Introduction

Welcome to our final project! The purpose of this project is to prepare you for the kind of research work you might find yourself doing after graduation. A lot of analytics work is working with large amounts of data and trying to make sense of it all.

One overarching goal of this project is to give you something to point to when applying for internships and job opportunities. You should feel welcomed to include a link to your project repository in your resumes and CVs.

Heads up!

This project will take about 2 to 3 times longer than the weekly homeworks. To help compensate, there will be no homework assigned for the last two modules. However, you are advised to start this project early.

Most of your time will most likely be spent on Part 1.

There may be small updates like clarifications made to this document. Any updates made will be highlighted in pale yellow with a date of when it was made.

Logistics

Who

Your teams will be made up of 2 people of your choosing. You will have until end-of-day on November 4th to find group members. If you prefer a surprise, you can wait until the end of the week and you will be assigned to a group by me or a CA.

Add your group to this spreadsheet. This will be inputted into Canvas as project groups for you.

What

You will be turning in a URL to the repository where your project code can be found. Provide the HTTPS link, *not* the link to clone the URL.[1] We suggest using GitHub to host your repository, but you may also use BitBucket or GitLab.

See Requirements & Constraints for exactly what is needed and expected.

---

[1] The HTTP link looks like `https://github.com/YourUsername/YourProject`. The clone URL looks like `git@github.com/YourUsername/YourProject.git` or `https://github.com/YourUsername/YourProject.git`

When

Group selection is due **Friday, November 4th at 11:59pm ET** via this spreadsheet; otherwise we'll choose a group for you.

The project is due **Sunday, December 11th at 11:59pm ET**. No late submissions will be accepted.

5% additional extra credit will be awarded if projects are completed and submitted a week earlier, by Sunday, December 4th at 11:59pm ET. See the extra credit section for more extra credit opportunities.

Problem Overview

In March 2022, Uber announced that they will allow its New York City-based users to hire yellow taxis from its app. While we each all have our own preferences for how we get home on a late night, let's see if we can see some trends to back up why Uber and taxis have clashed for years.

Using hired-ride trip data from Uber and NYC Yellow cab from January 2009 through June 2015, and joining with local historical weather data, you will be providing a single Jupyter notebook and SQL queries to answer particular questions by downloading and cleaning data, storing the data in a SQLite database, and providing visualizations.

Datasets

Updated Oct 30, 2022 *: Yellow Taxi trip data has been updated to a format called Parquet, instead of CSV.*
Updated *Nov 14, 2022* *: Shp ("shape") file information*

- Uber rides: a sample of Uber rides from 01-2009 through 06-2015 can be downloaded from here.
- Historical weather data: the complete catalog of local climatological data for 2009 through 2015, plus the dataset's documentation, can be found in this folder.
- Yellow Taxi trip data: use this URL to download the ~~CSV~~ Parquet files needed for the project, as instructed in Part 1 below.
    - It may be helpful to read the "Trip Record User Guide" and "Yellow Trips Data Dictionary" under the "Data Dictionaries and MetaData" section towards the bottom.
    - For some months, the pickup and dropoff locations are given as IDs, and not latitude/longitudes. Find & download the "Taxi Zone Shapefile (PARQUET)" file among the NYC Taxi record data site. It will be a zip file. Unzip the zip file, and place all the contents in the same directory as your project's notebook. *(updated Nov 14, 2022).*
    - We will *not* be using data for Green Taxi trips or FHV trips.

- Each Parquet file may take a few minutes or more to download, depending on your internet speed.

Specifications

Read these specifications closely. Failure to follow them will result in points being deducted.

Create a single Jupyter Notebook that is divided into four parts: Data Preprocessing, Storing Data, Understanding Data, and Visualizing Data.

For each part, use Markdown cells to describe what is going on. Construct and write your notebook as if you could hand it to a (not necessarily Python-inclined) friend, and they could understand it without needing to read the actual code.

When implementing your solutions to each part, write separate functions or specific tasks that help accomplish the goals of each part. This will make it easier to test different parts of the logic independently. Then in separate cells, execute the functions (so, don't just define functions; call them as well). I should be able to clear the output, then run all cells successfully to generate everything needed below.

