Final Project

May 1, 2023

0.0.1 Part 1: Data Preprocessing

[1]:
```
import os
import re
import csv
import glob
import folium
import zipfile
import sqlite3

import requests
import pyarrow
import pandas as pd
import numpy as np
import seaborn as sns
import geopandas as gpd
from scipy import stats
from datetime import datetime
from bs4 import BeautifulSoup
import matplotlib.pyplot as plt
from folium.plugins import HeatMap
from sqlalchemy import create_engine, Column, Integer, String, Float, Date,␣
 ↪Time, DateTime
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy.orm import sessionmaker
from sqlalchemy.sql import text
```

0.1 Part 1: Data Preprocessing

0.1.1 Yellow Taxi dataset

Downloading

[2]:
```
def fetch_yellow_taxi_links(base_url):
    """
    Fetches all links to the yellow taxi data from the given base URL.
```

```
    Args:
        base_url (str): The URL to fetch the links from.

    Returns:
        list: A list of all links to the yellow taxi data.
    """
    resp = requests.get(base_url)
    resp.raise_for_status()
    soup = BeautifulSoup(resp.text, 'html.parser')
    # Select all <a> elements with href attribute containing 'yellow_tripdata'
    # and either a '.parquet' or '.zip' extension.
    links = soup.find_all('a', href=re.compile(r'^(?=.*yellow_tripdata)(?=.
↪*(\d{4}-\d{2}\.parquet|\.zip)).*$'))
    return links
```

[3]:
```
def fetch_taxi_data(links, start_date, end_date, retrieved_files_dir):
    """
    Downloads and extracts the yellow taxi data within the specified date range.

    Args:
        links (list): A list of links to the yellow taxi data.
        start_date (datetime.datetime): The start date of the range to download.
        end_date (datetime.datetime): The end date of the range to download.
        retrieved_files_dir (str): The directory to save the downloaded files␣
↪in.
    """
    print('Checking availability of retrieved data...')
    # Create the retrieved files directory if it doesn't already exist.
    if not os.path.exists(retrieved_files_dir):
        os.makedirs(retrieved_files_dir)

    for link in links:
        url = link['href']
        file_name = url.split('/')[-1]
        date_str = file_name.split('_')[-1].split('.')[0]
        date_obj = datetime.strptime(date_str, '%Y-%m')

        if start_date <= date_obj <= end_date:
            file_path = os.path.join(retrieved_files_dir, file_name)

            if os.path.exists(file_path):
                print(f"File '{file_name}' already exists.")
            else:
                print(f"Downloading '{file_name}'...")
                response = requests.get(url)
                response.raise_for_status()
```

```
                with open(file_path, 'wb') as file:
                    file.write(response.content)

                if file_name.endswith('.zip'):
                    csv_file_name = file_name.replace('.zip', '.csv')
                    csv_file_path = os.path.join(retrieved_files_dir,␣
↪csv_file_name)

                    if os.path.exists(csv_file_path):
                        print(f"File '{csv_file_name}' already exists.")
                    else:
                        print(f"Extracting '{file_name}'...")
                        with zipfile.ZipFile(file_path, 'r') as zip_file:
                            zip_file.extractall(retrieved_files_dir)

                        os.remove(file_path)

    print('Data fetching completed.')
```

Cleaning, filtering & sampling

[4]:
```
def clean_and_sample_data(data: pd.DataFrame, columns_to_keep: list,␣
↪columns_to_rename: dict,
                          down_threshold: float, up_threshold: float,
                          left_threshold: float, right_threshold: float,␣
↪sample_size: int, year: int) -> pd.DataFrame:
    # Only keep the columns specified in columns_to_keep and create a copy of␣
↪the DataFrame
    data = data[columns_to_keep].copy()

    # Rename the columns specified in columns_to_rename
    data.rename(columns={old_name: new_name for old_name, new_name in␣
↪zip(columns_to_keep, columns_to_rename)}, inplace=True)

    if year < 2011:
        # Only keep rows where Start_Lat, End_Lat, Start_Lon, and End_Lon are␣
↪within specified thresholds
        data = data[(data['Start_Lat'] <= up_threshold) & (data['End_Lat'] <=␣
↪up_threshold) & (data['Start_Lat'] >= down_threshold) & (
            data['End_Lat'] >= down_threshold) & (data['Start_Lon'] <=␣
↪right_threshold) & (data['End_Lon'] <= right_threshold) & (
                    data['Start_Lon'] >= left_threshold) & (data['End_Lon'] >=␣
↪left_threshold)]
    else:
        pass  # Do nothing if year is greater than or equal to 2011

    if data.empty:
```

```
        return data
    else:
        # Sample the specified number of rows randomly and return the resulting␣
↪DataFrame
        return data.sample(sample_size, random_state=42)
```

[5]:
```
def compile_and_clean_taxi_data() -> pd.DataFrame:
    yellow_taxi_data = pd.DataFrame()

    # Set threshold values for latitude and longitude coordinates
    down_threshold = 40.560445
    up_threshold = 40.908524
    left_threshold = -74.242330
    right_threshold = -73.717047

    # Set the number of samples to take
    sample_size = 2500

    # Define the columns to rename
    columns_to_rename = ['Pickup_Datetime', 'Dropoff_Datetime', "Trip_Distance",
                         "Start_Lon", "Start_Lat", "End_Lon", "End_Lat",␣
↪"Fare_Amt", "Tip_Amt"]

    # Check if the directory for sampled files exists
    if not os.path.exists(sampled_files_dir):
        os.makedirs(sampled_files_dir)
    print('Check availability of sampled data ...')

    # Loop through the years from 2009 to 2015
    for year in range(2009, 2016):
        checked = False
        # Loop through the months from 1 to 12
        for month in range(1, 13):
            file_name = f"sampled_data_{year}.csv"
            file_path = os.path.join(sampled_files_dir, file_name)

            # If the file exists, read in the data
            if os.path.exists(file_path):
                yellow_taxi_data = pd.read_csv(file_path)
                if not checked:
                    print(f"Sampled data from {file_name} loaded successfully!")
                checked = True
            else:
                # If the file does not exist, process the data
                print(f"Sampling {year}-{month:02d}")

                # Define the columns to keep based on the year
```

```
            if year == 2009:
                columns_to_keep = ['Trip_Pickup_DateTime',␣
↪'Trip_Dropoff_DateTime', "Trip_Distance",
                                   "Start_Lon", "Start_Lat", "End_Lon",␣
↪"End_Lat", "Fare_Amt", "Tip_Amt"]
            elif year == 2010:
                columns_to_keep = ['pickup_datetime', 'dropoff_datetime',␣
↪"trip_distance", "pickup_longitude", "pickup_latitude",
                                   "dropoff_longitude", "dropoff_latitude",␣
↪"fare_amount", "tip_amount"]
            if year >= 2011:
                columns_to_keep = ['tpep_pickup_datetime',␣
↪'tpep_dropoff_datetime', 'trip_distance',
                                   'Start_Lon', 'Start_Lat', 'End_Lon',␣
↪'End_Lat', 'fare_amount', 'tip_amount']

            # Read in the data for the given year and month
            data = pd.read_parquet(f"{retrieved_files_dir}/
↪yellow_tripdata_{year}-{month:02d}.parquet")

            # If the year is 2011 or later, merge with the taxi zone data
            if year >= 2011:
                data = data.merge(taxi_zones, left_on='PULocationID',␣
↪right_on='LocationID', how='left') \
                    .rename(columns={'Lon': 'Start_Lon', 'Lat':␣
↪'Start_Lat'}) \
                    .drop(columns=['PULocationID', 'LocationID'])

                data = data.merge(taxi_zones, left_on='DOLocationID',␣
↪right_on='LocationID', how='left') \
                    .rename(columns={'Lon': 'End_Lon', 'Lat': 'End_Lat'}) \
                    .drop(columns=['DOLocationID', 'LocationID'])

            sampled_data = clean_and_sample_data(data, columns_to_keep,␣
↪columns_to_rename,
                                                 down_threshold,␣
↪up_threshold, left_threshold, right_threshold,
                                                 sample_size, year)

            yellow_taxi_data = yellow_taxi_data.append(sampled_data,␣
↪ignore_index=True)
        if year == 2015 and month >= 6:
            yellow_taxi_data.to_csv(file_path, index=False)
            break
    if not os.path.exists(file_path):
        yellow_taxi_data.to_csv(file_path, index=False)
```

```
            print(f"Sampled data saved as {file_path}")


    yellow_taxi_data.to_csv("sampled_taxi_data_2009_2015.csv", index=False)
    print(f"Sampled Yellow Taxi dataset saved as {file_path}")
    return yellow_taxi_data
```

[6]:
```
def load_taxi_zones():
    """
    Load and preprocess the NYC Taxi Zones shapefile into a GeoDataFrame.

    Returns:
        geopandas.GeoDataFrame: The GeoDataFrame with processed Taxi Zone data.
    """
    # load the Taxi Zones shapefile into a GeoDataFrame
    zones = gpd.read_file('taxi_zone_data/taxi_zones.shp')

    # convert the projection of the GeoDataFrame to EPSG 2263 (US feet) to␣
 ↪match the projection of the taxi data
    zones = zones.to_crs(epsg=2263)

    # add columns for the longitude and latitude of the centroid of each zone
    zones['Lon'] = zones.centroid.x
    zones['Lat'] = zones.centroid.y

    # drop columns that won't be used in the analysis
    zones = zones.drop(columns=['OBJECTID', 'Shape_Leng', 'Shape_Area', 'zone',␣
 ↪'borough', 'geometry'])

    # return the processed GeoDataFrame
    return zones
```

[7]:
```
# Define the main URL to fetch the yellow taxi data links from.
main_url = 'https://www1.nyc.gov/site/tlc/about/tlc-trip-record-data.page'
```

[8]:
```
# Define the start and end dates for the data range to download.
start_date = datetime(2009, 1, 1)
end_date = datetime(2015, 6, 30)
```

[9]:
```
# Define the directory to save the downloaded files in.
retrieved_files_dir = 'monthly_data'
# Define the directory to save the sampled files in.
sampled_files_dir = 'sampled_data'
```

