Divvy Bike Share System Analysis

March 27, 2025

0.1 Divvy Bike Share System

Analyzing User Behavior to Optimize Marketing Strategy ### **Project Goal:** Analyzing Divvy bike trip data (a bike-sharing system in Chicago) to identify differences in behavior between annual members and casual riders. The goal is to develop a marketing strategy based on the findings, aimed at attracting casual riders to annual membership.

0.1.1 1. Data Processing With Python

[ ]:
```
import pandas as pd
import numpy as np
import os
# Import all the necessary packages

pwd = os.getcwd()
# Create pwd path variable to make the code cleaner
```

[ ]:
```
ds1 = pd.read_csv(pwd + "/Trips_Data/24_01_tripdata.csv")
ds2 = pd.read_csv(pwd + "/Trips_Data/24_02_tripdata.csv")
ds3 = pd.read_csv(pwd + "/Trips_Data/24_03_tripdata.csv")
ds4 = pd.read_csv(pwd + "/Trips_Data/24_04_tripdata.csv")
ds5 = pd.read_csv(pwd + "/Trips_Data/24_05_tripdata.csv")
ds6 = pd.read_csv(pwd + "/Trips_Data/24_06_tripdata.csv")
ds7 = pd.read_csv(pwd + "/Trips_Data/24_07_tripdata.csv")
ds8 = pd.read_csv(pwd + "/Trips_Data/24_08_tripdata.csv")
ds9 = pd.read_csv(pwd + "/Trips_Data/24_09_tripdata.csv")
ds10 = pd.read_csv(pwd + "/Trips_Data/24_10_tripdata.csv")
ds11 = pd.read_csv(pwd + "/Trips_Data/24_11_tripdata.csv")
ds12 = pd.read_csv(pwd + "/Trips_Data/24_12_tripdata.csv")
# Import all the files
```

Since our data is divided into 12 separate files, where each file contains each month separately, we have to combine them into a whole DataFrame

[4]:
```
bikes_data = pd.concat([ds1, ds2, ds3, ds4, ds5, ds6, ds7, ds8, ds9, ds10,␣
↪ds11, ds12])
bikes_data
# Concatinatig them into one big DataFrame
```

[4]:

	ride_id	rideable_type	started_at	ended_at	start_station_name	start_station_id	end_station_name	end_station_id	start_lat	start_lng	end_lat	end_lng	member_casual
0	C1D650626C8C899A	electric_bike	2024-01-12 15:30:27	2024-01-12 15:37:59	Wells St & Elm St	KA1504000135	Kingsbury St & Kinzie St	KA1503000043	41.903267	-87.634737	41.889177	-87.638506	member
1	EECD38BDB25BFCB0	electric_bike	2024-01-08 15:45:46	2024-01-08 15:52:59	Wells St & Elm St	KA1504000135	Kingsbury St & Kinzie St	KA1503000043	41.902937	-87.634440	41.889177	-87.638506	member
2	F4A9CE78061F17F7	electric_bike	2024-01-27 12:27:19	2024-01-27 12:35:19	Wells St & Elm St	KA1504000135	Kingsbury St & Kinzie St	KA1503000043	41.902951	-87.634470	41.889177	-87.638506	member
3	0A0D9E15EE50B171	classic_bike	2024-01-29 16:26:17	2024-01-29 16:56:06	Wells St & Randolph St	TA1305000030	Larrabee St & Webster Ave	13193	41.884295	-87.633963	41.921822	-87.644140	member
4	33FFC9805E3EFF9A	classic_bike	2024-01-31 05:43:23	2024-01-31 06:09:35	Lincoln Ave & Waveland Ave	13253	Kingsbury St & Kinzie St	KA1503000043	41.948797	-87.675278	41.889177	-87.638506	member
…	…	…	…	…	…	…	…	…	…	…	…	…	…
178367	BD56BA20F42E4794	electric_bike	2024-12-11 08:23:46.564	2024-12-11 08:37:34.532	Clybourn Ave & Division St	TA1307000115	NaN	NaN	41.904634	-87.640518	41.880000	-87.630000	member
178368	3074643A6B60B300	electric_bike	2024-12-09 12:26:15.677	2024-12-09 12:37:32.712	Canal St & Jackson Blvd	13138	NaN	NaN					
178369	15602635C5DF484E	electric_bike	2024-12-31 17:10:03.113	2024-12-31 17:17:21.838	Albany Ave & Bloomingdale Ave	15655	California Ave & Milwaukee Ave	13084					
178370	F15ABBA961560B75	electric_bike	2024-12-01 14:39:47.216	2024-12-01 14:45:21.268	Albany Ave & Bloomingdale Ave	15655	California Ave & Milwaukee Ave	13084					
178371	8AF273287533B527	electric_bike	2024-12-17 06:38:32.320	2024-12-17 06:46:27.167	Albany Ave & Bloomingdale Ave	15655	NaN	NaN					

```
178368  41.878125 -87.639968  41.900000 -87.620000        member
178369  41.914027 -87.705126  41.922695 -87.697153        member
178370  41.914003 -87.705099  41.922695 -87.697153        member
178371  41.914027 -87.705126  41.920000 -87.690000        member

[5860568 rows x 13 columns]
```