Of course, you're encouraged to experiment, understand, and play around with the data and your code's implementation. Be as messy as you need. Create multiple notebooks if you want as you're exploring. The final result should be clean, readable, and tell a logical story in a single notebook.

Part 1: Data Preprocessing

Overview: For Part 1, you will be downloading Parquet files (both by hand and with Python code), cleaning and filtering for the relevant data, filling in missing data, and generating samples of these datasets.

Downloading: You have been provided links above to all the data needed. However, you will need to programmatically download the Yellow Taxi trip data (as in, write code in your notebook that downloads the Parquet files). Using the `re` module, write a regular expression to help pull out the desired links for Yellow Taxi Parquet files. You are required to use the 3rd-party packages `requests` and `BeautifulSoup`[2] to programmatically download the Yellow Taxi Parquet files.

Cleaning & filtering: You must use the `pandas`[3] and `geopandas` [4] packages to load and clean all the datasets. The process of cleaning & filtering the data should include:

---

[2] Both `requests` and `BeautifulSoup` are introduced in Module 8: "HTTP; REST APIs; Data formats & serialization"
[3] `pandas` is introduced in Module 11: "Data science with numpy and pandas"
[4] `geopandas` will not be introduced, but once familiar with pandas, it should be easy to pick up.

- looking up the latitude and longitude for some months where only location IDs are given for pickups and dropoffs (use the shp file linked above) – some location IDs may not be valid so those specific trips can be removed; *(updated Nov 3, 2022)*
- removing unnecessary columns and only keeping columns needed to answer questions in the other parts of this project;
- removing invalid data points (use your discretion!);
- normalizing column names;
- normalizing and using appropriate column types for the respective data;
- for Uber and Yellow Taxi data, removing trips that start and/or end outside of the following latitude/longitude coordinate box: (40.560445, -74.242330) and (40.908524, -73.717047).

Sampling: Each month of Yellow Taxi data contains millions of trips. However, the provided Uber dataset is only a sampling of all data. Therefore, you will need to generate a sampling of Yellow Taxi data that's roughly equal to the sample size of the Uber dataset.

Missing data: To help answer the questions later in the project, you will need to add to the datasets missing information that can be calculated. Mainly: the distance between a trip's starting point and ending point (as the crow flies, don't word about the exact path taken).

Therefore, you must define a function that calculates the distance between two coordinates in kilometers that only uses the `math` module from Python's standard library.

Then, use that distance calculation function to add a column to each Uber and Yellow Taxi dataset that contains the distance between the pickup and dropoff location.

Tips for Part 1:

- The Yellow Taxi Parquet files are huge – take a look at the section in the Appendix about dealing with a large amount of data.
- Relatedly, make use of your `.gitignore` file to avoid committing the large Yellow Taxi dataset Parquet files (and other unnecessary files) to your repo.
- The shp file that contains the location IDs for pickups/dropoffs contain geometric zones, and not specific latitude/longitude coordinates. You can use the "center" of the zones (polygons) as a point to look up the coordinates. Take a look at this StackOverflow answer to figure out how exactly to get those coordinates using geopandas. *(updated Nov 3, 2022)*
- Read ahead to figure out which columns are absolutely necessary for each dataset.
- Be mindful of the data types for each column, which will make it easier for yourself when storing and filtering data later on.
- You may find that the weather data you need later on does not exist at the frequency needed (daily vs hourly). You may calculate/generate samples from one to populate the other. Just document what you're doing so we can follow along.

Part 2: Storing Data

Overview: For Part 2, you will be taking the sample datasets generated from Part 1, and populating a SQLite database with tables generated from the datasets.

First, using `SQLAlchemy`[5], create a `SQLite`[6] database with which you'll load in your preprocessed datasets.

Next, create and populate four tables: one for your sampled datasets of Yellow Taxi trips, one for Uber trips, one for hourly weather information, and one for daily weather information. Use appropriate data types for each column.

Finally, create a `schema.sql` file that defines each table's schema. You can use `SQLAlchemy` within the notebook to help generate this file, another programmatic approach, or create this schema file by hand. You may elect to complete this either before or after you've populated your tables.

