CitiBike Bike Rentals Analysis

This analysis was conducted as a project from Codecademy's Data Engineer Career Path. The course provided the source data and the following prompt to start the project:

> A bike rental company has asked you to create a database to help their analysts understand the effects of weather on bike rentals. You've been given a year of bike rental data from the company and you'll source weather data from the government. You'll need to clean and validate both data sets, design a relational PostgreSQL database to store the data, and develop views for the database to assist the analytics team.

I completed the project in the following steps:

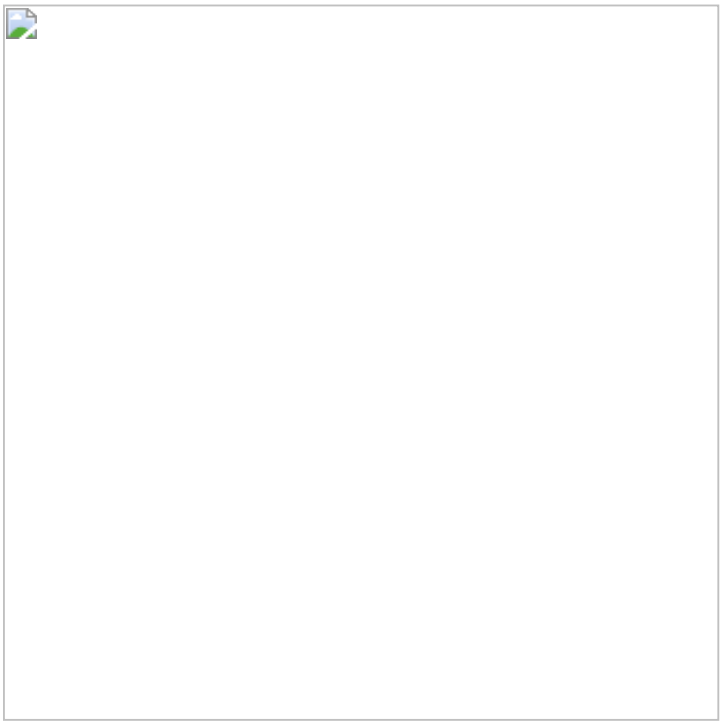
1. Prepare Data and Environment
2. Import, Inspect, Clean, and Export Files
3. Create a Database Schema
4. Import Data and Create Views for Analysis
5. Create Data Visualizations

Access the files from the project here:

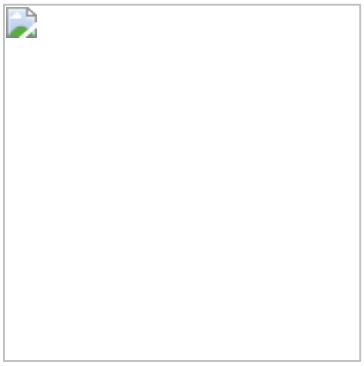
https://drive.google.com/drive/folders/11PMCJSuH7HAm5kFHQoEns1XK9wWOu41k?usp=share_link

(1.) Prepare Data and Environment *VSCode*

Download resources to a shared directory



Setup Jupyter Notebook in VSCode with a virtual environment (.venv) for the project



(2.) Import, Inspect, Clean, and Export files *Python*

Import files

```
import pandas as pd
import numpy as np
import glob

files = glob.glob("/Users/nate/Data_Projects/Bike Rental Portfolio
Project/bike-rental-starter-kit/data/JC-*-citibike-tripdata.csv")
df_list = []
for filename in files:
    data = pd.read_csv(filename)
    df_list.append(data)

bikedata = pd.concat(df_list)

weather = pd.read_csv("/Users/nate/Data_Projects/Bike Rental Portf
olio Project/bike-rental-starter-kit/data/newark_airport_2016.cs
v")
```

