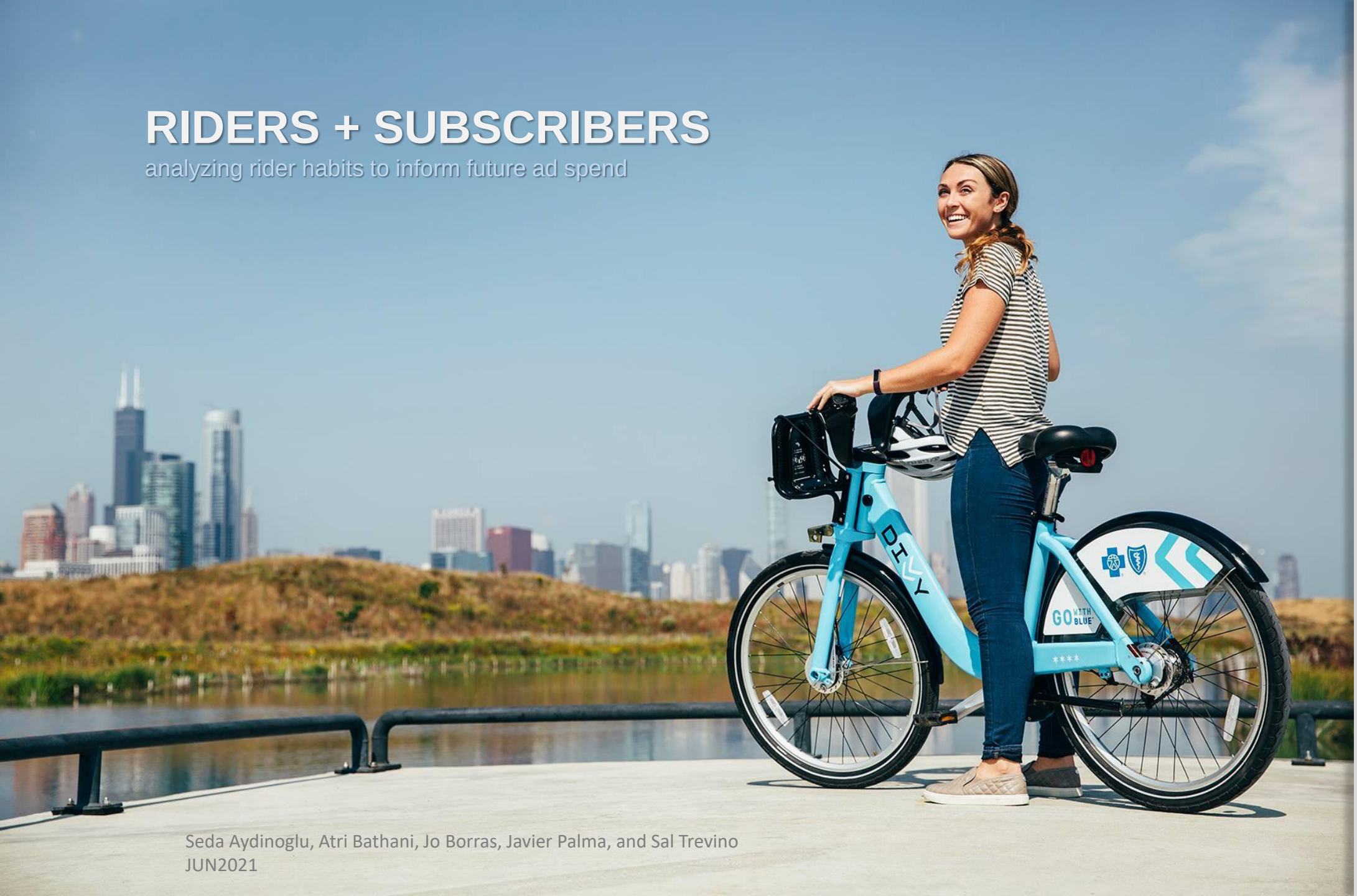
RIDERS + SUBSCRIBERS

analyzing rider habits to inform future ad spend



Seda Aydinoglu, Atri Bathani, Jo Borras, Javier Palma, and Sal Trevino

JUN2021

DIVY

PROJECT OUTLINE

- Project Summary
- Hypothesis
- Gathering the Data
  - Google Trends
  - City of Chicago
  - Baseball Almanac
  - NOAA Weather Data
  - Gas Prices
  - Oil Futures
- Cleaning the Data
  - Installing and Using New Libraries
  - Merging the Data
- Visualizing the Data
- Finding Correlations and Patterns
- Presenting Our Findings
- Libraries Summary
- Data Sources

PROJECT SUMMARY

We imagined ourselves working as analysts for Chicago bike-rental firm, Divvy, we will be analyzing trip and use data from both “as-needed” customer riders and regular subscribers and comparing that data to local weather and gas prices, as well as major sporting events, to see what trends emerge.

The end goal is to establish whether or not there is a correlation between seasonality, fuel, and local sporting events to Divvy rider, in order to create a better model for targeting ad dollars and convincing more people to ride Divvy.

HYPOTHESIS

We began the project trying to find correlations between the number of Divvy rides, local weather, major sporting events, and gas prices. We believed we would be able to show significant correlation and use that to advise the company's ad spend based on oil futures, game schedules, and meteorological almanac data.

Our findings showed that there was a correlation with Divvy use and average daily temp., as well as fuel prices. At first, there did not seem to be a strong correlation between Divvy bike rentals and oil futures prices, however, which indicated that "current conditions" are a stronger driver for Divvy use.