Tips for Part 2:

- Take a look at the Appendix for an example of a schema file.
- I should be able to run this schema file to create the tables in a database via the SQLite CLI tool. That is, I should be able to run the following command in a Jupyter notebook cell to create a database with the four required tables (it is not expected that you do this yourself for the project, but this is a good sanity check for it to succeed without error):

```
!sqlite3 project.db < schema.sql
```

- Some `pandas` functionality that might be helpful to look into (and not just for Part 2) (not exhaustive of what could be helpful):
  - `pd.read_sql_query` - read data from querying a SQL table
  - `pd.read_sql_table` - read entire SQL table
  - `df.to_sql` - add data from the dataframe to a SQL table
  - `pd.to_numeric` - Convert argument to a numeric type
  - `pd.concat` - Concatenate pandas objects along a particular axis with optional set logic along the other axes
  - `pd.merge` - Merge DataFrame or named Series objects with a database-style join
  - `pd.merge_asof` - Perform a merge by key distance. This is similar to a left-join except that we match on the nearest key rather than equal keys. Both DataFrames must be sorted by the key.

[5] `SQLAlchemy` is introduced in Module 10: "SQL fundamentals con't; SQLAlchemy"
[6] `SQLite` is introduced in Module 9: "Relational Databases; SQL fundamentals"

Part 3: Understanding Data

Overview: In Part 3, you will be crafting a set of SQL queries to develop a better understanding of the datasets we're working with.

For this part, define a SQL query for each of the following questions - one query per question. Save each query as a `.sql` file, naming it something illustrative of what the query is for, e.g. `top_10_hottest_days.sql`.

1. For 01-2009 through 06-2015, what hour of the day was the most popular to take a Yellow Taxi? The result should have 24 bins.
2. For the same time frame, what day of the week was the most popular to take an Uber? The result should have 7 bins.
3. What is the 95% percentile of distance traveled for *all* hired trips during July 2013?
4. What were the top 10 days with the highest number of hired rides for 2009, and what was the average distance for each day?
5. Which 10 days in 2014 were the windiest on average, and how many hired trips were made on those days?
6. During Hurricane Sandy in NYC (Oct 29-30, 2012), plus the week leading up and the week after, how many trips were taken each hour, and for each hour, how much precipitation did NYC receive and what was the sustained wind speed? There should be an entry for every single hour, even if no rides were taken, no precipitation was measured, or there was no wind.

For each query, be sure to execute it in the notebook so we can see your answers to the question.

Note: we will *not* be checking for exactness with the results. We're only concerned with the construction of your queries.

Tips for Part 3:

- You may wish to use **`SQLAlchemy`** within the notebook to help craft these queries and query files. You can also use **`pandas`** to help check the validity of your queries.
- You may want to familiarize yourself with the **`WITH <name> AS`** and **`WITH RECURSIVE <name> AS`** expressions in SQL. This is a good resource with a lot of examples to help get familiar.
    - There are quite a few examples using **`UNION ALL`** - this will help "flatten" tables that have common columns (e.g. fare amount, pickup dates, etc) into a singular shared column (for example, taxi fares and uber fares are in one column).
    - Look at the example query that starts with **`WITH RECURSIVE dates(x) AS`** for help with the 6th query.
- You may also want to familiarize yourself with the COALESCE expression in SQL. This is a decent tutorial to look through.

- This Stack Overflow post will be helpful when crafting queries for figuring out percentiles/quantiles.
- This Stack Overflow post will be helpful for grouping results by timeframe, e.g. by hour, day.

Part 4: Visualizing Data

Overview: For Part 4, you will be creating visualizations to enhance your understanding of the datasets.

For this final part, you will be creating a bunch of visualizations embedded in your notebook using `matplotlib`[7] and/or other visualization libraries of your choice.

This is where you can get creative with the look and feel of each visual. All that is required is that each visualization is immediately understandable without necessarily needing to read its associated function (i.e. labeled axes, titles, appropriate plot/graph/visual type, etc). You're welcome to use Markdown cells to introduce and/or explain each visualization.

