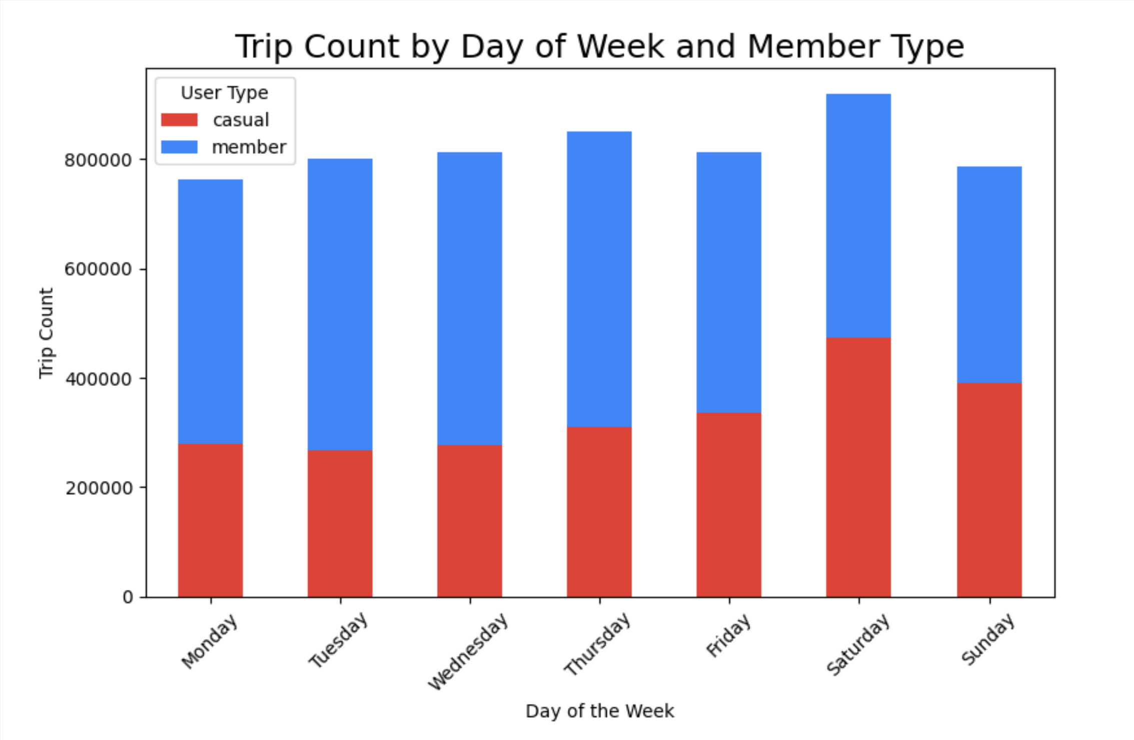
THIRD

3) Usage variations between weekdays and correlations between casual users and members aim to identify user patterns and profiles between weekdays and weekends. (stacked column chart)

INPUT

```
'''Trip Count by Day of Week and Member Type'''  # stacked column
chart

# converting column "day_of_week" to category type
df_tripdata['day_of_week'] =
pd.Categorical(df_tripdata['day_of_week'], categories=['Monday',
'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday', 'Sunday'],
ordered=True)

# group data by 'day_of_week' and 'member_casual' and count
occurrences
df_grouped = df_tripdata.groupby(['day_of_week',
'member_casual']).size().unstack()

# create the stacked column chart
colors = ['#DB4437', '#4285F4']
stacked = df_grouped.plot(kind='bar', stacked=True, color=colors)

# cConfigure axis title and labels
stacked.set_title('Trip Count by Day of Week and Member Type',
fontdict={'fontname': 'Roboto', 'fontsize': 18})
stacked.set_xlabel('Day of the Week')
stacked.set_ylabel('Trip Count')
plt.legend(title='User Type', loc='upper left')
plt.xticks(rotation=45)
plt.show()
```

OUTPUT



A quantidade de viagens totais por dia da semana é bastante estável. Mas quando consideramos os usuários casuais, nota-se um aumento significativo de uso aos Sábados.

Weekends seem like the best days to reach casual users.

