Link to github Repo: https://github.com/Sczaba135/206final

1. Our goal for this project was to utilize two API sources, create three different tables, join them together through a shared integer key, perform calculations using the combined data, and then visualize our findings. We chose historical weather data from New York City and historical crash data from New York City. Our initial idea was to identify matching dates between the datasets, determine whether it snowed on those days, and calculate the average number of crashes on snowy days compared to days without snow.
2. We ended up using different APIs than we initially planned, but we were able to stick to most of our project goals. Due to the volume of data in the crash dataset, which contained approximately 2,140,000 rows, we had to adjust our objectives to make the project more feasible. Instead of focusing solely on snow or no snow, we expanded our scope to include precipitation as a variable, allowing for more meaningful calculations.
   For the weather data, sourced from the Open-Meteo API, we selected the first 121 days of the year 2022 and matched the car crash data to these days. For the crash data, obtained from the NYC Open Data API by Socrata, we narrowed down the dataset because of the high volume of crashes in New York City. To manage this, we selected 25 random crashes for each day. We created three separate tables: car_crash, snowfall_data, and weather_summary. We used weather_id as a shared integer key to join the snowfall_data and weather_summary tables, effectively combining them into one table. We calculated the average number of injuries and deaths for each day and identified days when precipitation or snowfall exceeded 0.1 inches. Finally, we calculated the average number of injuries and deaths per accident for various weather conditions, including snow, rain, and no significant weather event. To present our findings, we created four visualizations to clearly communicate the results.
3. Throughout this project, we encountered numerous challenges at every step. Early on, we realized that our initial APIs were not viable, as they did not provide API keys and only offered data downloads in XML, CSV, or JSON formats, which was not what we wanted. This led us to spend a significant amount of time searching for alternative APIs, ensuring they were either open source or provided keys we could access from their respective websites. Both of us initially struggled to access the data, but we persisted and eventually got the APIs to work. However, we immediately faced another challenge—limiting the

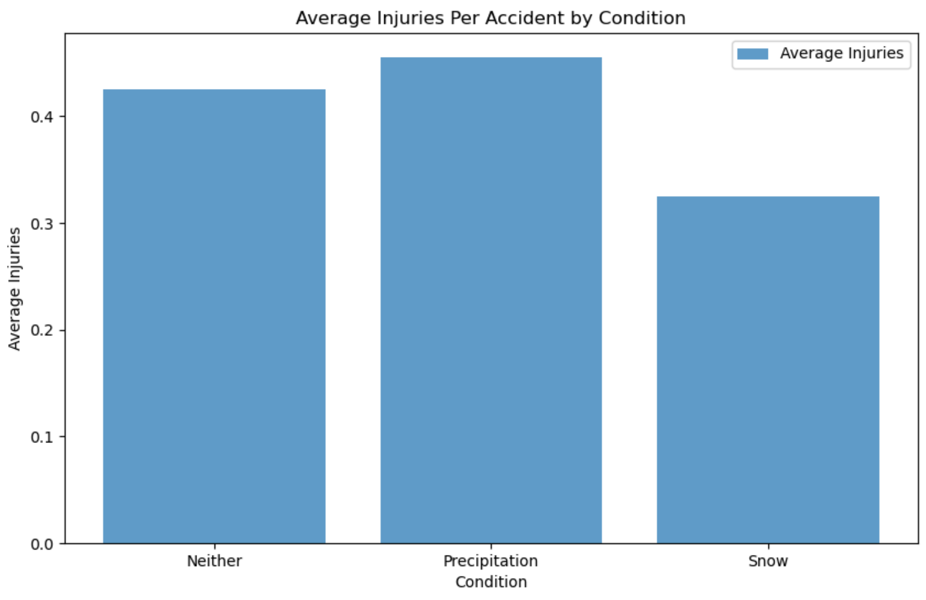
data to 25 or fewer items each time we ran our code. This required us to rework our approach to data retrieval and implement stricter filtering mechanisms. We also had disagreements about what calculations to perform, as feasibility issues arose due to the massive volume of data in our crash API. To address this, we each worked on our own calculations and ultimately chose the one that was more practical and meaningful. From there, we created visualizations based on the data in the resulting text file.