Inspect the data

```
pd.set_option('display.max_rows', 1000)
pd.set_option('display.max_columns',504)
pd.set_option('display.width',1000)

# print(bikedata.head(10))
# print(weather.head(10))



#bikedata ----------------------------------
print("len(bikedata): " + str(len(bikedata)))
bike_dup = bikedata.duplicated()
#print(bike_dup.value_counts())

print("number of bikes: " + str(len(bikedata['Bike ID'].unique
```

```
()))))

print("min(start_time): " + str(bikedata['Start Time'].min()))
print("max(start_time): " + str(bikedata['Start Time'].max()))
print("youngest rider: " + str(2016 - bikedata['Birth Year'].max
()) + " years old")
print("oldest rider: " + str(2016 - bikedata['Birth Year'].min())
+ " years old")
print("avg trip duration: " + str(round(bikedata['Trip Duration'].
mean() / 60, 2)) + " minutes")


# The oldest person known to have lived in New Jersey in 2016 was
Adele Dunlap,
# who at the time was the oldest living person in the United State
s. She was born on December 12, 1902,
# and lived in Hunterdon County, New Jersey. In 2016, she turned 1
14 years old. She passed away on
# February 5, 2017, at the age of 114 years and 51 days.


too_old = bikedata[bikedata.birth_year < 1902]
```

Output:

len(bikedata): 247584

number of bikes: 566

min(start_time): 2016-01-01 00:02:52

max(start_time): 2016-12-31 23:44:50

youngest rider: 16.0 years old

oldest rider: 116.0 years old

avg trip duration: 14.76 minutes

Clean the Data

▼ code

```
#adjust column names-----------------------------------------
-----------------------------------------------------------
--
bikedata.columns = bikedata.columns.str.lower().str.replace('
','_')
weather.columns = weather.columns.str.lower()

#get rid of riders that are too old
bikedata = bikedata[bikedata.birth_year >= 1902]
#-----------------------------------------------------------
-----------------------------------------------------------
--



#edit gender column ------------------------------------------
-----------------------------------------------------------
--
def gender(num):
    if num == 0:
        return 'U'
    elif num == 1:
        return 'M'
    else:
        return 'F'

bikedata.gender = bikedata.gender.astype(int).apply(gender)
#-----------------------------------------------------------
-----------------------------------------------------------
--




# change birth year to int -----------------------------------
-----------------------------------------------------------
--
bikedata.birth_year = bikedata.birth_year.astype(int)
```

```
#-------------------------------------------------------------
-------------------------------------------------------------
--



#create stations df ------------------------------------------
-------------------------------------------------------------
--
bikedata = bikedata.rename(columns = {'start_station_latitude':
'start_latitude',
                                      'start_station_longitud
e': 'start_longitude',
                                      'end_station_latitude':
'end_latitude',
                                      'end_station_longitude':
'end_longitude'})

start_stations = bikedata[['start_station_id', 'start_station_n
ame', 'start_latitude','start_longitude']].drop_duplicates()
start_stations.columns = start_stations.columns.str.replace('st
art_', '')

end_stations = bikedata[['end_station_id', 'end_station_name',
'end_latitude','end_longitude']].drop_duplicates()
end_stations.columns = end_stations.columns.str.replace('end_',
'')

stations = pd.concat([start_stations, end_stations]).drop_dupli
cates().reset_index(drop=True)
stations.columns = stations.columns.str.replace('station_', '')
#-------------------------------------------------------------
-------------------------------------------------------------
--



# create trip_data df -----------------------------------------
-------------------------------------------------------------
```

```
--
trip_data = bikedata[['bike_id', 'start_station_id', 'start_tim
e', 'end_station_id', 'stop_time', 'trip_duration', 'user_typ
e', 'birth_year', 'gender']]
#-------------------------------------------------------------
-------------------------------------------------------------
--



# clean weather table ---------------------------------------
-------------------------------------------------------------
--

for column in weather.columns:
    null_count = weather[column].isna().sum()
    print(f"{column}: {null_count} null values")

weather = weather.drop(columns = ['pgtm', 'tsun'])
weather.columns.str.lower()
weather = weather.rename(columns = {'awnd': 'awnd_mph', 'snwd':
'snw_dpth'})
weather = weather.drop(columns = ['station', 'name'])



#-------------------------------------------------------------
-------------------------------------------------------------
--

#return stations, trip_data, weather
```

Write to CSV

```
trip_data.to_csv('trip_data.csv', index=False)
weather.to_csv('weather.csv', index=False)
stations.to_csv('stations.csv', index=False)
```