DATA ACQUISITION

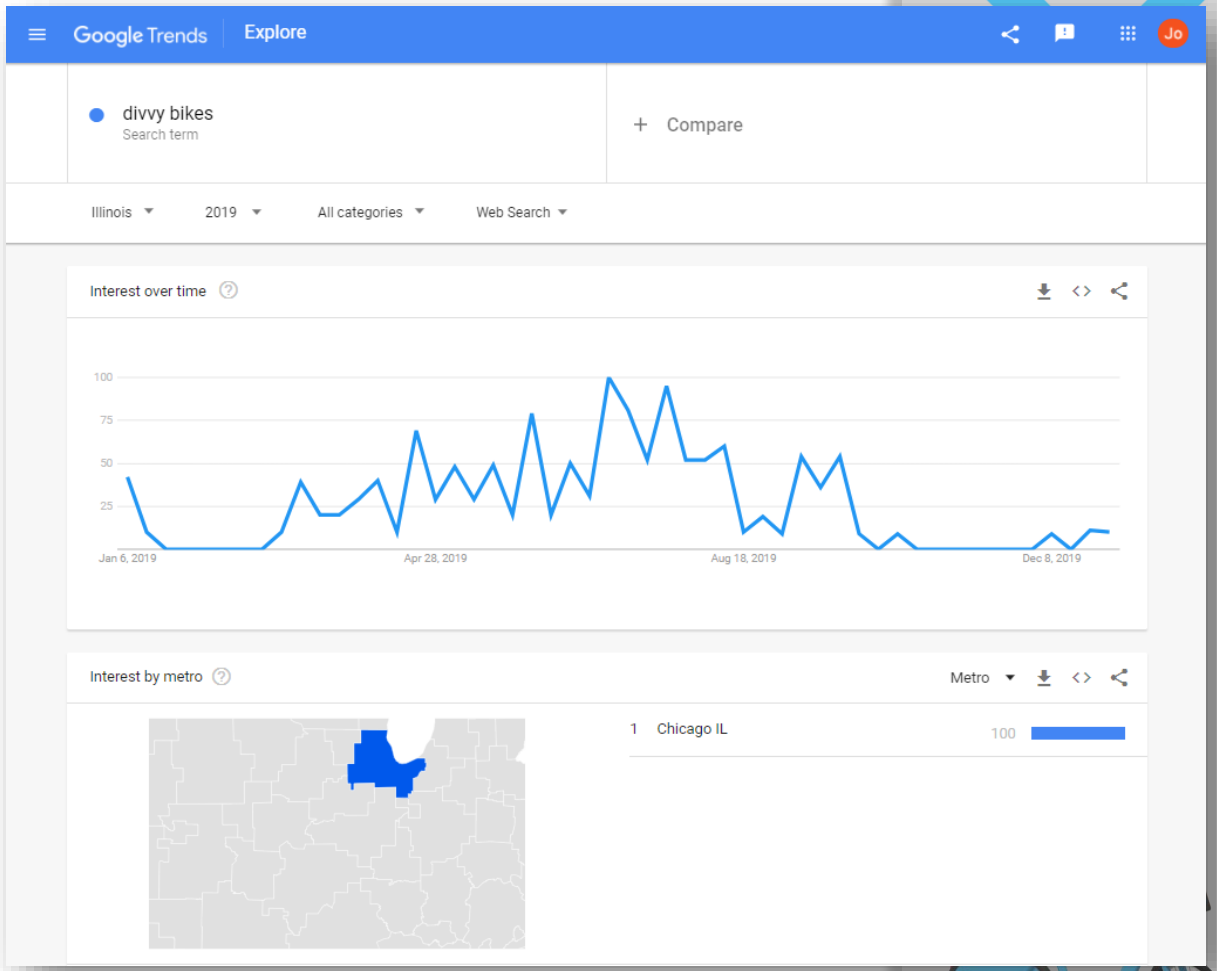
We used publicly-available Divvy rental data from 2019 pulled from the City of Chicago Data Portal, weather data from the NOAA, fuel prices came from the EIA, oil futures data came from Investing.com, broader Divvy search trends from Google Trends as the basis for our data gathering, using Zipfile to import and extract from large data sets.

We also pulled Chicago Cubs and White Sox game data from the Official Baseball Almanac site using read_clipboard, a Python package that extracts data from HTML tables on websites and converts them to dataframes that are useful in Python.

GOOGLE TRENDS

Search interest over the course of 2019 peaks in the summer months, with some additional higher search interest in the late weeks of December and into early January, coinciding with the holidays. A quick scan also shows search interest seems to be flat in the months of February and November.

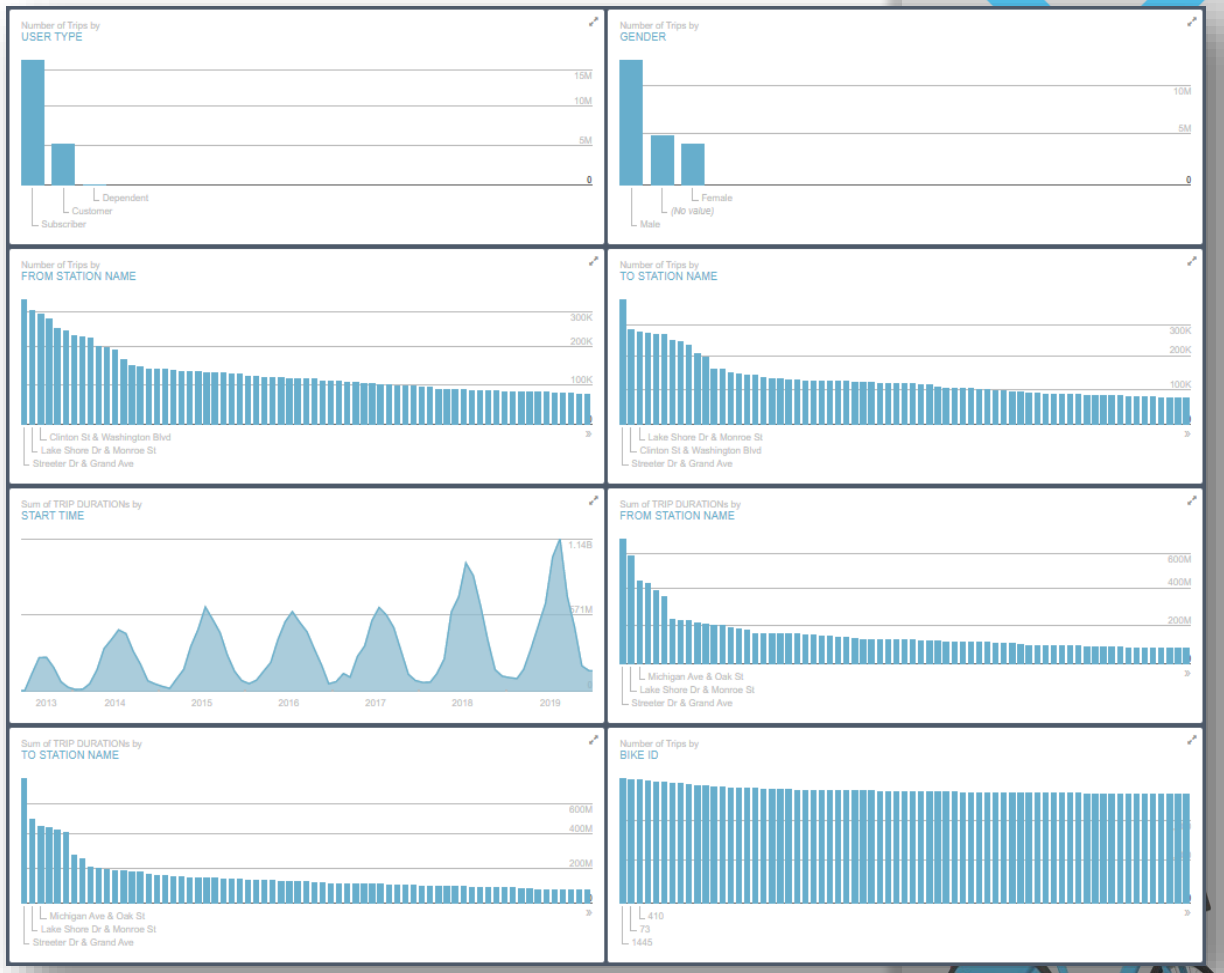
We downloaded this search data as a .csv file, and then read the .csv into a dataframe using Pandas for manipulation and further visualization.



