PROJECT REPORT

DIVVYS BIKESHARE

NEMA AENUGU

SUMANTH SRIVATSAV BALIJEPALLI

Table of Contents

Background

The bike share program for Chicagoland is called Divvy, and it has 6,000 bikes available at 570+ stations across Chicago and Evanston. Divvy offers locals and visitors a quick, enjoyable, and economical means of transportation for moving around and discovering Chicago.

Like other bike share programs, Divvy uses a fleet of sturdy, durable bikes that are locked into a system of docking stations spread out over the area. From one station, the bikes can be unlocked and then brought back to any other station in the network. To move around Chicago, travel to work or school, run errands, make it to appointments or social gatherings, and more, people use bike sharing.

Riders have access to all bikes and stations across the system using Divvy, which is available around the clock, every day of the week, 365 days a year.

Key questions

Cycling in groups promotes a healthier and more environmentally conscious way of life. The sharing bicycles might be powered by an intriguing technology. The objective is to learn:

1. What is the riding pattern?
   If you are aware of the times when people are using bikes, you can make more of them available at those times. You can also find out where they are getting their bikes from.

2. To what extent do they go?
   It would be necessary to address the possibility that others wouldn't have access to bikes during long bike rides at a particular period.

3. Which bike stations have high demand?
   By identifying the popular stations, one can pay more attention to them, learn what makes them special, and perhaps incorporate those qualities into other stations. They can also assist locate similar areas in the city and establish new stations.

4. Which days of the week see the most rides?
   Knowing the busiest days will allow you to allocate resources accordingly.

Data

Data source

Divvy Bike sharing data: Divvy Bikeshare data website[1]

The contains data for 7 years between 2015 and 2022. It is meant for public use of Divvy's travel information. This data can assist to map, animating, or otherwise bringing it to life. This will help a decision-maker in policy, an expert in transportation, a web developer, and a designer to study the bike-sharing program.

Data description

The downloaded from the source was in zip format. We downloaded individual files from the data-sharing website which formed our raw data. We unzipped them to obtain 49 files of 4.62 gigabytes. The files contained trip information for 7 years between 2015 and 2022. There were files containing information on the stations where the bikes are docked. Over the past 7 years, the data storing schema has changed multiple times which posed a challenge while combining all the files to build a master dataset.

After combining all the files and reformatting them, we were left with the following data attributes as shown in Table 1. Details on cleaning and process are provided in the following sections. Please note, the data contains basic information on the users if they are subscribed to the bike-sharing program else, there was no information on the users.

1 https://divvy-tripdata.s3.amazonaws.com/index.html

Table 1 Data attributes

Column Name	**Type**
TRIP ID	Number
START TIME	Date & Time
STOP TIME	Date & Time
BIKE ID	Plain Text
TRIP DURATION	Number
FROM STATION ID	Plain Text
FROM STATION NAME	Plain Text
TO STATION ID	Plain Text
TO STATION NAME	Plain Text
USER TYPE	Plain Text
GENDER	Plain Text
BIRTH YEAR	Number
FROM LATITUDE	Number
FROM LONGITUDE	Number
FROM LOCATION	Point
TO LATITUDE	Number
TO LONGITUDE	Number
TO LOCATION	Point

Data transformation/Exploratory data analysis

We developed a custom Python[2] script to perform all the data processes. The developed script is heavily based on the pandas[3] library. Pandas is an open-source, BSD[4]-licensed library that offers high-performance, user-friendly data structures, and tools for data analysis.

Data Reading

The first step is to read the data from CSV files into a data frame. This was one of the most time-consuming parts of the process.

We first retrieved a list of all the files ending in .csv to form a list of the input files. Then we merged all the files in one Pandas DataFrame. The DataFrame contained all the data with unformatted information and was in raw format. Figure 1 shows a piece of code to perform these actions.

[2] https://www.python.org/
[3] https://pandas.pydata.org/docs/
[4] https://en.wikipedia.org/wiki/BSD_licenses

Figure 1 Reading raw data from the input files.

```python
 # Import libraries for data manipulation

import numpy as np
import pandas as pd
# reading csv files

file_list = [f for f in glob.glob("./Data/Raw/*trip*.csv")]
raw_data = pd.DataFrame()

for ii in range(len(file_list)):
  raw_data = pd.concat([raw_data, pd.read_csv(file_list[ii])], axis=0,
ignore_index=True)
```

Once the files are read into one variable, next we reformatted the data to make all the information consistent and coherent.
The data from different years was stored in different formats under varying attribute names. Shows all the attribute names as they were read from the files.