(3.) Create a Database Schema *Lucid Chart*

(4.) Import Data and Create Views for Analysis

PostgreSQL

Import Data

First I imported the files I had cleaned with Python from my project folder:

▼ import files SQL statements

```
-- import stations.csv into the stations table
COPY stations(
  id,
  name,
  latitude,
  longitude
)
FROM '/Users/nate/Data_Projects/Bike Rental Portfolio Project/s
tations.csv'
DELIMITER ','
CSV HEADER;

-- import weather.csv into the weather table
COPY weather(
  date,
  awnd_mph,
  prcp,
  snow,
  snw_dpth,
  tavg,
  tmax,
  tmin,
  wdf2,
  wdf5,
  wsf2,
  wsf5
)
FROM '/Users/nate/Data_Projects/Bike Rental Portfolio Project/w
eather.csv'
DELIMITER ','
CSV HEADER;

-- import trip_data.csv into the trip_data table
```

```
COPY trip_data(
  trip_id,
  bike_id,
  start_station_id,
  start_time,
  end_station_id,
  stop_time,
  trip_duration,
  user_type,
  birth_year,
  gender
)
FROM '/Users/nate/Data_Projects/Bike Rental Portfolio Project/t
rip_data.csv'
DELIMITER ','
CSV HEADER;
```

Create Views

Next, I created views to help analysis of the data.

▼ avg_rides_hourly view

```
CREATE VIEW avg_rides_hourly AS (

WITH
hours AS (
  SELECT DISTINCT
    TO_CHAR(start_time, 'HH24') AS hour
    FROM trip_data
  ORDER BY 1
  ),

assign_trips AS (
  SELECT
    TO_CHAR(t.start_time, 'MM/DD/YYYY') AS date,
```

```
    h.hour,
    t.trip_id
  FROM trip_data t
  JOIN hours h ON h.hour = TO_CHAR(t.start_time, 'HH24')
  ORDER BY 1
  ),

aggr_trips AS (
  SELECT
     date,
     hour,
     COUNT(trip_id)
  FROM assign_trips
  GROUP BY date, hour
  ORDER BY date, hour
  )

SELECT DISTINCT
    hour,
  ROUND(AVG(count),0) AS avg_rides
FROM aggr_trips
GROUP BY hour
ORDER BY hour

);
```

avg_rides_hourly gives the average amount of trips made at each hour of the day over the past year. Analysts can draw from this view to determine high volume points throughout the day.

▼ daily_trips_weather view

```
CREATE VIEW daily_trips_weather AS (
  SELECT DISTINCT
    w.date,
    TO_CHAR(w.date, 'Day') AS weekday,
    COUNT(t.trip_id) OVER (
      PARTITION BY date
      ) AS trips,
```

```
    ROUND(AVG(t.trip_duration) OVER (
      PARTITION BY date
        ),2) AS avg_trip_duration,
    w.awnd_mph,
    w.prcp,
    w.snow,
    w.snw_dpth,
    w.tavg,
    w.tmin,
    w.tmax,
    w.wdf2,
    w.wdf5,
    w.wsf2,
    w.wsf5
  FROM trip_data t
  JOIN weather w ON DATE(w.date) = DATE(t.start_time)
  ORDER BY 1
);
```

daily_trips_weather gives general information about the weather alongside the amount and avg duration of trips that day.

