REPORT

Will Allison, Henry Hunt, Jackson Vontz

The goals for your project including what APIs/websites you planned to work with and what data you planned to gather (10 points)

Original Goals

Our original plan was to all work with car related APIs and gather information like average rating per car brand, AVG MPG per car make, etc. Our plan quickly changed once almost all of our original apis (https://vpic.nhtsa.dot.gov/api/ , https://www.kbb.com/ , https://consumerreports.org/, these are the parent websites) would not let us gain access unless approved by a dealership, which none of us wanted to go through the hassle to do. Individually, we sought out new APIs to use.

2. The goals that were achieved including what APIs/websites you actually worked with and what data you did gather (10 points)

Achieved Goals

Henry: I decided to work with the Washington Metro System (WMATA) api. I gathered each train currently operating and assigned it an ID, the time until its next station stop, the destination string and assigned each destination an integer ID, the length of each train, and the track number the train is arriving on. The goal was to show how many trains are operating on each line at one time and the average length of trains operating on each line. I added the number of cars in each train on the line then divided by the number of trains on the line.

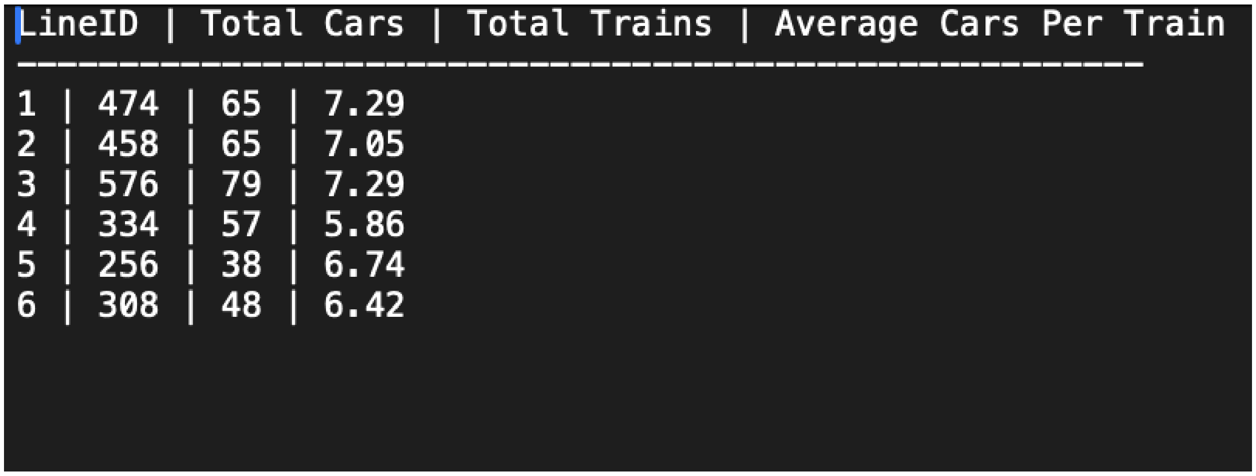
3. The problems that you faced (10 points)

Problems we Encountered

The main problems we encountered were accessing APIs and inserting the data into the database. Our biggest struggle with the NHTSA api was making it only add 25 rows at a time, seeing as the API typically uploaded thousands of rows of data when run. In the end this was the most satisfying thing to figure out, and allowed the file to run much more smoothly and made the graph easier to read. Additionally, formatting the tables to allow for no string duplicates was complicated at the beginning, requiring foresight we were not originally prepared for. This happened with the REST API and the train API. Coding as a team was also harder than expected, and required teamwork and collaboration. Setting up the database as a whole was the hardest part. Being able to find and navigate the API website to understand the variable names and ways to retrieve them was tricky. Once that happened, making sure you both fetched and stored the data proved to be a challenge, but in the end with practice and repetition became easier. The biggest challenge and problem we encountered was all creating our own dbs for each API. We overcame this but originally were very concerned as the deadline was approaching quickly.

4. The calculations from the data in the database (i.e. a screenshot) (10 points)
Calculations (5 total, 3 for project, 1 EC)

