Google Data Analytics Capstone Project

This capstone project is the final project in my Google Data Analytics Professional Certificate course. In this case study, I will be analyzing a public dataset for a fictional company provided by the course. I will be using Python for data analysis and PowerBi for visualizations.

About the company

In 2016, Cyclistic launched a successful bike-share offering having a fleet of 5,824 bicycles that are tracked 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 at any time.

The company's fleet has grown over the years and has reached 1380 station by august 2022.

Riders who have an annual subscription are called members while riders who are single-ride or full-day pass users are considered casual riders.

The director of marketing is looking to maximize the number of annual memberships as they are more profitable than single-ride or full-day passes. This strategy is believed to be the key to future growth.

The following data analysis steps will be followed: Ask, Prepare, Process, Analyze, Share, Act.

Ask

The questions that need to be answered are:

1. How do annual members and casual riders use Cyclistic bikes differently?
2. How to convert casual riders into members?

Prepare

The dataset follows the ROCCC Analysis as described below:

Reliable - yes, not biased

Original - yes, can locate the original public data

Comprehensive - yes, not missing important information

Current - yes, updated monthly

Cited - yes

I will be using the public dataset located here (https://divvy-tripdata.s3.amazonaws.com/index.html) The data has been made available by Motivate International Inc. under this license (https://ride.divvybikes.com/data-license-agreement).

Key Tasks Followed:

Downloaded data and copies have been stored on the computer.

I have downloaded the data for AUG 21- JUL 22 Period.

The data is in CSV (comma-separated values) format, and there are a total of 13 columns. **Installing and loading necessary packages**

Entrée [1]:

```
import pandas as pd
!pip install pandasql
from pandasql import sqldf
from datetime import datetime
import numpy as np

%matplotlib inline
import matplotlib as mpl
import matplotlib.pyplot as plt
```

```
Requirement already satisfied: pandasql in c:\users\acer\anaconda3\lib\site-
packages (0.7.3)
Requirement already satisfied: sqlalchemy in c:\users\acer\anaconda3\lib\sit
e-packages (from pandasql) (1.4.32)
Requirement already satisfied: pandas in c:\users\acer\anaconda3\lib\site-pa
ckages (from pandasql) (1.4.2)
Requirement already satisfied: numpy in c:\users\acer\anaconda3\lib\site-pac
kages (from pandasql) (1.21.5)
Requirement already satisfied: pytz>=2020.1 in c:\users\acer\anaconda3\lib\s
ite-packages (from pandas->pandasql) (2021.3)
Requirement already satisfied: python-dateutil>=2.8.1 in c:\users\acer\anaco
nda3\lib\site-packages (from pandas->pandasql) (2.8.2)
Requirement already satisfied: six>=1.5 in c:\users\acer\anaconda3\lib\site-
packages (from python-dateutil>=2.8.1->pandas->pandasql) (1.16.0)
Requirement already satisfied: greenlet!=0.4.17 in c:\users\acer\anaconda3\l
ib\site-packages (from sqlalchemy->pandasql) (1.1.1)
```

Importing data to Pandas DataFrame

Entrée [2]:

```
file1=pd.read_csv('202108-divvy-tripdata.csv')
file2=pd.read_csv('202109-divvy-tripdata.csv')
file3=pd.read_csv('202110-divvy-tripdata.csv')
file4=pd.read_csv('202111-divvy-tripdata.csv')
file5=pd.read_csv('202112-divvy-tripdata.csv')
file6=pd.read_csv('202201-divvy-tripdata.csv')
file7=pd.read_csv('202202-divvy-tripdata.csv')
file8=pd.read_csv('202203-divvy-tripdata.csv')
file9=pd.read_csv('202204-divvy-tripdata.csv')
file10=pd.read_csv('202205-divvy-tripdata.csv')
file11=pd.read_csv('202206-divvy-tripdata.csv')
file12=pd.read_csv('202207-divvy-tripdata.csv')
```

Merging data into a data frame

Entrée [3]:

```
data=pd.concat([file6, file5,file4,file3,file1,file2,file7,file8,file9,file10,file11,file12
```

Entrée [4]:

```
del file6, file5,file4,file3,file1,file2,file7,file8,file9,file10,file11,file12
```

Process

Cleaning and Preparing Data

Checking the data

Entrée [5]:

```
data.shape
```

Out[5]:

```
(5901463, 13)
```

Entrée [6]:

```
data.dtypes
```

Out[6]:

```
ride_id                object
rideable_type          object
started_at             object
ended_at               object
start_station_name     object
start_station_id       object
end_station_name       object
end_station_id         object
start_lat             float64
start_lng             float64
end_lat               float64
end_lng               float64
member_casual          object
dtype: object
```

Entrée [7]:

```
data.head()
```

Out[7]:

	ride_id	rideable_type	started_at	ended_at	start_station_name	start_station_ic
0	C2F7DD78E82EC875	electric_bike	2022-01-13 11:59:47	2022-01-13 12:02:44	Glenwood Ave & Touhy Ave	525
1	A6CF8980A652D272	electric_bike	2022-01-10 08:41:56	2022-01-10 08:46:17	Glenwood Ave & Touhy Ave	525
2	BD0F91DFF741C66D	classic_bike	2022-01-25 04:53:40	2022-01-25 04:58:01	Sheffield Ave & Fullerton Ave	TA1306000016
3	CBB80ED419105406	classic_bike	2022-01-04 00:18:04	2022-01-04 00:33:00	Clark St & Bryn Mawr Ave	KA1504000151
4	DDC963BFDDA51EEA	classic_bike	2022-01-20 01:31:10	2022-01-20 01:37:12	Michigan Ave & Jackson Blvd	TA1309000002

Entrée [8]:

```
data.isnull().sum()
```

Out[8]:

```
ride_id                    0
rideable_type              0
started_at                 0
ended_at                   0
start_station_name    860786
start_station_id      860784
end_station_name      919896
end_station_id        919896
start_lat                  0
start_lng                  0
end_lat                 5590
end_lng                 5590
member_casual              0
dtype: int64
```

Almost 1/6 of the end_station_name and start_station_name data is missing we cannot just drop it

Make stations tables

We set their coords as primary key to filter stations names

Entrée [9]:

```
stat_data=data
```

Entrée [10]:

```
stat_data=stat_data[stat_data['start_station_name'].notnull()]
```

Entrée [ ]:

```
stat_data.drop_duplicates(subset='start_station_name',inplace=True)
```

Entrée [12]:

```
len(stat_data)
```

Out[12]:

```
1381
```

Entrée [13]:

```
stat_data.head()
```

Out[13]:

	ride_id	rideable_type	started_at	ended_at	start_station_name	start_station_ic
0	C2F7DD78E82EC875	electric_bike	2022-01-13 11:59:47	2022-01-13 12:02:44	Glenwood Ave & Touhy Ave	525
2	BD0F91DFF741C66D	classic_bike	2022-01-25 04:53:40	2022-01-25 04:58:01	Sheffield Ave & Fullerton Ave	TA1306000016
3	CBB80ED419105406	classic_bike	2022-01-04 00:18:04	2022-01-04 00:33:00	Clark St & Bryn Mawr Ave	KA1504000151
4	DDC963BFDDA51EEA	classic_bike	2022-01-20 01:31:10	2022-01-20 01:37:12	Michigan Ave & Jackson Blvd	TA1309000002
5	A39C6F6CC0586C0B	classic_bike	2022-01-11 18:48:09	2022-01-11 18:51:31	Wood St & Chicago Ave	637

Entrée [ ]:

```
e','started_at','ended_at','end_station_name','end_station_id','member_casual','end_lat','en
```

Entrée [15]:

```
stat_data.reset_index( drop=True, inplace=True)
```

Entrée [ ]:

```
stat_data.rename(columns = {'start_lat':'station_lat','start_lng':'station_lng','start_stat
```

Create station tables with 4 decimals

A value in decimal degrees to 5 decimal places is accurate to 1.11m which can not be the case.