▼ travel_directions view

```
CREATE VIEW travel_directions AS (

WITH coords AS (
    SELECT
        t.trip_id,
        d.latitude AS lat1,
        d.longitude AS lon1,
        a.latitude AS lat2,
        a.longitude AS lon2
    FROM trip_data t
    JOIN stations d ON t.start_station_id = d.id
    JOIN stations a ON t.end_station_id = a.id
),

directions AS (
```

```
  SELECT
      trip_id,
      DEGREES(
          ATAN2(
              SIN(RADIANS(lon2 - lon1)) * COS(RADIANS(lat2)),
              COS(RADIANS(lat1)) * SIN(RADIANS(lat2)) -
              SIN(RADIANS(lat1)) * COS(RADIANS(lat2)) * COS(RAD
IANS(lon2 - lon1))
          )
      ) AS bearing_degrees
  FROM coords
)

SELECT
    trip_id,
  bearing_degrees,
    CASE
    WHEN bearing_degrees >= 337.5 OR bearing_degrees < 22.5 THE
N 'N'
    WHEN bearing_degrees >= 22.5 AND bearing_degrees < 67.5 THE
N 'NE'
    WHEN bearing_degrees >= 67.5 AND bearing_degrees < 112.5 TH
EN 'E'
    WHEN bearing_degrees >= 112.5 AND bearing_degrees < 157.5 T
HEN 'SE'
    WHEN bearing_degrees >= 157.5 AND bearing_degrees < 202.5 T
HEN 'S'
    WHEN bearing_degrees >= 202.5 AND bearing_degrees < 247.5 T
HEN 'SW'
    WHEN bearing_degrees >= 247.5 AND bearing_degrees < 292.5 T
HEN 'W'
    WHEN bearing_degrees >= 292.5 AND bearing_degrees < 337.5 T
HEN 'NW'
  END AS direction
FROM directions

);
```

travel_directions assigns the direction that the customer traveled in based on the locations of the starting and ending stations.

▼ travel_vs_wind view

```
CREATE VIEW travel_vs_wind AS (

WITH cte AS (
  SELECT
    w.date,
    d.trip_id,
    concat(s.name, ' to ', e.name) AS trip,
    t.trip_duration,
    d.direction as trip_direction,
    w.wdf2,
    w.wsf2,
    CASE
      WHEN w.wdf2 >= 337.5 OR w.wdf2 < 22.5 THEN 'N'
      WHEN w.wdf2 >= 22.5 AND w.wdf2 < 67.5 THEN 'NE'
      WHEN w.wdf2 >= 67.5 AND w.wdf2 < 112.5 THEN 'E'
      WHEN w.wdf2 >= 112.5 AND w.wdf2 < 157.5 THEN 'SE'
      WHEN w.wdf2 >= 157.5 AND w.wdf2 < 202.5 THEN 'S'
      WHEN w.wdf2 >= 202.5 AND w.wdf2 < 247.5 THEN 'SW'
      WHEN w.wdf2 >= 247.5 AND w.wdf2 < 292.5 THEN 'W'
      WHEN w.wdf2 >= 292.5 AND w.wdf2 < 337.5 THEN 'NW'
    END AS wdf2_direction,
    w.wdf5,
    w.wsf5,
    CASE
      WHEN w.wdf5 >= 337.5 OR w.wdf5 < 22.5 THEN 'N'
      WHEN w.wdf5 >= 22.5 AND w.wdf5 < 67.5 THEN 'NE'
      WHEN w.wdf5 >= 67.5 AND w.wdf5 < 112.5 THEN 'E'
      WHEN w.wdf5 >= 112.5 AND w.wdf5 < 157.5 THEN 'SE'
      WHEN w.wdf5 >= 157.5 AND w.wdf5 < 202.5 THEN 'S'
      WHEN w.wdf5 >= 202.5 AND w.wdf5 < 247.5 THEN 'SW'
      WHEN w.wdf5 >= 247.5 AND w.wdf5 < 292.5 THEN 'W'
      WHEN w.wdf5 >= 292.5 AND w.wdf5 < 337.5 THEN 'NW'
    END AS wdf5_direction
  FROM travel_directions d
```

```
  JOIN trip_data t ON t.trip_id = d.trip_id
  JOIN weather w ON DATE(w.date) = DATE(t.start_time)
  JOIN stations s ON t.start_station_id = s.id
  JOIN stations e ON t.end_station_id = e.id
  )

SELECT
  date,
  trip_id,
  trip,
  trip_duration,
  trip_direction,
  wdf2,
  wsf2,
  wdf2_direction,
  wdf5,
  wsf5,
  wdf5_direction,
  CASE
    WHEN
        (trip_direction = 'N' AND (wdf2_direction = 'S' OR wdf5
_direction = 'S'))
  OR    (trip_direction = 'NW' AND (wdf2_direction = 'SE' OR wd
f5_direction = 'SE'))
  OR    (trip_direction = 'W' AND (wdf2_direction = 'E' OR wdf5
_direction = 'E'))
  OR    (trip_direction = 'SW' AND (wdf2_direction = 'NE' OR wd
f5_direction = 'NE'))
  OR    (trip_direction = 'S' AND (wdf2_direction = 'N' OR wdf5
_direction = 'N'))
  OR    (trip_direction = 'SE' AND (wdf2_direction = 'NW' OR wd
f5_direction = 'NW'))
  OR    (trip_direction = 'E' AND (wdf2_direction = 'W' OR wdf5
_direction = 'W'))
  OR    (trip_direction = 'NE' AND (wdf2_direction = 'SW' OR wd
f5_direction = 'SW'))
  THEN 1
  ELSE 0
```

```
  END AS against_wind
FROM cte
WHERE trip_duration <= 28800
ORDER BY date
);
```