Figure 2 Raw data attributes

```python
raw_data.columns

'''
'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', 'trip_id', 'starttime', 'stoptime', 'bikeid',
'tripduration', 'from_station_id', 'from_station_name', 'to_station_id',
'to_station_name', 'usertype', 'gender', 'birthyear', 'start_time',
'end_time', '01 - Rental Details Rental ID',
'01 - Rental Details Local Start Time',
'01 - Rental Details Local End Time', '01 - Rental Details Bike ID',
'01 - Rental Details Duration In Seconds Uncapped',
'03 - Rental Start Station ID', '03 - Rental Start Station Name',
'02 - Rental End Station ID', '02 - Rental End Station Name',
'User Type', 'Member Gender',
'05 - Member Details Member Birthday Year']
'''
```

It can be seen multiple variable names appear to contain the same information but are named differently. To make it consistent for data analysis, we merged different columns containing the same information. This process took about 4 days of processing.
First, we worked on the date and time column. Columns related to date and time and the reformatting process are shown below in

Figure 3 Data attributes to merge

```
#Columns = ['usertype','User Type', 'member_casual']
#Columns = ['start_station_id', 'from_station_id', '03 - Rental Start Station
ID']
#Columns = ['end_station_id', 'to_station_id','02 - Rental End Station ID']
#Columns = ['ride_id', 'trip_id', '01 - Rental Details Rental ID']
#Columns = ['gender', 'Member Gender']
#Columns = ['birthyear','05 - Member Details Member Birthday Year']
#Columns = ['bikeid', '01 - Rental Details Bike ID']
#Columns = ['tripduration', '01 - Rental Details Duration In Seconds Uncapped']
```

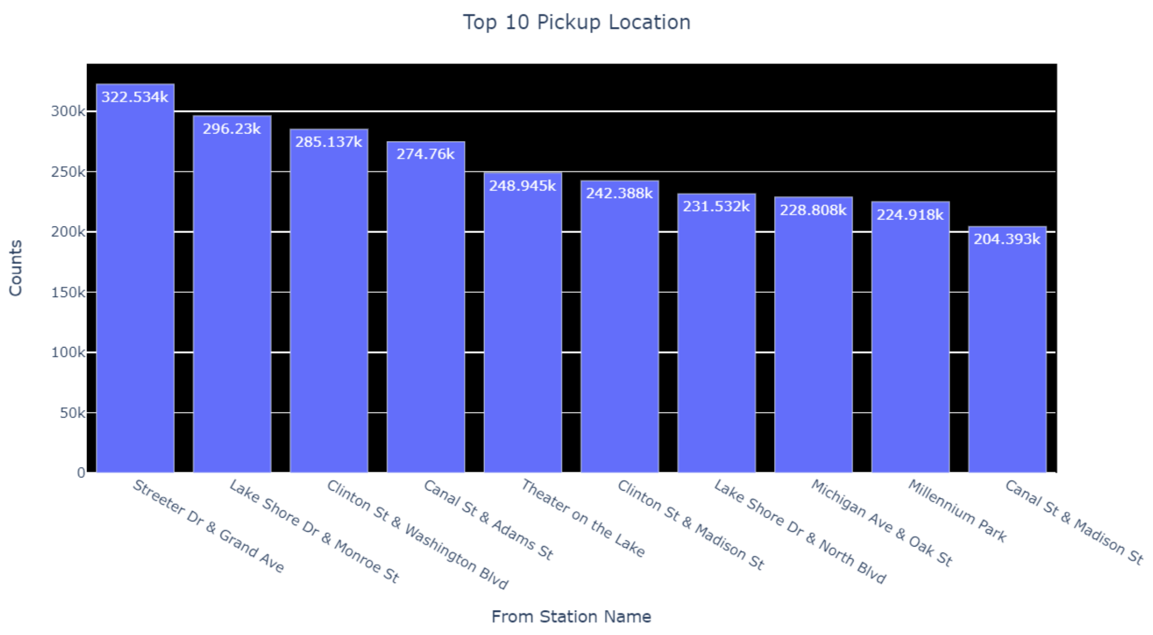
Once the coding was done, our machines were running for about 7 days to perform all the reading, reformatting, and cleaning of the data. To be safe from data loss during the crashing of code, we wrote the data to files and read them again to save the process until the checkpoint.

Analysis

Once the data is ready for analysis, we started with the managerial question

What are the TOP 10 popular stations to begin a ride at?

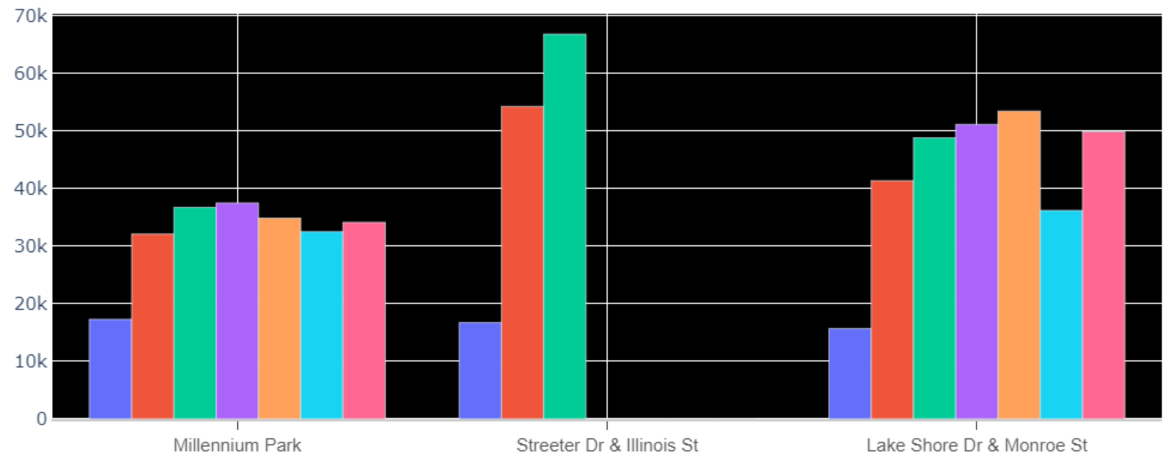
Figure 4 Popular stations



Average rides from each station are distributed with a mean of 28,399.39 and a median of 10,198.5 rides.