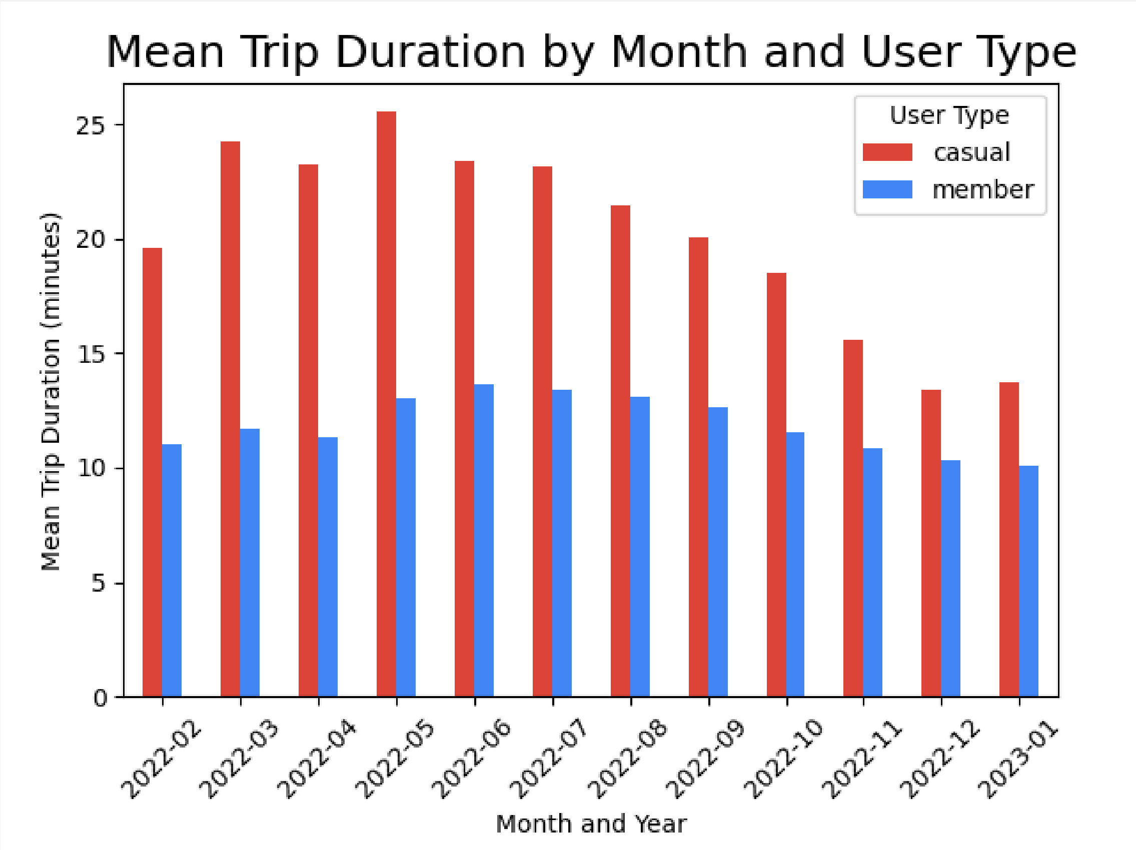
FOURTH

4) The overall average duration of trips per month and by user type should show whether there are differences and correlations between usage patterns. (chart of grouped columns)

INPUT

```
'''Mean Trip Duration by Month and User Type'''  # clustered bar chart

# calculates the average of trip times in minutes, separated by month,
year and type of user
mean_tsm =
df_tripdata.groupby([df_tripdata['started_at'].dt.strftime('%Y-%m'),
'member_casual'])['duration'].mean().dt.total_seconds() / 60

# converts the timedelta type data to float so that it can be inserted
into the chart
mean_tsm = mean_tsm.astype(float)

# creates the clustered bar chart
colors = ['#DB4437', '#4285F4']
mean_dur_user = mean_tsm.unstack().plot(kind='bar', rot=45,
color=colors)

# defines the axis title and labels
mean_dur_user.set_xlabel('Month and Year')
mean_dur_user.set_ylabel('Mean Trip Duration (minutes)')
mean_dur_user.set_title('Mean Trip Duration by Month and User Type',
fontdict={'fontname': 'Roboto', 'fontsize': 18})
plt.legend(title='User Type', loc='upper right')
plt.show()
```

OUTPUT



Casual users clearly tend to ride longer. Members uses the bikes for shorter periods.

This can point to a tendency for casual users to use the service to ride around and enjoy the day, while member users may tend to use more as a transportation way (such as going from home to work).

But that's just un assumption to be confirmed through surveys. For now, we will proceed with the analysis of the data we have.

FIFTH

5) Identifying the stations with the largest number of casual users and stations with the highest proportion of casual users can be a good tool for converting members.

