Car Recommendations and Bike Data Projects

Joe Rummel

Problems

- **Car Recommender System** - TrueCar would like to build a vehicle recommendation engine to suggest other vehicles a user should look at when they're browsing TrueCar.com.
- **Accessing and Analyzing Data from an API** - Collect data from the BikeStats API and see what changes have occurred from the last time the data was collected.

Car Recommender

- Runs from the terminal by typing “python recommend_engine.py”

- Reads the data into a Pandas Data Frame

- Asks user to enter a Trim ID from the keyboard

  - There is error checking

  - Trim ID is then found in the Data Frame and the row is stored in its own Data Frame

Car Recommender - Continuous Columns

- There were 4 columns with continuous data
- The columns with continuous data had to be scaled
  - “msrp" and “invoice” column values were in the tens of thousands
  - “pct_discount” column was between 0 and 1
  - “N_ind_transactions” was between 0 and 100
  - “pct_discount” would have little to no effect on the recommender since it is so much smaller than “msrp” and “invoice”, hence the need for scaling

Car Recommender - Discrete Columns

- “tc_body” column
    - Filtered the Data Frame to only include rows that had the same tc_body as the row identified by the Trim ID the user entered
    - If the user is looking for sedans, it makes no sense to recommend a truck
        - Filtered out the row identified by the Trim ID the user entered, so that row does not get recommended
- “trim” column
    - Created a new column in the Data Frame called “trim_similarity”
    - “trim_similarity” was a count of how many words in the “trim” column for a row were also in the “trim” column identified by the Trim ID the user entered.
- “year”, “make”, “model”
    - Did not use these columns

Car Recommender - Results

- Added a “Distance” column to the filtered Data Frame
- “Distance” is the Euclidean distance between a row in the Data Frame and the row identified by the Trim ID the user entered
- Paramaters were “msrp”, “invoice”, “pct_discount”, “n_ind_transactions”, and “trim_similarity”
- Sorted the Data Frame by distance in ascending order
- Outputted the top 5 Trim IDs

Car Recommender - Improvements

- I had to make assumptions of what was important to the user (i.e. price and number of sales had more of an influence than other parameters. Also, I didn't even include Make and Model as parameters, because I did not think they were important). If I knew what was important to the user, I could:
    - Add weights to certain parameters to give them more influence over others
    - Make use of other parameters (i.e. if I knew Make and Model were important to the user, I would add them as parameters and give them more weight. To me, make and model are not as significant as price and discounts).
- I could try other distance algorithms. Euclidean is not the only one out there. (Cosine, city block, manhattan, etc.)

API Data Fetch

- Runs from the terminal by typing “python api_fetch.py”
- Requests a JSON from the API
    - JSON can be easily put into a Python Dictionary
    - The CSV module in Python has a DictWriter class that allows you to easily export Python dictionaries to CSV files
    - DictWriter also allows you to specify what columns you want outputted and in what order
    - JSON doesn’t care about commas or quotes. With CSV, commas and quotes in the data can be troublesome.

Calculating Diffs

- Runs from the terminal by typing “python diff.py”
- Reads the old file into a Pandas Data Frame
- Reads the new file into a Pandas Data Frame

Calculating Diffs - Adds and Deletes

- A row was considered to have been added if the ID appears in the Data Frame with the new data, but does not appear in the Data Frame with the old data

- A row was considered to have been deleted if the ID appears in the Data Frame with the old data, but does not appear in the Data Frame with the new data

- Separate Data Frames were created to store the Adds and Deletes

Calculating Diffs - No Change

- For IDs that appear in both the file with the new data and the file with the old data, 2 new Data Frames are created
    - 1 Data Frame that contains rows that only have those IDs, but the data is from the old data file
    - 1 Data Frame that contains rows that only have those IDs, but the data is from the new data file
- A boolean column called “hasChange” is created in both Data Frames:
    - “hasChange” is False if a row has the same data in both Data Frames (i.e. no change occurred)
    - “hasChange” is True if a row has data that is different from one Data Frame to the other (i.e. a change occurred)
    - A new Data Frame was created to contain only the rows where “hasChange” was False

Calculating Diffs - Has Change

- Filters both Data Frames that contain rows with IDs that appear in both the new and old file on “hasChange” equals True

- Put the two Data Frames in a Pandas Panel

- Apply a function to the Panel that shows the diffs between rows

- Create a new Data Frame that contains the diffs

Calculating Diffs - Final Results

- Concatenate the add, delete, no change, and changes Data Frames

- Sort on ascending ID

- Export to “changes.csv”

- There were 649 rows added, 3 rows deleted, 94 rows modified, and 3 rows were unchanged

- For the 94 row that changed, changes only occurred in the emptySlots and bikesAvailable columns

Calculating Diffs - Larger Data Set

- If there were 23 million bike stations:
    - My current solution would probably take a considerable amount of time and memory to run
        - Pandas Data Frames are stored in memory. It is possible to have data that fits on a CSV but is too large for a Data Frame
        - Python can be slow. This data set (<1000) took 0.9 seconds to run. For 23 million, it would take over 20,000 seconds (about 6 hours)
    - Importing from CSV to Pandas, as I am doing now, would probably not be the best solution

Calculating Diffs - Large Data Sets

- Things I would do differently:
    - Load the data into a database
        - SQL might work best because I can load the old and new data in separate tables and use joins
            - Make sure to do indexing and anything else SQL has to offer to decrease run time
- Do the processing to determine if a row was added, deleted, modified, or unchanged in SQL using one of Python's SQL modules like sqlite
- Use Pandas only to determine the diffs on rows that changed and put that information into the database.
- Viewing changes would then be a simple database query