DIVVY CHICAGO DATA

The City of Chicago Data Portal provided highly detailed, usable data for our Divvy project that included demographic information about riders' age and gender, start and end locations, average rental time (defined using "Start Time" and "End Time", and– critically– whether or not the "User Type" is a "Subscriber" or "Customer".

We were able to call the data as an API, then trim away the information we deemed unnecessary by dropping columns and combining certain other variables.



READING ZIPPED DATA

You can see some of the process that went into reading in the Divvy Trips Data in the code, at right, which shows the earliest part of the imports but is significant because of the zipfile library, which allowed us to pull in compressed data and extract the information we needed quickly.

This is one of the “new” libraries we used throughout the project that were not covered in class– **this satisfies one of the requirements of this presentation**.

In [1]:
```
# Initializing imports.
from pathlib import Path
import os
import zipfile
import requests,io
import pandas as pd
import datetime as dt
```

In [2]:
```
#
Divvy_Trips_2019_Q1_link = 'https://divvy-tripdata.s3.amazonaws.com/Divvy_Trips_2019_Q1.zip'
Divvy_Trips_2019_Q2_link = 'https://divvy-tripdata.s3.amazonaws.com/Divvy_Trips_2019_Q2.zip'
Divvy_Trips_2019_Q3_link = 'https://divvy-tripdata.s3.amazonaws.com/Divvy_Trips_2019_Q3.zip'
Divvy_Trips_2019_Q4_link = 'https://divvy-tripdata.s3.amazonaws.com/Divvy_Trips_2019_Q4.zip'
```

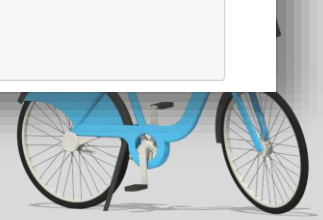
In [3]:
```
#
getfolder1 = requests.get(Divvy_Trips_2019_Q1_link)
getfolder2 = requests.get(Divvy_Trips_2019_Q2_link)
getfolder3 = requests.get(Divvy_Trips_2019_Q3_link)
getfolder4 = requests.get(Divvy_Trips_2019_Q4_link)
```

In [4]:
```
#
unzipfolder1 = zipfile.ZipFile(io.BytesIO(getfolder1.content))
unzipfolder2 = zipfile.ZipFile(io.BytesIO(getfolder2.content))
unzipfolder3 = zipfile.ZipFile(io.BytesIO(getfolder3.content))
unzipfolder4 = zipfile.ZipFile(io.BytesIO(getfolder4.content))
```

In [5]:
```
#
unzipfolder1.extractall('./Q1/Divvy_Trips_2019_Q1')
unzipfolder2.extractall('./Q2/Divvy_Trips_2019_Q2')
unzipfolder3.extractall('./Q3/Divvy_Trips_2019_Q3')
unzipfolder4.extractall('./Q4/Divvy_Trips_2019_Q4')
```

In [6]:
```
#
os.rename("./Q1/Divvy_Trips_2019_Q1/Divvy_Trips_2019_Q1","./Q1/Divvy_Trips_2019_Q1/Divvy_Trips_2019_Q1.csv")
os.rename("./Q2/Divvy_Trips_2019_Q2/Divvy_Trips_2019_Q2","./Q2/Divvy_Trips_2019_Q2/Divvy_Trips_2019_Q2.csv")
```

In [7]:
```
#
Q1_df = pd.read_csv('./Q1/Divvy_Trips_2019_Q1/Divvy_Trips_2019_Q1.csv')
Q2_df = pd.read_csv('./Q2/Divvy_Trips_2019_Q2/Divvy_Trips_2019_Q2.csv')
Q3_df = pd.read_csv('./Q3/Divvy_Trips_2019_Q3/Divvy_Trips_2019_Q3.csv')
Q4_df = pd.read_csv('./Q4/Divvy_Trips_2019_Q4/Divvy_Trips_2019_Q4.csv')
```