Let's make some quick checks to learn more about our data

[5]:
```
bikes_data.dtypes
```

[5]:
```
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
```

[ ]:
```
bikes_data.isnull().sum()
# Check for NaN values
```

[ ]:
```
ride_id                   0
rideable_type             0
started_at                0
ended_at                  0
start_station_name    1073951
start_station_id      1073951
end_station_name      1104653
end_station_id        1104653
start_lat                 0
start_lng                 0
end_lat                7232
end_lng                7232
member_casual             0
dtype: int64
```

[ ]:
```
bikes_data.duplicated().sum()
# Check for duplicates
```

[ ]:
```
np.int64(0)
```

Fine, the data has been checked, so it's time to start cleaning and transforming.

[ ]:
```
bikes_data = bikes_data.drop(columns=["ride_id", "start_station_id",␣
 ↪"start_station_name", "end_station_name", "end_station_id",
                  "start_lat", "start_lng", "end_lat", "end_lng"])
bikes_data.head()
# Drop unnecessary columns
```

[ ]:

	rideable_type	started_at	ended_at	member_casual
0	electric_bike	2024-01-12 15:30:27	2024-01-12 15:37:59	member
1	electric_bike	2024-01-08 15:45:46	2024-01-08 15:52:59	member
2	electric_bike	2024-01-27 12:27:19	2024-01-27 12:35:19	member
3	classic_bike	2024-01-29 16:26:17	2024-01-29 16:56:06	member
4	classic_bike	2024-01-31 05:43:23	2024-01-31 06:09:35	member

[ ]:
```
bikes_data["started_at"] = pd.to_datetime(bikes_data["started_at"].str.split('.
 ↪').str[0], format="%Y-%m-%d %H:%M:%S")
bikes_data["ended_at"] = pd.to_datetime(bikes_data["ended_at"].str.split('.').
 ↪str[0], format="%Y-%m-%d %H:%M:%S")
# Coverting coumns into a datetime type
# I also drop miliseconds to make the data easier to use

print(bikes_data.dtypes)
# A quick check
```

```
rideable_type            object
started_at       datetime64[ns]
ended_at         datetime64[ns]
member_casual            object
dtype: object
```

Make a new column **"ride__time__minutes"** to know a trip duration for each record.

[ ]:
```
bikes_data["ride_time_minutes"] = round((bikes_data["ended_at"] -␣
 ↪bikes_data["started_at"]).dt.total_seconds() / 60).astype(int)
bikes_data.head()
# Substract the strat time from end time
# Dividing by 60 to convet into minutes
```

[ ]:

	rideable_type	started_at	ended_at	member_casual	ride_time_minutes
0	electric_bike	2024-01-12 15:30:27	2024-01-12 15:37:59	member	8
1	electric_bike	2024-01-08 15:45:46	2024-01-08 15:52:59	member	
2	electric_bike	2024-01-27 12:27:19	2024-01-27 12:35:19	member	
3	classic_bike	2024-01-29 16:26:17	2024-01-29 16:56:06	member	
4	classic_bike	2024-01-31 05:43:23	2024-01-31 06:09:35	member	

```
1                  7
2                  8
3                 30
4                 26
```

Let's make a few basic calculations on our new column.

[ ]:
```
bikes_data.groupby('member_casual')['ride_time_minutes'].mean()
# Calculate the average
```

[ ]:
```
member_casual
casual    25.147047
member    12.768463
Name: ride_time_minutes, dtype: float64
```

[ ]:
```
bikes_data.groupby('member_casual')['ride_time_minutes'].min()
# Calculate the minimum
```

[ ]:
```
member_casual
casual    -160
member   -2748
Name: ride_time_minutes, dtype: int64
```

[ ]:
```
bikes_data.groupby('member_casual')['ride_time_minutes'].max()
# Calculate the maximum
```

[ ]:
```
member_casual
casual    1560
member    1560
Name: ride_time_minutes, dtype: int64
```

As a result, I see that we have some negative values in our column which seems incorrect to me, let's find out exactly how many of them there are and find a solution.

[ ]:
```
negative_count = (bikes_data["ride_time_minutes"] < 0).sum()
print(negative_count)
# Let's see how many of these negative values we have
```

```
99
```

Understanding the size of our data, in this case we can simply eliminate those rows whitout significante affect on the whole picture.

[ ]:
```
bikes_data = bikes_data[bikes_data["ride_time_minutes"] >= 0]
# Leave only the data in the column, that is equal to or greater than zero

negative_count = (bikes_data["ride_time_minutes"] < 0).sum()
print(negative_count)
# Let's see if everyting went well
```

0

Now I want to create a new column where the duration of each trip will be categorized, which will make it easier to understand the data in the future when it is visualized.