Therefore we choose 4 (number of stations with 4 decimals in coors is the same number of stations with non-rounded coors)

Entrée [17]:

```
station_coor=[]
for i in range(0,len(stat_data)):
    a=(round(stat_data['station_lat'].values[i],4),round(stat_data['station_lng'].values[i]
    station_coor.append(a)
```

Entrée [18]:

```
stat_data.insert(4,'station_coor',station_coor)
```

Entrée [19]:

```
del station_coor
```

Entrée [20]:

```
stat_data.head()
```

Out[20]:

	station_name	station_id	station_lat	station_lng	station_coor
0	Glenwood Ave & Touhy Ave	525	42.012800	-87.665906	(42.0128, -87.6659)
1	Sheffield Ave & Fullerton Ave	TA1306000016	41.925602	-87.653708	(41.9256, -87.6537)
2	Clark St & Bryn Mawr Ave	KA1504000151	41.983593	-87.669154	(41.9836, -87.6692)
3	Michigan Ave & Jackson Blvd	TA1309000002	41.877850	-87.624080	(41.8778, -87.6241)
4	Wood St & Chicago Ave	637	41.895634	-87.672069	(41.8956, -87.6721)

Entrée [21]:

```
len(stat_data['station_coor'].unique())
```

Out[21]:

```
996
```

we have 997 station coor(even if not rounded to 4) while we also have 1382 station whicch means that due to gps

related data entry imprecision we will have to assume that the stations with the same coor are one

Entrée [ ]:

```
stat_data.drop_duplicates(subset='station_coor',inplace=True)
```

Entrée [23]:

```
len(stat_data['station_id'].unique())
```

Out[23]:

```
965
```

we have 998 station and 966 station id which means there exists different stations with same id which means that

station_id is not a primary key thus irrelevant

NB: we only use this table to replace missing values and not to change any value that already exists

Entrée [ ]:

```
stat_data.drop(['station_id'], axis=1,inplace=True)
```

Entrée [25]:

```
data.drop(['start_station_id','end_station_id'], axis=1,inplace=True)
```

Entrée [26]:

```
stat_data.to_csv('station_data4.csv')
```

Create station data with rounded coor all the way to 1 decimal(to match imprecision coor)

Our goal is to get an approximative location where riders start and end their rides in order to do so we round the coordinates to get the closest station to the location the gps detects

Entrée [ ]:

```
for j in range(1,4):
    station_coor=[]
    for i in range(0,len(stat_data)):
        a=(round(stat_data['station_lat'].values[i],j),round(stat_data['station_lng'].value
        station_coor.append(a)
    stat_data.drop(['station_coor'], axis=1,inplace=True)
    stat_data.insert(4,'station_coor',station_coor)
    stat_data=stat_data.drop_duplicates('station_coor',ignore_index=True)
    stat_data.to_csv("station_data"+str(j)+".csv")
```

Entrée [ ]:

```
del station_coor
```

Entrée [ ]:

```

```

Replace missing values of station data in table

We have 5590 rides with no end coor and these same rows have no station name neither id which

means there is no way of knowing the end station

Entrée [27]:

```
data=data[data['end_lat'].notnull()]
```

Entrée [ ]:

```
data.isnull().sum()
```

Out[15]:

```
ride_id                    0
rideable_type              0
started_at                 0
ended_at                   0
start_station_name    860786
start_station_id      860784
end_station_name      914306
end_station_id        914306
start_lat                  0
start_lng                  0
end_lat                    0
end_lng                    0
member_casual              0
dtype: int64
```

I tried to replace it by using loops but that just takes too much time(5m rows and 1000 stations results to 5b iteration)

In order to work with sqldf we have to convert all the dataframe columns into str

Entrée [ ]:

```
data=data.applymap(str)
```

we create a subset of missing data

Entrée [ ]:

```
missing_stat=sqldf("select * from data where start_station_name='nan' or end_station_name='
```

Entrée [ ]:

```
missing_stat.head()
```

Out[22]:

	ride_id	rideable_type	started_at	ended_at	week_day	ride_length	start_statio
0	99103BB87CC6C1BB	electric_bike	2021-08-10 17:15:49	2021-08-10 17:22:44	1	0 days 00:06:55	
1	EAFCCCFB0A3FC5A1	electric_bike	2021-08-10 17:23:14	2021-08-10 17:39:24	1	0 days 00:16:10	
2	9EF4F46C57AD234D	electric_bike	2021-08-21 02:34:23	2021-08-21 02:50:36	5	0 days 00:16:13	
3	5834D3208BFAF1DA	electric_bike	2021-08-21 06:52:55	2021-08-21 07:08:13	5	0 days 00:15:18	
4	CD825CB87ED1D096	electric_bike	2021-08-19 11:55:29	2021-08-19 12:04:11	3	0 days 00:08:42	

We subtitute all the missing data from our dataframe

Entrée [ ]:

```
data=sqldf("select * from data where start_station_name!='nan' and end_station_name!='nan'
```

how many rows we should replace?

Entrée [ ]:

```
long=len(missing_stat)
```

create dataframe for clean data

Entrée [ ]:

```
columns = missing_stat.columns
```

Entrée [ ]:

```
columns
```

Out[18]:

```
Index(['ride_id', 'rideable_type', 'started_at', 'ended_at',
       'start_station_name', 'end_station_name', 'start_lat', 'start_lng',
       'end_lat', 'end_lng', 'member_casual'],
      dtype='object')
```

Entrée [ ]:

```
notmissing_stat=pd.DataFrame(columns = columns)
```

Replace missing values in 'start_station_name'

Entrée [ ]:

```
notmissing_stat=pd.concat([notmissing_stat,sqldf("select * from missing_stat where start_st
```