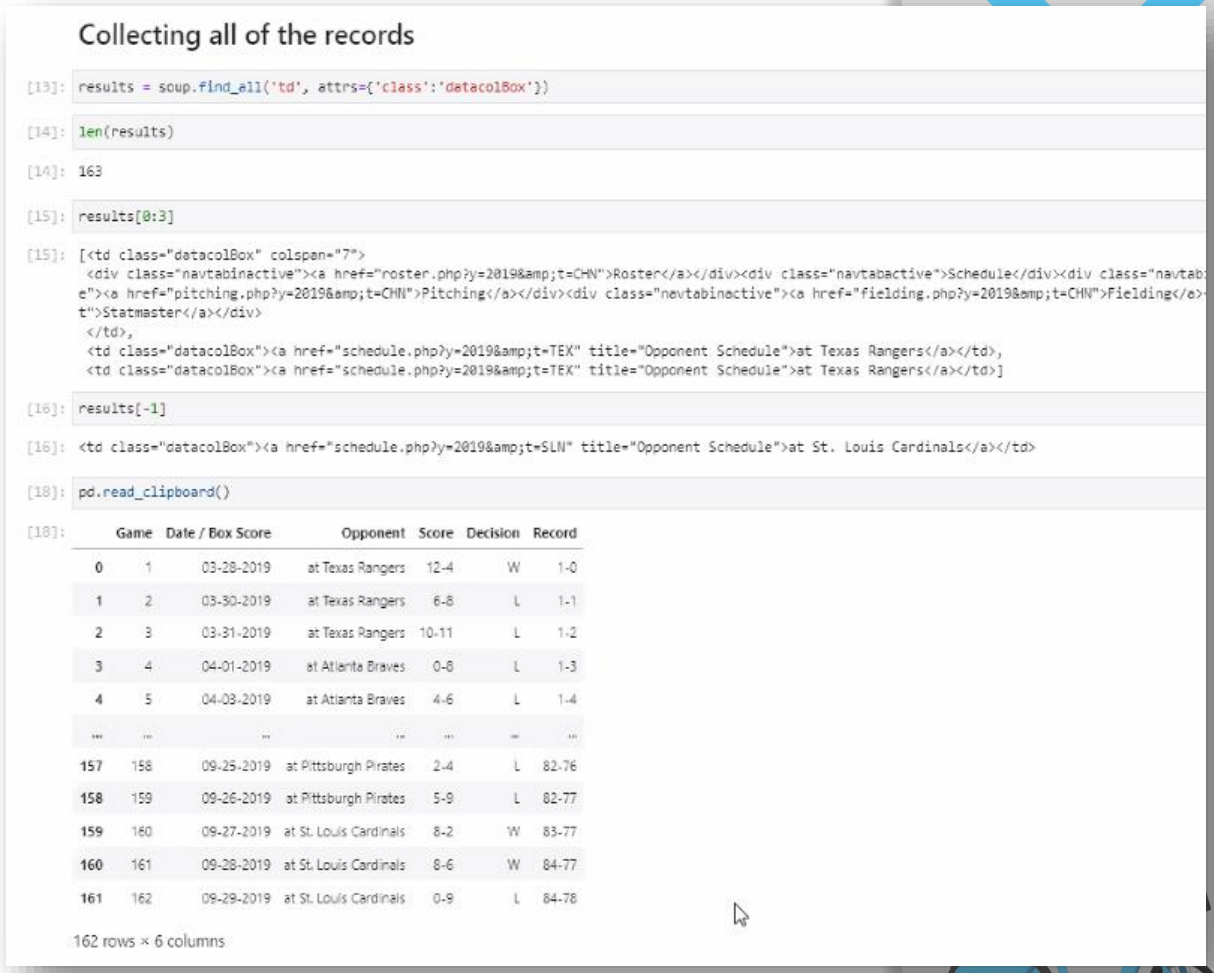


DIVVY

BASEBALL ALMANAC DATA

We pulled the 2019 Cubs and Sox game schedules, as well as location data and scores, using the **pd.read_clipboard** function in Pandas.

We used the regular expression ".str.contains('vs', regex = False)" to determine which games were home games (relevant) vs. away games (irrelevant).



Collecting all of the records

```
[13]: results = soup.find_all('td', attrs={'class':'datacolBox'})

[14]: len(results)

[14]: 163

[15]: results[0:3]

[15]: [<td class="datacolBox" colspan="7">
       <div class="navtabinactive"><a href="roster.php?y=2019&amp;t=CHN">Roster</a></div><div class="navtabactive">Schedule</div><div class="navtab
      e"><a href="pitching.php?y=2019&amp;t=CHN">Pitching</a></div><div class="navtabinactive"><a href="fielding.php?y=2019&amp;t=CHN">Fielding</a>
      t">Statmaster</a></div>
       </td>,
       <td class="datacolBox"><a href="schedule.php?y=2019&amp;t=TEX" title="Opponent Schedule">at Texas Rangers</a></td>,
       <td class="datacolBox"><a href="schedule.php?y=2019&amp;t=TEX" title="Opponent Schedule">at Texas Rangers</a></td>]

[16]: results[-1]

[16]: <td class="datacolBox"><a href="schedule.php?y=2019&amp;t=SLN" title="Opponent Schedule">at St. Louis Cardinals</a></td>

[18]: pd.read_clipboard()
```

[18]:

	Game	Date / Box Score	Opponent	Score	Decision	Record
0	1	03-28-2019	at Texas Rangers	12-4	W	1-0
1	2	03-30-2019	at Texas Rangers	6-8	L	1-1
2	3	03-31-2019	at Texas Rangers	10-11	L	1-2
3	4	04-01-2019	at Atlanta Braves	0-6	L	1-3
4	5	04-03-2019	at Atlanta Braves	4-6	L	1-4
...	...	...	...	...	...	...
157	158	09-25-2019	at Pittsburgh Pirates	2-4	L	82-76
158	159	09-26-2019	at Pittsburgh Pirates	5-9	L	82-77
159	160	09-27-2019	at St. Louis Cardinals	8-2	W	83-77
160	161	09-28-2019	at St. Louis Cardinals	8-6	W	84-77
161	162	09-29-2019	at St. Louis Cardinals	0-9	L	84-78

162 rows × 6 columns

READING WEATHER DATA

We imported the weather data from the Chicago O'Hare weather stations using data compiled by the NOAA (National Oceanic and Atmospheric Association) using the **pd.read_csv** technique used in the exercises.

There was a significant amount of data, including high and low temperatures (tmax, tmin), precipitation types and amount, as well as weather "type" summary. We used the tavg (average daily temperature), and calculated the weekly mean temp using that data.

```
[48]: # Assigning local path and reading Weather csv file.
      file_path = Path('OHAREWEATHERDATA_2019.csv')
      chi_weather = pd.read_csv('OHAREWEATHERDATA_2019.csv', infer_datetime_format = True)

[49]: # Dropping columns that we will not use in final df.
      chi_weather=chi_weather.drop(columns = ['STATION','NAME','AWND', 'PRCP', 'SNOW','SNWD','TMAX','TMIN','WDF2', 'WDF5','WSF2',

[50]: # Converting dates into the datetime format.
      chi_weather['DATE']=pd.to_datetime(chi_weather["DATE"])

[51]: # Creating a new daily date df with the date range of 1/1/19 to 12/31/19.
      weeks_2019_weather = pd.date_range('2019-1-1', '2019-12-31', freq = "D").to_series()

[52]: # Getting the number of the week for each date on daily df.
      weeks_2019_weather = weeks_2019_weather.dt.isocalendar().week

[53]: # Converting new Weeks seies to df, resetting index and dropping additional index column and renaming 'Week' column.
      Weeks_2019_df = weeks_2019_weather.to_frame().reset_index().drop(columns=['index']).rename(columns={'week':'Week'})

[54]: # Changing column name to Week.
      new_names = ['Week']
      Weeks_2019_df.columns = new_names

[55]: # Since 12/30/31 is technically Week #1 of Jan. 2020, we needed to convert to Week 53. This will prevent adding values to W
      Weeks_2019_df.iloc[[363],0]=53

[56]: # Same with this date, since 12/31/31 is technically Week #1 of Jan. 2020, we needed to convert to Week 53. This will preve
      Weeks_2019_df.iloc[[364],0]=53
```

READING GAS PRICES

We read in the weekly gas prices from EIA (Energy Information Administration) data using the **pd.read_excel** function in Pandas, as well as the xlrd library, then deleted the unnecessary columns.