Year-wise top stations

Figure 5 Popular Stations



How many round trips?

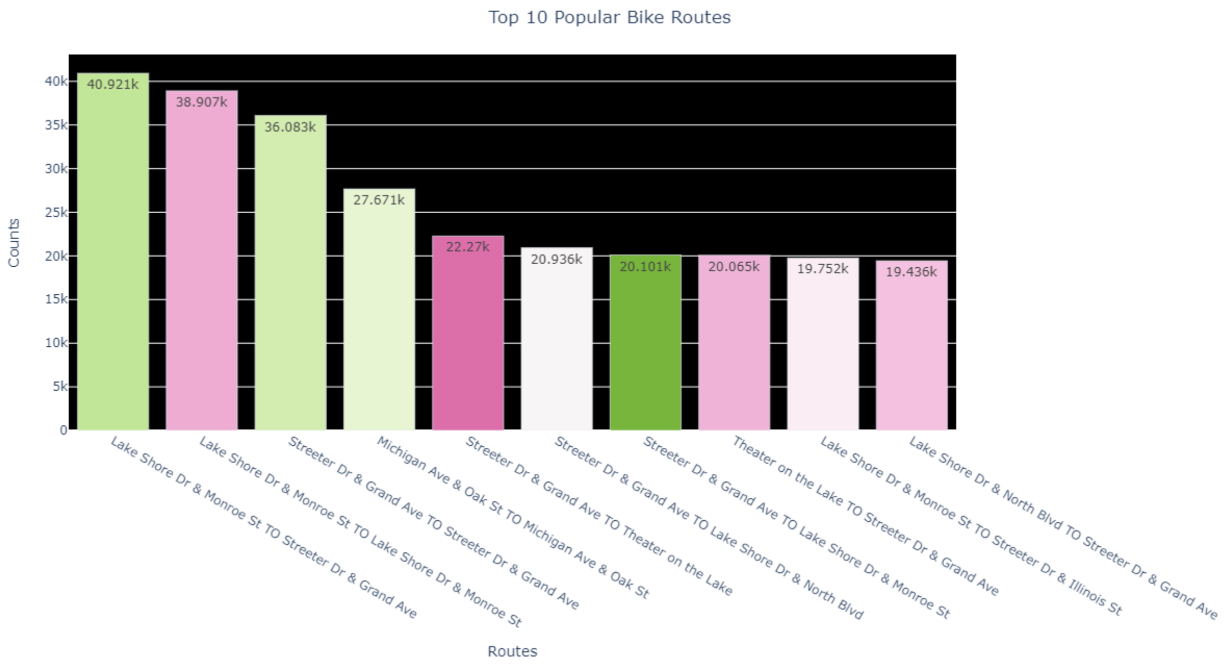
Here, we count and calculate the percentage of round trips, i.e., rides 'FROM' and 'TO' stations are the same. This is important to know because the service provider needs to rebalance the bikes from low-demand to high-demand areas.

Total Round Trips: 787,236

Percentage of Round Trips: 3.71%

What are the most popular paths?