travel_vs_wind is a view which allows analysts to see any affects the direction of wind might have made on the customer's trip.

▼ weather_trips_and_duration

```
CREATE VIEW weather_trips_and_duration AS (
WITH cte AS (
  SELECT DISTINCT
    w.date,
    t.trip_id,
    t.trip_duration,
    CASE
      WHEN w.prcp > 0
      THEN 1
      ELSE 0
    END AS is_rain,
    CASE
      WHEN (w.snow > 0 OR w.snw_dpth > 2)
      THEN 1
      ELSE 0
    END AS is_snow,
    CASE
      WHEN (w.wsf2 > 20 OR w.wsf5 > 20)
      THEN 1
      ELSE 0
    END AS is_windy
  FROM trip_data t
  JOIN weather w ON DATE(w.date) = DATE(t.start_time)
  ORDER BY 1
  ),

clear AS (
  SELECT
```

```
    date,
    COUNT(trip_id) AS ct_trips,
    ROUND(AVG(trip_duration),2) AS avg_duration
    FROM cte
  WHERE (is_rain = 0 AND is_snow = 0 AND is_windy = 0)
  GROUP BY date
  ORDER BY date
    ),

rain AS (
  SELECT
    date,
    COUNT(trip_id) AS ct_trips,
    ROUND(AVG(trip_duration),2) AS avg_duration
    FROM cte
  WHERE is_rain = 1
  GROUP BY date
  ORDER BY date
    ),

snow AS (
  SELECT
    date,
    COUNT(trip_id) AS ct_trips,
    ROUND(AVG(trip_duration),2) AS avg_duration
    FROM cte
  WHERE is_snow = 1
  GROUP BY date
  ORDER BY date
    ),

windy AS (
  SELECT
    date,
    COUNT(trip_id) AS ct_trips,
    ROUND(AVG(trip_duration),2) AS avg_duration
    FROM cte
  WHERE is_windy = 1
```

```
  GROUP BY date
  ORDER BY date
    )

SELECT
    ROUND(AVG(clear.ct_trips),2) AS avg_clear_trips,
  ROUND(AVG(clear.avg_duration),2) AS avg_clear_duration,
  ROUND(AVG(rain.ct_trips),2) AS avg_rain_trips,
  ROUND(AVG(rain.avg_duration),2) AS avg_rain_duration,
  ROUND(AVG(snow.ct_trips),2) AS avg_snow_trips,
  ROUND(AVG(snow.avg_duration),2) AS avg_snow_duration,
  ROUND(AVG(windy.ct_trips),2) AS avg_windy_trips,
  ROUND(AVG(windy.avg_duration),2) AS avg_windy_duration
FROM clear, rain, snow, windy

);
```

weather_trips_and_duration is a view used to compare the behavior of customers based on the weather on a given day.

Export Views as CSV

Finally, I exported these back to my project folder.