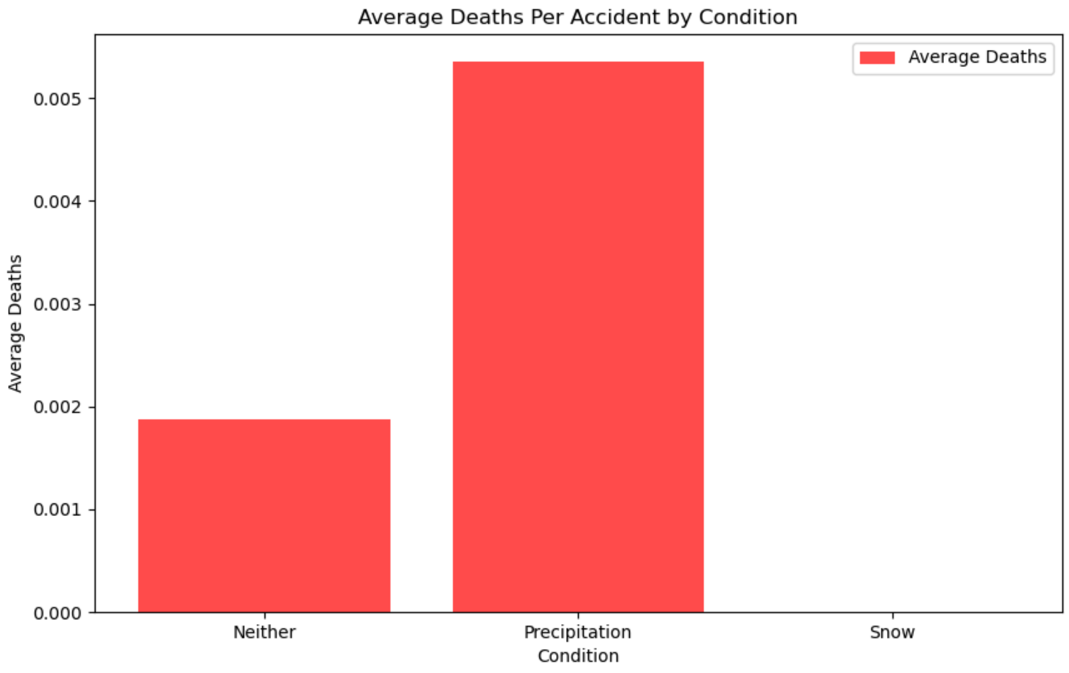
4.