[10]:
```
# Download and extract the yellow taxi data within the specified date range.
yellow_taxi_links = fetch_yellow_taxi_links(main_url)
fetch_taxi_data(yellow_taxi_links, start_date, end_date, retrieved_files_dir)
```

```
Checking availability of retrieved data…
File 'yellow_tripdata_2015-01.parquet' already exists.
File 'yellow_tripdata_2015-02.parquet' already exists.
File 'yellow_tripdata_2015-03.parquet' already exists.
File 'yellow_tripdata_2015-04.parquet' already exists.
File 'yellow_tripdata_2015-05.parquet' already exists.
File 'yellow_tripdata_2015-06.parquet' already exists.
File 'yellow_tripdata_2014-01.parquet' already exists.
File 'yellow_tripdata_2014-02.parquet' already exists.
File 'yellow_tripdata_2014-03.parquet' already exists.
File 'yellow_tripdata_2014-04.parquet' already exists.
File 'yellow_tripdata_2014-05.parquet' already exists.
File 'yellow_tripdata_2014-06.parquet' already exists.
File 'yellow_tripdata_2014-07.parquet' already exists.
File 'yellow_tripdata_2014-08.parquet' already exists.
File 'yellow_tripdata_2014-09.parquet' already exists.
File 'yellow_tripdata_2014-10.parquet' already exists.
File 'yellow_tripdata_2014-11.parquet' already exists.
File 'yellow_tripdata_2014-12.parquet' already exists.
File 'yellow_tripdata_2013-01.parquet' already exists.
File 'yellow_tripdata_2013-02.parquet' already exists.
File 'yellow_tripdata_2013-03.parquet' already exists.
File 'yellow_tripdata_2013-04.parquet' already exists.
File 'yellow_tripdata_2013-05.parquet' already exists.
File 'yellow_tripdata_2013-06.parquet' already exists.
File 'yellow_tripdata_2013-07.parquet' already exists.
File 'yellow_tripdata_2013-08.parquet' already exists.
File 'yellow_tripdata_2013-09.parquet' already exists.
File 'yellow_tripdata_2013-10.parquet' already exists.
File 'yellow_tripdata_2013-11.parquet' already exists.
File 'yellow_tripdata_2013-12.parquet' already exists.
File 'yellow_tripdata_2012-01.parquet' already exists.
File 'yellow_tripdata_2012-02.parquet' already exists.
File 'yellow_tripdata_2012-03.parquet' already exists.
File 'yellow_tripdata_2012-04.parquet' already exists.
File 'yellow_tripdata_2012-05.parquet' already exists.
File 'yellow_tripdata_2012-06.parquet' already exists.
File 'yellow_tripdata_2012-07.parquet' already exists.
File 'yellow_tripdata_2012-08.parquet' already exists.
File 'yellow_tripdata_2012-09.parquet' already exists.
File 'yellow_tripdata_2012-10.parquet' already exists.
File 'yellow_tripdata_2012-11.parquet' already exists.
File 'yellow_tripdata_2012-12.parquet' already exists.
File 'yellow_tripdata_2011-01.parquet' already exists.
File 'yellow_tripdata_2011-02.parquet' already exists.
File 'yellow_tripdata_2011-03.parquet' already exists.
File 'yellow_tripdata_2011-04.parquet' already exists.
File 'yellow_tripdata_2011-05.parquet' already exists.
```

```
File 'yellow_tripdata_2011-06.parquet' already exists.
File 'yellow_tripdata_2011-07.parquet' already exists.
File 'yellow_tripdata_2011-08.parquet' already exists.
File 'yellow_tripdata_2011-09.parquet' already exists.
File 'yellow_tripdata_2011-10.parquet' already exists.
File 'yellow_tripdata_2011-11.parquet' already exists.
File 'yellow_tripdata_2011-12.parquet' already exists.
File 'yellow_tripdata_2010-01.parquet' already exists.
File 'yellow_tripdata_2010-02.parquet' already exists.
File 'yellow_tripdata_2010-03.parquet' already exists.
File 'yellow_tripdata_2010-04.parquet' already exists.
File 'yellow_tripdata_2010-05.parquet' already exists.
File 'yellow_tripdata_2010-06.parquet' already exists.
File 'yellow_tripdata_2010-07.parquet' already exists.
File 'yellow_tripdata_2010-08.parquet' already exists.
File 'yellow_tripdata_2010-09.parquet' already exists.
File 'yellow_tripdata_2010-10.parquet' already exists.
File 'yellow_tripdata_2010-11.parquet' already exists.
File 'yellow_tripdata_2010-12.parquet' already exists.
File 'yellow_tripdata_2009-01.parquet' already exists.
File 'yellow_tripdata_2009-02.parquet' already exists.
File 'yellow_tripdata_2009-03.parquet' already exists.
File 'yellow_tripdata_2009-04.parquet' already exists.
File 'yellow_tripdata_2009-05.parquet' already exists.
File 'yellow_tripdata_2009-06.parquet' already exists.
File 'yellow_tripdata_2009-07.parquet' already exists.
File 'yellow_tripdata_2009-08.parquet' already exists.
File 'yellow_tripdata_2009-09.parquet' already exists.
File 'yellow_tripdata_2009-10.parquet' already exists.
File 'yellow_tripdata_2009-11.parquet' already exists.
File 'yellow_tripdata_2009-12.parquet' already exists.
Data fetching completed.
```

[11]:
```
# Load the taxi zone lookup data.
taxi_zones = load_taxi_zones()
```

[12]:
```
# Compile and clean the downloaded taxi data, and save the sampled data to disk.
compiled_taxi_data = compile_and_clean_taxi_data()
```

```
Check availability of sampled data …
Sampled data from sampled_data_2009.csv loaded successfully!
Sampled data from sampled_data_2010.csv loaded successfully!
Sampled data from sampled_data_2011.csv loaded successfully!
Sampled data from sampled_data_2012.csv loaded successfully!
Sampled data from sampled_data_2013.csv loaded successfully!
Sampled data from sampled_data_2014.csv loaded successfully!
Sampled data from sampled_data_2015.csv loaded successfully!
Sampled Yellow Taxi dataset saved as sampled_data/sampled_data_2015.csv
```

[13]:
```
compiled_taxi_data.head()
```

[13]:

	Pickup_Datetime	Dropoff_Datetime	Trip_Distance	Start_Lon	Start_Lat	End_Lon	End_Lat	Fare_Amt	Tip_Amt
0	2009-01-04 07:53:32	2009-01-04 08:10:30	11.0	-73.862767	40.769043	-73.992901	40.697823	25.3	0.00
1	2009-01-25 03:29:10	2009-01-25 03:35:52	2.4	-73.977883	40.745888	-73.956754	40.772334	8.2	1.23
2	2009-01-31 20:34:17	2009-01-31 20:47:42	0.5	-73.987889	40.749865	-73.987705	40.755688	8.2	0.00
3	2009-01-21 15:05:16	2009-01-21 15:21:11	1.9	-73.990142	40.731772	-74.008403	40.725475	9.7	0.00
4	2009-01-26 19:59:38	2009-01-26 20:14:16	3.1	-73.970618	40.755777	-74.004455	40.742319	11.9	0.00

distance calculation

[14]:
```
def haversine_distance(lat1: float, lon1: float, lat2: float, lon2: float) ->␣
↪float:
    # Haversine formula for calculating the distance between two points on Earth
    R = 6371  # Earth's radius in kilometers
    lat1, lon1, lat2, lon2 = map(np.radians, [lat1, lon1, lat2, lon2])
    dlat = lat2 - lat1
    dlon = lon2 - lon1
    a = np.sin(dlat / 2) ** 2 + np.cos(lat1) * np.cos(lat2) * np.sin(dlon / 2)␣
↪** 2
    c = 2 * np.arctan2(np.sqrt(a), np.sqrt(1 - a))
    return R * c
```

Adding distance feature

[15]:
```
def add_distance_features(data: pd.DataFrame) -> pd.DataFrame:
    """
    Add columns to a Pandas dataframe with distance-related features calculated␣
↪using the haversine distance formula.

    Args:
        data (pd.DataFrame): The dataframe to which distance-related features␣
↪will be added.

    Returns:
        pd.DataFrame: The dataframe with new columns for distance-related␣
↪features.
    """
```

```
    # apply the haversine_distance function to each row of the dataframe to␣
 ↪calculate the distance between the start and end coordinates
    data['Real_Distance'] = data.apply(lambda row:␣
 ↪haversine_distance(row["Start_Lat"], row["Start_Lon"], row["End_Lat"],␣
 ↪row["End_Lon"]), axis=1)

    # return the dataframe with the new columns added
    return data
```

[16]:
```
# load the CSV file into a Pandas dataframe
yellow_taxi_dataset = pd.read_csv("sampled_taxi_data_2009_2015.csv")
```

[17]:
```
# apply the 'add_distance_features' function to the dataframe to add new␣
 ↪columns with distance-related features
yellow_taxi_dataset = add_distance_features(yellow_taxi_dataset)
```

[18]:
```
yellow_taxi_dataset.head()
```

[18]:

	Pickup_Datetime	Dropoff_Datetime	Trip_Distance	Start_Lon	Start_Lat	End_Lon	End_Lat	Fare_Amt	Tip_Amt	Real_Distance
0	2009-01-04 07:53:32	2009-01-04 08:10:30	11.0	-73.862767	40.769043	-73.992901	40.697823	25.3	0.00	13.525673
1	2009-01-25 03:29:10	2009-01-25 03:35:52	2.4	-73.977883	40.745888	-73.956754	40.772334	8.2	1.23	3.437221
2	2009-01-31 20:34:17	2009-01-31 20:47:42	0.5	-73.987889	40.749865	-73.987705	40.755688	8.2	0.00	0.647674
3	2009-01-21 15:05:16	2009-01-21 15:21:11	1.9	-73.990142	40.731772	-74.008403	40.725475	9.7	0.00	1.690572
4	2009-01-26 19:59:38	2009-01-26 20:14:16	3.1	-73.970618	40.755777	-74.004455	40.742319	11.9	0.00	3.219327