You can use `pandas` to help parse data before generating a visualization, but you ***must*** read the data from your `SQLite` database; do not read data directly from Parquet files. That is, you are able to use `pandas` dataframes to help with your visualization. But you should be creating those dataframes from querying the SQL tables you need and not by reading from a Parquet file.

Create the following 6 visualizations. You must define a function for each visualization, then call these functions in separate cells to render each visualization.

1. Create an appropriate visualization for the first query/question in part 3.
2. Create a visualization that shows the average distance traveled per month (regardless of year - so group by each month) for both taxis and Ubers combined. Include the 90% confidence interval around the mean in the visualization.
3. Define three lat/long coordinate boxes around the three major New York airports: LGA, JFK, and EWR (you can use bboxfinder to help). Create a visualization that compares what day of the week was most popular for drop offs for each airport.
4. Create a heatmap of all hired trips over a map of the area. Consider using KeplerGL or another library that helps generate geospatial visualizations.
5. Create a scatter plot that compares tip amount versus distance for Yellow Taxi rides. You may remove any outliers how you see fit.
6. Create another scatter plot that compares tip amount versus precipitation amount for Yellow Taxi rides. You may remove any outliers how you see fit.

Tips for Part 4:

[7] `matplotlib` is introduced in Module 12: "Data visualization with matplotlib"

- Check out matplotlib's gallery for inspiration on which visualization to choose. Click on a visualization for sample code that generates it.

Requirements & Constraints

Top-level requirements:

- Your project must use Python 3.8 or higher.
- The project must be under Git version control.
- Your notebook must be able to execute completely and successfully in order.
- All code must be clear and follow Python's style guide as outlined in PEP 8[8].
- Any and all functions, classes, and class methods must have docstrings, and follow PEP 257.[9]

Your project will need to include the following:

- Source code
  - One (and only one) Jupyter Notebook. Leave your notebook executed (you don't need to clear the output). GitHub will therefore automatically render all generated visualizations for you.
  - Required SQL files (schema & queries).
- `README.md` file
  - A file describing the project, its use, and its behavior. See the Appendix below for more detail on what's expected.
- `requirements.txt` file
  - This should include any Python package that you've had to `conda install` or `pip install`.
  - Be mindful that the Anaconda installation provides additional 3rd party libraries so you don't have to install them yourself. Your `requirements.txt` file must include anything that Anaconda already makes available.
- `.gitignore` file
  - A file to ignore files that should not be committed (e.g. pycache files, CSV files, Parquet files, `SQLite` files)

Do *not* include:

- Any Yellow Taxi Parquet files - there's no need to include data that will be downloaded by your code.
- Any database/`SQLite` files, other than your schema file and query files.
- Any images of your visualizations - they should be automatically embedded in your notebook when the appropriate cells are executed.
- You can certainly generate the above files as you are working, just do not include them in your repository

---

[8] You might find this resource helpful. This library can help with formatting code within Jupyter Notebook to conform to PEP 8. This library is similar, but for users of Jupyter Lab.
[9] You might find this resource helpful.

    - Tip: make use of the `.gitignore` file.

Grading

The project will be out of 100 points. Everyone in the group will share the same grade.

- 10% - Clarity of code (including PEP8[7] + PEP257[8] style)
- 70% - Meets the above criteria (Specifications, Requirements & Constraints)
- 20% - Thorough use of Git
    - Your team has used Git to save incremental progress in your work. Each member has roughly the same number of commits, additions, and subtractions.

The full detailed grading rubric can be found here.

Extra Credit

You have the opportunity to earn up to 15% in extra credit, for a maximum total of 115 points on the project. You may complete all extra credit opportunities, or pick and choose which you want to complete.