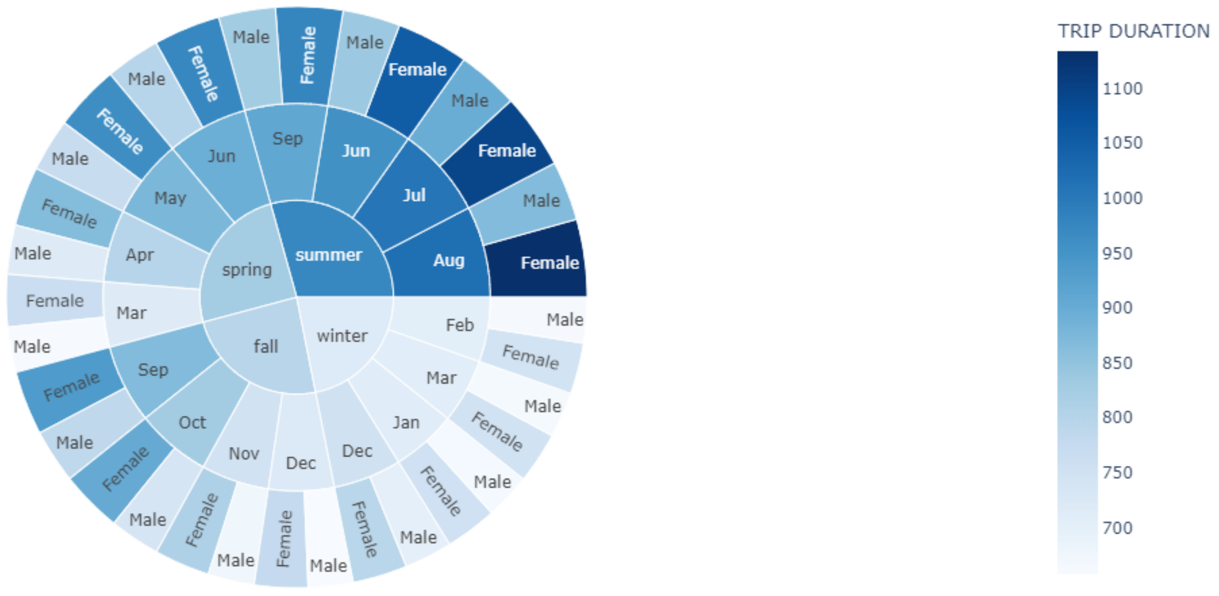
Figure 6 Popular Paths



According to the data, Lake Shore Drive & Monroe Street to Streeter & Grand Ave is the most popular route.

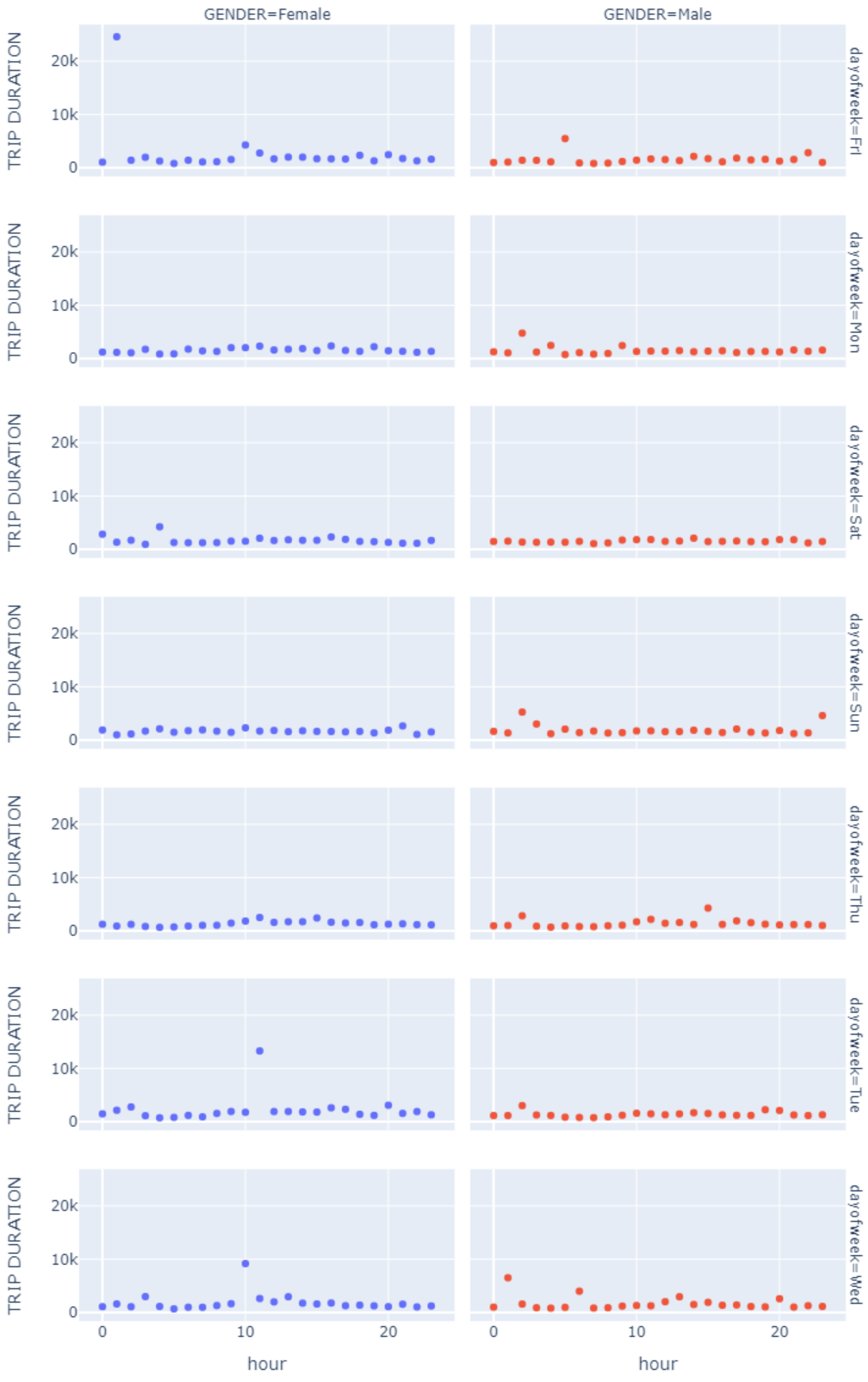
What is the trip distribution?

Figure 7 Seasonal Distribution of trips



Summer season realized most of the trips, where female riders made more trips than men riders.

Figure 8 Trip distribution by gender and day of the week



Findings and managerial implications

When it comes to how frequently people utilize bikes throughout the year, both genders have developed a consistent pattern over time. The numbers are low in the winter and gradually increase during the year, reaching their peak in the summer. The same holds if you look at it every week over time; both genders follow a similar pattern, with low numbers on Saturdays and Sundays and high numbers on Tuesdays. There are 4,112,418 female users and 12,235,452 male users, i.e., 25% female and 75% male users. Summer season months realize most of the riders, followed by spring, fall, and lastly by winter. This makes sense as the city of Chicago gets extreme winter in the mid-west area. Of the Summer seasons, August and Jul are the busiest months.