0.1.2 Uber dataset

[19]:
```
uber_data = pd.read_csv("uber_rides_sample.csv")
```

[20]:
```
def load_and_clean_uber_data(csv_file) -> pd.DataFrame:
    csv_file = csv_file.dropna()

    # Set the latitude and longitude
    csv_file = csv_file[(csv_file['pickup_latitude'] >= 40.560445) &
                        (csv_file['pickup_latitude'] <= 40.908524) &
                        (csv_file['pickup_longitude'] >= -74.242330) &
                        (csv_file['pickup_longitude'] <= -73.717047) &
                        (csv_file['dropoff_latitude'] >= 40.560445) &
                        (csv_file['dropoff_latitude'] <= 40.908524) &
```

```
                        (csv_file['dropoff_longitude'] >= -74.242330) &
                        (csv_file['dropoff_longitude'] <= -73.717047)]

    # Convert pickup_datetime column to datetime format
    csv_file.loc[:, 'pickup_datetime'] = pd.
↪to_datetime(csv_file['pickup_datetime'], format='%Y-%m-%d %H:%M:%S UTC')

    # Create new columns for pickup date and time, as well as year, month, day,␣
↪and hour
    csv_file.loc[:, 'Pickup_Date'] = csv_file['pickup_datetime'].dt.date
    csv_file.loc[:, 'Pickup_Time'] = csv_file['pickup_datetime'].dt.time
    csv_file.loc[:, 'Year'] = csv_file['pickup_datetime'].dt.year
    csv_file.loc[:, 'Month'] = csv_file['pickup_datetime'].dt.month
    csv_file.loc[:, 'Day'] = csv_file['pickup_datetime'].dt.day
    csv_file.loc[:, 'Hour'] = csv_file['pickup_datetime'].dt.hour

    # Convert pickup datetime to datetime object
    csv_file.loc[:, 'Pickup'] = pd.to_datetime(csv_file['pickup_datetime'])

    # Extract day of the week from pickup datetime
    csv_file.loc[:, 'DayofWeek'] = csv_file['Pickup'].dt.dayofweek

    # Combine latitude and longitude columns into tuples
    csv_file['Start_point'] = list(zip(csv_file['pickup_latitude'],␣
↪csv_file['pickup_longitude']))
    csv_file['End_point'] = list(zip(csv_file['dropoff_latitude'],␣
↪csv_file['dropoff_longitude']))

    csv_file = csv_file.drop(columns=['key', 'Unnamed: 0', 'passenger_count',␣
↪'Pickup_Time', 'pickup_datetime'])\
                .rename(columns={'pickup_longitude': 'Start_Lon',
                                 'pickup_latitude': 'Start_Lat',
                                 'dropoff_longitude': 'End_Lon',
                                 'dropoff_latitude': 'End_Lat',
                                 'fare_amount': 'Fare_Amt',
                                 'Pickup_Date': 'Date'})


    return csv_file
```

[21]:
```
def get_uber_data() -> pd.DataFrame:
    # Load and clean the Uber dataset, and add distance features to the dataset
    uber_dataframe = load_and_clean_uber_data(uber_data)
    add_distance_features(uber_dataframe)

    # Remove unnecessary columns from the dataset, and convert the 'Pickup'␣
↪column to datetime format
```

```
    uber_dataframe.drop(columns=["Start_point", "End_point"], inplace=True)
    uber_dataframe["Pickup"] = pd.to_datetime(uber_dataframe["Pickup"])

    # Extract date from the 'Pickup' column and create a new column␣
↪'Pickup_Date'
    uber_dataframe['Pickup_Date'] = uber_dataframe['Pickup'].dt.date

    # Convert the 'Hour' and 'DayofWeek' columns to integers, and rename the␣
↪'Real_Distance' column to 'Trip_distance'
    uber_dataframe['Hour'] = uber_dataframe['Hour'].astype(int)
    uber_dataframe['DayofWeek'] = uber_dataframe['DayofWeek'].astype(int)
    uber_dataframe.rename(columns={'Real_Distance': 'Trip_distance'},␣
↪inplace=True)

    return uber_dataframe
```

[22]:
```
uber_dataset = get_uber_data()
```

[23]:
```
uber_dataset.head()
```

[23]:

	Fare_Amt	Start_Lon	Start_Lat	End_Lon	End_Lat	Date	Year	Month	Day	Hour	Pickup	DayofWeek	Trip_distance	Pickup_Date
0	7.5	-73.999817	40.738354	-73.999512	40.723217	2015-05-07	2015	5	7	19	2015-05-07 19:52:06	3	1.683323	2015-05-07
1	7.7	-73.994355	40.728225	-73.994710	40.750325	2009-07-17	2009	7	17	20	2009-07-17 20:04:56	4	2.457590	2009-07-17
2	12.9	-74.005043	40.740770	-73.962565	40.772647	2009-08-24	2009	8	24	21	2009-08-24 21:45:00	0	5.036377	2009-08-24
3	5.3	-73.976124	40.790844	-73.965316	40.803349	2009-06-26	2009	6	26	8	2009-06-26 08:22:21	4	1.661683	2009-06-26
4	16.0	-73.925023	40.744085	-73.973082	40.761247	2014-08-28	2014	8	28	17	2014-08-28 17:47:00	3	4.475450	2014-08-28

0.1.3 Whether Data

[24]:
```
import os
import pandas as pd


# Define a list of column names for the weather data
WEATHER_COLUMNS = [
    'DATE',
    'LATITUDE',
    'LONGITUDE',

    'Sunrise',
```

```
    'Sunset',

    'DailyPeakWindSpeed',
    'DailyPrecipitation',
    'DailySustainedWindSpeed',
    'DailyAverageWindSpeed',

    'HourlyWindSpeed',
    'HourlyPrecipitation'
]


# Define a function to get all CSV files in a directory
def get_csv_files(directory: str) -> list:
    return [os.path.join(directory, file) for file in os.listdir(directory) if␣
 ↪file.endswith('.csv')]


# Define a function to merge multiple CSV files into a single DataFrame
def merge_csv_files(files: list, columns: list) -> pd.DataFrame:
    return pd.concat([pd.read_csv(file, usecols=columns) for file in files])


# Define a function to split the 'DATE' column into separate 'Date' and 'Time'␣
 ↪columns
def split_datetime_column(weather_data: pd.DataFrame) -> None:
    datetime_data = pd.to_datetime(weather_data['DATE'])
    weather_data['Date'] = datetime_data.dt.date.astype(str)
    weather_data['Time'] = datetime_data.dt.time.astype(str)
    weather_data.drop(columns=['DATE'], inplace=True)


# Define a function to save the merged weather data to a CSV file
def save_weather_data(weather_data: pd.DataFrame, output_file: str) -> None:
    weather_data.to_csv(output_file, index=False)
    weather_data.dropna(inplace=True)


# Define a function to merge all weather files in a directory, split the 'DATE'␣
 ↪column, and save the result to a CSV file
def merge_weather_files() -> None:
    files = get_csv_files('weather_data')
    weather_data = merge_csv_files(files, WEATHER_COLUMNS)
    split_datetime_column(weather_data)
    save_weather_data(weather_data, 'WeatherData.csv')
```

[25]:
```
def get_hourly_weather_data(csv_file) -> pd.DataFrame:
    # Select rows with Time column containing "51:00"
    weather_hourly = csv_file[csv_file['Time'].str.contains("51:00")]

    # Extract HourlyWindSpeed, HourlyPrecipitation, Time, and Date columns into␣
 ↪a new dataframe
    weather_hourly = weather_hourly.loc[:, ["HourlyWindSpeed",␣
 ↪"HourlyPrecipitation", "Time", "Date"]]

    # Replace missing and "T" values in HourlyPrecipitation column with 0.005
    weather_hourly["HourlyPrecipitation"] =␣
 ↪weather_hourly["HourlyPrecipitation"].fillna(0).str.replace("T", "0.005")

    # Replace missing, "T", and "nan" values in HourlyWindSpeed column with 0,␣
 ↪and convert to numeric data type
    weather_hourly["HourlyWindSpeed"] = pd.
 ↪to_numeric(weather_hourly["HourlyWindSpeed"].fillna(0).replace(["T", "nan"],␣
 ↪0.005).replace("nan", 0))

    # Combine Date and Time columns to create a new datetime column
    weather_hourly['Record_Time'] = pd.to_datetime(weather_hourly['Date'] + ' '␣
 ↪+ weather_hourly['Time'])

    # Extract Year, Month, Day, and Hour from the datetime column
    weather_hourly['Year'] = weather_hourly['Record_Time'].dt.year
    weather_hourly['Month'] = weather_hourly['Record_Time'].dt.month
    weather_hourly['Day'] = weather_hourly['Record_Time'].dt.day
    weather_hourly['Hour'] = weather_hourly['Record_Time'].dt.hour

    # Drop Unnecessary Columns
    weather_hourly.drop(columns=['Time'], inplace=True)
    weather_hourly.dropna(inplace=True)
    return weather_hourly


def get_daily_weather_data(csv_file) -> pd.DataFrame:
    # Select relevant columns and replace T with 0.005
    weather_daily = csv_file.loc[:, ['DailyPrecipitation',␣
 ↪'DailySustainedWindSpeed', 'DailyPeakWindSpeed', 'DailyAverageWindSpeed',␣
 ↪'Time', 'Date']]
    weather_daily['DailyPrecipitation'] = weather_daily['DailyPrecipitation'].
 ↪replace('T', 0.005)

    # Drop rows with any missing values
    weather_daily.dropna(inplace=True)
```

```
    # Combine date and time columns to create 'Record_Time'
    weather_daily['Record_Time'] = pd.to_datetime(weather_daily['Date'] + ' ' +␣
↪weather_daily['Time'])

    # Extract year, month, day, and hour from Record_Time column
    weather_daily['Year'], weather_daily['Month'], weather_daily['Day'],␣
↪weather_daily['Hour'] = \
        weather_daily['Record_Time'].dt.year, weather_daily['Record_Time'].dt.
↪month, \
        weather_daily['Record_Time'].dt.day, weather_daily['Record_Time'].dt.
↪hour

    # Drop the Time column
    weather_daily.drop(columns=['Time'], inplace=True)
    weather_daily.dropna(inplace=True)
    return weather_daily




def get_daily_sun_data(csv_file) -> pd.DataFrame:

    weather_sun = csv_file[['Sunrise', 'Sunset', 'Date']].dropna().
↪reset_index(drop=True)


    weather_sun[['Sunrise', 'Sunset']] = weather_sun[['Sunrise', 'Sunset']].
↪apply(
        lambda x: pd.to_datetime(x, format='%H%M').dt.time)



    # Concatenate Date and Sunrise columns to create a new column␣
↪Sunrise_DateTime
    weather_sun['Sunrise_DateTime'] = pd.to_datetime(weather_sun['Date'] + ' '␣
↪+ weather_sun['Sunrise'].astype(str))

    # Convert Sunrise_DateTime column to Sunrise column and drop␣
↪Sunrise_DateTime column
    weather_sun['Sunrise'] = weather_sun['Sunrise_DateTime']
    weather_sun.drop(columns=['Sunrise_DateTime'], inplace=True)

    # Concatenate Date and Sunset columns to create a new column Sunset_DateTime
    weather_sun['Sunset_DateTime'] = pd.to_datetime(weather_sun['Date'] + ' ' +␣
↪weather_sun['Sunset'].astype(str))
```

```
    # Convert Sunset_DateTime column to Sunset column and drop Sunset_DateTime column
    weather_sun['Sunset'] = weather_sun['Sunset_DateTime']
    weather_sun.drop(columns=['Sunset_DateTime'], inplace=True)
    weather_sun.dropna(inplace=True)

    return weather_sun
```