The most reliable (gov't) gas price information was only available as a weekly average, this dictated our data formatting throughout the project and is the reason we organized all the other data by week.

xlrd is one of the "new" libraries we used throughout the project that were not covered in class– **this satisfies one of the requirements of this presentation**.

```
[31]: # Reading in the web address link to access the "Data 1" excel sheet for our Retail Gas Price df.
      xls = pd.ExcelFile('https://www.eia.gov/dnav/pet/xls/PET_PRI_GND_DCUS_YORD_W.xls')
      retail_gas_df = pd.read_excel(xls, 'Data 1')

[32]: # Creating a cols variable to store the columns that we are going to want to delete.
      cols = [2, 3, 4, 5, 6, 7, 8]

[33]: # Deleting unnecessary columns that will not be used in our final df.
      retail_gas_df.drop(retail_gas_df.columns[cols], axis=1, inplace=True)

[ ]: # Deleting first row which has info that is not needed for the df then resetting index count and deleting new index col.
     retail_gas_df = retail_gas_df.drop(labels=0, axis=0).reset_index().drop(columns=['index'])

[ ]: # Creating variable with range of rows that we'd like to delete, then deleting.
     row_x = range(1, 970, 1)
     retail_gas_df = retail_gas_df.drop(retail_gas_df.index[[row_x]]).reset_index().drop(columns=['index'])

[ ]: # Creating variable for second range of rows that we'd like to delete, then deleting.
     row_y = range(54, 129, 1)
     retail_gas_df = retail_gas_df.drop(retail_gas_df.index[[row_y]]).reset_index().drop(columns=['index'])

[ ]: # Deleting rows and cols that will not be used in final df.
     retail_gas_df = retail_gas_df.drop(
         labels=[0,54], axis=0).reset_index().drop(
         columns=['index']).rename(
         columns={'Back to Contents':'Date'
                  , 'Data 1: Chicago Gasoline and Diesel Retail Prices':'Retail_Gas_Prices'
                 })

[ ]: # Converting dates into the datetime format.
     retail_gas_df['Date']=pd.to_datetime(retail_gas_df.Date)

[ ]: # Converting gas prices into decimal.
     retail_gas_df['Retail_Gas_Prices'] = retail_gas_df['Retail_Gas_Prices'].astype(float)
```

DIVY

OIL FUTURES DATA

We imported the oil futures prices using the webbrowser library to get the data directly from the online source. The data was presented as a .csv, which we then converted to a dataframe using the **pd.read_csv** function in Pandas.

This is another one of the “new” libraries we used throughout the project that were not covered in class– **this satisfies one of the requirements of this presentation**.

```
[41]: # Accessing historical futures crude oil prices by opening the browser in a new tab via the Webbrowser Library.
      #******Copy Sorted Ascending Weekly Date Range: 12/17/18 to 12/30/19******
      webbrowser.open_new('https://www.investing.com/commodities/crude-oil-historical-data')

[41]: True

[42]: # Reading in the data that was copied to our clipboard and converting it to a df.
      ## we used the futures_crude_oil_prices = pd.read_clipboard()

[43]: # Exporting the futures oil price df stored on our clipboard to a csv file.
      ## file_path = Path('futures_crude_oil_prices.csv')
      ## futures_crude_oil_prices = futures_crude_oil_prices.to_csv('futures_crude_oil_prices.csv', index = False)

[44]: # Reading in the Futures Crude Oil df that we just created.
      file_path = Path('futures_crude_oil_prices.csv')
      futures_crude_oil_prices = pd.read_csv('futures_crude_oil_prices.csv')

[45]: # Resetting index, renaming columns, and dropping columns that are not needed.
      futures_crude_oil_prices = futures_crude_oil_prices.reset_index().rename(
          columns={'Dec 23, 2018' : 'Date','45.33': 'Crude_Oil_Price'}).drop(columns=['index','45.45','47.00','42.36','2.34M','-0

[46]: # Converting dates into the datetime format.
      futures_crude_oil_prices['Date']=pd.to_datetime(retail_gas_df.Date)

[47]: # Final Futures Crude Oil Price df by week:
      futures_crude_oil_prices
```

[47]:

	Date	Crude_Oil_Price
0	2018-12-31	47.96
1	2019-01-07	51.59
2	2019-01-14	53.80

DIVY

CLEANING THE DATA

As before, we needed to organize all the data into “Weeks” so it could be plotted against the weekly average gas prices. Since there are not baseball games every week, we filled in the missing weeks with “0” games.

Next, we split the Divvy data from “Date” into month and day, then grouped each day by its respective week before concating that data with the fuels df.

```
[ ]: # Filling blanks with 0 int.
     Final_df_05['No_Cubs_Games'] = Final_df_05['No_Cubs_Games'].fillna(0).astype(int)
     Final_df_05['No_Sox_Games'] = Final_df_05['No_Sox_Games'].fillna(0).astype(int)

[*]: # Converted the dates in the 'Dates' column into a DateTime type.
     joined_Divvy_data_split['Date']=pd.to_datetime(joined_Divvy_data_split.Date)

[ ]: # Reformatted to a month/date/year.
     joined_Divvy_data_split['Date']=joined_Divvy_data_split['Date'].dt.strftime('%m/%d/%Y')

[ ]: # Combined the Date df with the oritinal df.
     joined_Divvy_data_all = pd.concat([joined_Divvy_data, joined_Divvy_data_split], axis="columns", join="inner")

[9]: # Appending Q1-Q4 into a single df in chronological order.
     joined_Divvy_data = Q1_df.append([Q2_df,Q3_df,Q4_df], ignore_index=True, sort=False)
```