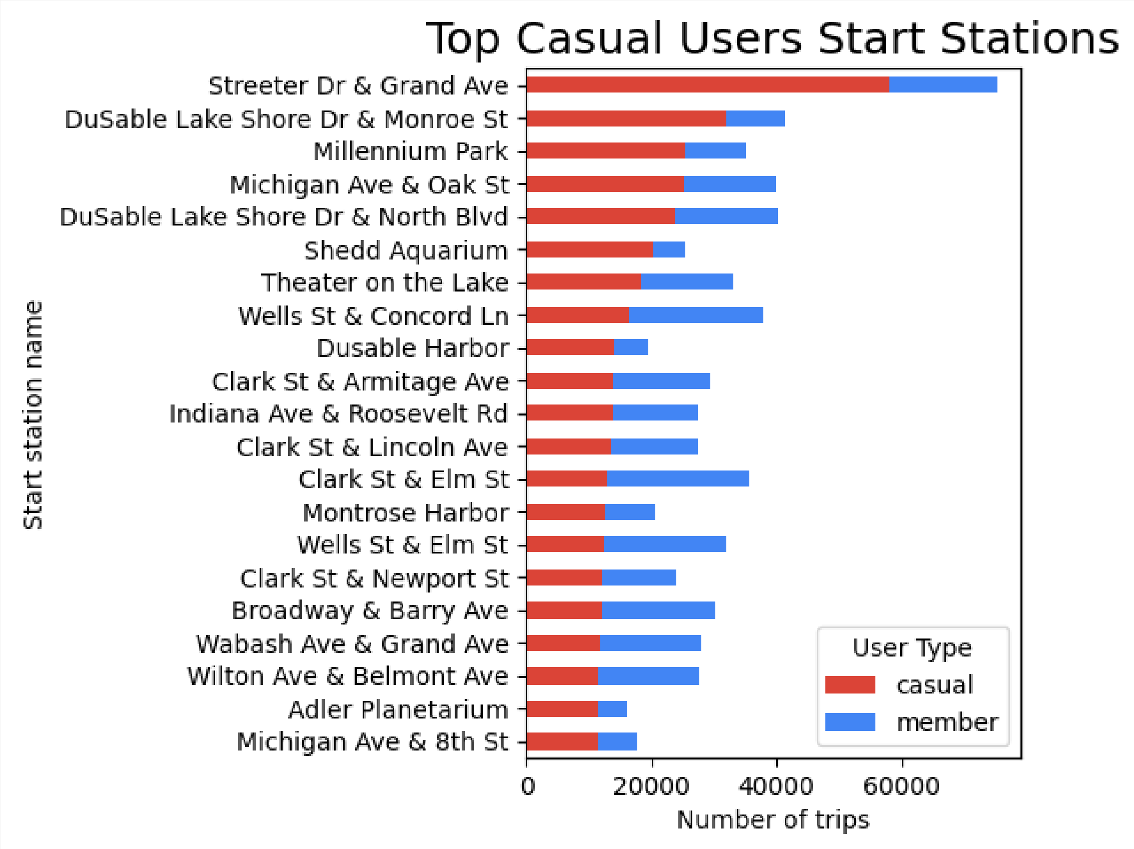
INPUT

```
'''Top Stations by Casual Users'''

# eliminate rows with null values in column 'member_casual'
df_clean_member_casual = df_tripdata.dropna(subset=['member_casual'])
```

INPUT

```
'''Top Start Start Stations'''

# group by station and user type and count the number of occurrences
df_grouped_start_station =
df_clean_member_casual.groupby(['start_station_name',
'member_casual']).size().unstack()

# select only the top 21 casual type counts
df_grouped_start_station =
df_grouped_start_station.sort_values(by='casual',
ascending=False).head(21)
df_grouped_start_station =
df_grouped_start_station.sort_values(by='casual', ascending=True)

# create clustered bar chart
colors = ['#DB4437', '#4285F4']
df_grouped_start_station.plot(kind='barh', stacked=True, color=colors)

# configure the chart
plt.title('Top Casual Users Start Stations', fontdict={'fontname':
'Roboto', 'fontsize': 18})
plt.xlabel('Number of trips')
plt.ylabel('Start station name')
plt.legend(title='User Type', loc='lower right')
plt.show()
```