Stations around Lake Shore, Streeter Drive, Grand Ave, and Michigan Avenue are the most popular areas. These locations are prime tourist attractions and have numerous summer activities. Of all the popular locations, Millennium Park is one the most famous destination.

Conclusions

The bike share program for Chicagoland is called Divvy, and it has 6,000 bikes available at 570+ stations across Chicago and Evanston. Divvy offers locals and visitors a quick, enjoyable, and economical means of transportation for moving around and discovering Chicago.

We retrieved Divvy Bike sharing data containing 7 years in 49 files of 4.62 gigabytes. We developed a custom Python script to perform all the data processes. The developed script is heavily based on the pandas library. The data from different years was stored in different formats under varying attribute names. Shows all the attribute names as they were read from the files. the coding was done, and our machines were running for about **7 days non-stop** to perform all the reading, reformatting, and cleaning of the data. To be safe from data loss during the crashing of code, we wrote the data to files and read them again to save the process until the checkpoint.

Average rides from each station are distributed with a mean of 28,399.39 and a median of 10,198.5 rides. Here, we count and calculate the percentage of round trips, i.e., rides 'FROM' and 'TO' stations are the same. This is important to know because the service provider needs to rebalance the bikes from low-demand to high-demand areas. Total Round Trips: 787,236 and Percentage of Round Trips: 3.71%. According to the data, Lake Shore Drive & Monroe Street to Streeter & Grand Ave is the most popular route. Summer season realized most of the trips, where female riders made more trips than men riders. There are 4,112,418 female users and 12,235,452 male users, i.e., 25% female and 75% male users. Summer season months realize most of the riders, followed by spring, fall, and lastly by winter. This makes sense as the city of Chicago gets extreme winter in the mid-west area. Of the Summer seasons, August and Jul are the busiest months.