MERGING THE DATA

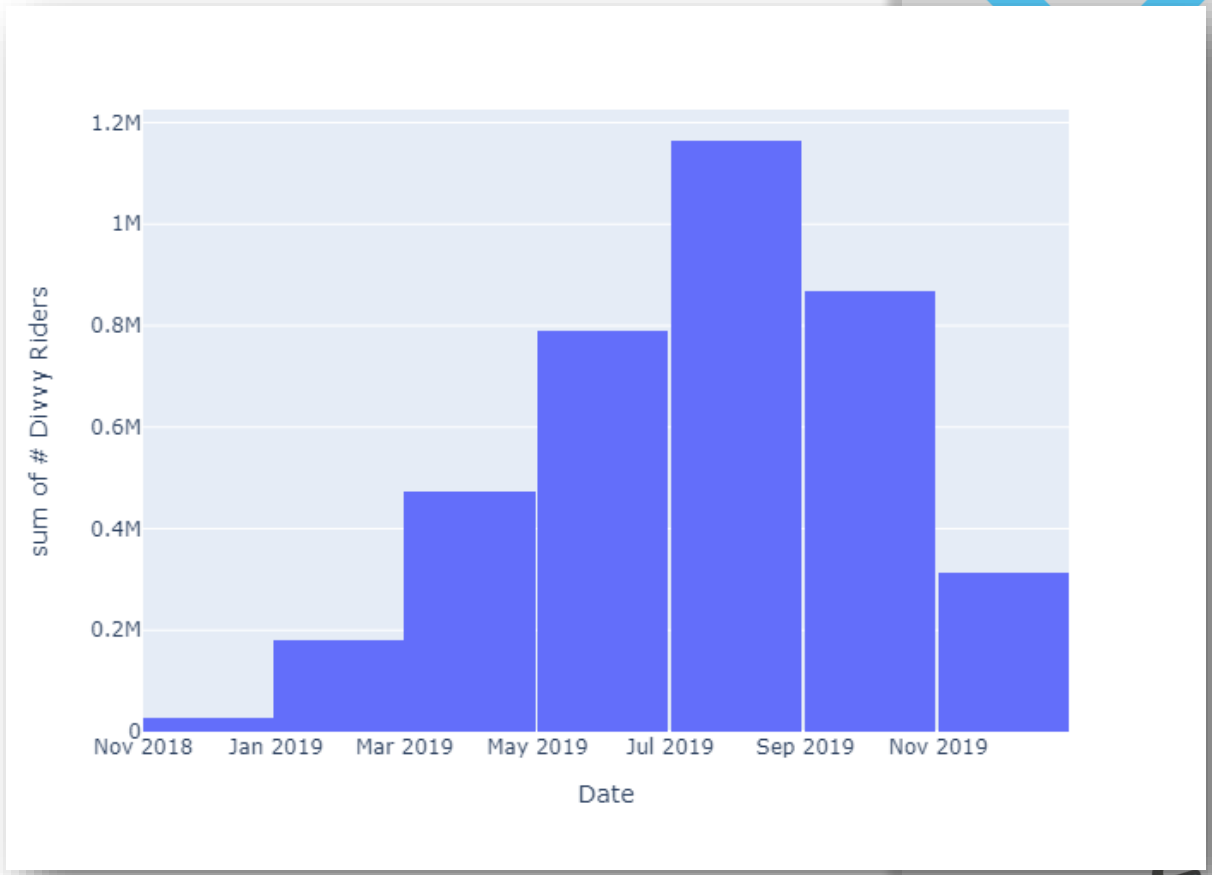
Once we applied similar logic to all the available data, we merged all the dataframes into a one single df that we then used to generate our plots and look for correlation.

```
[ ]: # Merging df by common "Week" column and deleting any extra Date column that is not needed.
Final_df_01 = pd.merge(Weekly_Divvy_Users_df, joined_weather_n_weeks_df, how='outer', left_on = 'Week', right_on = 'Week')
Final_df_02 = pd.merge(Final_df_01, google_trend_Divvy_searches_2019, how='outer', left_on = 'Week', right_on = 'Week')
Final_df_03 = pd.merge(Cubs_2019_Home_Games, Sox_2019_Home_Games, how='outer', left_on = 'Week', right_on = 'Week')
Final_df_04 = pd.merge(Final_df_01, Final_df_03, how='outer', left_on = 'Week', right_on = 'Week')
Final_df_05 = pd.merge(Final_df_04, google_trend_Divvy_searches_2019, how='outer', left_on = 'Week', right_on = 'Week').dro

[ ]: # Filling blanks with 0 int.
Final_df_05['No_Cubs_Games'] = Final_df_05['No_Cubs_Games'].fillna(0).astype(int)
Final_df_05['No_Sox_Games'] = Final_df_05['No_Sox_Games'].fillna(0).astype(int)

[ ]: # Merging gas and oil prices by common "Date" column and rounding decimal to hundredths.
Final_df_06 = pd.merge(retail_gas_df, futures_crude_oil_prices, how='outer', left_on = 'Date', right_on = 'Date')
Final_df_06['Retail_Gas_Prices'] = Final_df_06['Retail_Gas_Prices'].round(2).astype(float)

[ ]: # Combining Petro df with the rest of the df and deleting "Week" column since it's no longer needed.
Final_df_07 = pd.concat([Final_df_05, Final_df_06], axis = 'columns', join = 'outer').drop(columns=['Week'])

[ ]: # Converting Google Search data to a number.
Final_df_07['Divvy_Google_Searches'] = Final_df_07['Divvy_Google_Searches'].astype(int)

[ ]: # Rearranging final df columns.
Final_df_07 = Final_df_07[['Date'
                           , 'Total_2019_Users'
                           , 'Retail_Gas_Prices'
                           , 'Crude_Oil_Price'
                           , 'Avg_Temp'
                           , 'No_Cubs_Games'
                           , 'No_Sox_Games'
                           , 'Divvy_Google_Searches']]
```

DIVVY

VISUALIZING DATA (1)

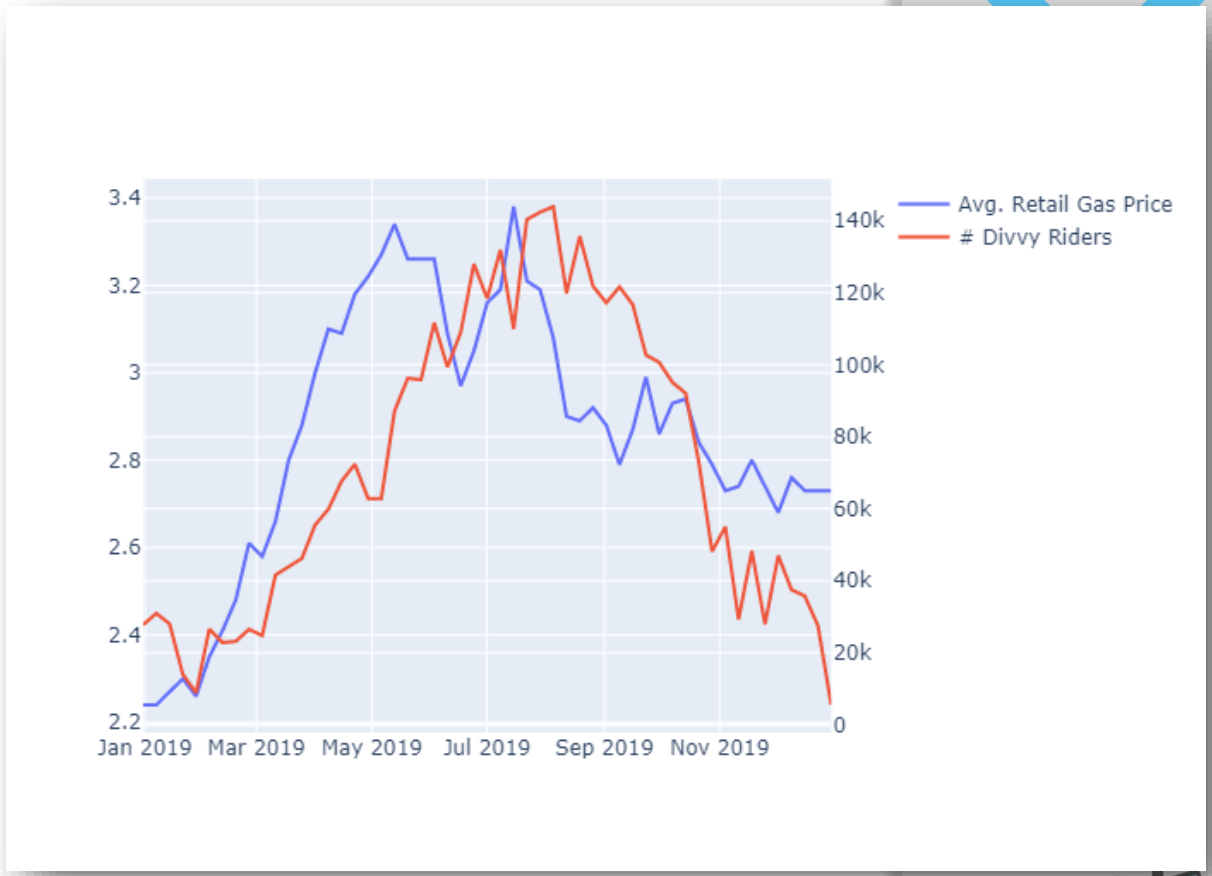
The first plot we did showed the sum total of Divvy riders throughout the course of the year. This followed our assumption that there would be more Divvy use in the warmer summer months than the colder winter months.