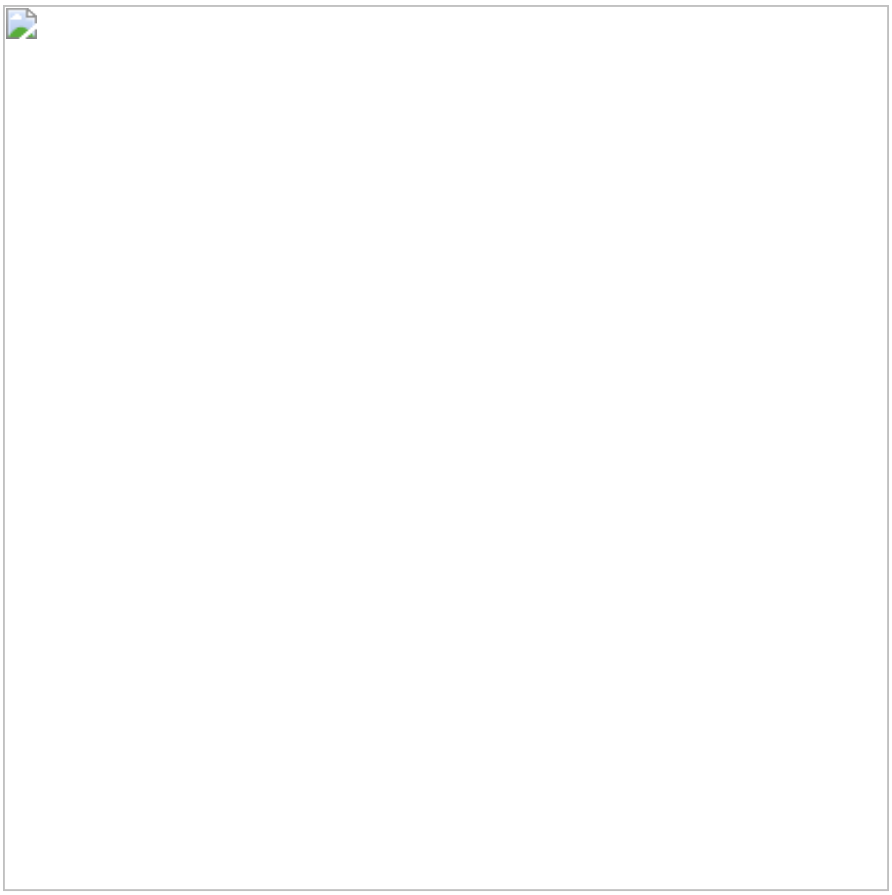
```
COPY (SELECT * FROM avg_rides_hourly) TO '/Users/nate/Data_Project
s/Bike Rental Portfolio Project/avg_rides_hourly.csv' DELIMITER
',' CSV HEADER;
COPY (SELECT * FROM daily_trips_weather) TO '/Users/nate/Data_Proj
ects/Bike Rental Portfolio Project/daily_trips_weather.csv' DELIMI
TER ',' CSV HEADER;
COPY (SELECT * FROM travel_directions) TO '/Users/nate/Data_Projec
ts/Bike Rental Portfolio Project/travel_directions.csv' DELIMITER
',' CSV HEADER;
COPY (SELECT * FROM travel_vs_wind) TO '/Users/nate/Data_Projects/
Bike Rental Portfolio Project/travel_vs_wind.csv' DELIMITER ',' CS
V HEADER;
```

```
COPY (SELECT * FROM weather_trips_and_duration) TO '/Users/nate/Da
ta_Projects/Bike Rental Portfolio Project/weather_trips_and_durati
on.csv' DELIMITER ',' CSV HEADER;
```