Appendix: python codes with proper documentation

```python
# Data analysis libraries

import numpy as np

import pandas as pd


import matplotlib.pyplot as plt
import plotly.express as px


import plotly.graph_objects as go

import glob


# Import libraries for data manipulation

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

plt.style.use('seaborn')

import seaborn as sns


# reading csv files
file_list = [f for f in glob.glob("./Data/Raw/*trip*.csv")]

raw_data = pd.DataFrame()



for ii in range(len(file_list)):
    raw_data = pd.concat([raw_data, pd.read_csv(file_list[ii])], axis=0,
ignore_index=True)


raw_data.to_csv("./Data/Merged_raw_data2.csv", index=False)


raw_data.columns
```

```
'''
'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', 'trip_id', 'starttime', 'stoptime', 'bikeid',
'tripduration', 'from_station_id', 'from_station_name', 'to_station_id',
'to_station_name', 'usertype', 'gender', 'birthyear', 'start_time',
'end_time', '01 - Rental Details Rental ID',
'01 - Rental Details Local Start Time',
'01 - Rental Details Local End Time', '01 - Rental Details Bike ID',
'01 - Rental Details Duration In Seconds Uncapped',
'03 - Rental Start Station ID', '03 - Rental Start Station Name',
'02 - Rental End Station ID', '02 - Rental End Station Name',
'User Type', 'Member Gender',
'05 - Member Details Member Birthday Year']
'''


# Read data file
raw_data     =     pd.read_csv("./Data/Merged_raw_data.csv",     sep     =",",
low_memory=False)


#raw_data2 = raw_data


#raw_data = raw_data2[:1000]


#raw_data.to_csv("./Data/Merged_raw_data_small.csv", index=False)


#raw_data    =    pd.read_csv("./Data/Merged_raw_data_small.csv",    sep    =",",
low_memory=False)
```

```
columns  =  ['started_at','starttime','start_time','01  -  Rental  Details  Local 
Start Time','03 - Rental Start Station ID']


def convert_date(x):

    try:

        return pd.to_datetime(x).strftime('%m/%d/%Y %H:%M:%S')

    except:

        return None


raw_data['START TIME'] = pd.to_datetime(0)

for ii in range (len(raw_data)):

  #print(ii)

  for c_ii in range(len(columns)):

    arg = (convert_date(raw_data[columns[c_ii]][ii]))

    if (arg != None):

      raw_data['START TIME'][ii] = arg

      continue


raw_data.to_csv("./Data/Merged_raw_data.csv",index=False)


columns = ['ended_at', 'end_time', '01 - Rental Details Local End Time']


raw_data['STOP TIME'] = pd.to_datetime(0)

for ii in range (len(raw_data)):

  #print(ii)

  for c_ii in range(len(columns)):

    arg = (convert_date(raw_data[columns[c_ii]][ii]))

    if (arg != None):

      raw_data['STOP TIME'][ii] = arg
      continue


raw_data['New_Duration_Seconds'] = 0

for ii in range (len(raw_data)):

                     raw_data['New_Duration_Seconds'][ii]                    =
((pd.to_datetime(raw_data['ended_at'][ii])                                      -
pd.to_datetime(raw_data['started_at'][ii])).seconds)


#Columns = ['usertype','User Type', 'member_casual']
#Columns = ['start_station_id', 'from_station_id', '03 - Rental Start Station
ID']

#Columns = ['end_station_id', 'to_station_id','02 - Rental End Station ID']

#Columns = ['ride_id', 'trip_id', '01 - Rental Details Rental ID']

#Columns = ['gender', 'Member Gender']

#Columns = ['birthyear','05 - Member Details Member Birthday Year']

#Columns = ['bikeid', '01 - Rental Details Bike ID']

#Columns = ['tripduration', '01 - Rental Details Duration In Seconds Uncapped']


Columns = ['bikeid', '01 - Rental Details Bike ID']


raw_data["New_BikeId"] = -1

for ii in range (len(raw_data)):

  for c_ii in range(len(Columns)):

    arg = str(raw_data[Columns[c_ii]][ii])

    if (arg != 'nan'):

      raw_data["New_BikeId"][ii] = arg

      continue


Columns = ['birthyear','05 - Member Details Member Birthday Year']


raw_data["New_BirthYear"] = -1
```

```
for ii in range (len(raw_data)):
  for c_ii in range(len(Columns)):
    arg = str(raw_data[Columns[c_ii]][ii])
    if (arg != 'nan'):
      raw_data["New_BirthYear"][ii] = arg
      continue

Columns = ['ride_id', 'trip_id', '01 - Rental Details Rental ID']

# Male = 1, Female = 0
raw_data["New_Gender"] = -1
for ii in range (len(raw_data)):
  for c_ii in range(len(Columns)):
    arg = str(raw_data[Columns[c_ii]][ii])
    if (arg == 'Male'):
      raw_data["New_Gender"][ii] = 1
      continue
    elif (arg == "Female"):
      raw_data["New_Gender"][ii] = 0
      continue

Columns = ['ride_id', 'trip_id', '01 - Rental Details Rental ID']

raw_data["New_TripID"] = -1
for ii in range (len(raw_data)):
  for c_ii in range(len(Columns)):
    arg = str(raw_data[Columns[c_ii]][ii])
    if (arg != 'nan'):
      raw_data["New_TripID"][ii] = arg
      continue


Columns = ['end_station_id', 'to_station_id','02 - Rental End Station ID']


raw_data["New_End_StationID"] = -1
for ii in range (len(raw_data)):
  for c_ii in range(len(Columns)):
    arg = str(raw_data[Columns[c_ii]][ii])
    if (arg != 'nan'):
      raw_data["New_End_StationID"][ii] = arg
      continue


Columns = ['start_station_id', 'from_station_id', '03 - Rental Start Station ID']

raw_data["New_Start_StationID"] = -1
for ii in range (len(raw_data)):
  for c_ii in range(len(Columns)):
    arg = str(raw_data[Columns[c_ii]][ii])
    if (arg != 'nan'):
      raw_data["New_Start_StationID"][ii] = arg
      continue