Entrée [ ]:

```
rounded=4
while len(notmissing_stat)!=long:
    if rounded==0:
        missing_stat.drop(['start_station_coor'], axis=1,inplace=True)
        break

    start_station_coor=[]

    for i in range(0,len(missing_stat)):
        a=(round(float(missing_stat['start_lat'].values[i]),rounded),round(float(missing_st
        start_station_coor.append(a)

    if rounded!=4:
        missing_stat.drop(['start_station_coor'], axis=1,inplace=True)


    missing_stat.insert(11,'start_station_coor',start_station_coor)
    missing_stat=missing_stat.applymap(str)

    stat_data=pd.read_csv("station_data"+str(rounded)+".csv")


    a=sqldf("select * from (select * from missing_stat where start_station_name='nan')as mi
    a['start_station_name']=a['station_name']

    a.drop(['station_name','station_lat','station_lng','station_coor','Unnamed: 0'], axis=1


    a.drop(['start_station_coor'], axis=1,inplace=True)


    notmissing_stat=pd.concat([notmissing_stat,a], ignore_index=True)

    missing_stat=pd.concat([missing_stat,a], ignore_index=True)

    missing_stat.drop_duplicates('ride_id',keep=False,inplace=True,ignore_index=True)

    del a
    rounded-=1
```

Entrée [ ]:

```
notmissing_stat
```

Out[27]:

	ride_id	rideable_type	started_at	ended_at	start_station_name	end_statio
0	C1E408B2F6190D9D	electric_bike	2021-08-06 09:30:09	2021-08-06 09:35:39	Aberdeen St & Jackson Blvd	
1	76A83D4BC639464D	electric_bike	2021-08-14 11:10:25	2021-08-14 11:39:33	Kingsbury St & Kinzie St	
2	81C3530FF8F4EC68	electric_bike	2021-08-05 16:34:48	2021-08-05 17:01:47	Michigan Ave & Oak St	
3	A75B5B09D5F7522F	electric_bike	2021-08-18 16:32:03	2021-08-18 16:40:52	Kingsbury St & Kinzie St	
4	F9DFE91D7C378E28	electric_bike	2021-08-26 04:09:36	2021-08-26 04:42:33	Larrabee St & Armitage Ave	
...	...	...	...	...	...	
416404	1C28A39170F1BFCF	electric_bike	2021-10-22 15:48:03	2021-10-22 15:55:30	Meade Ave & Diversey Ave	
416405	E23D9291AAE08CD0	electric_bike	2021-10-15 12:41:55	2021-10-15 13:11:02	Meade Ave & Addison St	Campb Mont
416406	6463C8C4BCD3FE51	electric_bike	2021-10-01 17:58:46	2021-10-01 18:16:57	Desplaines St & Kinzie St	Michigan A
416407	B32EBE63A8E44A23	electric_bike	2021-10-15 12:41:54	2021-10-15 13:11:05	Meade Ave & Addison St	Campb Mont
416408	E51924A2CE9A1A2A	electric_bike	2021-10-09 19:16:49	2021-10-09 19:33:39	Clark St & Leland Ave	Campb Mont

416409 rows × 11 columns

Entrée [ ]:

```
missing_stat=notmissing_stat
```

Entrée [ ]:

```
notmissing_stat=0
notmissing_stat=pd.DataFrame(columns = columns)
```

Entrée [ ]:

```
notmissing_stat=pd.concat([notmissing_stat,sqldf("select * from missing_stat where end_stat
```

Entrée [ ]:

```
missing_stat.drop(['end_station_coor'], axis=1,inplace=True)
```

Entrée [ ]:

```
rounded=4
while len(notmissing_stat)!=long:
    if rounded==0:
        missing_stat.drop(['end_station_coor'], axis=1,inplace=True)
        break


    end_station_coor=[]
    for i in range(0,len(missing_stat)):

        b=(round(float(missing_stat['end_lat'].values[i]),rounded),round(float(missing_stat
        end_station_coor.append(b)
    if rounded!=4:
        missing_stat.drop(['end_station_coor'], axis=1,inplace=True)


    missing_stat.insert(11,'end_station_coor',end_station_coor)
    missing_stat=missing_stat.applymap(str)

    stat_data=pd.read_csv("station_data"+str(rounded)+".csv")


    a=sqldf("select * from (select * from missing_stat where end_station_name='nan')as miss
    a['end_station_name']=a['station_name']



    a.drop(['station_name','station_lat','station_lng','station_coor','Unnamed: 0'], axis=1



    a.drop(['end_station_coor'], axis=1,inplace=True)
    notmissing_stat=pd.concat([notmissing_stat,a], ignore_index=True)

    missing_stat=pd.concat([missing_stat,a], ignore_index=True)

    missing_stat.drop_duplicates('ride_id',keep=False,inplace=True,ignore_index=True)

    del a
    rounded-=1
```

Entrée [ ]:

```
data=pd.concat([data,notmissing_stat], ignore_index=True)
```

Entrée [ ]:

```
del notmissing_stat,missing_stat
```

Entrée [ ]:

```
data.isnull().sum()
```

Out[5]:

```
Unnamed: 0            0
ride_id               0
rideable_type         0
started_at            0
ended_at              0
week_day              0
ride_length           0
start_station_name    0
end_station_name      0
start_lat             0
start_lng             0
end_lat               0
end_lng               0
member_casual         0
dtype: int64
```

Entrée [ ]:

```
len(data)
```

Out[6]:

```
5672423
```

Create ride length is seconds

Entrée [ ]:

```
ride_length=[]
for i in range(0,len(data)):
    start_time=data['started_at'].values[i].split()
    end_time=data['ended_at'].values[i].split()
    if datetime.strptime(end_time[1], '%H:%M:%S')>datetime.strptime(start_time[1], '%H:%M:%
        a=datetime.strptime(end_time[1], '%H:%M:%S')-datetime.strptime(start_time[1], '%H:%
        if str(a)[2]!=':':
            ride_length.append(int(str(a)[0])*3600+int(str(a)[2]+str(a)[3])*60+int(str(a)[5
        else:
            ride_length.append(int(str(a)[0]+str(a)[1])*3600+int(str(a)[3]+str(a)[4])*60+in

    else:
        ride_length.append(0)
```

Entrée [ ]:

```
data.insert(4,'ride_length',ride_length)
```

Entrée [ ]:

```
del ride_length
```

Entrée [33]:

```
data['ride_length'].dtype
```

Out[33]:

```
dtype('int64')
```

Checking consistency of Ride length

Some ride length duration are negative, i.e start time is actually greater than the end time.