OUTPUT

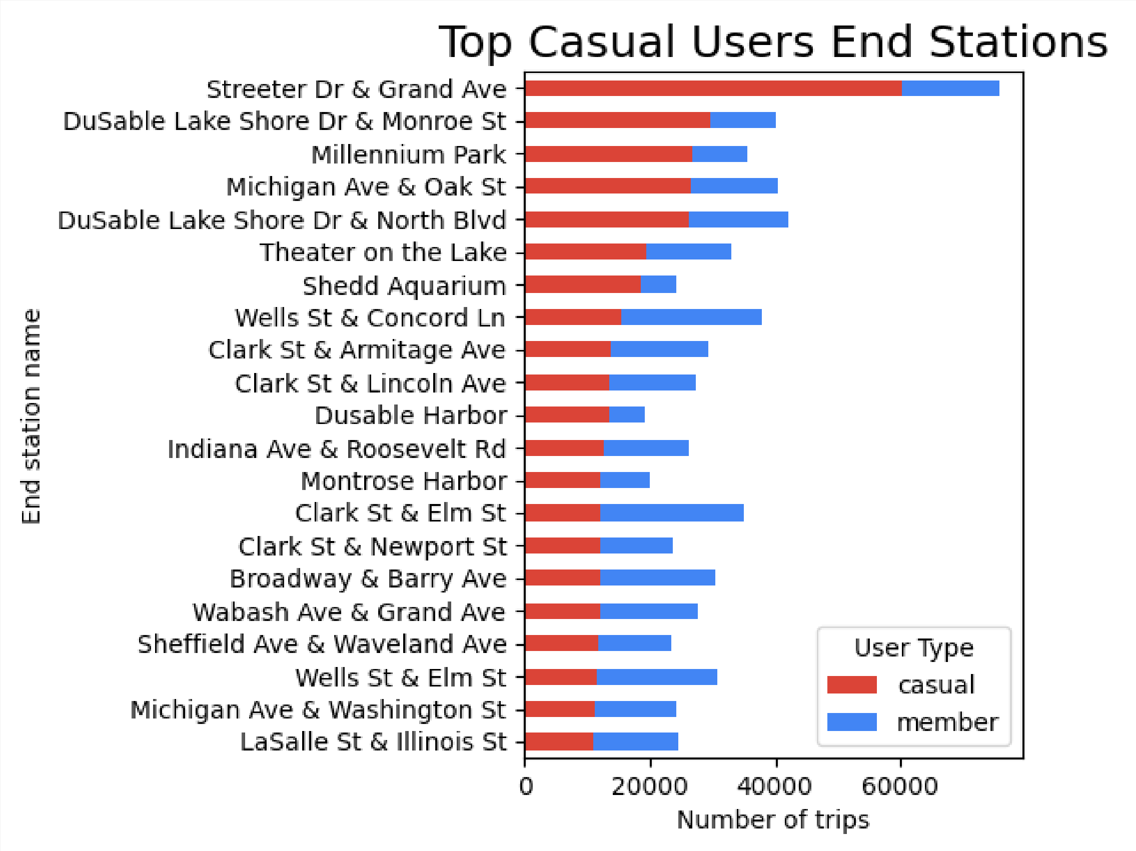


The "Streeter Dr & Grand Ave" station is notably the most widely used starting point for casual users. In addition to the quantity, it is interesting to note the proportions between user types. There are stations that are used in smaller total quantity, but that have a very large proportion of casual users. All these stations should be taken into consideration for a better understanding of the target users and marketing actions.

INPUT

```
'''Top End Stations'''

# group by station and user type and count the number of occurrences
df_grouped_end_station =
df_clean_member_casual.groupby(['end_station_name',
'member_casual']).size().unstack()

# select only the top 21 casual type counts
df_grouped_end_station =
df_grouped_end_station.sort_values(by='casual',
ascending=False).head(21)
df_grouped_end_station =
df_grouped_end_station.sort_values(by='casual', ascending=True)

# create clustered bar chart
colors = ['#DB4437', '#4285F4']
df_grouped_end_station.plot(kind='barh', stacked=True, color=colors)

# configure the chart
plt.title('Top Casual Users End Stations', fontdict={'fontname':
'Roboto', 'fontsize': 18})
plt.xlabel('Number of trips')
plt.ylabel('End station name')
plt.legend(title='User Type', loc='lower right')
plt.show()
```

OUTPUT



With some differences, the volume of use of stations as endpoint is quite similar to the collection data.

Let's create a DataFrame that shows the total usage value of the stations, separated by member and casual users, and add the geographic data so that we can view the information on a map.

INPUT

```
''' Top Stations DataFrame with Geolocation'''  # df_top_stations
DataFrame

# sums the start and end values for the 21 stations most used by
casual users
df_top_stations =
df_grouped_start_station.head(21).add(df_grouped_end_station.head(21),
fill_value=0, axis=0)

# ranks the stations with the highest number of casual users
df_top_stations = df_top_stations.sort_values('casual',
ascending=False)

# adds the geolocation data for each station
df_top_stations = df_top_stations.reset_index()
df_top_stations.rename(columns={'index': 'start_station_name'},
inplace=True)

df_geo_info = df_tripdata[['start_station_name', 'start_lat',
'start_lng']]

df_top_stations =
df_top_stations.merge(df_geo_info.groupby('start_station_name')
                                        .first(),
on='start_station_name', how='left')

df_top_stations.rename(columns={'start_station_name': 'station_name',
'start_lat': 'lat', 'start_lng': 'lng'},
                       inplace=True)

print(df_top_stations)
```