VISUALIZING DATA (2)

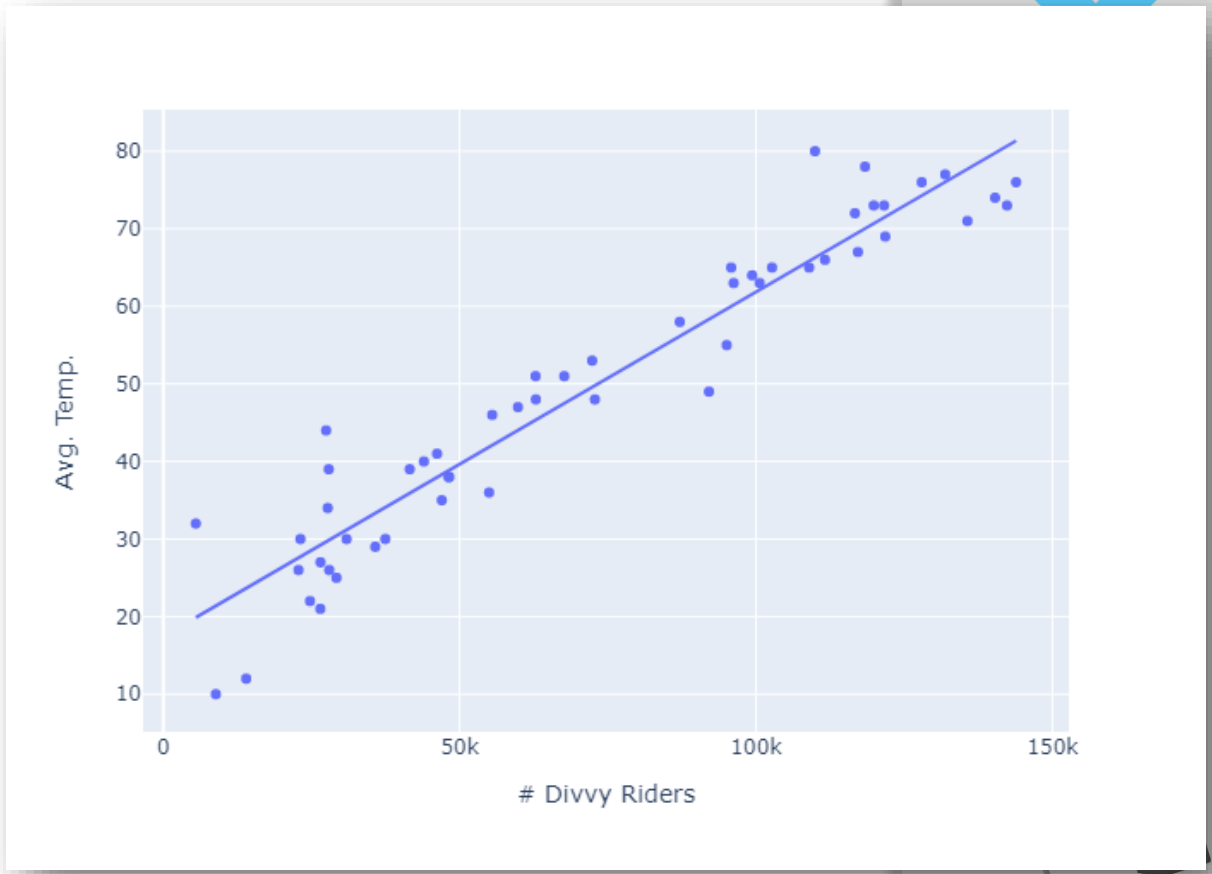
The first comparative plot we did showed the Avg. Retail Gas Price plotted against the number of Divvy Riders. There was a strong correlation here (0.7) as well as a similarly strong correlation between Gas Price and Google Search traffic related to Divvy bikes (0.68).

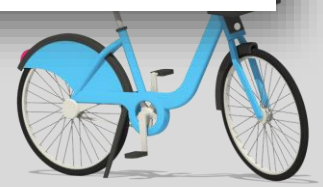
We observed that Divvy ridership, while correlated, lagged Average Gas Price by about four (4) weeks, implying that one or two expensive “fill-ups” would be enough to convince someone to try a Divvy on their commute.



VISUALIZING DATA (3)

Looking at scatter plot for temperature and Divvy use, you can see that very strong correlation play out.