Entrée [ ]:

```
data=data[data['ride_length'] !=0]
```

Create week day

Entrée [ ]:

```
week_day=[]
for i in range(0,len(data)):
    date=data['started_at'].values[i].split()

    date[0]=date[0].replace('-','/')
    a=datetime.strptime(date[0], '%Y/%m/%d').weekday()
    if a==0 :
        a='Monday'
    elif a==1 :
        a='Tuesday'
    elif a==2 :
        a='wednesday'
    elif a==3 :
        a='Thursday'
    elif a==4 :
        a='Friday'
    elif a==5 :
        a='Saturday'
    elif a==6 :
        a='Sunday'
    week_day.append(a)
```

Entrée [ ]:

```
data.insert(4,'week_day',week_day)
```

Entrée [ ]:

```
del week_day
```

Entrée [37]:

```
data.head()
```

Out[37]:

	ride_id	rideable_type	started_at	ended_at	week_day	ride_length	start_statio
0	4CA9676997DAFFF6	classic_bike	2021-11-26 10:27:28	2021-11-26 11:22:13	Friday	3285	Michigan Av
1	F3E84A230AF2D676	classic_bike	2021-11-15 09:35:03	2021-11-15 09:42:08	Monday	425	Clark St & (
2	A1F2C92308007968	electric_bike	2021-11-10 16:27:02	2021-11-10 17:04:28	wednesday	2246	Leamingtc H
3	9B871C3B14E9BEC4	classic_bike	2021-11-09 19:51:36	2021-11-09 20:11:17	Tuesday	1181	Desplai I
4	2A81E957DD24A3DC	classic_bike	2021-11-06 19:14:10	2021-11-06 19:33:19	Saturday	1149	Larral Armit

Entrée [38]:

```
data.dtypes
```

Out[38]:

```
ride_id                object
rideable_type          object
started_at             object
ended_at               object
week_day               object
ride_length             int64
start_station_name     object
end_station_name       object
start_lat             float64
start_lng             float64
end_lat               float64
end_lng               float64
member_casual          object
dtype: object
```

Analyse

Key tasks

1. Aggregate your data so it’s useful and accessible.
2. Organize and format your data.
3. Perform calculations.
4. Identify trends and relationships.

Ride Analysis

Entrée [ ]:

```
avg=np.average(data['ride_length'])
print('average ride bike length: '+str(int(avg//3600))+' hours '+str(int((avg%3600)//60))+'
```

```
average ride bike length: 0 hours 16 mins 53 seconds
```

Entrée [ ]:

```
maxi=max(data['ride_length'])
print('max ride bike length: '+str(int(maxi//3600))+' hours '+str(int((maxi%3600)//60))+' m
```

```
max ride bike length: 23 hours 44 mins 42 seconds
```

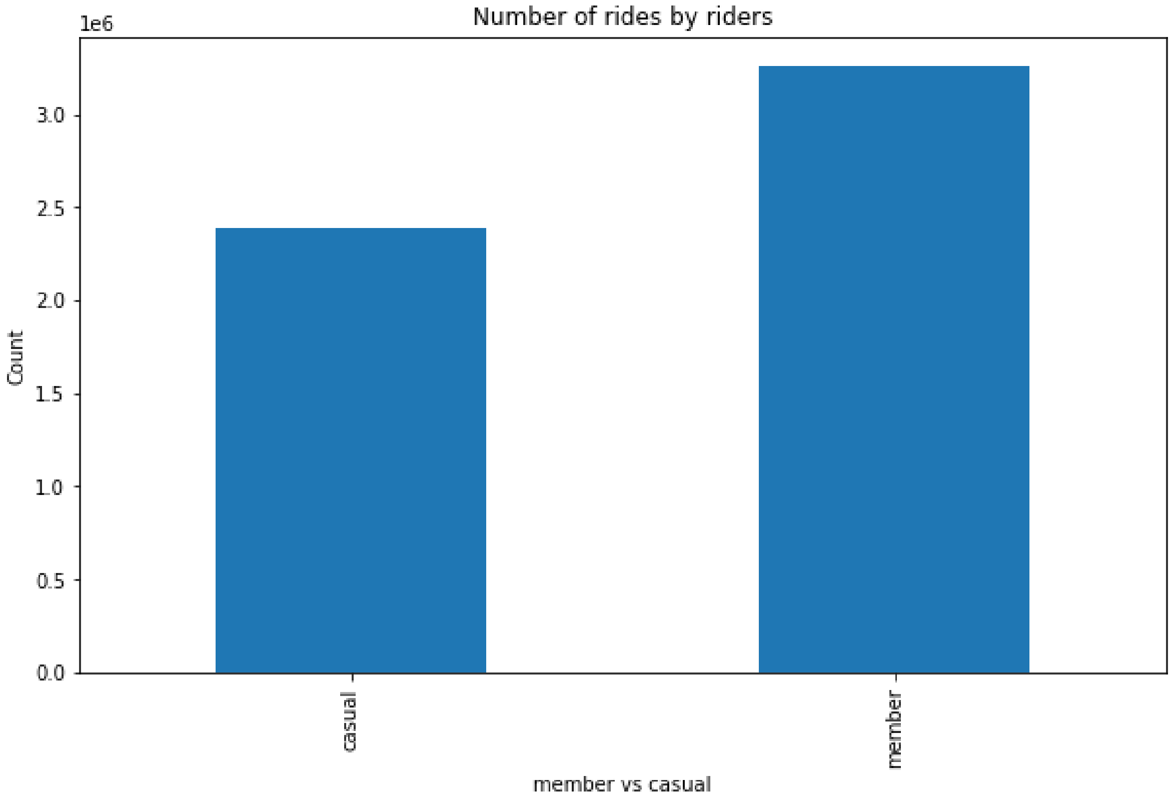
Compare Members and Casual riders

Entrée [ ]:

```
data.groupby(['member_casual'])['ride_id'].count().plot(kind='bar', figsize=(10, 6))

plt.xlabel('member vs casual') # add to x-label to the plot
plt.ylabel('Count') # add y-label to the plot
plt.title('Number of rides by riders') # add title to the plot

plt.show()
```



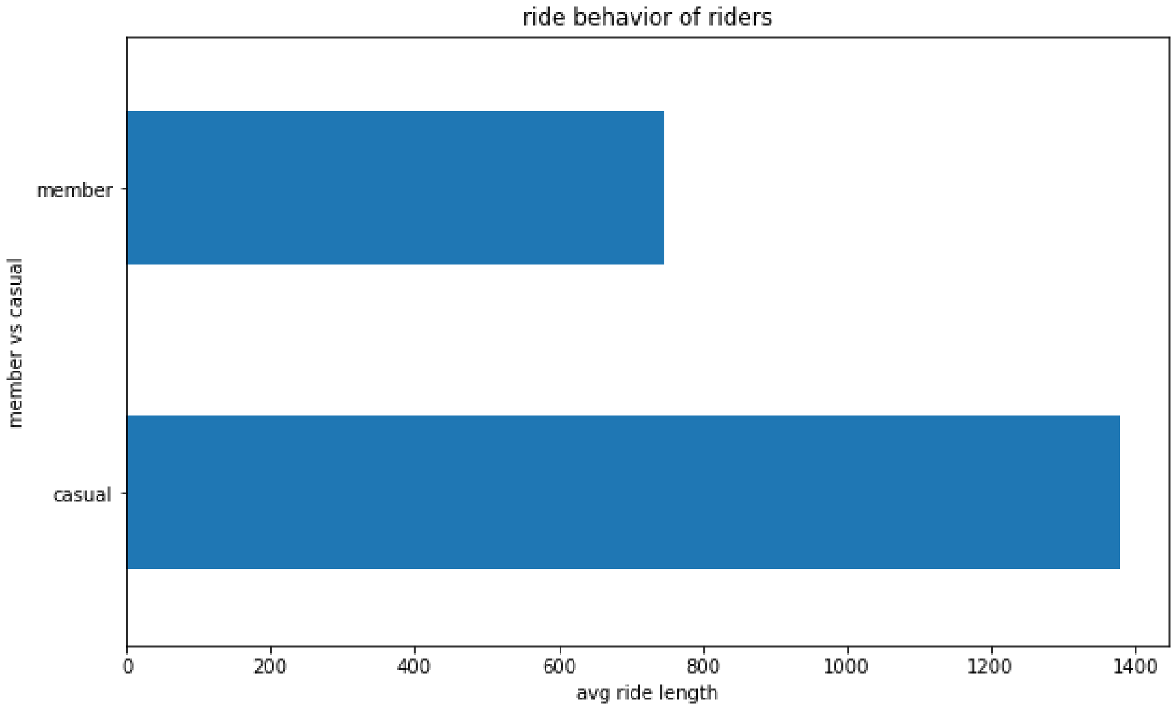
Ride behavior of members and casual riders