raw_data["New_member_type"] = -1
Columns = ['usertype','User Type', 'member_casual']
for ii in range (len(raw_data)):
  #arg = ""
  for c_ii in range(len(Columns)):
    arg = str(raw_data[Columns[c_ii]][ii])
    if (arg == 'member' or 'Subscriber'):
      raw_data["New_member_type"][ii] = 1
      continue
    elif (arg == "Customer" or "casual"):
      raw_data["New_member_type"][ii] = 0
      continue

Columns = ['usertype','User Type', 'member_casual']
Columns += ['start_station_id', 'from_station_id', '03 - Rental Start Station ID']
Columns += ['end_station_id', 'to_station_id','02 - Rental End Station ID']
Columns += ['ride_id', 'trip_id', '01 - Rental Details Rental ID']
Columns += ['gender', 'Member Gender']
Columns += ['birthyear','05 - Member Details Member Birthday Year']
Columns += ['bikeid', '01 - Rental Details Bike ID']
Columns += ['tripduration', '01 - Rental Details Duration In Seconds Uncapped']

Columns

raw_data = raw_data.drop(Columns, axis=1)

# Final Data Writing for future usage
raw_data.to_csv("./Data/MasterTable1.csv", index=False)

Data = pd.read_csv("./Data/MasterTable.csv", sep =",", low_memory=False)

#Adding New Features
# Adding 'date' column
Data['START TIME'] = pd.to_datetime(Data['START TIME'])
Data['STOP TIME']  = pd.to_datetime(Data['STOP TIME'])
```

```python
Data['date']   = Data['START TIME'].dt.date
Data['hour'] = Data['START TIME'].dt.hour

# Adding 'weekend' column

Data['weekend'] = np.where(Data['START TIME'].dt.weekday > 4 , 1, 0)


# Adding 'dayofweek' column

Data['dayofweek'] = Data['START TIME'].dt.day_name().str[:3]


monthMap = {1: 'Jan', 2: 'Feb', 3: 'Mar', 4: 'Apr', 5: 'May', 6: 'Jun',
            7: 'Jul', 8: 'Aug', 9: 'Sep', 10: 'Oct', 11: 'Nov', 12: 'Dec'}


Data['month'] = Data['START TIME'].dt.month

Data['year'] = Data['START TIME'].dt.year


Data['months'] = Data['month'].map(monthMap)


# Adding 'dayofmonth' column
Data['dayofmonth'] = Data['START TIME'].dt.day


# Adding 'dayofyear' column

Data['dayofyear'] = Data['START TIME'].dt.dayofyear


#Trip count

Data['trip_count'] = 1


#Adding Seasons
'''

Typically in Northern Hemisphere Season Starts:


Spring: March 21 - June 20
```

```
Summer: June 21 - September 20

Fall: September 21 - December 20

Winter: December 21 - March 20

'''

Data['season'] = ''


for x in range(Data.year.min() - 1, Data.year.max() + 1):

    # Spring: March 21 - June 20

    Data.loc[(Data['START TIME'] >= str(x) +'-03-21 00:00:00') & (Data['START
TIME'] <= str(x) +'-06-20 23:59:59'),  'season'] = 'spring'

    # Summer: June 21 - September 20

    Data.loc[(Data['START TIME'] >= str(x) +'-06-21 00:00:00') & (Data['START
TIME'] <= str(x) +'-09-20 23:59:59'),  'season'] = 'summer'

    # Fall: September 21 - December 20

    Data.loc[(Data['START TIME'] >= str(x) +'-09-21 00:00:00') & (Data['START
TIME'] <= str(x) +'-12-20 23:59:59'),  'season'] = 'fall'

    # Winter: December 21 - March 20

    Data.loc[(Data['START TIME'] >= str(x) +'-12-21 00:00:00') & (Data['START
TIME'] <= str(x + 1) +'-03-20 23:59:59'),'season'] = 'winter'


# Final Data Writing for future usage
Data.to_csv("./Data/MasterTable2.csv", index=False)


# What are the TOP 10 popular stations to begin a ride at?
df_StartStation_Count         =         pd.DataFrame(Data['FROM         STATION
NAME'].value_counts().reset_index(name = 'Counts'))

df_StartStation_Count.columns = ['FROM STATION NAME', 'Counts']

df_top10 = df_StartStation_Count.head(10)

print(df_top10)


Data['FROM STATION NAME'].value_counts().head(10).index
```

```python
fig = (
    px.bar(
        df_top10,
        x = 'FROM STATION NAME',
        y = 'Counts',
        text_auto = True,
        labels = {"FROM STATION NAME": "From Station Name",},
        width = 1000,
        height = 550
    )
)

fig.update_traces(hoverinfo = "all")

fig.update_layout(
        # update layout with titles
        title = {
            "text": "Top 10 Pickup Location",
            "x": 0.5,
        },
        yaxis_title = "Counts",
        plot_bgcolor = 'black',


)

fig.update_xaxes(categoryorder = "total descending")

# display the figure
```

```
fig.show()


# Average rides from each station


round(Data['FROM      STATION      NAME'].value_counts().reset_index(name      =
'Counts').mean()[0], 2)


round(Data['FROM      STATION      NAME'].value_counts().reset_index(name      =
'Counts').median()[0], 2)


Data.columns


# 'TRIP ID', 'START TIME', 'STOP TIME', 'BIKE ID', 'TRIP DURATION',
# 'FROM STATION ID', 'FROM STATION NAME', 'TO STATION ID',
# 'TO STATION NAME', 'USER TYPE', 'GENDER', 'BIRTH YEAR', 'FROM LATITUDE',
# 'FROM LONGITUDE', 'FROM LOCATION', 'TO LATITUDE', 'TO LONGITUDE',
# 'TO LOCATION', 'date', 'weekend', 'dayofweek', 'month', 'year',
# 'dayofmonth', 'dayofyear', 'trip_count', 'season']


#Per year Top 3 popular stations to begin a ride at?
dfyr = pd.crosstab(Data['FROM STATION NAME'], Data.year)
dfyr = dfyr.reset_index()


dfyr.sort_values(by = dfyr.columns[1:].to_list(), ascending = False, inplace =
True)
dfyr.reset_index(inplace = True)
dfyr = dfyr.iloc[0:3]


dfyr.columns
dfyr.columns.to_list()[0]
dfyr.drop(columns = [dfyr.columns.to_list()[0]], axis = 1, inplace = True)


fig = go.Figure()


for i in dfyr.columns[1:].to_list():


    fig.add_trace(go.Bar(x = dfyr['FROM STATION NAME'],
                          y = dfyr[i],