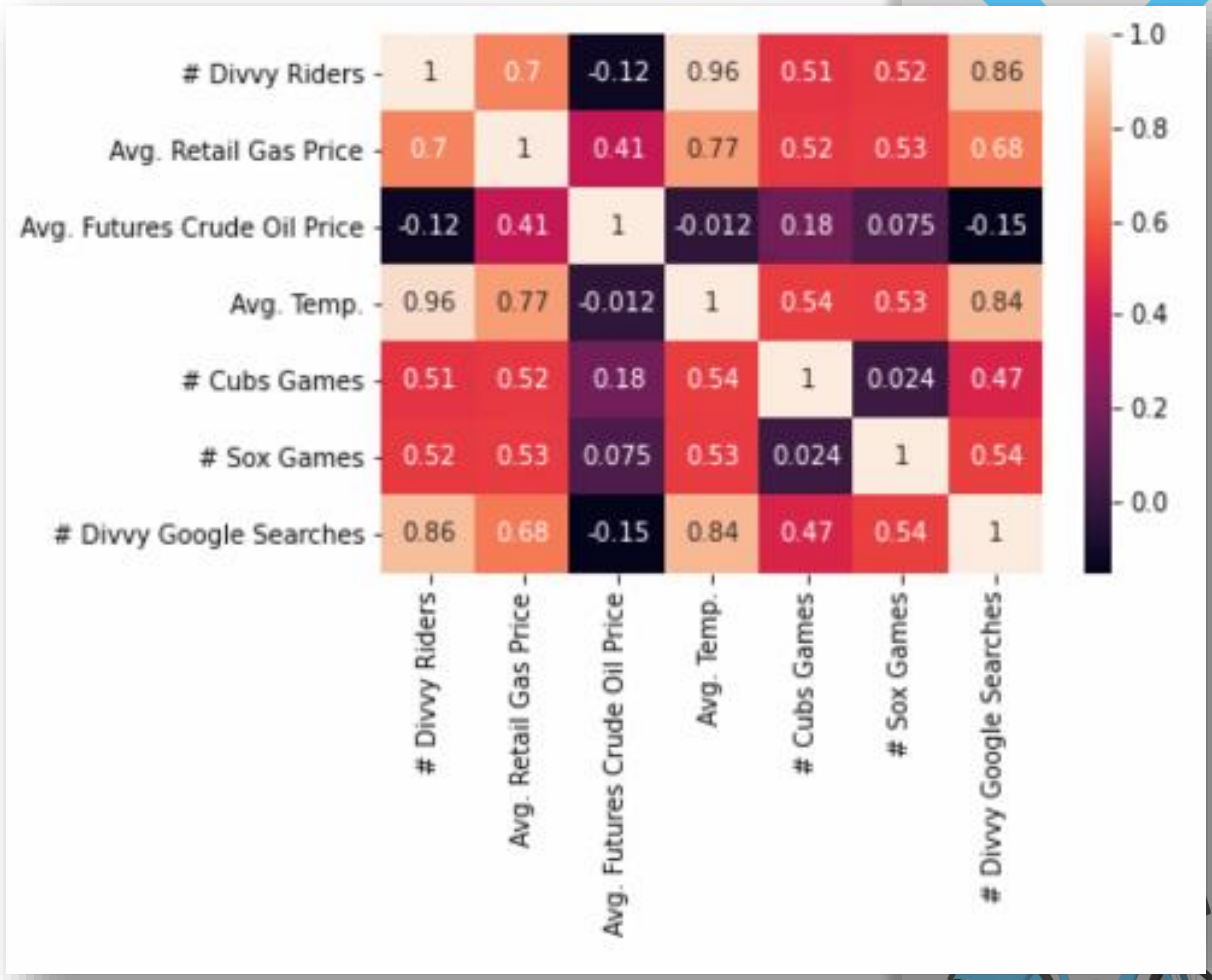




FINDING CORRELATIONS

Interestingly, the number of home baseball games (used as “major sporting events”) did not have a major impact on Divvy ridership (0.51 and 0.52 for Cobs and Sox, respectively).

We took this to indicate that people attended major sporting events using transportation that was already familiar to them. People who took public transport regularly would use public transport to a game, people who used Divvy would used Divvy, etc. This could present an opportunity to increase Divvy use in the future, if it were presented as a better alternative to the riders’ “usual” transportation choice.



PRESENTING FINDINGS

Our original hypothesis was that fuel prices and Divvy rides would be strongly linked, and they were– but, initially, there wasn't a clear correlation between oil futures (one indicator of future fuel prices).

Upon re-examination, we realized that the futures prices reflected prices 90-days out. Peak futures corresponded to peak Divvy use 90-days later, **which means that we actually *can* use futures prices to predict demand**.

We would recommend our ads and maintenance teams use spikes in oil futures pricing as a guide to plan out their bike deployment to commuter-heavy stations and increase web advertising in anticipation of search spikes.

DIVVY

NEW LIBRARIES SUMMARY

By the end of the project, we had used a number of new libraries including zipfile, webbrowser, warnings, and xlrd to import and manage different data.

[1]:
```
# Initializing imports.
from pathlib import Path
import os
import requests,io
import pandas as pd
import datetime as dt
import seaborn as sn
import matplotlib.pyplot as plt
import plotly.express as px
import plotly.graph_objects as go
from plotly.subplots import make_subplots

# New Libraries that have not been used in class.
import zipfile
import webbrowser

# Static image generation requires Kaleido package.
if not os.path.exists("images"):
    os.mkdir("images")
```

[2]:
```
# Assigning web address link to variable.
Divvy_Trips_2019_Q1_link = 'https://divvy-tripdata.s3.amazonaws.com/Divvy_Trips_2019_Q1.zip'
Divvy_Trips_2019_Q2_link = 'https://divvy-tripdata.s3.amazonaws.com/Divvy_Trips_2019_Q2.zip'
Divvy_Trips_2019_Q3_link = 'https://divvy-tripdata.s3.amazonaws.com/Divvy_Trips_2019_Q3.zip'
Divvy_Trips_2019_Q4_link = 'https://divvy-tripdata.s3.amazonaws.com/Divvy_Trips_2019_Q4.zip'
```

[3]:
```
# Getting Zip Folders from Divvy Trip Data Website and saving locally.
getfolder1 = requests.get(Divvy_Trips_2019_Q1_link)
getfolder2 = requests.get(Divvy_Trips_2019_Q2_link)
getfolder3 = requests.get(Divvy_Trips_2019_Q3_link)
getfolder4 = requests.get(Divvy_Trips_2019_Q4_link)
```

[4]:
```
# Unzipping Divvy Q1-Q4 folders with new Zipfile library.
unzipfolder1 = zipfile.ZipFile(io.BytesIO(getfolder1.content))
unzipfolder2 = zipfile.ZipFile(io.BytesIO(getfolder2.content))
unzipfolder3 = zipfile.ZipFile(io.BytesIO(getfolder3.content))
unzipfolder4 = zipfile.ZipFile(io.BytesIO(getfolder4.content))
```

DIVVY

DATA SOURCES

.csv files are stored in our Group Project repo. Below are the links to the data described throughout the project.

Divvy Rental Data
https://data.cityofchicago.org/Transportation/Divvy-Trips/fg6s-gzvg

NOAA Weather Data
https://www.noaa.gov/

EIA Gas Prices
https://www.eia.gov/dnav/pet/hist/LeafHandler.ashx?n=PET&s=EMM_EPM0_PTE_NUS_DPG&f=W

Investing.com Oil Futures Prices
https://www.investing.com/commodities/crude-oil-streaming-chart

Baseball Games Data
https://www.baseball-almanac.com/teamstats/schedule.php?y=2019&t=CHN Cubs
https://www.baseball-almanac.com/teamstats/schedule.php?y=2019&t=CHA Sox