Entrée [ ]:

```
data.groupby(['member_casual'])['ride_length'].mean().plot(kind='barh', figsize=(10, 6))

plt.xlabel('avg ride length') # add to x-label to the plot
plt.ylabel('member vs casual') # add y-label to the plot
plt.title('ride behavior of riders') # add title to the plot

plt.show()
```



Casual riders take longer rides than members.

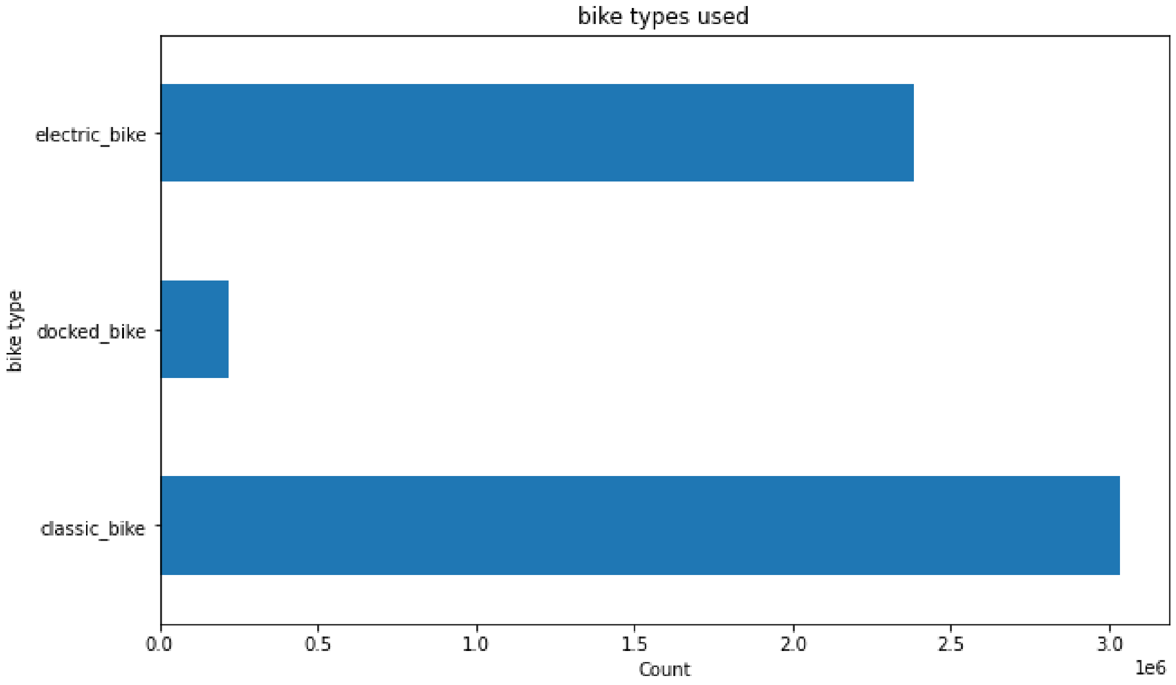
Comparing Bikes

Entrée [ ]:

```
data.groupby(['rideable_type'])['ride_id'].count().plot(kind='barh', figsize=(10, 6))

plt.xlabel('Count') # add to x-label to the plot
plt.ylabel('bike type') # add y-label to the plot
plt.title('bike types used') # add title to the plot

plt.show()
```



Classic Bike is more preferred than Electric Bike.

Docked Bike is the least preferred Bike.