OUTPUT

```
                          station_name    casual    member        lat
lng
      Streeter Dr & Grand Ave  118214.0  32790.0  41.892278 -87.612043
DuSable Lake Shore Dr & Monroe   61559.0  19945.0  41.880958 -87.616743
             Millennium Park   52423.0  18155.0  41.881050 -87.624100
       Michigan Ave & Oak St   51815.0  28307.0  41.900905 -87.623783
DuSable Lake Shore Dr & North  49839.0  32551.0  41.911722 -87.626804
              Shedd Aquarium   39208.0  10629.0  41.867226 -87.615355
         Theater on the Lake   37888.0  28217.0  41.926277 -87.630834
       Wells St & Concord Ln   31969.0  43824.0  41.912003 -87.634631
     Clark St & Armitage Ave   27747.0  30872.0  41.918348 -87.636283
              Dusable Harbor   27529.0  11164.0  41.886976 -87.612813
      Clark St & Lincoln Ave   27063.0  27678.0  41.915848 -87.634655
   Indiana Ave & Roosevelt Rd   26474.0  27273.0  41.867888 -87.623041
           Clark St & Elm St   25315.0  45331.0  41.902973 -87.631280
             Montrose Harbor   24932.0  15632.0  41.963982 -87.638181
       Clark St & Newport St   24343.0  23584.0  41.944519 -87.654811
        Broadway & Barry Ave   24271.0  36393.0  41.937582 -87.644098
           Wells St & Elm St   24247.0  38521.0  41.903222 -87.634324
      Wabash Ave & Grand Ave   23853.0  31701.0  41.891437 -87.627001
 Sheffield Ave & Waveland Ave   11766.0  11712.0  41.949196 -87.654481
     Wilton Ave & Belmont Ave   11675.0  16026.0  41.940180 -87.653040
           Adler Planetarium   11500.0   4643.0  41.866095 -87.607267
       Michigan Ave & 8th St   11420.0   6402.0  41.872597 -87.624089
 Michigan Ave & Washington St   11356.0  12979.0  41.883984 -87.624684
     LaSalle St & Illinois St   11105.0  13355.0  41.890893 -87.631486
```

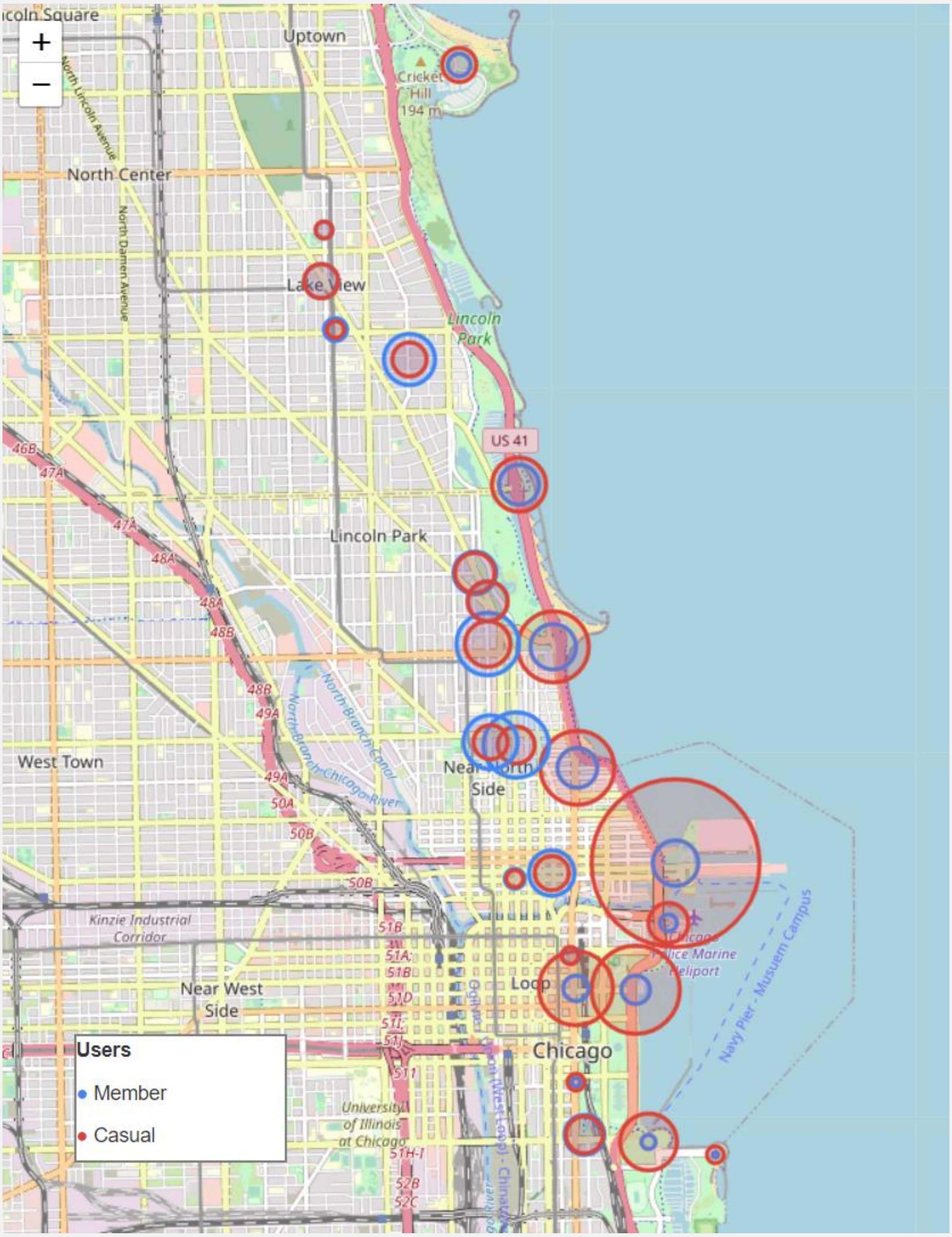
Now we can use this DataFrame on a map to get a better understanding of the types of users per station throughout the city.