- 2.5%: Add one or more animations (matplotlib animations for example) and/or widgets (Jupyter widgets, matplotlib widgets) for any of your visualizations. You may use any 3rd party packages/libraries if you wish. You may wish to add this to one of the 6 required visualizations, or create a brand new visualization based off of your own question/prompt.
- 2.5%: Define at least one unit test for every function and class method you've defined. While we go over unit testing at the end of the course, you can learn about it on your own with this video tutorial series, this written one or this other written one (open either in incognito if you don't want to make an account with the site).
- 2.5%: Add type hints to all functions & class methods you've defined. Type hinting is covered at the end of the course, so you can learn about it on your own with this video tutorial, this written one (skip over the sections "Pydantic models" and "Type hints in Fast API"), or this one.
- 2.5%: Add an additional SQL table that contains daily sunset/sunrise data (which can be found in the original weather data). Write a question that considers this new table in relation to one or more existing tables, and then create either a SQL query or a visualization that answers that question.
- 5.0%: Submit your project for grading a week early, by Sunday, December 4th at 11:59pm ET.

Academic Integrity

Students are expected to follow Columbia's academic integrity policy. Any academic misconduct discovered will be reported to the Graduate Student Conduct Team and graded accordingly.

Appendix

What's a README?

A README file is a that sits at the root of the project, repo, or directory of an application and describes what the application does and how to use it. It is often in Markdown (`README.md`), reStructuredText (`README.rst`), or simple text (`README.txt`). Here are some good examples of READMEs.

- https://github.com/python/cpython/blob/master/README.rst
- https://github.com/pandas-dev/pandas/blob/master/README.md
- https://github.com/tensorflow/tensorflow/blob/master/README.md

Your README needs to include:

- A description of what has been implemented
    - This only needs to be a paragraph or less of a description. More is welcome but not needed.
- Your group name
    - Eg. Project Group 51
- A list containing the UNI for each member on the team
    - Of the form: "UNIs: [uni1, uni2, uni3]". Eg. "UNIs: [ab1234, cd7847, ef9873]"

Handling a lot of data

This project deals with a lot of data that can be too much for a personal computer to handle effectively.

For instance, each month of yellow taxi data is about *2GB in size*. Therefore, it's imperative that you remove any unnecessary and invalid data, use appropriate data types when possible, and take samples.

Downloading Yellow Taxi data can take a while per file since each file is so large. Consider saving either the original Parquet files and/or the cleaned & sampled data for each month locally to your computer in case you need to step away, and load it back in when you return.

If your notebook is struggling with memory or disk space, you have a few options:

- Download the Parquet files one at a time, either manually or programmatically. If manually, be sure you have code that works for downloading programmatically. Save the Parquet files locally, then once downloaded, use Pandas to read only a certain number of rows in order to get an idea of what to clean. Be sure you still have code that cleans and samples the entire dataset.
- Use the school's computer labs. You can go physically to the labs, or remote into a computer in the lab from your own computer.

- Use a virtual machine on Google Cloud Platform. You will receive credits for using GCP week of `Oct 24, 2022`. Follow this guide for creating a virtual machine (VM) with a Jupyter Notebook environment.
- You might be able to use Google Colab for a remote notebook environment, however it too might be constrained by memory and space. Note that uploading/downloading files might go to the notebook's session storage. Use your GDrive storage to make sure data files you need are persisted.

What's a Schema File?

The example below is copied from this gist:

```
CREATE TABLE IF NOT EXISTS people (
   peopleId INTEGER PRIMARY KEY AUTOINCREMENT,
   gn TEXT,
   sn TEXT,
   birthday DATE
);

CREATE TABLE IF NOT EXISTS peopleLog (
   peopleLogId INTEGER PRIMARY KEY AUTOINCREMENT,
   peopleIdNew INTEGER,
   peopleIdOld INTEGER,
   gnNew TEXT,
   gnOld TEXT,
   snNew TEXT,
   snOld TEXT,
   birthdayNew DATE,
   birthdayOld DATE,
   action TEXT,
   timestamp DATE
);

CREATE TABLE IF NOT EXISTS books (
   bookId INTEGER PRIMARY KEY AUTOINCREMENT,
   title TEXT,
   author INTEGER
);

CREATE TABLE IF NOT EXISTS booksLog (
   booksLogId INTEGER PRIMARY KEY AUTOINCREMENT,
   bookIdNew INTEGER,
   bookIdOld INTEGER,
   titleNew TEXT,
   titleOld TEXT,
   authorOld INTEGER,
```

```
   authorNew INTEGER,
   action TEXT,
   timestamp DATE
);
```