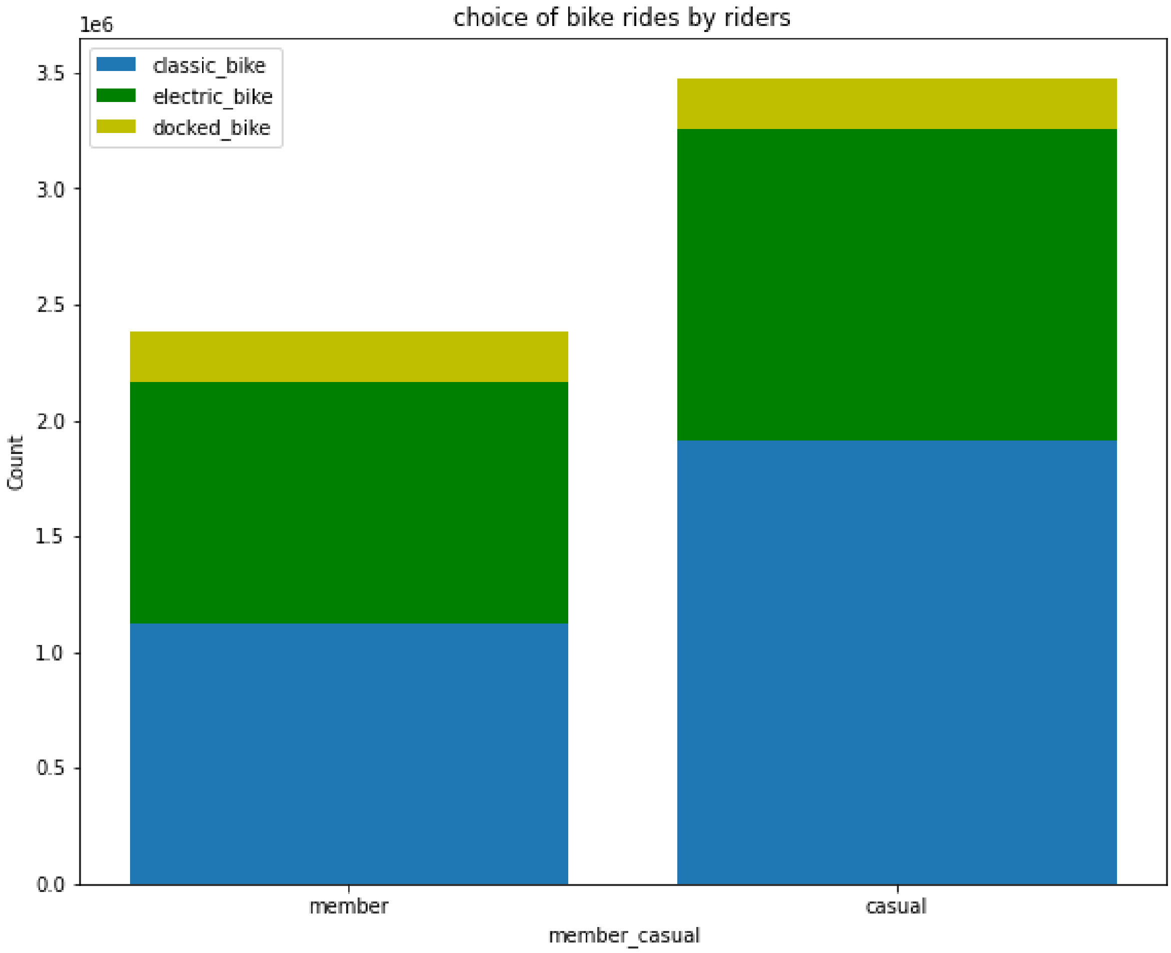
Choice of Bikes by Riders

Entrée [ ]:

```
x=['member','casual']
y1=data[data['rideable_type']=='classic_bike'].groupby(['member_casual'])['ride_id'].count(
y2=data[data['rideable_type']=='electric_bike'].groupby(['member_casual'])['ride_id'].count
y3=data[data['rideable_type']=='docked_bike'].groupby(['member_casual'])['ride_id'].count()
```

Entrée [ ]:

```
f, ax = plt.subplots(figsize=(10,8))
plt.bar(x,y1)
plt.bar(x, y2, bottom=y1, color='g')
plt.bar(x, y3, bottom=y1+y2, color='y')
plt.xlabel("member_casual")
plt.ylabel("Count")
plt.legend(["classic_bike", "electric_bike", "docked_bike"])
plt.title("choice of bike rides by riders")
plt.show()
```

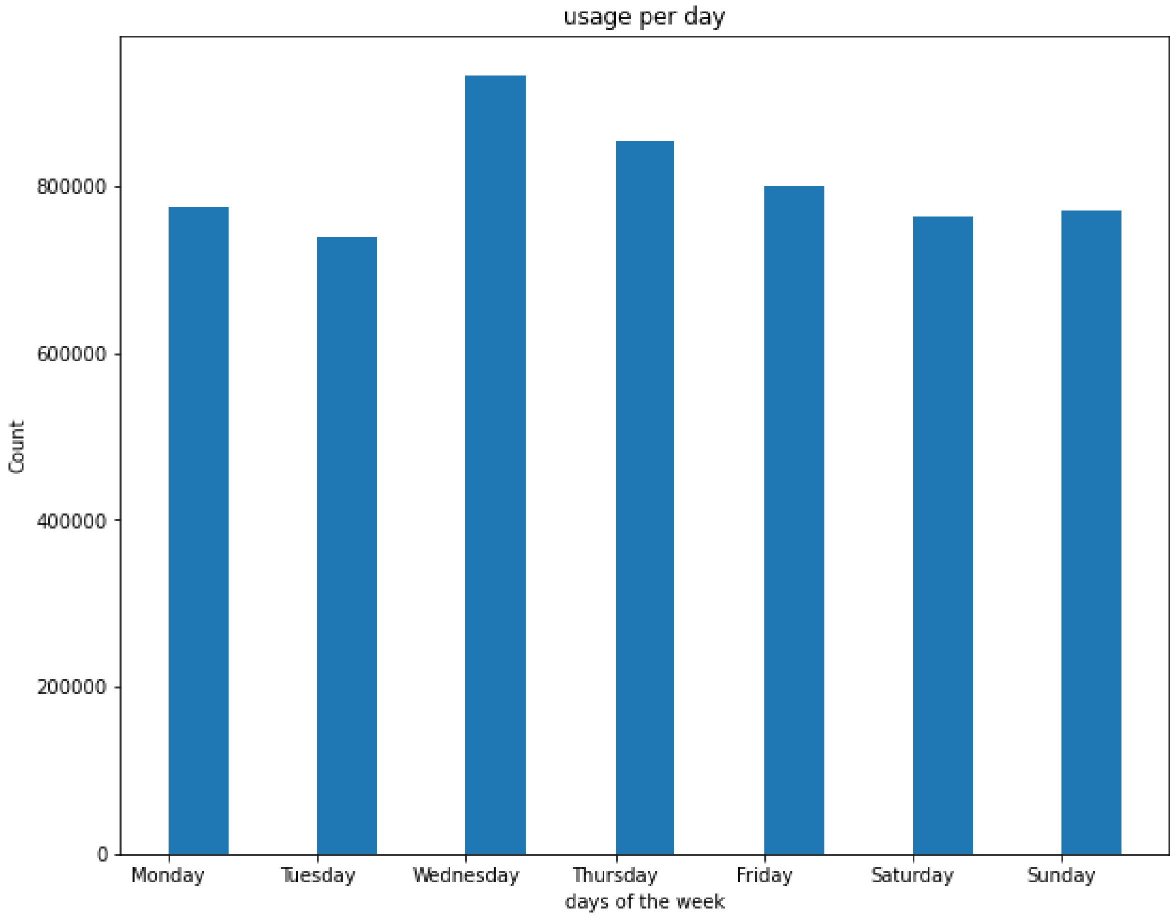


Weekday Rides (usage on different days)

Entrée [ ]:

```
x=['Monday', 'Tuesday', 'Wednesday','Thursday', 'Friday', 'Saturday','Sunday']
y=data.groupby(['week_day'])['ride_id'].count()
f, ax = plt.subplots(figsize=(10,8))
plt.bar(x,y, align='edge', width=0.4)
plt.xlabel('days of the week') # add to x-label to the plot
plt.ylabel('Count') # add y-label to the plot
plt.title('usage per day') # add title to the plot

plt.show()
```