Hunt (1)
Average Cars Per Train = Total Cars/Total Trains



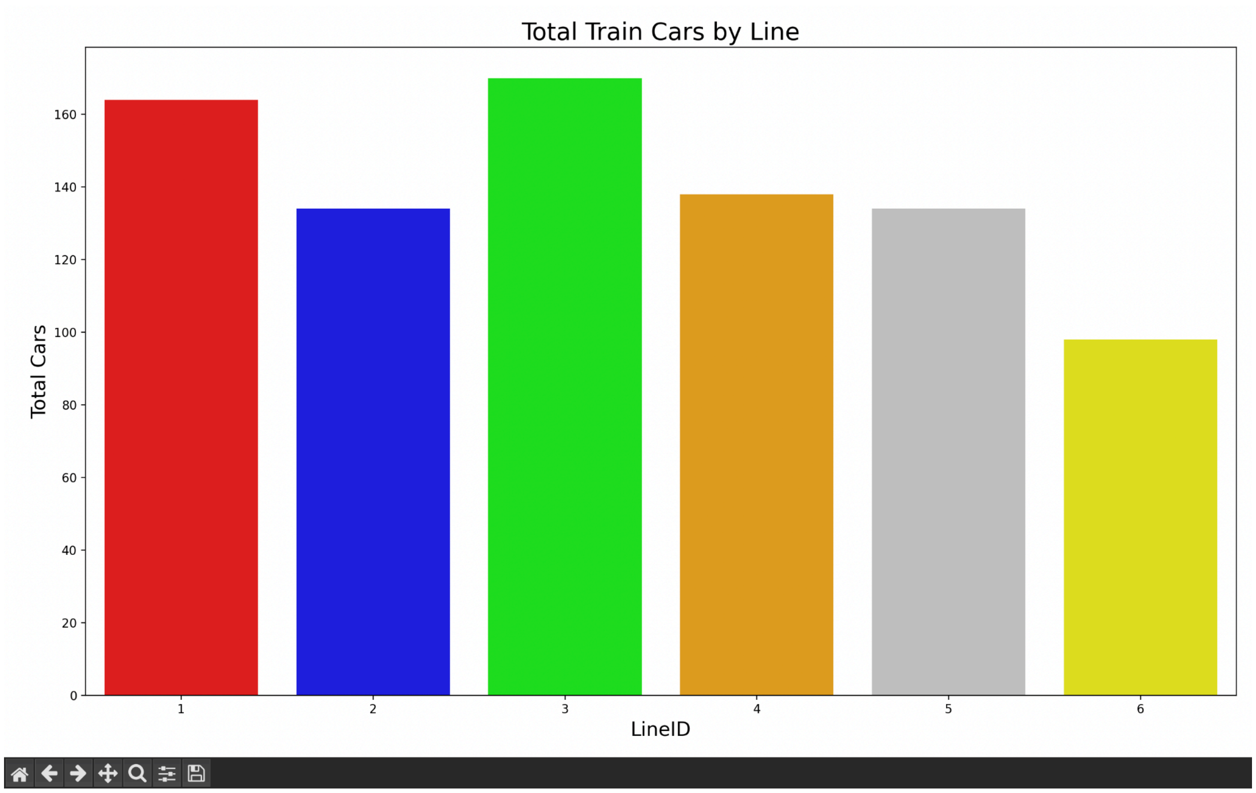
```
LineID | Total Cars | Total Trains | Average Cars Per Train
-------------------------------------------------------
1 | 474 | 65 | 7.29
2 | 458 | 65 | 7.05
3 | 576 | 79 | 7.29
4 | 334 | 57 | 5.86
5 | 256 | 38 | 6.74
6 | 308 | 48 | 6.42
```

NHTSA
Models per Make Count

5. The visualization that you created (i.e. screenshot or image file) (10 points)

Visualizations (5 total, 3 for project, 2 EC)

Hunt



6. Instructions for running your code (10 points)
Instructions for running your code

Hunt

Very straightforward to run. To clear the database uncomment drop table if exists then once run for the first time and first 25 items populate then you can comment again. You can then run over and over again to get data. The calculations file you just simply run and it will automatically overwrite the previous calculations file and create a new visual.

7. Documentation for each function that you wrote. This includes describing the input and output for each function (20 points)

Hunt

1. **fetch_train_data**

Fetches real-time train arrival data from the Washington Metropolitan Area Transit Authority (WMATA) API.

- Parameters:
    - api_key (str): The API key required to authenticate requests.
    - station_codes (list): A list of station codes to fetch data for.
    - limit (int): Maximum number of train data entries to return.
- Returns:
    - list: A list of dictionaries containing train data if the request is successful; each dictionary includes train details such as destination, time until arrival, etc.
    - If the request fails, prints an error message specific to the station code that failed.

2. **create_destination_id_map**

Creates a mapping from train destination names to unique identifiers.

- Parameters:
    - train_data (list): A list of dictionaries, each containing train information.
- Returns:
    - dict: A dictionary where keys are destination names and values are unique destination IDs, sorted alphabetically by destination name.

3. **create_database**

Sets up and populates an SQLite database with train prediction data.

- Parameters:
  - train_data (list): A list of dictionaries containing train data.
  - line_id_map (dict): A dictionary mapping line codes to line IDs.
  - destination_id_map (dict): A dictionary mapping destination names to destination IDs.
- Output:
  - Creates a table TrainPredictions if it does not already exist.
  - Inserts data into the TrainPredictions table.
  - Prints the total number of records in the database after insertion.

4. main

Fetches then processes and inserts train data into database.

- Output:
  - Fetches train data using the fetch_train_data function.
  - Generates line and destination ID mappings.
  - Creates a database and inserts fetched data using the create_database function.
  - Prints a message upon successful loading of data into the SQLite database.