                         name = str(i),
                         hoverinfo = 'all',
                         #marker_color = np.random.randn(500),
                         #marker_color = 'sandybrown',
                    ))
fig.update_layout(
    xaxis = dict(
        showline = True,
        showgrid = True,
        showticklabels = True,
        linecolor = 'rgb(204, 204, 204)',
        linewidth = 2,
        ticks = 'outside',
            tickfont = dict(
                family = 'Arial',
                size = 12,
                color = 'rgb(82, 82, 82)',
                ),
    ),


    yaxis = dict(
        showgrid = True,

        zeroline = True,

        showline = True,

        showticklabels = True,

    ),


    width = 900,

    height = 450,

    #autosize = True,

    # margin = dict(

    #     autoexpand = True,

    #     l = 100,

    #     r = 20,

    #     t = 110,

    # ),

    showlegend = False,

    plot_bgcolor = 'black',

    title = 'from_station_name Count on Yearly Basis',

)


fig.show()


#  Count  and  calculate  the  percentage  of  round  trips:  meaning  rides
from_station_name and to_station_name must be the same

dfSameStation = pd.DataFrame(Data[Data['FROM STATION NAME'] == Data['TO STATION
NAME']][['FROM STATION NAME', 'TO STATION NAME']])

print(f"Total Round Trips: {round(len(dfSameStation), 2)}")

print(f"Percentage  of  Round  Trips:  {round((len(dfSameStation)  /  len(Data)  *
100), 2)}%")
```

```
# popular routes

dfPopularRoutes = Data[["FROM STATION NAME", "TO STATION NAME"]]

dfPopularRoutes['popular_routes'] = Data['FROM STATION NAME'] + ' TO ' +
Data['TO STATION NAME']

dfPopularRoutes =
pd.DataFrame(dfPopularRoutes['popular_routes'].value_counts().reset_index(name
= 'Counts'))

dfPopularRoutes.sort_values(by = 'Counts', ascending  = False, inplace = True)

dfPopularRoutes = dfPopularRoutes.head(10)


fig = (

    px.bar(

        dfPopularRoutes,

        x = dfPopularRoutes.columns[0],

        y = 'Counts',

        text_auto = True,

        labels = {"index": "Routes",},

        width = 1200,

        height = 650

    )

)


fig.update_traces(hoverinfo = "all", marker_color = np.random.randn(100))


fig.update_layout(

        # update layout with titles

        title = {

            "text": "Top 10 Popular Bike Routes",

            "x": 0.5,

                },
        yaxis_title = "Counts",

        plot_bgcolor = 'black',

        #showlegend = True,

)


fig.update_xaxes(categoryorder = 'total descending', color = '#444')


# display the figure

fig.show()


#Sunburst

'''

'TRIP ID', 'START TIME', 'STOP TIME', 'BIKE ID', 'TRIP DURATION',

'FROM STATION ID', 'FROM STATION NAME', 'TO STATION ID',

'TO STATION NAME', 'USER TYPE', 'GENDER', 'BIRTH YEAR', 'FROM LATITUDE',

'FROM LONGITUDE', 'FROM LOCATION', 'TO LATITUDE', 'TO LONGITUDE',

'TO LOCATION', 'date', 'weekend', 'dayofweek', 'month', 'year',

'dayofmonth', 'dayofyear', 'trip_count', 'season', 'months'

'''


df_compareTrip           =          Data.groupby(['season','months','GENDER'],
as_index=False).agg({'TRIP DURATION': 'mean'})


fig_tripduration       =        px.sunburst(df_compareTrip,       path       =
['season','months','GENDER'],

                        values = 'TRIP DURATION',

                        color = 'TRIP DURATION',

                        color_continuous_scale = 'blues',

                        maxdepth = -1)
```

```
fig_tripduration.update_layout(margin = dict(t = 10, b = 10, r = 10, l = 10),
                               showlegend = False,
                               plot_bgcolor = 'black',
                               )

fig_tripduration.show()

df1 = Data.copy()

'''
'TRIP ID', 'START TIME', 'STOP TIME', 'BIKE ID', 'TRIP DURATION',
'FROM STATION ID', 'FROM STATION NAME', 'TO STATION ID',
'TO STATION NAME', 'USER TYPE', 'GENDER', 'BIRTH YEAR', 'FROM LATITUDE',
'FROM LONGITUDE', 'FROM LOCATION', 'TO LATITUDE', 'TO LONGITUDE',
'TO LOCATION', 'date', 'weekend', 'dayofweek', 'month', 'year',
'dayofmonth', 'dayofyear', 'trip_count', 'season', 'months'

'''


df1 = Data[['hour', 'GENDER', 'months', 'dayofweek', 'USER TYPE',
'weekend', 'season', 'TRIP DURATION', 'trip_count']]

featureColumns = ['hour', 'GENDER', 'months', 'dayofweek', 'USER TYPE',
'weekend', 'season']

df1 = df1.groupby(featureColumns).aggregate({'TRIP DURATION': 'mean',
                                              'trip_count': 'sum',
                                              }).reset_index()

df1['TRIP DURATION'] = df1['TRIP DURATION'].apply(lambda x: round(x, 2))
```

```
df2 = df1.groupby(['hour', 'GENDER', 'dayofweek']).aggregate({'TRIP DURATION':
'mean',

                                            'trip_count': 'mean',

                                            }).reset_index()


fig = px.scatter(df2, x = 'hour', y = 'TRIP DURATION', color = 'GENDER',
facet_col = 'GENDER', facet_row = 'dayofweek', width = 800, height = 1200)

fig.update_layout(title = {"text": "'Day of week' Vs 'Av. Trip Duration' Through
out the Day", "x": 0.5,}, title_font_color = 'blue', showlegend = False,)
#plot_bgcolor = 'black',

fig.update_xaxes(tickmode = 'auto', showgrid = True) #tickmode = 'linear'

fig.update_yaxes(showgrid = True)

fig.show()


sum(Data['GENDER'] == 'Female')


sum(Data['GENDER'] == 'Male')
```