(5.) Create Data Visualizations *Tableau Public*

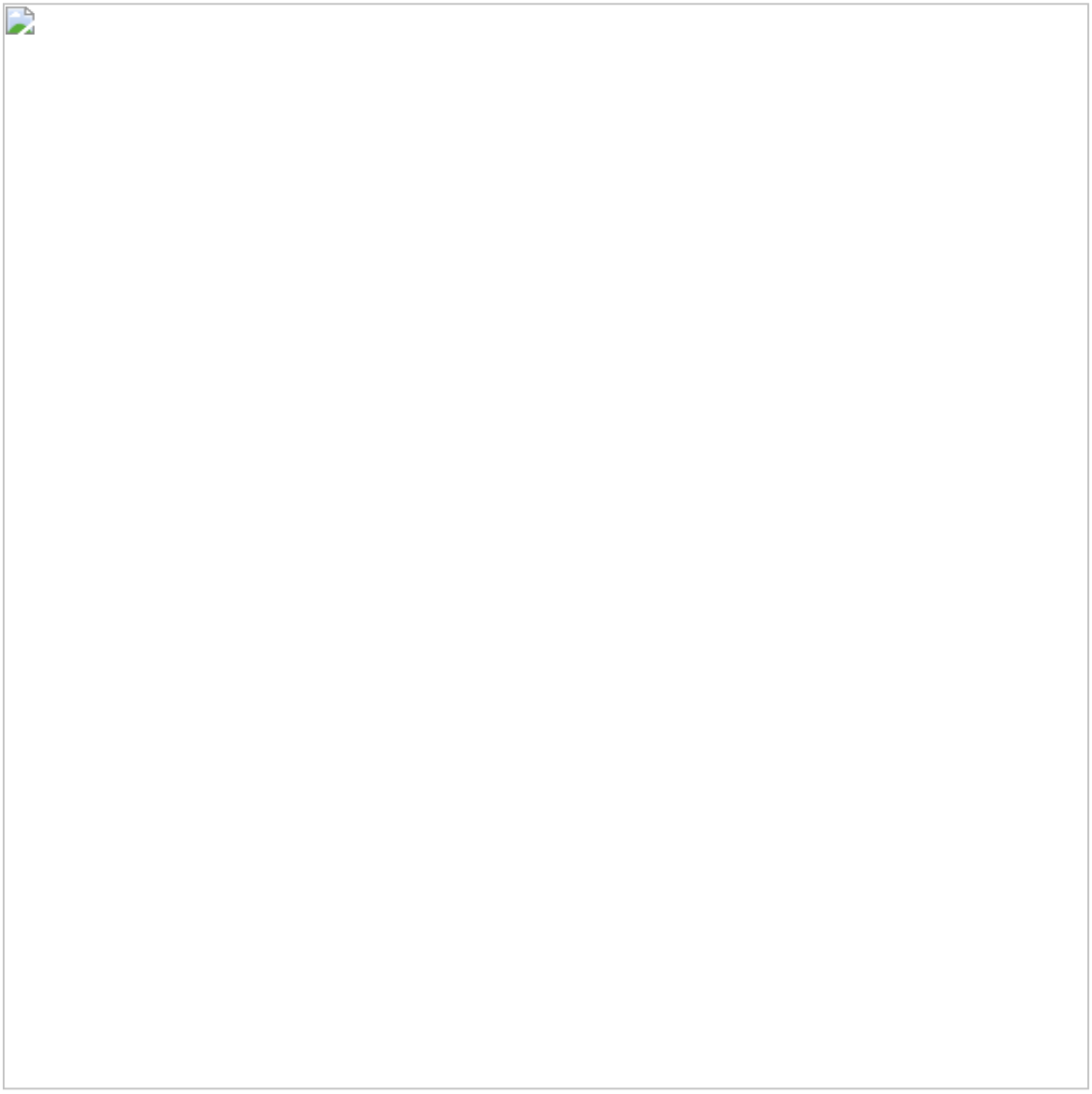
I wanted to create some quick visualizations just to put the data to use so I put together two simple visuals with Tableau:

Using a top 10 filter and counting the records we can see what trips were most popular in 2016.