Let's create the map:

INPUT

```
'''Top Stations Interactive Map'''  # interactive map
# create an interactive map centered on Chicago
map_chicago = folium.Map(location=[41.9000, -87.6298], zoom_start=13)

# sets the scale factor and add circles for each station
scale_factor = 0.0005
for index, row in df_top_stations.iterrows():
    folium.CircleMarker(location=[row['lat'], row['lng']],
radius=row['member']*scale_factor, color='#4285F4',
                        fill=True, fill_color='#4285F4',
popup=row['station_name']).add_to(map_chicago)
    folium.CircleMarker(location=[row['lat'], row['lng']],
radius=row['casual']*scale_factor, color='#DB4437',
                        fill=True, fill_color='#DB4437',
popup=row['station_name']).add_to(map_chicago)

# add title and legend to the map
title_html = '''
             <h3 align="center" style="font-size:20px"><b>Top Stations
Rides</b></h3>
             '''
map_chicago.get_root().html.add_child(folium.Element(title_html))
legend_html = '''
                <div style="position: fixed;
                            bottom: 50px; left: 50px; width: 150px;
height: 90px;
                            border:2px solid grey; z-index:9999; font-
size:14px;
                            background-color:white;
                            ">
                            <p><b>    Users</b></p>
                            <p><span
style='color:#4285F4;'>&#9679;</span>  Member</p>
                            <p><span
style='color:#DB4437;'>&#9679;</span>  Casual</p>
                </div>
                '''
map_chicago.get_root().html.add_child(folium.Element(legend_html))

# adds markers with station names
for index, row in df_top_stations.iterrows():
    folium.Marker(location=[row['lat'], row['lng']],
popup=row['station_name'],
                  tooltip=row['station_name']).add_to(map_chicago)

# adds controls layer
folium.LayerControl().add_to(map_chicago)

# saves the map in html format
map_chicago.save('Data/map_chicago.html')
```

OUTPUT



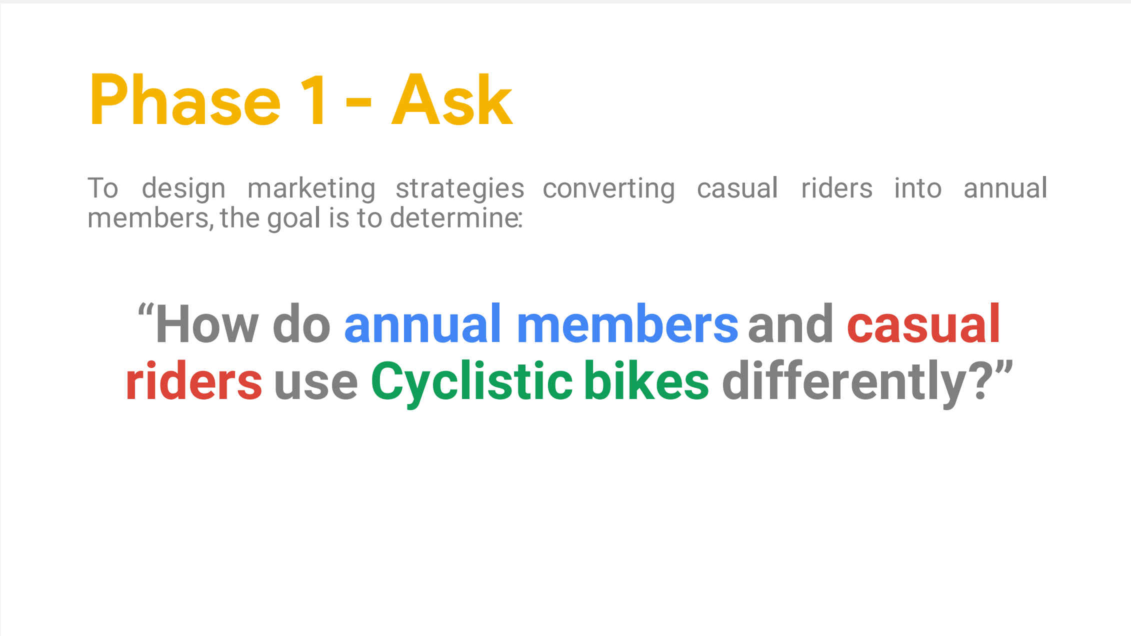
The red circles represent casual users and blue circles represent the members. Circle sizes are proportional to the travel quantity.

This can help to visualize the location of the most used stations by casual users for a better understanding and conduct of surveys and marketing actions.

Now that we've identified usage patterns, days of the year, week days, and stations with the most casual users, we know when and where to focus marketing actions. It is important to use this information to apply surveys to better understand the profile of casual users and to know the reasons why they do not become members.

Phase 5 - Share

For the presentation, let's insert the charts into Power Point and point out the most important information and analysis.