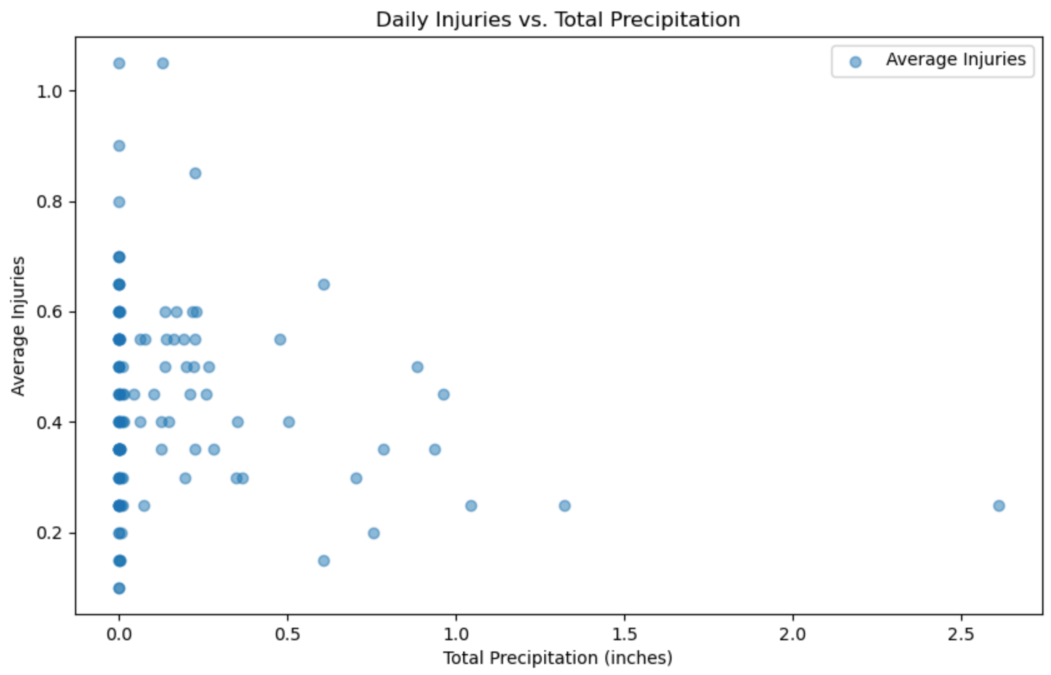
```
import sqlite3
import pandas as pd

# Connect to the database
db_path = 'project_data.db'
conn = sqlite3.connect(db_path)

# Step 1: Calculate daily average injuries, deaths, and snow/precipitation indicators
query_daily = """
SELECT
   dm.date AS date,
   AVG(cc.number_of_persons_injured) AS avg_injuries,
   AVG(cc.number_of_persons_killed) AS avg_deaths,
   MAX(CASE WHEN sd.snowfall_sum > 0.1 THEN 1 ELSE 0 END) AS snow_over_0_1,
   MAX(CASE WHEN ws.precipitation_sum > 0.1 THEN 1 ELSE 0 END) AS
precipitation_over_0_1,
   ws.precipitation_sum AS total_precipitation,
   sd.snowfall_sum AS total_snowfall
FROM car_crash cc
LEFT JOIN day_mapping dm ON cc.date_id = dm.date_id
LEFT JOIN weather_summary ws ON dm.date = ws.date
LEFT JOIN snowfall_data sd ON ws.weather_id = sd.weather_id
GROUP BY dm.date;
"""

# Step 2: Calculate injuries and deaths per accident for different conditions
query_conditions = """
SELECT
   CASE
       WHEN sd.snowfall_sum > 0.1 AND ws.precipitation_sum > 0.1 THEN 'Both'
       WHEN sd.snowfall_sum > 0.1 THEN 'Snow'
       WHEN ws.precipitation_sum > 0.1 THEN 'Precipitation'
       ELSE 'Neither'
   END AS day_condition,
   AVG(cc.number_of_persons_injured) AS avg_injuries_per_accident,
   AVG(cc.number_of_persons_killed) AS avg_deaths_per_accident
FROM car_crash cc
LEFT JOIN day_mapping dm ON cc.date_id = dm.date_id
LEFT JOIN weather_summary ws ON dm.date = ws.date
LEFT JOIN snowfall_data sd ON ws.weather_id = sd.weather_id
GROUP BY day_condition;
"""

# Step 3: Additional optional categories
query_totals = """
SELECT
   COUNT(*) AS total_accidents,
   SUM(cc.number_of_persons_injured) AS total_injuries,
   SUM(cc.number_of_persons_killed) AS total_deaths
FROM car_crash cc;
"""

# Execute queries
daily_results = pd.read_sql_query(query_daily, conn)
conditions_results = pd.read_sql_query(query_conditions, conn)
totals_results = pd.read_sql_query(query_totals, conn)

# Write the results to a text file
output_file = 'accident_analysis_results.txt'
with open(output_file, 'w') as f:
   # Daily Averages
   f.write("Daily Averages and Snow/Precipitation Indicators:\n")
   f.write(daily_results.to_string(index=False))
   f.write("\n\n")

   # Injuries/Deaths Per Condition
   f.write("Average Injuries and Deaths Per Accident by Condition:\n")
   f.write(conditions_results.to_string(index=False))
   f.write("\n\n")

   # Totals
   f.write("Total Accidents, Injuries, and Deaths:\n")
   f.write(totals_results.to_string(index=False))
```

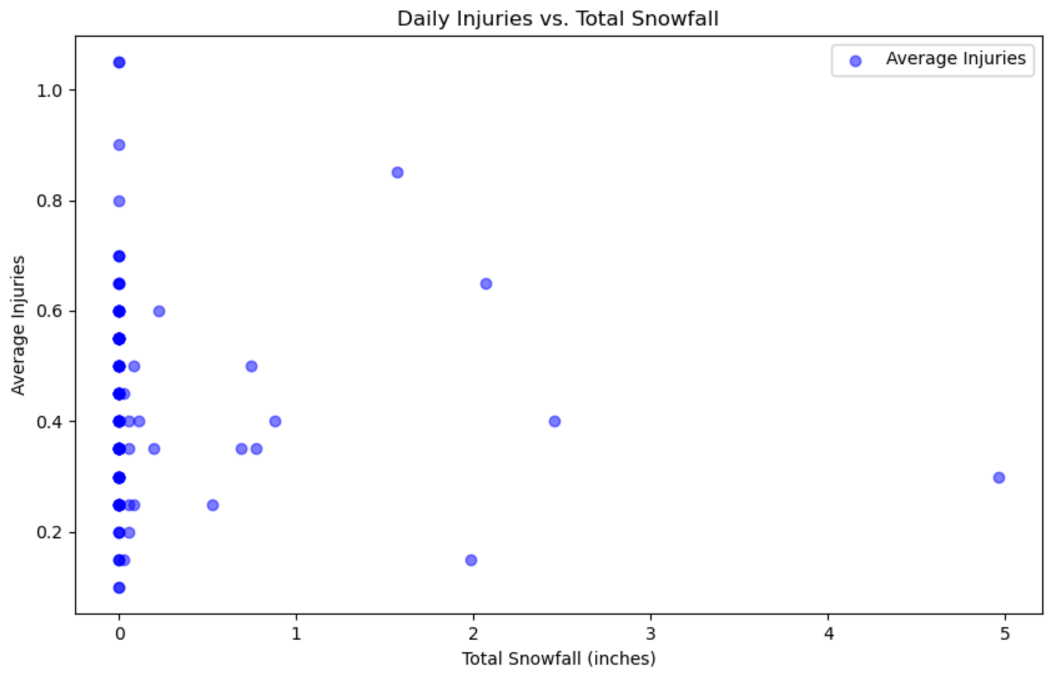
```
# Close the database connection
conn.close()
```