[ ]:
```
def if_el(duration):
    if duration < 10:
        return "Under 10"
    elif 10 <= duration < 20:
        return "10 to 20"
    elif 20 <= duration < 40:
        return "20 to 40"
    elif 40 <= duration < 60:
        return "40 to 60"
    else:
        return "Over 60"
# Creating our own function if_el to categorize duration
# Categorize trip durations into the following intervals: "Under 10", "10 to␣
 ↪20", "20 to 40", "40 to 60", and "Over 60"

bikes_data["trip_duration"] = bikes_data["ride_time_minutes"].apply(if_el)
# Apply the function to the chosen column
bikes_data.head()
```

[ ]:

	rideable_type	started_at	ended_at	member_casual	ride_time_minutes	trip_duration
0	electric_bike	2024-01-12 15:30:27	2024-01-12 15:37:59	member	8	Under 10
1	electric_bike	2024-01-08 15:45:46	2024-01-08 15:52:59	member	7	Under 10
2	electric_bike	2024-01-27 12:27:19	2024-01-27 12:35:19	member	8	Under 10
3	classic_bike	2024-01-29 16:26:17	2024-01-29 16:56:06	member	30	20 to 40
4	classic_bike	2024-01-31 05:43:23	2024-01-31 06:09:35	member	26	20 to 40

Now it's time to make a few new columns in order to dive deeper and see if there is any patterns in our trips data

[ ]:
```
bikes_data["month"] = bikes_data["started_at"].dt.month_name()
bikes_data["day"] = bikes_data["started_at"].dt.day_name()
bikes_data["hour"] = bikes_data["started_at"].dt.hour
# Create three new columns with month name, day of the week and hour
bikes_data.head()
# Another check if everyting went well
```

[ ]:
```
   rideable_type          started_at            ended_at member_casual  \
0  electric_bike 2024-01-12 15:30:27 2024-01-12 15:37:59        member
1  electric_bike 2024-01-08 15:45:46 2024-01-08 15:52:59        member
2  electric_bike 2024-01-27 12:27:19 2024-01-27 12:35:19        member
3   classic_bike 2024-01-29 16:26:17 2024-01-29 16:56:06        member
4   classic_bike 2024-01-31 05:43:23 2024-01-31 06:09:35        member

   ride_time_minutes trip_duration    month        day  hour
0                  8      Under 10  January     Friday    15
1                  7      Under 10  January     Monday    15
2                  8      Under 10  January   Saturday    12
3                 30      20 to 40  January     Monday    16
4                 26      20 to 40  January  Wednesday     5
```

Now that we have finished working with the time and dates, we can get rid of unnecessary columns.

[18]:
```
bikes_data = bikes_data.drop(columns=["started_at", "ended_at"])
bikes_data.head()
# Perform the drop process
```

[18]:
```
   rideable_type member_casual  ride_time_minutes trip_duration    month  \
0  electric_bike        member                  8      Under 10  January
1  electric_bike        member                  7      Under 10  January
2  electric_bike        member                  8      Under 10  January
3   classic_bike        member                 30      20 to 40  January
4   classic_bike        member                 26      20 to 40  January

         day  hour
0     Friday    15
1     Monday    15
2   Saturday    12
3     Monday    16
4  Wednesday     5
```

Almost finished! Let's group our data together and count them.

[19]:
```
bikes_data_final = bikes_data.groupby(["rideable_type", "member_casual",
                      "ride_time_minutes", "trip_duration", "month", "day",␣
 ↪"hour"]).size().reset_index(name="num_of_rides")
# Grouping all the data and adding a new column "num_of_rides" with the count
bikes_data_final
# Let's take a look at what we have done
```

[19]:
```
           rideable_type member_casual  ride_time_minutes trip_duration  \
0           classic_bike        casual                  0      Under 10
1           classic_bike        casual                  0      Under 10
2           classic_bike        casual                  0      Under 10
3           classic_bike        casual                  0      Under 10
```

```
4           classic_bike        casual                  0      Under 10
…                  …           …                   …           …
424808  electric_scooter        member                116       Over 60
424809  electric_scooter        member                183       Over 60
424810  electric_scooter        member                220       Over 60
424811  electric_scooter        member                256       Over 60
424812  electric_scooter        member                480       Over 60

            month       day  hour  num_of_rides
0           April    Friday     0             1
1           April    Friday     8             1
2           April    Friday     9             3
3           April    Friday    10             2
4           April    Friday    11             5
…           …       …   …           …
424808  September    Sunday    22             1
424809  September    Monday    18             1
424810  September    Sunday    13             1
424811  September  Thursday    14             1
424812  September  Saturday    15             1

[424813 rows x 8 columns]
```

[ ]:
```
bikes_data_final.to_csv("Divvy_Bike_Share_System.csv", index=False)
# Save the data in csv format
```

0.1.2 2. Analyze Data With Tableau

Since the data has been processed, grouped, and saved in CSV format. Now we can use this file to create a dashboard and analyze it with Tableau.

0.1.3 3. Share Conclusions and Recommendations

The entire process of working with the project, a description of the research and conclusions drawing are available in the Read.me file attached to the published project on Github