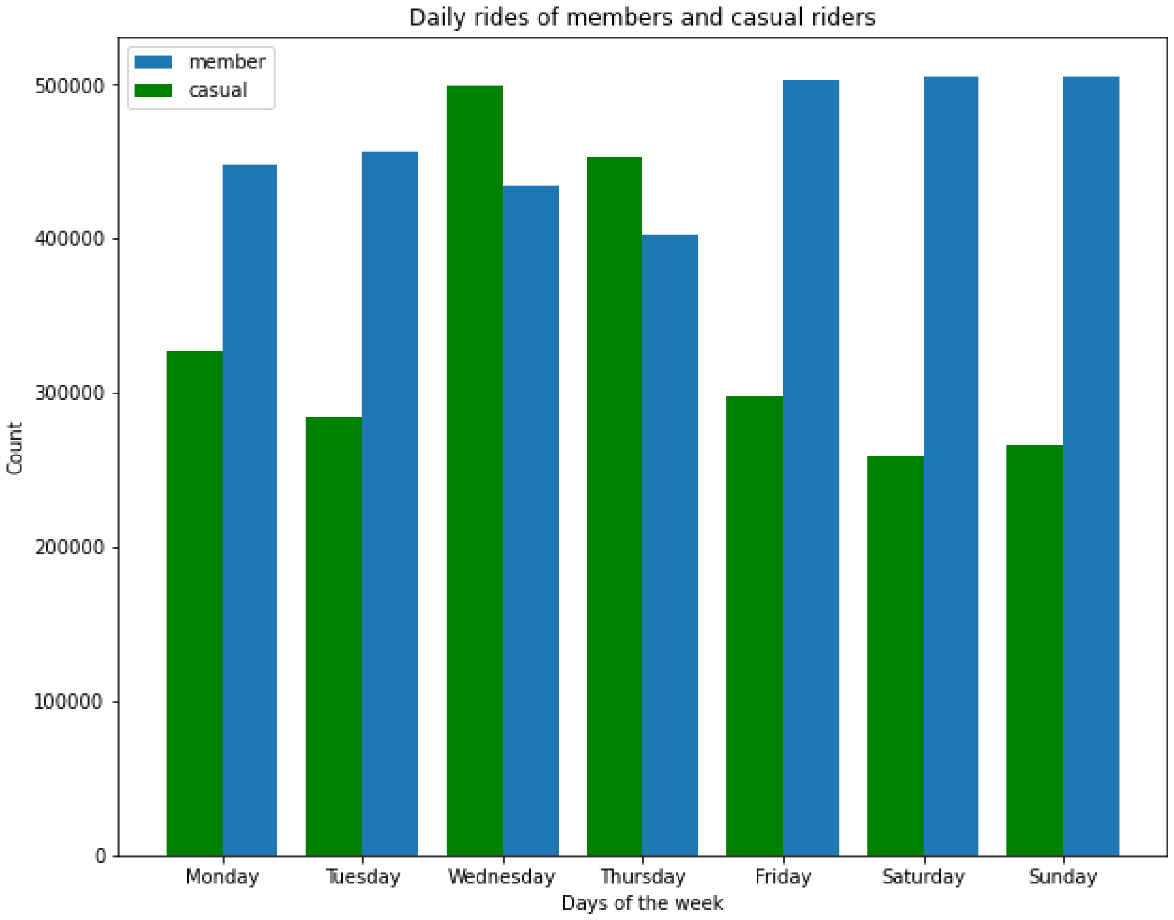
Daily rides of members and casual riders

Entrée [ ]:

```
x=['Monday', 'Tuesday', 'Wednesday','Thursday', 'Friday', 'Saturday','Sunday']
y1=data[data['member_casual']=='member'].groupby(['week_day'])['ride_id'].count()
y2=data[data['member_casual']=='casual'].groupby(['week_day'])['ride_id'].count()


f, ax = plt.subplots(figsize=(10,8))
plt.bar(x,y1, align='edge', width=0.4)
plt.bar(x, y2, color='g',align='edge', width=-0.4)

plt.xlabel("Days of the week")
plt.ylabel("Count")
plt.legend(["member", "casual"])
plt.title("Daily rides of members and casual riders")
plt.show()
```



Member's usage is high throughout friday and the weekend.

Casual Riders prefer to ride on wednesday.

Monthly Trends of rides of members and casual riders

Entrée [ ]:

```
x=['January','February','March','April','May' ,'June','July','August','September' ,'October
month_rider_type=pd.DataFrame()
for i in range(1,10):
    a=data[data.started_at.str.contains('-0'+str(i)+'-')].groupby(['member_casual'])['ride_
    month_rider_type=pd.concat([month_rider_type,a], axis=1)
    month_rider_type.rename(columns={'ride_id': x[i-1]}, inplace=True)
for i in range(10,13):
    a=data[data.started_at.str.contains('-'+str(i)+'-')].groupby(['member_casual'])['ride_i
    month_rider_type=pd.concat([month_rider_type,a], axis=1)
    month_rider_type.rename(columns={'ride_id': x[i-1]}, inplace=True)
#month_rider_type.reset_index(inplace=True)
#month_rider_type.rename(columns={'index': 'casual_member'}, inplace=True)
```

Entrée [ ]:

```
month_rider_type['casual_member']=month_rider_type.index
```

Entrée [ ]:

```
month_rider_type
```

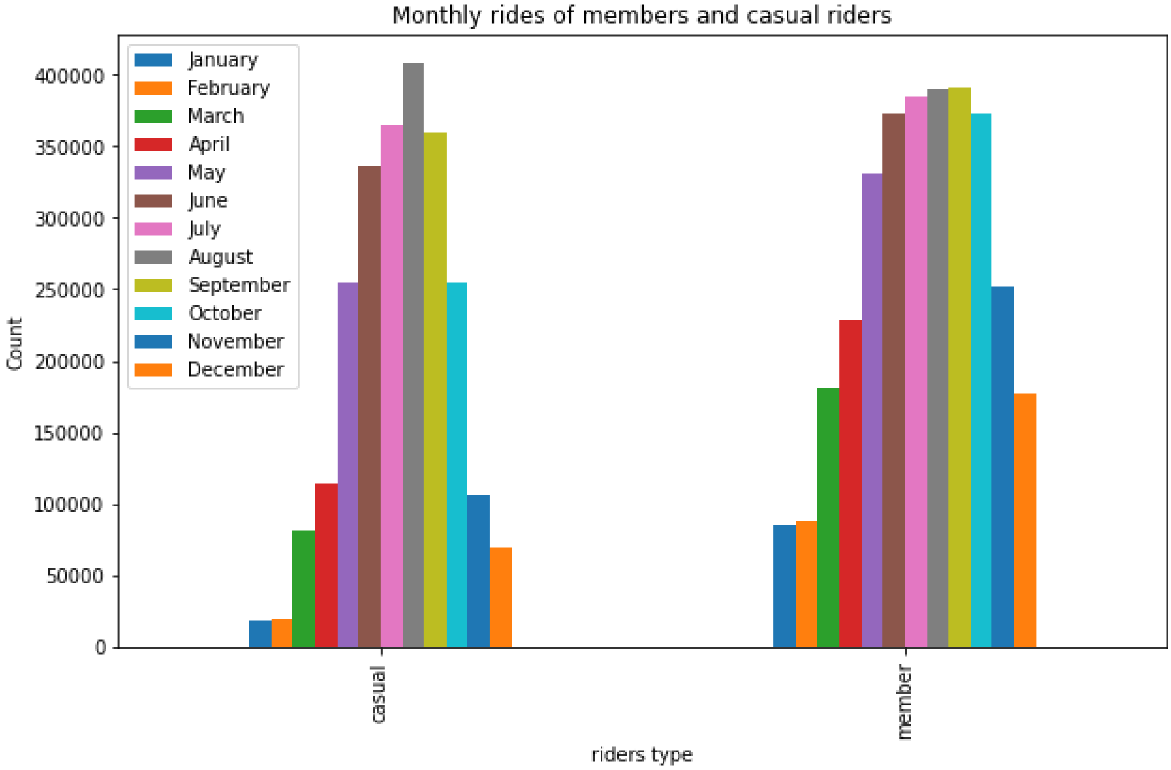
Out[145]:

	January	February	March	April	May	June	July	August	September	Oc
casual	18333	19083	81196	113878	254894	335721	365071	407654	359766	2
member	84999	88398	181203	227991	330650	373015	384966	390152	390762	3