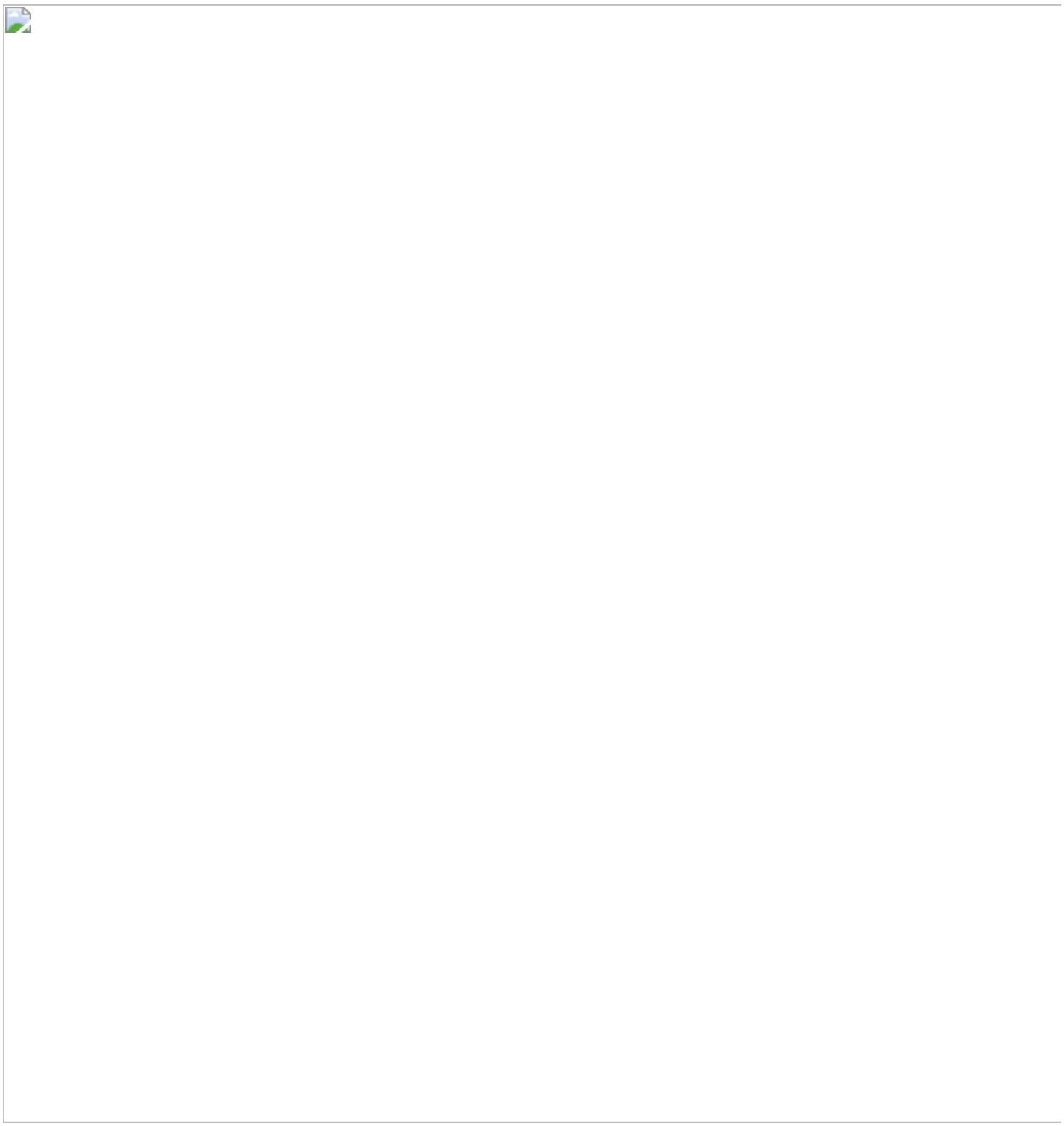


My next idea was to use a visual to see if there was a significant affect on ride duration if riders were traveling against strong oncoming wind. Unfortunately, it didn't seem like

there was any affect. In fact, the average trip duration was longer when there was no oncoming wind.



Lastly, I used the weather_trips_and_duration view to visualize the effect that the weather has on the number of rides daily and their average duration.
From the graph you can see that customers are far less likely to ride in the rain or snow than they are on clear or windy days. Unlike the graph above, it does seem here that wind has a large effect on ride duration, regardless of wind being head-on or not.



This CitiBike project analyzed a year's worth of bike rental data to understand the impact of weather on riding patterns. Using Python for data processing and PostgreSQL for database management, I conducted a comprehensive analysis of the dataset. Key findings include identifying the most popular bike trips in 2016 and, the effect of

headwinds on trip duration, and the overall effect on weather on daily ride activity. This was a great exercise to practice my skills in preparing data for analysis, and I'm looking forward to what else Codecademy has to offer in its Data Engineering Career Path.