Phase 2 - Prepare

Using Cyclistic's historical trip data to analyze and identify trends:

202202-divvy-tripdata.zip	Mar 2nd 2022, 05:22:47 pm	4.30 MB	ZIP file
202203-divvy-tripdata.zip	Apr 6th 2022, 02:07:41 pm	10.39 MB	ZIP file
202204-divvy-tripdata.zip	May 3rd 2022, 01:33:19 pm	13.36 MB	ZIP file
202205-divvy-tripdata.zip	Jun 3rd 2022, 11:08:02 pm	22.68 MB	ZIP file
202206-divvy-tripdata.zip	Jul 15th 2022, 12:27:59 pm	26.90 MB	ZIP file
202207-divvy-tripdata.zip	Aug 5th 2022, 07:27:33 pm	29.51 MB	ZIP file
202208-divvy-tripdata.zip	Sep 8th 2022, 07:20:19 pm	27.13 MB	ZIP file
202209-divvy-tripdata.zip	Oct 11th 2022, 12:59:39 pm	25.31 MB	ZIP file
202210-divvy-tripdata.zip	Nov 8th 2022, 07:47:10 pm	20.08 MB	ZIP file
202211-divvy-tripdata.zip	Dec 5th 2022, 03:17:32 pm	12.36 MB	ZIP file
202212-divvy-tripdata.zip	Jan 3rd 2023, 05:19:01 pm	6.75 MB	ZIP file
202301-divvy-tripdata.zip	Feb 7th 2023, 04:58:38 pm	6.78 MB	ZIP file

Phase 3 - Process

- Checked the data for errors using PyCharm.

- Transformed the data into a unique DataFrame to work with it effectively.

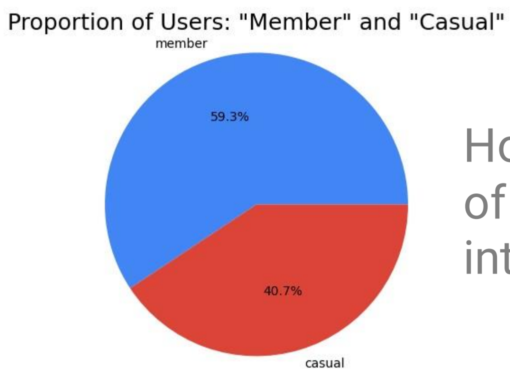
- Documented the cleaning process.

Phase 4 - Analyze

5 main analyzes were carried out:

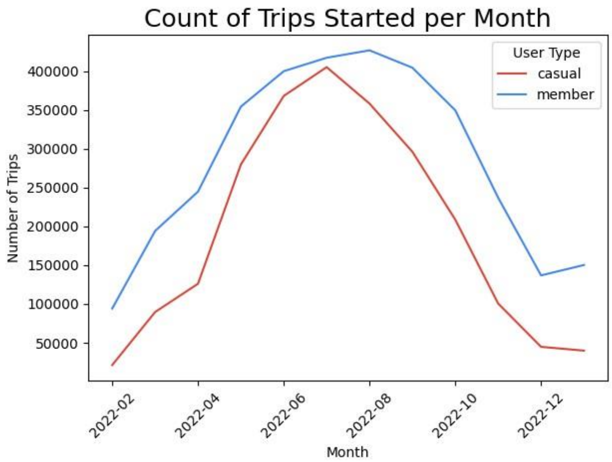
1) The ratio of casual users to members in the year
2) The variations in use among the year
3) The usage variations between weekdays
4) The overall average duration of trips per month
5) Identifying the stations with the largest number of casual users

Phase 5 - Share



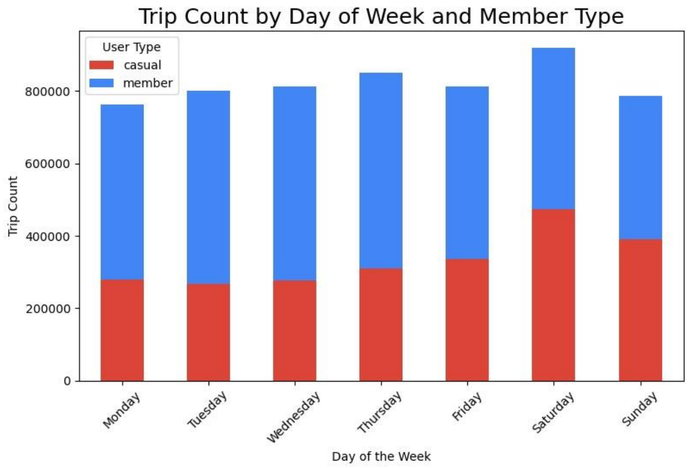
How can we revert some of the 40,7% casual users into members?