[26]:
```
merge_weather_files()
weather_dataset = pd.read_csv('WeatherData.csv')
```

[27]:
```
# Call the 'get_hourly_weather_data' function to get hourly weather data
hourly_weather_dataset = get_hourly_weather_data(weather_dataset)
hourly_weather_dataset.head()
```

[27]:
```
     HourlyWindSpeed HourlyPrecipitation        Date         Record_Time  \
24               3.0                0.01  2012-01-01 2012-01-01 18:51:00
25               7.0               0.005  2012-01-01 2012-01-01 19:51:00
26               9.0                0.04  2012-01-01 2012-01-01 20:51:00
27              11.0               0.005  2012-01-01 2012-01-01 21:51:00
269             11.0               0.005  2012-01-11 2012-01-11 21:51:00

     Year  Month  Day  Hour
24   2012      1    1    18
25   2012      1    1    19
26   2012      1    1    20
27   2012      1    1    21
269  2012      1   11    21
```

[28]:
```
# Call the 'get_daily_weather_data' function to get daily weather data
daily_weather_dataset = get_daily_weather_data(weather_dataset)
daily_weather_dataset.head()
```

[28]:
```
     DailyPrecipitation  DailySustainedWindSpeed  DailyPeakWindSpeed  \
5796               0.00                     10.0                15.0
5850               0.64                      9.0                15.0
5876               0.00                     12.0                19.0
5902               0.00                      9.0                17.0
5935               0.00                     12.0                21.0

      DailyAverageWindSpeed        Date         Record_Time  Year  Month  Day  \
5796                    3.8  2012-07-31 2012-07-31 23:59:00  2012      7   31
5850                    2.3  2012-08-01 2012-08-01 23:59:00  2012      8    1
5876                    2.7  2012-08-02 2012-08-02 23:59:00  2012      8    2
5902                    3.5  2012-08-03 2012-08-03 23:59:00  2012      8    3
5935                    3.1  2012-08-04 2012-08-04 23:59:00  2012      8    4
```

```
      Hour
5796    23
5850    23
5876    23
5902    23
5935    23
```

[29]:
```
# Call the 'get_daily_sun_data' function to get daily sunrise and sunset times
daily_sun_dataset = get_daily_sun_data(weather_dataset)
daily_sun_dataset.head()
```

[29]:
```
                Sunrise              Sunset        Date
0 2012-01-01 07:20:00 2012-01-01 16:39:00  2012-01-01
1 2012-01-10 07:20:00 2012-01-10 16:48:00  2012-01-10
2 2012-01-11 07:20:00 2012-01-11 16:49:00  2012-01-11
3 2012-01-12 07:19:00 2012-01-12 16:50:00  2012-01-12
4 2012-01-13 07:19:00 2012-01-13 16:51:00  2012-01-13
```

0.2 Part 2: Storing Data

0.2.1 create a SQLite database:

[30]:
```
SCHEMA_FILE = "schema.sql"
QUERY_DIRECTORY = "queries"
```

[31]:
```
conn = sqlite3.connect("project.db")

# Write data to the tables
yellow_taxi_dataset.to_sql("yellow_taxi_database", conn, if_exists="replace",␣
 ↪index=False)
uber_dataset.to_sql("uber_database", conn, if_exists="replace", index=False)
hourly_weather_dataset.to_sql("hourly_weather_database", conn,␣
 ↪if_exists="replace", index=False)
daily_weather_dataset.to_sql("daily_weather_database", conn,␣
 ↪if_exists="replace", index=False)

# Close the connection
conn.close()
```

[32]:
```
### create table
DAILY_WEATHER_SCHEMA = """
CREATE TABLE IF NOT EXISTS daily_weather_database
(
    id INTEGER PRIMARY KEY,
    DailyPrecipitation FLOAT,
    DailySustainedWindSpeed FLOAT,
```

```
    DailyPeakWindSpeed FLOAT,
    DailyAverageWindSpeed FLOAT,
    Date TIMESTAMP,
    Record_Time TIMESTAMP,
    Year INTEGER,
    Month INTEGER,
    Day INTEGER,
    Hour INTEGER
);
"""

HOURLY_WEATHER_SCHEMA = """
CREATE TABLE IF NOT EXISTS hourly_weather_database
(
    id INTEGER PRIMARY KEY,
    HourlyWindSpeed FLOAT,
    HourlyPrecipitation FLOAT,
    Date TIMESTAMP,
    Record_Time TIMESTAMP,
    Year INTEGER,
    Month INTEGER,
    Day INTEGER,
    Hour INTEGER
);
"""

TAXI_TRIPS_SCHEMA = """
CREATE TABLE IF NOT EXISTS yellow_taxi_database
(
    id INTEGER PRIMARY KEY,
    Date TIMESTAMP,
    Pickup TIMESTAMP,
    Pickup_Time TIMESTAMP,
    Trip_Distance FLOAT,
    Start_Lon FLOAT,
    Start_Lat FLOAT,
    End_Lon FLOAT,
    End_Lat FLOAT,
    Fare_Amt FLOAT,
    Tip_Amt FLOAT,
    Year INTEGER,
    Month INTEGER,
    Day INTEGER,
    Time INTEGER,
    DayofWeek INTEGER
);
"""
```

```
UBER_TRIPS_SCHEMA = """
CREATE TABLE IF NOT EXISTS uber_database
(
    id INTEGER PRIMARY KEY,
    Fare_Amt FLOAT,
    Start_Lon FLOAT,
    Start_Lat FLOAT,
    End_Lon FLOAT,
    End_Lat FLOAT,
    Date TIMESTAMP,
    Year INTEGER,
    Month INTEGER,
    Day INTEGER,
    Hour INTEGER,
    Pickup TIMESTAMP,
    DayofWeek INTEGER,
    Trip_Distance FLOAT,
    Pickup_Date TIMESTAMP
);
"""
```

[33]:
```
# Create the required schema.sql file
with open(SCHEMA_FILE, "w") as f:
    f.write(HOURLY_WEATHER_SCHEMA)
    f.write(DAILY_WEATHER_SCHEMA)
    f.write(TAXI_TRIPS_SCHEMA)
    f.write(UBER_TRIPS_SCHEMA)

# Connect to the SQLite database
conn = sqlite3.connect("project.db")
```

[34]:
```
# Create the tables with the schema files
with conn:
    conn.execute(HOURLY_WEATHER_SCHEMA)
    conn.execute(DAILY_WEATHER_SCHEMA)
    conn.execute(TAXI_TRIPS_SCHEMA)
    conn.execute(UBER_TRIPS_SCHEMA)

# Close the connection
conn.close()
```

0.3 Part 3: Understanding the Data

[35]:
```
# Connect to the SQLite database
conn = sqlite3.connect("project.db")
```

[36]:
```
# Query 1
query1 = """
    SELECT strftime('%Y-%m-%d', Pickup_Datetime) as Date, strftime('%H',␣
 ↪Pickup_Datetime) as Hour, COUNT(*) as Trips
    FROM yellow_taxi_database
    WHERE strftime('%Y-%m-%d', Pickup_Datetime) BETWEEN '2009-01-01' AND␣
 ↪'2015-06-30'
    GROUP BY Date, Hour
    ORDER BY Trips DESC;
    """
result1 = pd.read_sql_query(query1, conn)
result1.to_sql('result1', conn, index=False, if_exists='replace')
print(result1)
```

```
             Date Hour  Trips
0      2013-08-22   20     17
1      2015-04-19   14     16
2      2009-02-14   23     15
3      2010-02-13   20     15
4      2013-05-04   19     15
…             …  …     …
50796  2015-06-28   18      1
50797  2015-06-29   01      1
50798  2015-06-29   03      1
50799  2015-06-30   05      1
50800  2015-06-30   12      1

[50801 rows x 3 columns]
```

[37]:
```
# Define the SQL query
query2 = """
    SELECT Pickup_Date as Weekday, COUNT(*) as Trips
    FROM uber_database
    WHERE Pickup_Date BETWEEN '2009-01-01' AND '2015-06-30'
    GROUP BY Weekday
    ORDER BY Trips DESC;
"""

# Execute the query and store the results in a Pandas dataframe
result2 = pd.read_sql_query(query2, conn)
result2.to_sql('result2', conn, index=False, if_exists='replace')
```

```
# Print the dataframe
print(result2)
```

```
         Weekday  Trips
0     2009-12-11    127
1     2011-04-27    125
2     2009-10-23    123
3     2011-06-08    123
4     2012-03-23    122
…            …    …
2367  2014-12-25     25
2368  2015-01-27     22
2369  2012-10-29     21
2370  2010-12-27     13
2371  2011-08-28      7

[2372 rows x 2 columns]
```

[38]:
```
# Query 3
query3 = '''
    WITH combined_trips AS (
        SELECT Trip_Distance,
               CASE
                   WHEN Pickup_Datetime IS NOT NULL THEN strftime('%Y-%m-%d',␣
 ↪Pickup_Datetime)
                   ELSE strftime('%Y-%m-%d', Pickup_Date)
               END as Trip_Date
        FROM (
            SELECT Trip_Distance, Pickup_Datetime, NULL as Pickup_Date
            FROM yellow_taxi_database
            WHERE Pickup_Datetime BETWEEN '2013-07-01' AND '2013-07-31'
            UNION ALL
            SELECT Trip_Distance, NULL as Pickup_Datetime, Pickup_Date
            FROM uber_database
            WHERE Pickup_Date BETWEEN '2013-07-01' AND '2013-07-31'
        )
    ),
    ordered_trips AS (
        SELECT Trip_Distance,
               ROW_NUMBER() OVER (ORDER BY Trip_Distance) AS row_num,
               (SELECT COUNT(*) FROM combined_trips) AS total_count
        FROM combined_trips
    )
    SELECT Trip_Distance
    FROM ordered_trips
    WHERE row_num = ROUND(total_count * 0.95);
    '''
```

```
result3 = pd.read_sql_query(query3, conn)
result3.to_sql('result3', conn, index=False, if_exists='replace')

print(result3)
```

```
   Trip_Distance
0      10.216086
```