5. All metrics of precipitation/snowfall are in inches* and this is the average over the first 121 days in the year









6. Instructions for running code:

1. Open Github Repository and clone repository to local file

2. Then you will want to run the weather_data_api.py 5 times to get 121 days/rows of data to populate in DB Browser for SQLite.
3. Run car_crash_api.py 121 times (I know its a lot) to have 20 rows of car crashes populate for every single day for the first 121 days of the year and that will populate into DB Browser for SQLite.
4. Then run the analysis.py and have the accident_analysis_data.txt populate.
5. Then you are going to run visuals.py to get all four visuals to populate.

7. Each Function and what it does
    1. Weather Data Api.py
        a. fetch_weather_data_for_months()
            i. Description: Retrieves weather data for entire year from the weather api and process into pandas dataframe
                1. Input none
                2. Output: date, temp_max, temp_min, precipitation_sum, snowfall_sum
        b. setup_databse()
            i. Description: This sets up the SQLite database by creating the necessary tables:
                1. Input is none
                2. Output: creates two tables: weather_summary, and snowfall_data
        c. insert_weather_data(dataframe, limit=25)
            i. Description: this function inserts weather data from dataframe into the SQLite database, adding at most limit rows during each execution to prevent more than 25 rows
                1. Input: daily weather data frame: date, temp_max, temp_min precipitation, snow, limit
                2. Output: adds limit to rows in SQLite, adds data into weather_summary and adds data into snowfall_data
        d. retry(session, retries=5, backoff_factor=0.2)
            i. input : session (an http session), retries (the maximum number of times to retry a failed request), and back_off factor(how long to wait between retries)
            ii. Output: a wrapped session that automatically retries failed requests returning the normal responses (we had chat make this function) .
        e. (if __name__ == "__main__":)
            i. Input none
            ii. Output: executes the previous 3 functions
    2. Car_crash_api.py
        a. setup_database()
            i. Purpose is to create database tables (car crash and day mapping)
                1. Input none

            2. Output: the two tables mentioned above
    b. get_next_date_id()
        i. Fetches the next available date_id from day mapping
            1. Input is none
            2. Output is an integer (next_date_id)
    c. fetch_and_insert_car_crah_data()
        i. Gets crash data from the api and inserts it into car crash (max 20 rows at a time)
            1. Input is api data
            2. Output is rows added to tables
3. visuals.py (no explicit functions in this but don't want to get marked down)
    a. open(), read(), split(), strip(), dataframe(), astype(), figure(), bar(),scatter(), title(), xlabel(),ylabel(), legend(), savefig(), close()
    b. Data parsing
    c. Bar charts for injuries and deaths
    d. Data filtering for snow and precipitation days
    e. Barcharts for snowfall and precipitation bins
4. analysis.py
    a. pd.read_sql_query(): to execute SQL queries on the database and return results as pandas dataframe (learned this in si 305)
    b. open(): self-explanatory
    c. f.write(): self-explanatory
    d. conn.close(): close SQL database

Date	Issue Description	Location of Resource	Result (did it solve the issue?
12/9/2024	API collection	https://weatherstack.com/	no
12/9/2024	API collection	https://catalog.data.gov/dataset/motor-vehicle-collisions-crashes	no

12/9/2024	API collection	Best Weather API : r/learnpython reddit	no
12/11/24	API Collection	weather api	Yes
12/11/24	API Collection	NY crash API	Yes
12/11/24	Loading data into weather_summary and limiting to 25 rows per run	Chat GPT	Yes
12/11/2024	Loading data into car_crash_api and limiting to 25 rows per run	Chat GPT	Yes
12/11/2024	Writing SQL functions	W3 Schools https://www.w3schools.com/SQL/deFault.asp	Kind of but not really
12/11/2024	Learning about SQLite and the database	SQLite Documentation SQLite https://www.sqlite.org › docs	No
12/11/2024	Learning about SQLite and the database	DB Browser for SQLite	Yes this was a great website on sql database
12/11/2024	Calculation help	Chat GPT	Yes/No
12/12/2024	Visual ideation	seaborn.pydata.org	Kind of helped
12/12/2024	Creating visuals	ChatGPT	yes