Phase 5 - Share



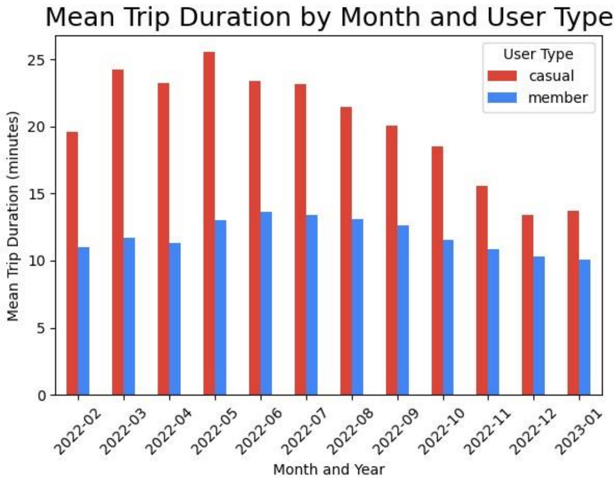
A marketing action to convert casual users into members should be more effective in the summer, especially in June and July.

Phase 5 - Share



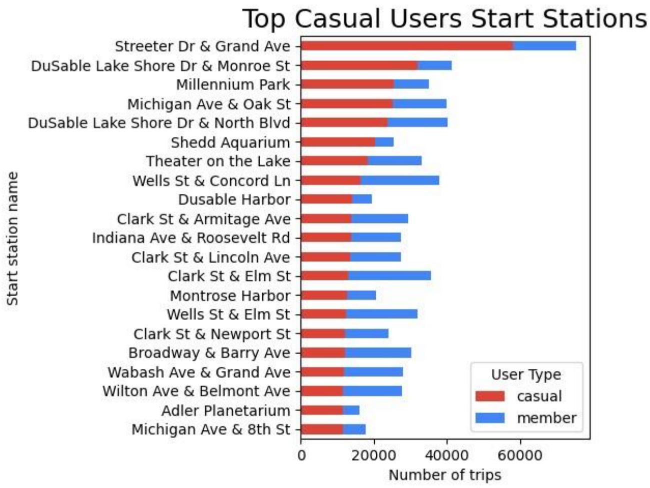
Weekends seem like the best days to reach casual users.

Phase 5 - Share



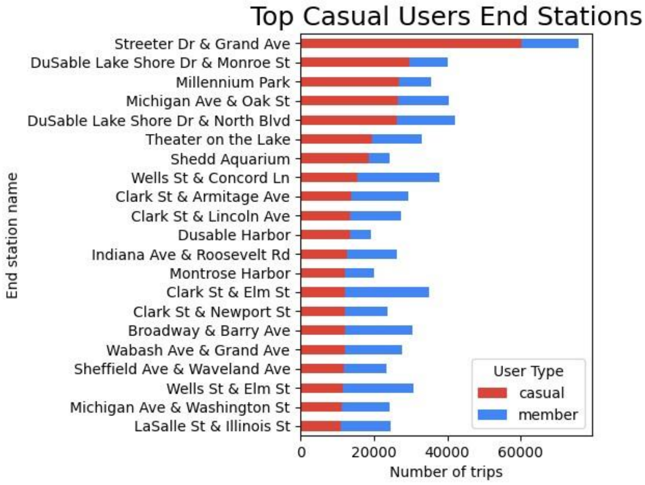
Casual users clearly tend to ride longer. Members uses the bikes for shorter periods.

Phase 5 - Share



In addition to the quantity, it is interesting to note the proportions between user types.

Phase 5 - Share



Stations with a big quantity and proportion of casual users may be the best ones to focus the marketing strategies.

Phase 5 - Share

Stations with a big quantity and proportion of casual users may be the best ones to focus the marketing strategies.

Phase 6 - Act

- 40,7% are casual users.
- Usage increases a lot in the summer.
- Weekends seem like the best days to reach casual users.
- Casual users clearly tend to ride longer.
- Marketing strategies should focus in the top casual user stations.

Phase 6 - Act

This study helped us better understand "How do annual members and casual riders use Cyclistic bikes differently".

Now we know when and where to focus marketing actions.

It is suggested to carry out a survey at the main stations to understand the casual user's profile and the reasons for not becoming a member.

Google
Data Analytics Professional Certificate Capstone

Google Data Analytics Capstone – Ask, Prepare, Process, Analyze, Share and Act

Author : Felipe Seleme Ribeiro
E-mail: felipeselemeribeiro@gmail.com

Date: February 2023

Phase 6 – Act

- 40,7% are casual users.
- Usage increases a lot in the summer.
- Weekends seem like the best days to reach casual users.
- Casual users clearly tend to ride longer.
- Marketing strategies should focus in the top casual user stations.

This study helped us better understand "How do annual members and casual riders use Cyclistic bikes differently".

Now we know when and where to focus marketing actions.

Actions should prioritize the summer period, especially on weekends, in stations with a higher concentration of casual users.

It is suggested to carry out a survey at the main stations to understand the casual user's profile and the reasons for not becoming a member.

Google

Data Analytics Capstone – Ask, Prepare, Process, Analyze, Share and Act

Author: Felipe Seleme Ribeiro

E-mail: felipeselemeribeiro@gmail.com

Date: February 2023