[39]:
```
# What were the top 10 days with the highest number of hired rides for 2009,␣
 ↪and what was the average distance for each day?

# Query 4
query4 = """
    WITH Combined_Trips AS (
        SELECT strftime('%Y-%m-%d', Pickup_Datetime) as Date, Trip_Distance
        FROM yellow_taxi_database
        WHERE strftime('%Y', Date) = '2009'
        UNION ALL
        SELECT strftime('%Y-%m-%d', Pickup_Date) as Date, Trip_Distance
        FROM uber
        WHERE strftime('%Y', Date) = '2009'
    )
    SELECT Date, COUNT(*) as Rides, AVG(Trip_Distance) as Avg_Distance
    FROM Combined_Trips
    GROUP BY Date
    ORDER BY Rides DESC
    LIMIT 10;
    """
result4 = pd.read_sql_query(query4, conn)

result4.to_sql('result4', conn, index=False, if_exists='replace')
print(result4)
```

```
         Date  Rides  Avg_Distance
0  2009-12-11    234      2.789095
1  2009-01-31    225      2.389727
2  2009-02-19    213      2.867974
3  2009-06-19    211      2.896776
4  2009-12-04    210      2.507848
5  2009-02-18    209      2.980824
6  2009-08-14    209      2.978543
7  2009-05-08    206      2.916524
8  2009-01-27    204      2.478543
9  2009-04-18    204      3.161798
```

[40]:
```
# Query 5
query5 = """
    WITH
        yellow_trips AS (
            SELECT
                strftime('%Y-%m-%d', Pickup_Datetime) as Date,
                COUNT(*) as Trips
            FROM yellow_taxi_database
            WHERE Date BETWEEN '2014-01-01' AND '2014-12-31'
            GROUP BY Date
        ),
        uber_trips AS (
            SELECT
                strftime('%Y-%m-%d', Pickup_Date) as Date,
                COUNT(*) as Trips
            FROM uber_database
            WHERE Date BETWEEN '2014-01-01' AND '2014-12-31'
            GROUP BY Date
        ),
        daily_stats AS (
            SELECT
                Date,
                AVG(DailyAverageWindSpeed) as Avg_Wind_Speed
            FROM daily_weather
            WHERE Date BETWEEN '2014-01-01' AND '2014-12-31'
            GROUP BY Date
        )
    SELECT
        daily_stats.Date,
        daily_stats.Avg_Wind_Speed,
        COALESCE(yellow_trips.Trips, 0) + COALESCE(uber_trips.Trips, 0) AS␣
 ↪Hired_Trips
    FROM
        daily_stats
        LEFT JOIN yellow_trips ON daily_stats.Date = yellow_trips.Date
        LEFT JOIN uber_trips ON daily_stats.Date = uber_trips.Date
    ORDER BY
        daily_stats.Avg_Wind_Speed DESC
    LIMIT
        10;
    """
result5 = pd.read_sql_query(query5, conn)

result5.to_sql('result5', conn, index=False, if_exists='replace')
print(result5)
```

```
         Date  Avg_Wind_Speed  Hired_Trips
```

```
0  2014-03-13            14.1          198
1  2014-01-07            13.1          145
2  2014-02-13            12.6          123
3  2014-01-02            12.2          123
4  2014-03-26            11.9          189
5  2014-12-07            11.8          175
6  2014-12-08            11.5          158
7  2014-03-29            10.8          184
8  2014-11-02            10.8          170
9  2014-01-03            10.4           99
```

[41]:
```
# Query 6
query6 = '''
WITH hourly_trips AS (
    SELECT COUNT(*) AS num_rides, strftime('%Y-%m-%d %H', Pickup_Datetime) AS␣
 ↪hour
    FROM (
        SELECT Pickup_Datetime FROM yellow_taxi_database WHERE␣
 ↪strftime('%Y-%m-%d %H', Pickup_Datetime) BETWEEN '2012-10-22 00' AND␣
 ↪'2012-11-06 23'
        UNION ALL
        SELECT Pickup_Date FROM uber_database WHERE strftime('%Y-%m-%d %H',␣
 ↪Pickup_Date) BETWEEN '2012-10-22 00' AND '2012-11-06 23'
    ) all_trips
    GROUP BY hour
),
hourly_weather_data AS (
    SELECT strftime('%Y-%m-%d %H', Record_Time) AS hour, AVG(HourlyWindSpeed)␣
 ↪AS avg_wind_speed, SUM(HourlyPrecipitation) AS total_precipitation
    FROM hourly_weather_database
    WHERE strftime('%Y-%m-%d %H', Record_Time) BETWEEN '2012-10-22 00' AND␣
 ↪'2012-11-06 23'
    GROUP BY hour
),
all_hours AS (
    SELECT strftime('%Y-%m-%d %H', Record_Time) AS hour
    FROM hourly_weather_database
    WHERE strftime('%Y-%m-%d %H', Record_Time) BETWEEN '2012-10-22 00' AND␣
 ↪'2012-11-06 23'
)
SELECT all_hours.hour, COALESCE(hourly_trips.num_rides, 0) AS num_rides,␣
 ↪COALESCE(hourly_weather_data.total_precipitation, 0) AS total_precipitation,␣
 ↪COALESCE(hourly_weather_data.avg_wind_speed, 0) AS avg_wind_speed
FROM all_hours
LEFT JOIN hourly_trips ON all_hours.hour = hourly_trips.hour
LEFT JOIN hourly_weather_data ON all_hours.hour = hourly_weather_data.hour
ORDER BY all_hours.hour;
```

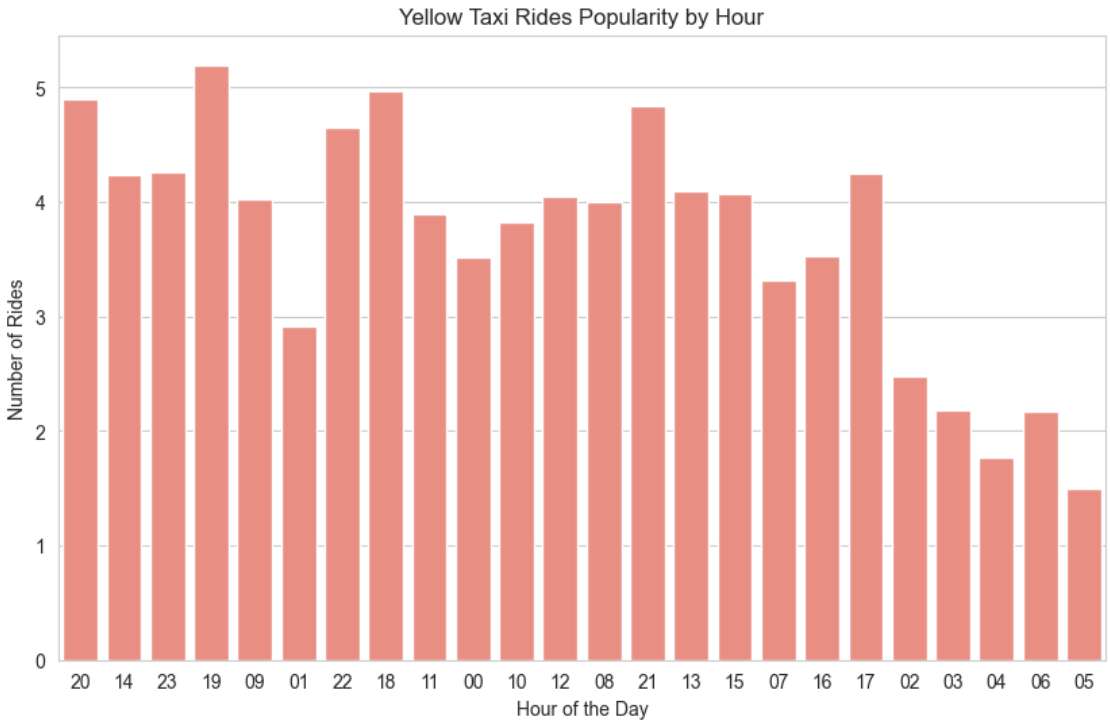
```
'''


result6 = pd.read_sql_query(query6, conn)
result6.to_sql('result2', conn, index=False, if_exists='replace')

print(result6)
```

```
              hour  num_rides  total_precipitation  avg_wind_speed
0    2012-10-22 00         76                0.025        6.200000
1    2012-10-22 01          0                0.015        5.266667
2    2012-10-22 02          0                0.035        5.266667
3    2012-10-22 03          0                0.040        6.800000
4    2012-10-22 04          1                0.005        6.000000
..             …        …                    …              …
361  2012-11-06 19          7                0.000        0.000000
362  2012-11-06 20          2                0.000        0.000000
363  2012-11-06 21          3                0.000        0.000000
364  2012-11-06 22          5                0.000        0.000000
365  2012-11-06 23          3                0.000        0.000000

[366 rows x 4 columns]
```

[42]:
```
#### part 4
```

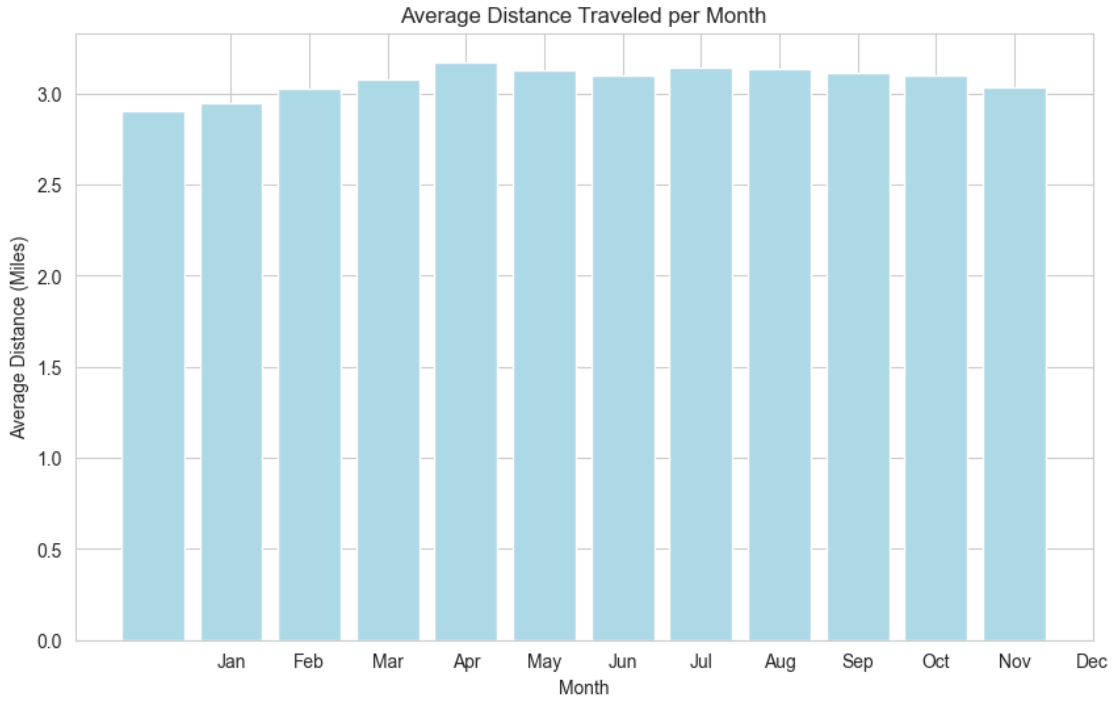
[43]:
```
# Set the seaborn style
sns.set_style("whitegrid")
```