Entrée [ ]:

```
month_rider_type.loc[['casual','member'], x].plot(kind='bar', figsize=(10, 6))
plt.xlabel("riders type")
plt.ylabel("Count")

plt.title("Monthly rides of members and casual riders")
plt.show()
```

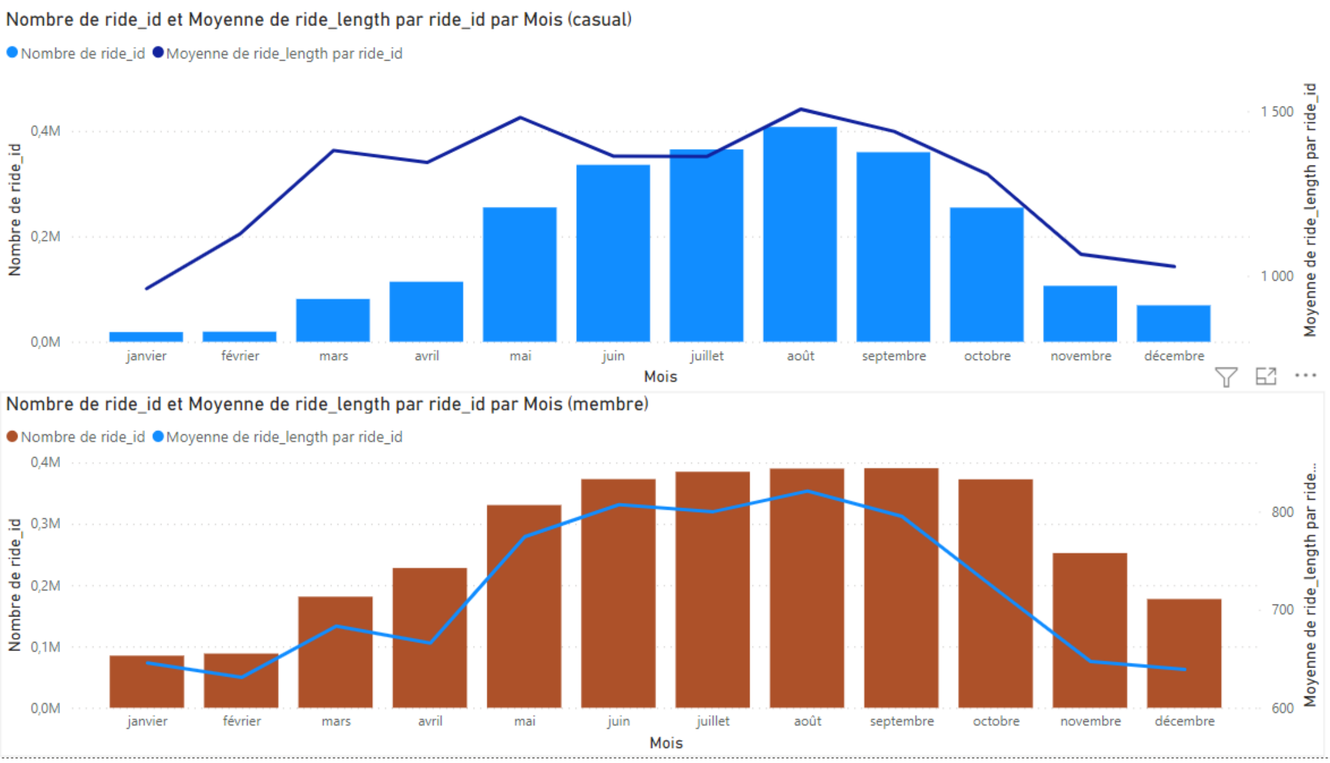


Casual riders numbers spikes up in the summer.

Maybe a good way to deal with that is issuing summer subscription.

For further analysis we are going to use advanced visualisations with

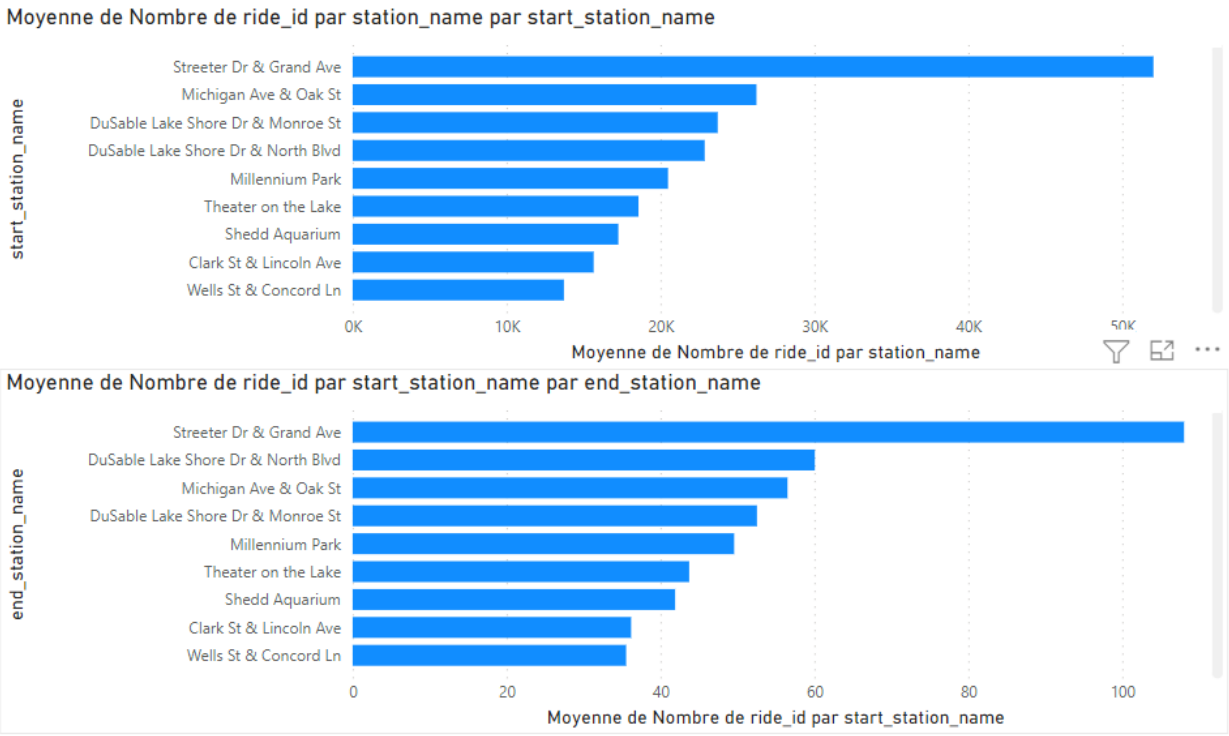
PowerBi



Members take more rides than casual Riders.

Casual Riders take longer rides than Members.

Most used stations by casual riders



stations les plus visité par les casual riders pendant la haute saison(mai-septembre)

These are the stations where our publicity is most effective

Share

Recommendations

To convert casual riders into annual members, the following marketing strategies can be implemented:

Offer occasional membership discounts to casual riders during the summer.

Put banners or special discount advertisements at the 5 most used station by casual riders.

Entrée [ ]:

```

```