

Greendeck Data Science Intern Challenge

Project uses Flask and Python to build the backend. Postman was used for testing POST requests. App is hosted on Heroku. The url for the Heroku server is: <https://enigmatic-mountain-99944.herokuapp.com/>

NOTE: Since there is no front end to the application, the user will get a Method Not Allowed response. Kindly use Postman to check for responses to POST requests.

It should also be noted that due to timeout error issues, I could not host the API with the test data on Heroku. I tried the template file given by Greendeck but it did not work. Then I tried zipping the file into my google drive, storing in my local machine, pushing to the heroku repo and then extracting it in the server. It still gave me timeout errors. A lot of effort was wasted trying to resolve this issue, therefore for demo purposes, I have stuck to the netaporter_gb_similar.json. However, I have tested this app with the test data on my local machine and it has worked throughout my testing.

Therefore, it would be appreciated if you run this repo on your local machine to get it running on test data.

Function Descriptions

QueryProcessing Class (queries.py)

These functions have been made such that most of the queries can be run even without a filter, which returns generic results on the entire dataset. Additionally, multiple queries can be passed by adding a '|' after every query code in the POST request body. For eg. `{"query_type": "discounted_products_list|avg_discount"}`. This would give 2 responses, a list of ids and an int for average discount.

It is very **IMPORTANT** to note that if no filter is specified with the queries, an empty list should be passed along with the queries. For eg. `{"query_type": "discounted_products_list|avg_discount", "filters": []}`

- **discounted_product_list()**
parameters :
 - subset - json data as a list>> returns a **list** of NAP product ids that are discounted.
- **discounted_product_count()**
parameters:
 - subset - json data as a list>> returns an **int** i.e. the number of NAP products that are discounted.
- **avg_discount()**
parameters:
 - subset - json data as a list>> returns a **float** i.e. the average discount of all the products in the catalogue
- **expensive_list()**
parameters:
 - subset - json data as a list>> returns a **list** of NAP products whose basket price is higher than any of the competing websites. **NOTE:** The basket price of the competitor is taken from the 'min_price' section. This is because it is only logical that a customer will buy the product with the minimum price first. Therefore, when comparing the 2 websites, the product will compare minimum prices available.
- **competition_discount_diff_list()**
parameters:
 - subset - json data as a list
 - filters - the set of filters passed with the POST request>> returns a **list** of NAP product ids where the percentage difference between basket_prices of the NAP product and the competing product aligns with the discount_diff parameter ('>', '<', '==').
This function depends on 2 filters (competition and discount_diff) that need to be present in the POST request. If not, will return error statement.

FilterProcessing Class (filters.py)

The functions in this class require the **operands** and the **operator** that is passed along with the POST request.

- **discount()**
parameters:
 - subset - json data as a list
 - op - either '==', '<' or '>'

- op2 - an **int** that specifies the percentage discount of items
- >> returns a **subset** of the data that contains only the products which match the discount criteria.
- **brandname()**
parameters:
 - subset - json data as a list
 - op2 - a **str** which specifies the brand name as found in the dataset.
 >> returns a **subset** of the data that contains only the products which match the brandname criteria.
- **discount_diff()**
parameters:
 - subset_from_competition - **json data** got after using the competition filter
 - op - either '==', '<' or '>'
 - op2 - an **int** that specifies the difference of percentage discount between basket_prices of the NAP product and the competing product.
 - competing_id - **str** id of the competing website.
 >> returns returns a **subset** of the data that contains only the products which match the discount_diff criteria. **NOTE:** The discount_diff filter is only valid when there is a competition filter used as well. Otherwise, it will return an error statement.
- **competition()**
parameters:
 - subset - json data as a list
 - op2 - a **str** id of the competing website.
 >> returns a **subset** of the data that contains only the products which match the competition criteria.

API Processing

The API works in 2 steps. First the "filters" dictionary is grabbed from the request body and data is filtered accordingly. Since each filter function returns a subset of the data, that subset can then be used for another filter and so on. This way there is no confusion of different return values for filters and can be scaled or modified for further use.

Once the filters are run, it will return a subset of the data that complies with all constraints. This can then be passed through the query functions which return their respective results, either a list of ids, a float etc.

This processing is handled by the **request_processing** function in the **data_processing.py** file.

Additionally, there is a need to explain the API entry point, i.e the **api.py** file. This is where the app is loaded.

There are 2 optional functions defined in this file, namely **init_files()** and the **extract_zip()**. The first function is used to download the data from a google drive link. The file_path is the parameter. The file itself is not restricted to json, but can also be a zip file. The second function is used only if a zipped dataset is downloaded. In that case, this method needs to be used before sending the data into the prepare_dataset method for further processing.

Demo Request and Response and Steps to run on Heroku or localhost

Steps to run on localhost

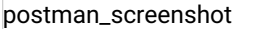
1. Make sure you have installed flask, flask_cors, gdown and zipfile along with python3 to get everything running.
2. Depending upon the type of test file you want to test on, either download the data or use the demo data(which is the default data used). Check the api.py file and uncomment the both or the init_files() function call displayed by the arrow in the file.
3. In a terminal, type `python api.py`
4. Should open the app on the default location `localhost:5000` . Still, check the terminal to see where the app is hosted. Go to that url.
5. Open Postman. Open a new tab in it. Change request type to **POST**. By default it is at **GET**. Type the app URL in the bar beside the request type.
6. Select **Body** from the menu bar below and then select **raw** from the menu bar above the blank space.
7. Type request in this format:


```
{ "query_type": "discounted_products_list|avg_discount|discounted_product_count", "filters": [{
  "operand1": "discount", "operator": ">", "operand2": 30 }] }
```
8. Check the output in the response space below.

Steps to run on Heroku

As mentioned earlier, the URL is : <https://enigmatic-mountain-99944.herokuapp.com/>

1. Since the server is already running , there is no need to go the url.
2. Simply copy the URL and paste in the URL bar as was done for localhost in the earlier steps.
3. Give the request in the same way, and get the output.

This is a screenshot of Postman after the POST request is processed on the Heroku server. 

Check out the **sample.jpg** in the root directory to see the screenshot if the picture is not being displayed in the pdf file.