[44]:
```
# Define a function for the first visualization from Query 1 in Part 3.
def visualize_query1():
    conn = sqlite3.connect("project.db")
    query1 = "SELECT * FROM result1;"
    df = pd.read_sql_query(query1, conn)

    plt.figure(figsize=(10, 6))
    sns.barplot(data=df, x='Hour', y='Trips', color="salmon", errorbar=None)
    plt.title('Yellow Taxi Rides Popularity by Hour')
    plt.xlabel('Hour of the Day')
    plt.ylabel('Number of Rides')
    plt.show()

visualize_query1()
```



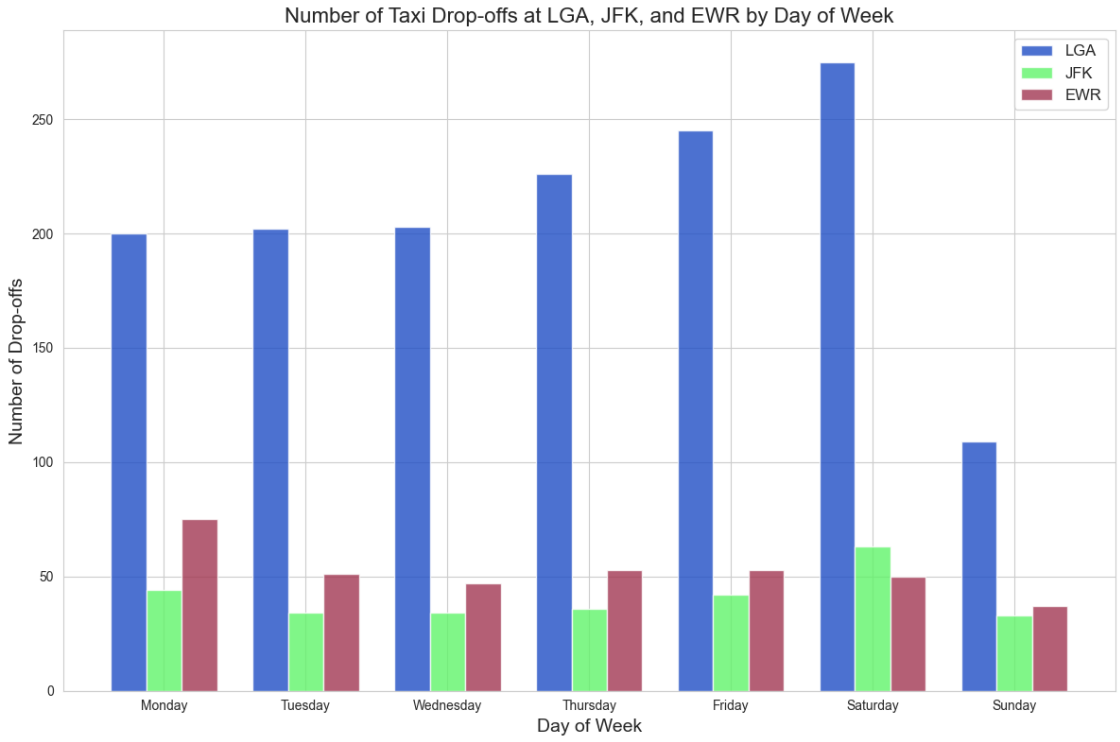
[45]:
```
# Define a function for the second visualization.
def visualize_avg_distance_per_month():
    query = """
    SELECT strftime('%m', Pickup_Date) as Month, AVG(trip_distance) as␣
 ↪Avg_Distance
    FROM (
        SELECT Pickup_Datetime as Pickup_Date, trip_distance
        FROM yellow_taxi_database
        UNION ALL
        SELECT Pickup_Date, trip_distance
        FROM uber_database
    )
    GROUP BY Month;

    """
    df = pd.read_sql_query(query, conn)

    plt.figure(figsize=(10, 6))
    plt.bar(x=df['Month'], height=df['Avg_Distance'], color="lightblue")
    plt.title('Average Distance Traveled per Month')
    plt.xlabel('Month')
    plt.ylabel('Average Distance (Miles)')
```

```
    plt.xticks(range(1, 13), ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul',␣
 ↪'Aug', 'Sep', 'Oct', 'Nov', 'Dec'])
    plt.show()
visualize_avg_distance_per_month()
```



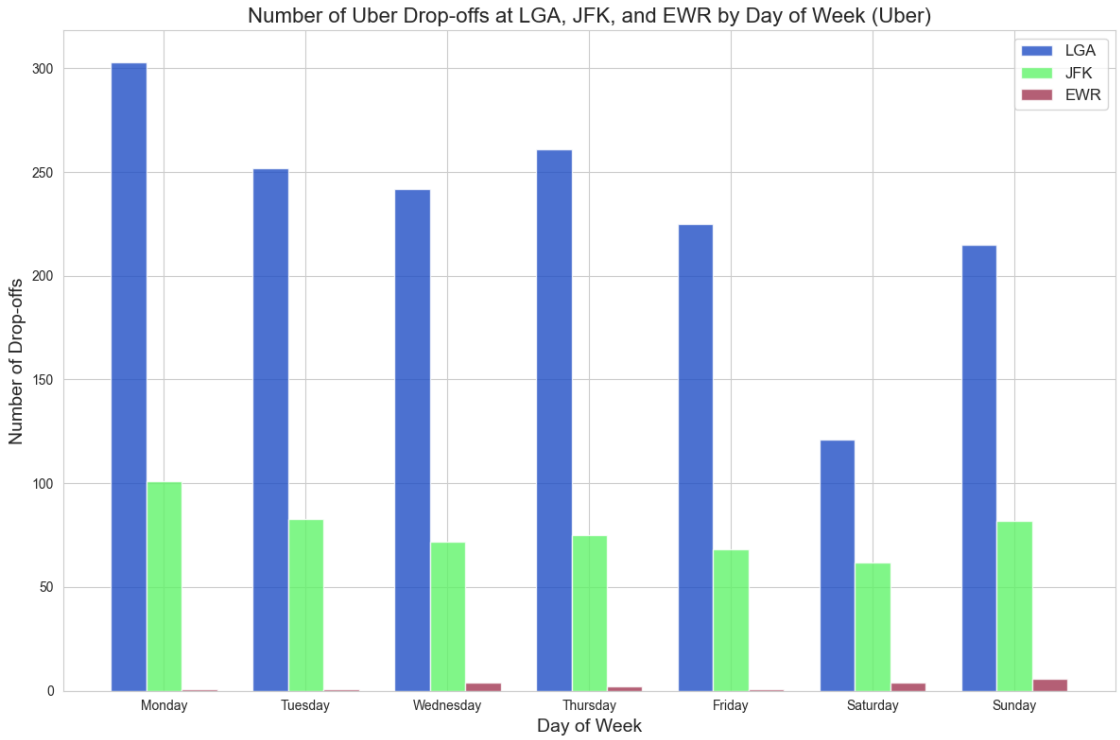
[46]:
```
Query_Visual_3 = '''
SELECT
strftime('%w', DayOfWeek) as DayOfWeek,
SUM(CASE WHEN (End_Lat BETWEEN 40.7614 AND 40.7747) AND (End_Lon BETWEEN -73.
 ↪8690 AND -73.8542) THEN 1 ELSE 0 END) as LGA,
SUM(CASE WHEN (End_Lat BETWEEN 40.6413 AND 40.6551) AND (End_Lon BETWEEN -73.
 ↪7781 AND -73.7617) THEN 1 ELSE 0 END) as JFK,
SUM(CASE WHEN (End_Lat BETWEEN 40.6594 AND 40.7108) AND (End_Lon BETWEEN -74.
 ↪2060 AND -74.1232) THEN 1 ELSE 0 END) as EWR
FROM (
SELECT strftime('%w', Pickup_Datetime) as DayOfWeek, End_Lat, End_Lon
FROM yellow_taxi_database
WHERE (End_Lat BETWEEN 40.5 AND 40.9) AND (End_Lon BETWEEN -74.3 AND -73.7)
UNION ALL
SELECT strftime('%w', Pickup) as DayOfWeek, End_Lat, End_Lon
FROM uber_database
WHERE (End_Lat BETWEEN 40.5 AND 40.9) AND (End_Lon BETWEEN -74.3 AND -73.7)
)
GROUP BY DayOfWeek;
```

```
'''

# Execute the query and store the results in a pandas dataframe.
airport_dropoffs_df = pd.read_sql_query(Query_Visual_3, conn)

# Create a new figure and axis object to plot the data.
fig, ax = plt.subplots(figsize=(12, 8))

# Define the positions of the bars and their width, and set the opacity.
x = np.arange(7)
bar_width = 0.25
opacity = 0.8

# Plot the data for each airport as a bar chart.
rects1 = ax.bar(x - bar_width, airport_dropoffs_df['LGA'], bar_width,␣
 ↪alpha=opacity, color='#1F4EC5', label='LGA')
rects2 = ax.bar(x, airport_dropoffs_df['JFK'], bar_width, alpha=opacity,␣
 ↪color='#61F46A', label='JFK')
rects3 = ax.bar(x + bar_width, airport_dropoffs_df['EWR'], bar_width,␣
 ↪alpha=opacity, color='#A23752', label='EWR')

# Set the labels for the x-axis, y-axis, and title of the plot, and the ticks␣
 ↪and tick labels for the x-axis.
ax.set_xlabel('Day of Week', fontsize=14)
ax.set_ylabel('Number of Drop-offs', fontsize=14)
ax.set_title('Number of Taxi Drop-offs at LGA, JFK, and EWR by Day of Week',␣
 ↪fontsize=16)
ax.set_xticks(x)
ax.set_xticklabels(['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday',␣
 ↪'Saturday', 'Sunday'])

# Add a legend to the plot.
ax.legend(fontsize=12)

# Ensure that all elements of the plot are visible, and display the plot.
fig.tight_layout()
plt.show()
```



[47]:
```
# Define the SQL query to extract the number of drop-offs at LGA, JFK, and EWR␣
 ↪airports by day of the week from the 'uber' table.
Query_Visual_3_uber = '''
SELECT
    DayofWeek,
    SUM(CASE WHEN (Start_Lat BETWEEN 40.7614 AND 40.7747) AND (Start_Lon␣
 ↪BETWEEN -73.8690 AND -73.8542) THEN 1 ELSE 0 END) as LGA,
    SUM(CASE WHEN (Start_Lat BETWEEN 40.6413 AND 40.6551) AND (Start_Lon␣
 ↪BETWEEN -73.7781 AND -73.7617) THEN 1 ELSE 0 END) as JFK,
    SUM(CASE WHEN (Start_Lat BETWEEN 40.6594 AND 40.7108) AND (Start_Lon␣
 ↪BETWEEN -74.2060 AND -74.1232) THEN 1 ELSE 0 END) as EWR
FROM uber
GROUP BY DayofWeek;
'''

# Execute the query and store the results in a pandas dataframe.
airport_dropoffs_df = pd.read_sql_query(Query_Visual_3_uber, conn)

# Create a new figure and axis object to plot the data.
fig, ax = plt.subplots(figsize=(12, 8))

# Define the positions of the bars and their width, and set the opacity.
x = np.arange(7)
```

```
bar_width = 0.25
opacity = 0.8

# Plot the data for each airport as a bar chart.
rects1 = ax.bar(x - bar_width, airport_dropoffs_df['LGA'], bar_width,␣
 ↪alpha=opacity, color='#1F4EC5', label='LGA')
rects2 = ax.bar(x, airport_dropoffs_df['JFK'], bar_width, alpha=opacity,␣
 ↪color='#61F46A', label='JFK')
rects3 = ax.bar(x + bar_width, airport_dropoffs_df['EWR'], bar_width,␣
 ↪alpha=opacity, color='#A23752', label='EWR')

# Set the labels for the x-axis, y-axis, and title of the plot, and the ticks␣
 ↪and tick labels for the x-axis.
ax.set_xlabel('Day of Week', fontsize=14)
ax.set_ylabel('Number of Drop-offs', fontsize=14)
ax.set_title('Number of Uber Drop-offs at LGA, JFK, and EWR by Day of Week␣
 ↪(Uber)', fontsize=16)
ax.set_xticks(x)
ax.set_xticklabels(['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday',␣
 ↪'Saturday', 'Sunday'])

# Add a legend to the plot.
ax.legend(fontsize=12)

# Ensure that all elements of the plot are visible, and display the plot.
fig.tight_layout()
plt.show()
```

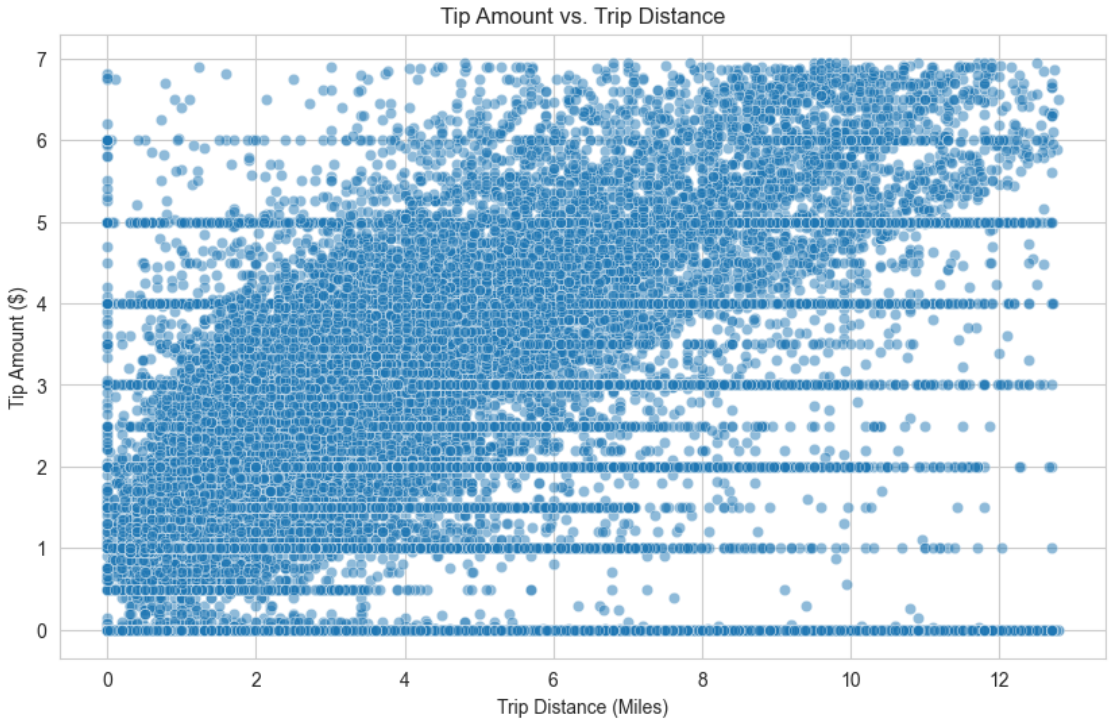


[48]:
```
def visualize_heatmap():
    query = """
    SELECT start_lat, start_lon
    FROM (
        SELECT start_lat, start_lon
        FROM yellow_taxi_database
        WHERE start_lat IS NOT NULL AND start_lon IS NOT NULL
        UNION ALL
        SELECT start_lat, start_lon
        FROM uber_database
        WHERE start_lat IS NOT NULL AND start_lon IS NOT NULL
    );
    """
    df = pd.read_sql_query(query, conn)

    # Center map on NYC
    nyc_map = folium.Map(location=[40.7128, -74.0060], zoom_start=12)

    # Create a heatmap
    heatmap_data = [[row['start_lat'], row['start_lon']] for index, row in df.
↪iterrows()]
    HeatMap(heatmap_data).add_to(nyc_map)
```

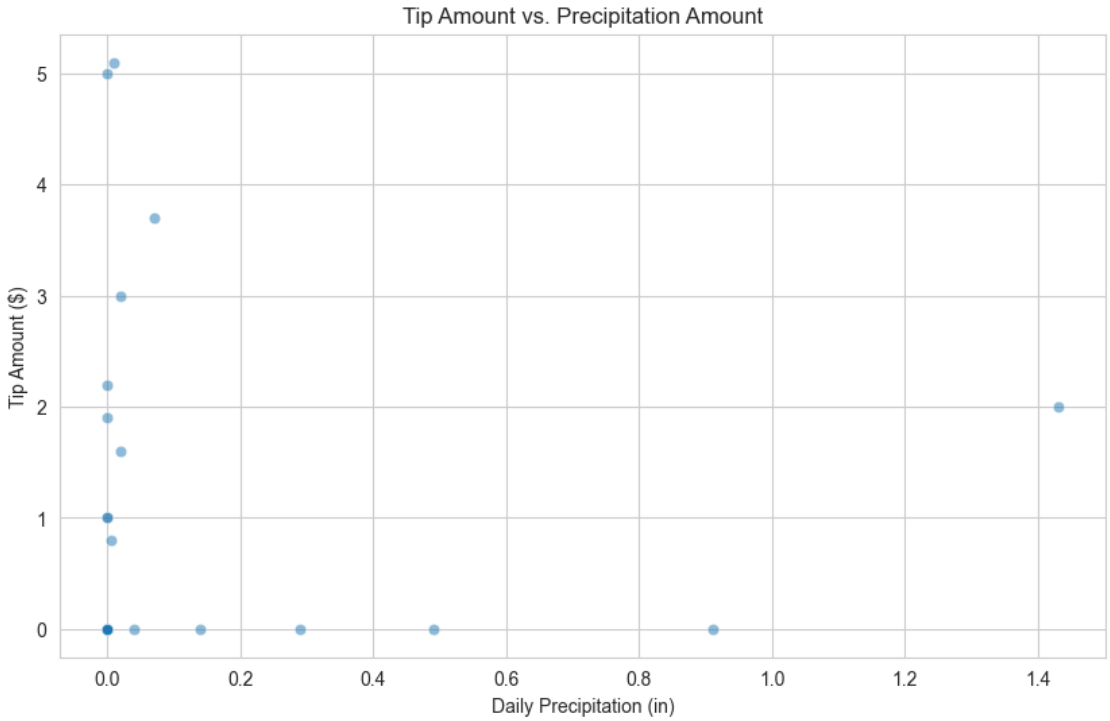
```
    return nyc_map

visualize_heatmap()
```

[48]: <folium.folium.Map at 0x7fd93af01860>

[49]:
```
def visualize_scatter_tip_distance():
    query = "SELECT Tip_Amt, Trip_Distance FROM yellow_taxi_database;"
    df = pd.read_sql_query(query, conn)

    # Remove outliers
    df = df[(np.abs(stats.zscore(df)) < 3).all(axis=1)]

    plt.figure(figsize=(10, 6))
    sns.scatterplot(data=df, x='Trip_Distance', y='Tip_Amt', alpha=0.5)
    plt.title('Tip Amount vs. Trip Distance')
    plt.xlabel('Trip Distance (Miles)')
    plt.ylabel('Tip Amount ($)')
    plt.show()

visualize_scatter_tip_distance()
```



[50]:
```
def visualize_scatter_tip_precipitation():
    query = """
    SELECT Tip_Amt, DailyPrecipitation
    FROM yellow_taxi_database
    JOIN daily_weather_database ON Pickup_Datetime = Record_Time;
    """
    try:
        df = pd.read_sql_query(query, conn)
        df['Tip_Amt'] = pd.to_numeric(df['Tip_Amt'], errors='coerce')
        df['DailyPrecipitation'] = pd.to_numeric(df['DailyPrecipitation'],␣
 ↪errors='coerce')

        # Remove outliers using z-score normalization
        df = df[(np.abs(stats.zscore(df)) < 3).all(axis=1)]

        # Create scatter plot
        plt.figure(figsize=(10, 6))
        sns.scatterplot(data=df, x='DailyPrecipitation', y='Tip_Amt', alpha=0.5)

        plt.title('Tip Amount vs. Precipitation Amount')
        plt.xlabel('Daily Precipitation (in)')
        plt.ylabel('Tip Amount ($)')
        plt.show()
    except Exception as e:
        print("Error:", e)
        print("Query:", query)

visualize_scatter_tip_precipitation()
```



[51]:
```
!pip freeze
```

```
WARNING: Ignoring invalid distribution -p
(/Users/botaozhang/miniconda3/lib/python3.7/site-packages)
WARNING: Ignoring invalid distribution -3p
(/Users/botaozhang/miniconda3/lib/python3.7/site-packages)
WARNING: Ignoring invalid distribution -2p
(/Users/botaozhang/miniconda3/lib/python3.7/site-packages)
WARNING: Ignoring invalid distribution -1p
(/Users/botaozhang/miniconda3/lib/python3.7/site-packages)
WARNING: Ignoring invalid distribution -p
(/Users/botaozhang/miniconda3/lib/python3.7/site-packages)
WARNING: Ignoring invalid distribution -0p
(/Users/botaozhang/miniconda3/lib/python3.7/site-packages)
WARNING: Ignoring invalid distribution -ip
(/Users/botaozhang/miniconda3/lib/python3.7/site-packages)
WARNING: Ignoring invalid distribution -
(/Users/botaozhang/miniconda3/lib/python3.7/site-packages)
absl-py==1.2.0
alembic==1.10.3
appdirs==1.4.4
appnope @ file:///opt/concourse/worker/volumes/live/4f734db2-9ca8-4d8b-5b29-6ca1
5b4b4772/volume/appnope_1606859466979/work
asn1crypto==0.24.0
astroid==2.2.5
astunparse==1.6.3
async-generator==1.10
atari-py==0.1.15
attrs==19.3.0
backcall @ file:///home/ktietz/src/ci/backcall_1611930011877/work
backports.zoneinfo==0.2.1
beautifulsoup4==4.12.0
bleach==3.1.1
box2d-py==2.3.8
branca==0.6.0
bs4==0.0.1
cachetools==4.1.0
catzzz==0.1.6
certifi @ file:///private/var/folders/sy/f16zz6x50xz3113nwtb9bvq00000gp/T/abs_47
7u68wvzm/croot/certifi_1671487773341/work/certifi
certipy==0.1.3
```

```
cffi==1.12.2
chardet==3.0.4
charset-normalizer==2.1.1
click==8.0.3
click-plugins==1.1.1
cligj==0.7.2
conda==23.1.0
conda-package-handling @ file:///opt/concourse/worker/volumes/live/d106838d-eaa7
-40fd-5437-9d95a7db5458/volume/conda-package-handling_1618262135990/work
cryptography==2.6.1
cycler==0.10.0
Cython==0.29.10
debugpy==1.6.7
decorator @ file:///opt/conda/conda-bld/decorator_1643638310831/work
defusedxml==0.6.0
dnspython==2.3.0
entrypoints @ file:///opt/concourse/worker/volumes/live/194c0a28-55ce-4e83-6a87-
0d9f2e06ab2c/volume/entrypoints_1649926487944/work
fastjsonschema==2.16.3
filelock @ file:///tmp/build/80754af9/filelock_1638521398314/work
Fiona==1.9.3
flake8==5.0.4
Flask==2.2.3
Flask-Cors==3.0.10
flatbuffers==2.0.7
folium==0.14.0
future==0.17.1
fuzzywuzzy==0.18.0
gast==0.3.3
gensim==4.2.0
geojson==3.0.1
geopandas==0.10.2
glfw==1.8.1
gmaps==0.9.0
google-auth==1.16.0
google-auth-oauthlib==0.4.1
google-pasta==0.2.0
greenlet==2.0.2
grpcio==1.29.0
gurobipy==9.5.2
-e git+https://github.com/openai/gym@13f32c7689b9c8738d6db7f6b850ec216cb4d781#eg
g=gym
h5py==2.10.0
huggingface-hub==0.2.1
idna==2.8
imageio==2.5.0
importlib-metadata==6.5.0
importlib-resources==5.12.0
```

```
ipykernel==6.16.2
ipython==7.34.0
ipython-genutils==0.2.0
ipywidgets==8.0.6
isort==4.3.20
itsdangerous==2.1.2
jedi @ file:///opt/concourse/worker/volumes/live/c9d2fa99-8bc1-4572-41e7-6beba63
91441/volume/jedi_1644315238822/work
Jinja2==3.1.2
joblib @ file:///tmp/build/80754af9/joblib_1635411271373/work
json5==0.9.2
jsonschema==3.2.0
jupyter==1.0.0
jupyter-console==6.6.3
jupyter-telemetry==0.1.0
jupyter_client==7.4.9
jupyter_core==4.12.0
jupyterhub==3.1.1
jupyterlab==2.0.1
jupyterlab-pygments==0.2.2
jupyterlab-server==1.0.7
jupyterlab-widgets==3.0.7
keras==2.11.0
Keras-Applications==1.0.8
keras-nightly==2.11.0.dev2022091607
Keras-Preprocessing==1.1.0
kiwisolver==1.1.0
lazy-object-proxy==1.4.1
libclang==14.0.6
lockfile==0.12.2
lxml==4.9.1
Mako==1.2.4
Markdown==3.2.2
MarkupSafe==2.1.2
matplotlib==3.2.0
matplotlib-inline @ file:///private/var/folders/sy/f16zz6x50xz3113nwtb9bvq00000g
p/T/abs_9ddl71oqte/croots/recipe/matplotlib-inline_1662014471815/work
mccabe==0.7.0
mistune==2.0.5
mkl-fft==1.0.15
mkl-random==1.1.0
mkl-service==2.3.0
# Editable install with no version control (mujoco-py==1.50.1.68)
-e /Users/botaozhang/miniconda3/lib/python3.7/site-packages
multitasking==0.0.11
munch==2.5.0
nbclient==0.7.3
nbconvert==7.3.1
```

```
nbformat==5.8.0
neo4j==5.6.0
nest-asyncio @ file:///private/var/folders/sy/f16zz6x50xz3113nwtb9bvq00000gp/T/a
bs_64pfm74mxq/croot/nest-asyncio_1672387129786/work
nltk==3.6.5
notebook==6.0.3
numpy==1.21.6
oauthlib==3.1.0
opencv-python==4.1.0.25
opt-einsum==3.2.1
packaging @ file:///tmp/build/80754af9/packaging_1637314298585/work
pamela==1.0.0
pandas==1.3.5
pandocfilters==1.4.2
parrot==1.0
parso @ file:///opt/conda/conda-bld/parso_1641458642106/work
patsy==0.5.3
pexpect @ file:///tmp/build/80754af9/pexpect_1605563209008/work
pickleshare @ file:///tmp/build/80754af9/pickleshare_1606932040724/work
Pillow==6.0.0
pipenv==2018.11.26
plotly==5.10.0
pluggy @ file:///opt/concourse/worker/volumes/live/b4f0e253-7a56-4c04-781c-49509
af26e8e/volume/pluggy_1648042585672/work
prometheus-client==0.7.1
prompt-toolkit==3.0.38
protobuf==3.12.2
psutil==5.9.5
psycopg==3.1.8
psycopg-binary==3.1.8
ptyprocess @ file:///tmp/build/80754af9/ptyprocess_1609355006118/work/dist/ptypr
ocess-0.7.0-py2.py3-none-any.whl
py4j==0.10.9.7
pyarrow==11.0.0
pyasn1==0.4.8
pyasn1-modules==0.2.8
pycodestyle==2.9.1
pycosat==0.6.3
pycparser==2.19
pyflakes==2.5.0
pyglet==1.3.2
Pygments @ file:///opt/conda/conda-bld/pygments_1644249106324/work
pylint==2.3.1
pymongo==4.3.3
pyOpenSSL==19.0.0
pyparsing @ file:///tmp/build/80754af9/pyparsing_1635766073266/work
pyproj==3.2.1
pyrsistent==0.15.7
```

```
PySocks==1.6.8
pyspark==3.4.0
python-dateutil @ file:///tmp/build/80754af9/python-dateutil_1626374649649/work
python-json-logger==2.0.7
python-Levenshtein==0.12.2
pytube==12.1.0
pytz==2019.3
PyYAML==5.3.1
pyzmq==25.0.2
qtconsole==5.4.2
QtPy==2.3.1
regex==2021.11.10
requests==2.29.0
requests-oauthlib==1.3.0
rsa==4.0
ruamel.yaml.clib @ file:///private/var/folders/sy/f16zz6x50xz3113nwtb9bvq00000gp
/T/abs_c7s0zxy4t2/croot/ruamel.yaml.clib_1666302244557/work
ruamel_yaml==0.15.46
sacremoses==0.0.46
scikit-learn==0.22.2.post1
scipy==1.4.1
seaborn==0.12.2
Send2Trash==1.5.0
sentence-transformers==2.1.0
sentencepiece==0.1.96
shapely==2.0.1
shell-functools==0.3.0
six==1.12.0
smart-open==6.3.0
soupsieve==2.4
spicy==0.16.0
SQLAlchemy==1.3.4
statsmodels==0.13.5
tb-nightly==2.11.0a20220916
tenacity==8.0.1
tensorboard==2.11.2
tensorboard-data-server==0.6.1
tensorboard-plugin-wit==1.6.0.post3
tensorflow==2.11.0
tensorflow-estimator==2.11.0
tensorflow-io-gcs-filesystem==0.27.0
termcolor==1.1.0
terminado==0.8.3
testpath==0.4.4
textblob==0.17.1
tf-estimator-nightly==2.11.0.dev2022091608
tf-nightly==2.11.0.dev20220916
tinycss2==1.2.1
```

```
tokenizers==0.10.3
toml==0.10.0
toolz @ file:///private/var/folders/sy/f16zz6x50xz3113nwtb9bvq00000gp/T/abs_a7gk
swah88/croot/toolz_1667464082910/work
torch==1.10.0
torchvision==0.11.1
tornado @ file:///private/var/folders/sy/f16zz6x50xz3113nwtb9bvq00000gp/T/abs_1f
imz6o0gc/croots/recipe/tornado_1662061695695/work
tqdm @ file:///tmp/build/80754af9/tqdm_1635330843403/work
traitlets==5.9.0
transformers==4.13.0
typing_extensions==4.5.0
urllib3==1.24.1
virtualenv==16.6.0
virtualenv-clone==0.5.3
wcwidth @ file:///Users/ktietz/demo/mc3/conda-bld/wcwidth_1629357192024/work
webencodings==0.5.1
Werkzeug==2.2.3
widgetsnbextension==4.0.7
wrapt==1.11.1
yfinance==0.1.86
zipp @ file:///private/var/folders/sy/f16zz6x50xz3113nwtb9bvq00000gp/T/abs_b71z7
9bye2/croot/zipp_1672387125902/work
```

[ ]: