Meta-Search Engine for Groceries/Food/Household Items

A platform to check and compare grocery prices from different retailers

https://github.com/VipulR2709/metasearch-engine-for-groceries

---

Background

The idea behind this project occurs from the rising price variety of different food and grocery items across supermarkets. As international students, we have experienced this issue and understand the hassle that one has to go through, in order to find the cheapest grocery items. Hence, comes the need for a database that can be used as a single source of information to take cost-effective decisions.

Objective

Entity-Relationship Diagram

Final Database (dealtraders_v2)

Final SQL Queries

Following are the SQL statements that were used to create tables in the database.

Target:

```
CREATE TABLE `target_products_details` (
  `product_id` text,
  `product_title` text,
  `product_url` text,
  `vendor_ids` text,
  `oos_all_store` text,
  `shipping_min_date` text,
  `shipping_max_date` text,
  `two_days_shipping_availability` text,
  `store_ids` int DEFAULT NULL,
  `pickup_date` text,
  `delivery_availability` text
);

CREATE TABLE `target_products_pricing` (
  `product_title` text,
  `price` double DEFAULT NULL
);


CREATE TABLE `target_store_name_mapping` (
  `store_ids` int DEFAULT NULL,
  `store_names` text
) ;
```

Walgreens:

```
CREATE TABLE `walgreens_products_details` (
`product_title` VARCHAR(100),
`availability_list` VARCHAR(100),
`ratings_list` double DEFAULT NULL,
`product_url` VARCHAR(200),
`packet_size_list` VARCHAR(100),
`prod_type_list` VARCHAR(100),
`packet_size_unit_list` VARCHAR(100),
`product_ids` VARCHAR(100),
PRIMARY KEY (product_ids),
CONSTRAINT FK_product_title FOREIGN KEY (product_title) REFERENCES walgreens_products_details(product_ids)
) ;

CREATE TABLE `walgreens_products_pricing` (
`product_title` VARCHAR(100),
`price` VARCHAR(100),
PRIMARY KEY(product_title),
CONSTRAINT FK_product_title FOREIGN KEY (product_title) REFERENCES walgreens_products_details(product_ids)
)
```

Star Market:

```
CREATE TABLE `starmarket_products_details` (
`product_ids` VARCHAR(100),
`product_title` VARCHAR(100),
`delivery_availability_list` VARCHAR(100),
`ratings_list` int DEFAULT NULL,
`packet_size_unit_list` VARCHAR(100),
`prod_type_list` VARCHAR(100),
`inStore_availability_list` BOOLEAN,
`pickup_availability_list` VARCHAR(100),
PRIMARY KEY (product_ids),
CONSTRAINT FK_product_title FOREIGN KEY (product_title) REFERENCES starmarket_products_details(Product_ID)
);

CREATE TABLE `starmarket_products_pricing` (
`product_title` VARCHAR(100),
`price` double DEFAULT NULL,
CONSTRAINT FK_product_title FOREIGN KEY (product_title) REFERENCES starmarket_products_details(product_ids)
);
```

Traderjoes:

```
CREATE TABLE `traderjoes_products_details` (
`product_title` VARCHAR(100),
`availability_list` VARCHAR(100),
`product_url` VARCHAR(100),
`packet_size_list` double DEFAULT NULL,
`packet_size_unit_list` VARCHAR(100),
`product_ids` VARCHAR(100),
`ingredient_list` VARCHAR(100),
PRIMARY KEY(product_title),
CONSTRAINT FK_product_title FOREIGN KEY (product_title) REFERENCES traderjoes_products_details(product_ids)
) ;
```

```
CREATE TABLE `traderjoes_products_pricing` (
`product_title` VARCHAR(100),
`price` double DEFAULT NULL,
CONSTRAINT FK_product_title FOREIGN KEY (product_title) REFERENCES traderjoes_products_details(product_ids)
);
```

Walmart:

```
CREATE TABLE `walmart_products_details` (
  `PRODUCT_URL` text,
  `PRODUCT_SIZE` double DEFAULT NULL,
  `product_title` text,
  `CATEGORY` text,
  `BRAND` text,
  `product_ids` text
) ;

CREATE TABLE `walmart_category_mapping` (
  `DEPARTMENT` text,
  `CATEGORY` text
) ;

CREATE TABLE `walmart_products_pricing` (
  `product_title` text,
  `price` double DEFAULT NULL
) ;
```

Mega_Store

```
CREATE TABLE `mega_store_data_final` (
`product_title` VARCHAR(100),
`availability_list` VARCHAR(100),
`product_url` VARCHAR(200),
`product_ids` VARCHAR(100),
`price` double DEFAULT NULL,
`Store` VARCHAR(100),
PRIMARY KEY (product_ids)
) ;
```

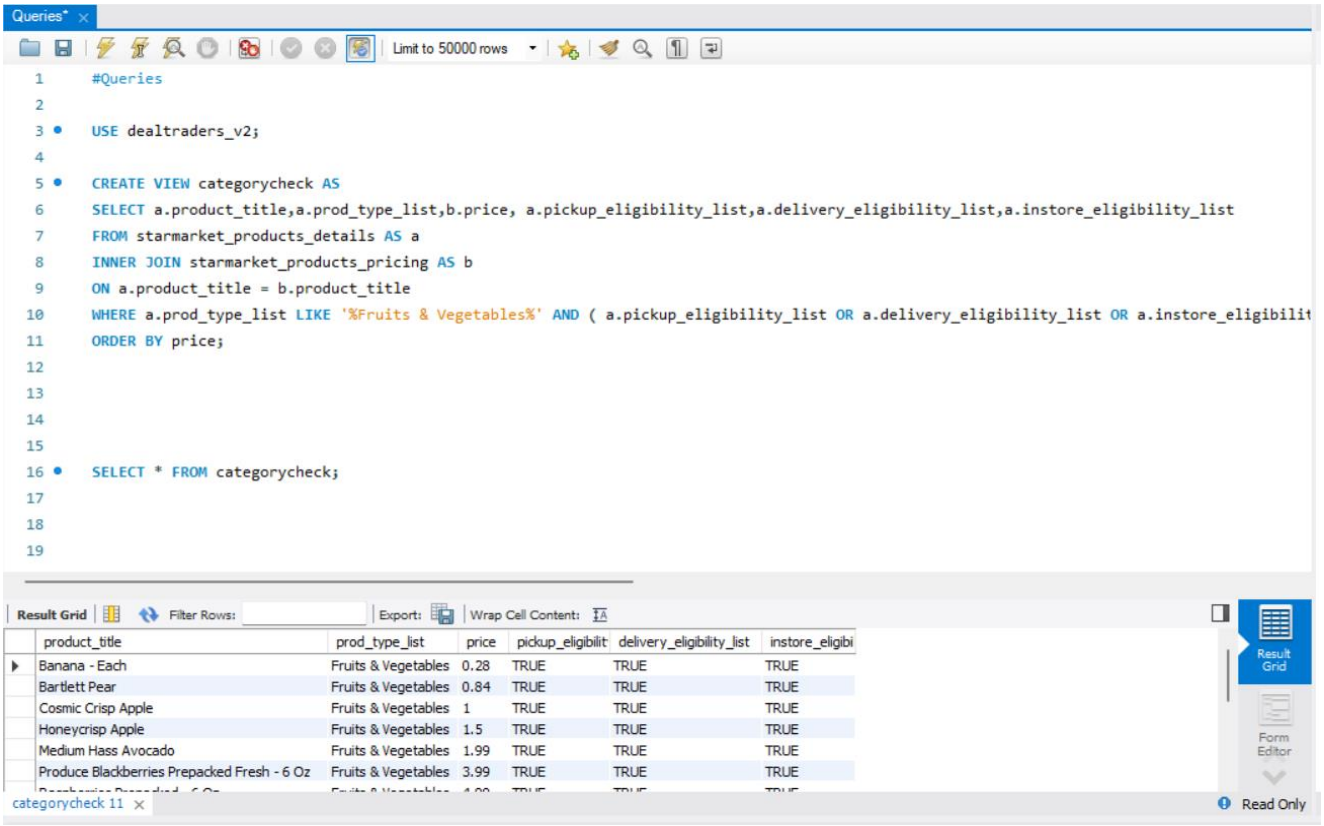
Major Use-cases

A few major user questions that can be answered from the database are as follow:

1. Use Case: Types of availability for fruits and vegetables
   Description: User should be able to view availability for fruits and vegetables
   Actor: User
   Precondition: User should have selected fruits and fresh foods category
   Steps:
   Actor action: User should be able to view availability for the selected category
   System Responses: Displays the modes of availability for selected category on screen
   Post Conditions: User can see a list of products within the selected category

   SQL Query:

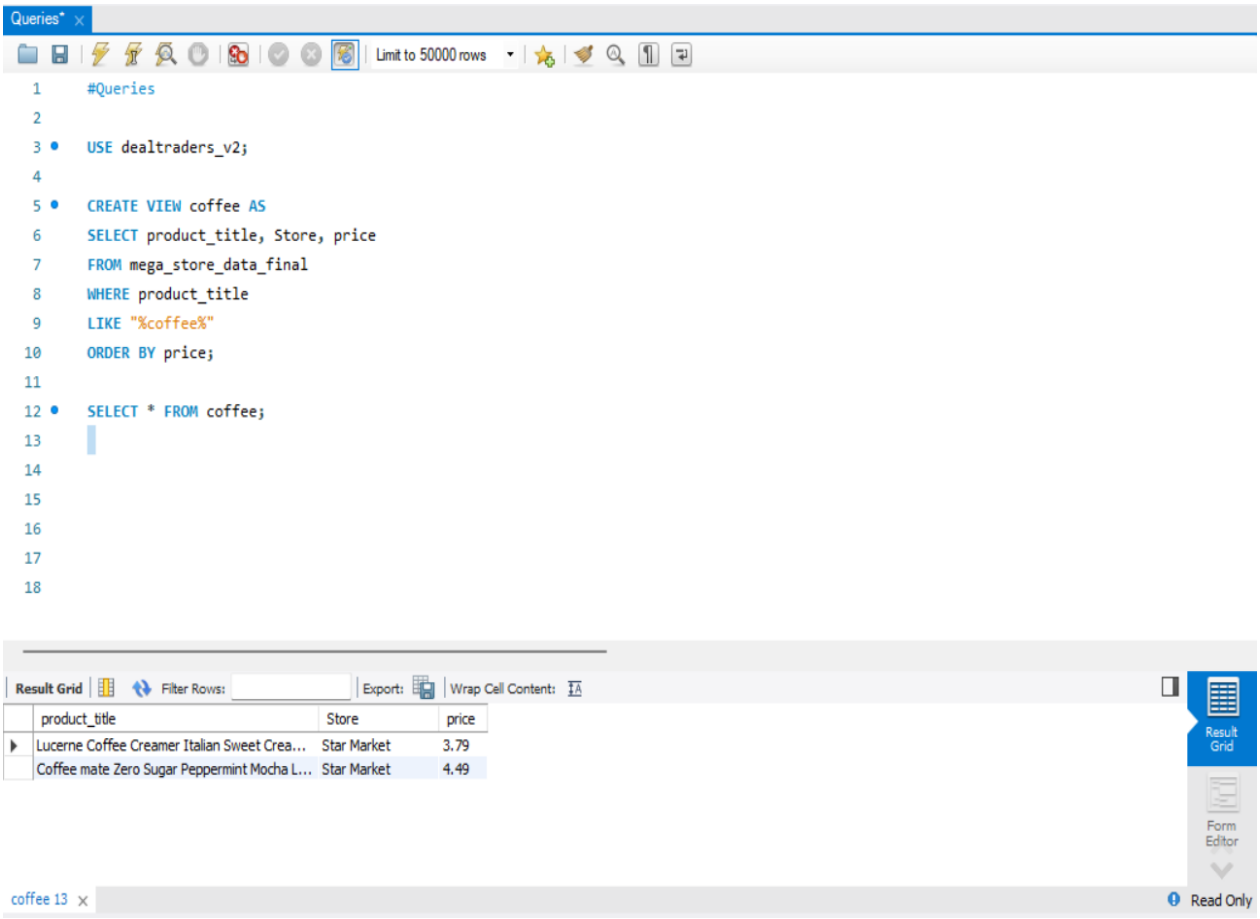
```
CREATE VIEW categorycheck AS
SELECT a.product_title,a.prod_type_list,b.price,
a.pickup_eligibility_list,a.delivery_eligibility_list,a.instore_eligibility_list
FROM starmarket_products_details AS a
INNER JOIN starmarket_products_pricing AS b
ON a.product_title = b.product_title
WHERE a.prod_type_list LIKE '%Fruits & Vegetables%' AND ( a.pickup_eligibility_list OR a.delivery_eligibility_list
OR a.instore_eligibility_list) = 'TRUE'
ORDER BY price;
```



2. Use Case: Cheapest coffee available at any store
Description: User should be able to view cheapest coffee
Actor: User
Precondition: User should know coffee category
Steps:
Actor action: User should be able to view the cheapest coffee available
System Responses: Displays the cheapest coffee
Post Conditions: User can see a list of coffee with the cheapest coffee on top

SQL Query:

```
CREATE VIEW coffee AS
SELECT product_title, Store, price
FROM mega_store_data_final
WHERE product_title
LIKE "%coffee%"
ORDER BY price LIMIT 2;
```



3. Use Case: Cheapest coffee available at any store
Description: User should be able to view cheapest coffee
Actor: User
Precondition: User should coffee category
Steps:
Actor action: User should be able to view the cheapest coffee available
System Responses: Displays the cheapest coffee
Post Conditions: User can see a list of coffee with the cheapest coffee on top

SQL Query:

```
CREATE VIEW milk AS
```

```
SELECT product_title, Store, price
FROM mega_store_data_final
WHERE product_title
LIKE "%milk%"
ORDER BY price
LIMIT 2;
```

4. Use Case: Stores with Bananas in stock
   Description: User should be able to see all the stores with bananas in stock
   Actor: User
   Precondition: User must have entered into fruits and fresh foods category
   Steps:
   Actor action: User views stores with various banana types
   System Responses: Displays bananas that are in stock at stores
   Post Condition: User can view all the stores in which bananas are in stock
   SQL Query:

```
CREATE VIEW banana AS
SELECT product_title, Store, price
FROM mega_store_data_final
WHERE product_title
LIKE "%banana%" AND availability_list = "IN_STOCK" ORDER BY price ;
```

```
1
2 •   SELECT product_title, Store, price
3     FROM mega_store_data_final
4     WHERE product_title
5     LIKE "%banana%" AND availability_list = "IN_STOCK" ORDER BY price ;
6
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

product_title	Store	price
Banana - Each	Star Market	0.28

5. Use Case: Availability of eggs in Megastore
   Description: User should be able to see availability of eggs
   Actor: User
   Precondition: User must have selected dairy and poultry category
   Steps:
   Actor action: User views eggs of various protein types at megastore
   System Responses: Displays price, title and protein types of eggs
   Post Condition: User can view multiple eggs

   SQL Query:

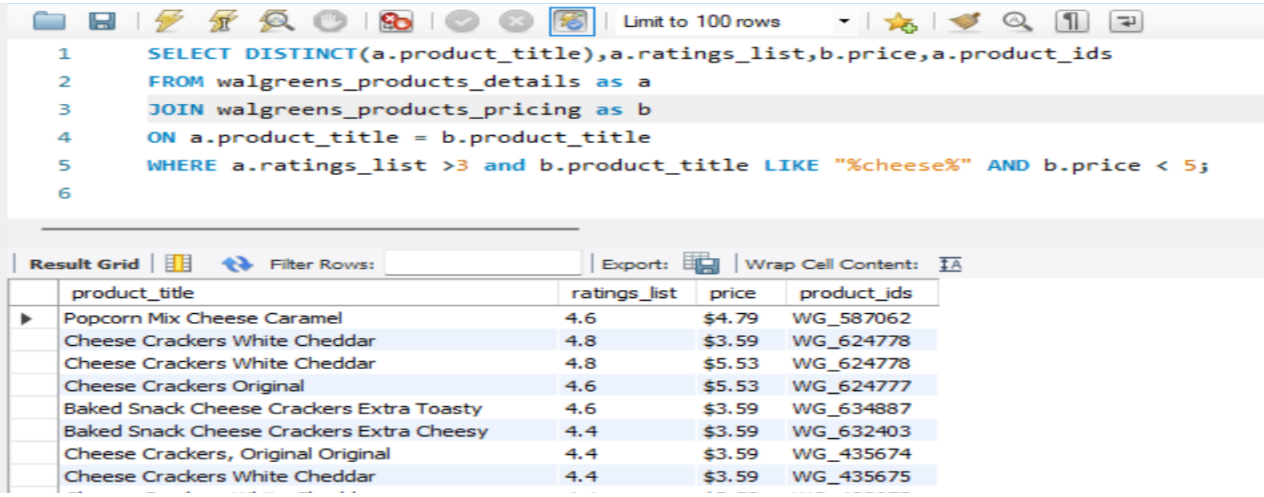
```
CREATE VIEW egg AS
SELECT product_title, Store, price
FROM mega_store_data_final
WHERE product_title
```

LIKE "%egg%"
ORDER BY price
LIMIT 2;

6. Checking all cheese products with ratings more than 3 at Walgreens and having price less than $5

SQL Query:

```
CREATE VIEW walgreenscheese AS
SELECT DISTINCT(a.product_title),a.ratings_list,b.price,a.product_ids
FROM walgreens_products_details as a
JOIN walgreens_products_pricing as b
ON a.product_title = b.product_title
WHERE a.ratings_list >3 and b.product_title LIKE "%cheese%" AND b.price < 5;
```



```
SELECT DISTINCT(a.product_title),a.ratings_list,b.price,a.product_ids
FROM walgreens_products_details as a
JOIN walgreens_products_pricing as b
ON a.product_title = b.product_title
WHERE a.ratings_list >3 and b.product_title LIKE "%cheese%" AND b.price < 5;
```

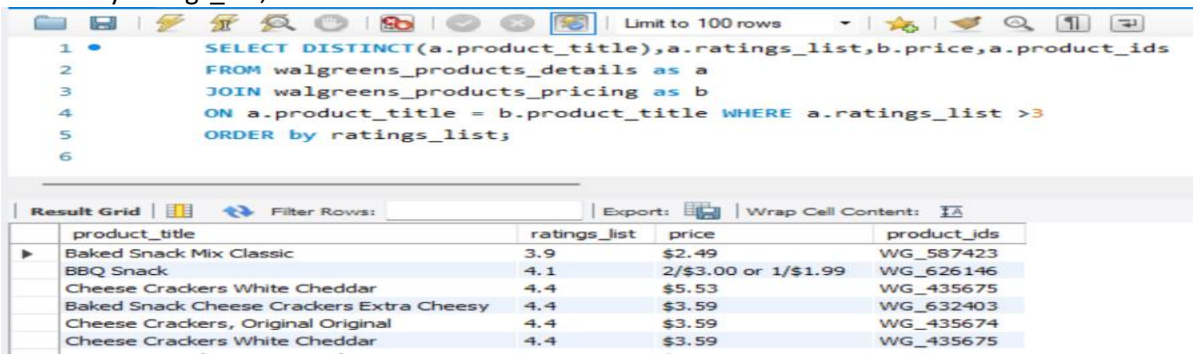
Result Grid | Filter Rows: | Export: | Wrap Cell Content:

product_title	ratings_list	price	product_ids
Popcorn Mix Cheese Caramel	4.6	$4.79	WG_587062
Cheese Crackers White Cheddar	4.8	$3.59	WG_624778
Cheese Crackers White Cheddar	4.8	$5.53	WG_624778
Cheese Crackers Original	4.6	$5.53	WG_624777
Baked Snack Cheese Crackers Extra Toasty	4.6	$3.59	WG_634887
Baked Snack Cheese Crackers Extra Cheesy	4.4	$3.59	WG_632403
Cheese Crackers, Original Original	4.4	$3.59	WG_435674
Cheese Crackers White Cheddar	4.4	$3.59	WG_435675

7. Use Case: Products with ratings greater than 3 at walgreens
Description: User should select walgreens as store and filter it by products with rating greater than 3
Actor: User
Precondition: Walgreens should be selected as store
Steps:
Actor action: User will select walgreens as store and select filter by ratings more than 3
System Responses: If the customer can see products with ratings more than 3
Post Conditions: User can see all the products with multiple category and variety with ratings greater than 3

SQL Query:

```
CREATE VIEW ratings AS
SELECT DISTINCT(a.product_title),a.ratings_list,b.price,a.product_ids
FROM walgreens_products_details as a
JOIN walgreens_products_pricing as b
ON a.product_title = b.product_title WHERE a.ratings_list >3
ORDER by ratings_list;
```



```
SELECT DISTINCT(a.product_title),a.ratings_list,b.price,a.product_ids
FROM walgreens_products_details as a
JOIN walgreens_products_pricing as b
ON a.product_title = b.product_title WHERE a.ratings_list >3
ORDER by ratings_list;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

product_title	ratings_list	price	product_ids
Baked Snack Mix Classic	3.9	$2.49	WG_587423
BBQ Snack	4.1	2/$3.00 or 1/$1.99	WG_626146
Cheese Crackers White Cheddar	4.4	$5.53	WG_435675
Baked Snack Cheese Crackers Extra Cheesy	4.4	$3.59	WG_632403
Cheese Crackers, Original Original	4.4	$3.59	WG_435674
Cheese Crackers White Cheddar	4.4	$3.59	WG_435675

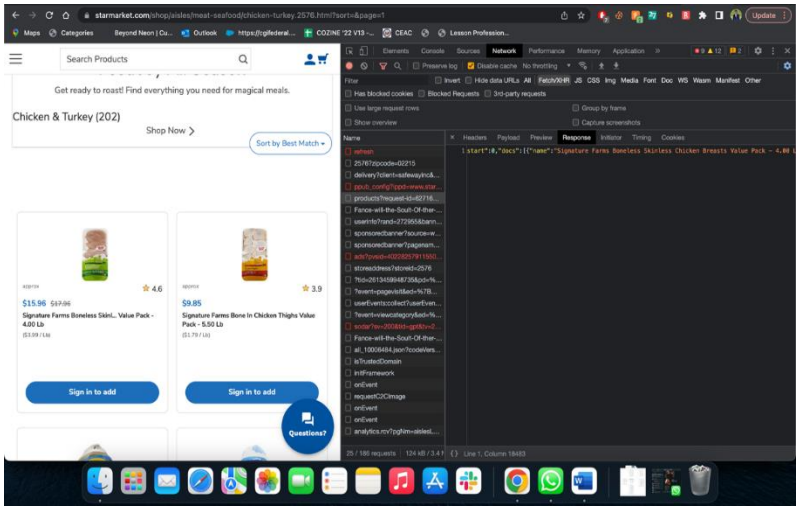
Steps performed to get the final database

1. Web scraping from different stores' websites
2. Data cleaning and munging
3. Checking normalization forms and data processing

Data Sources

Fetched data from: Star Market, Target, Walgreens and Trader Joes
Reference Links -

1. https://www.target.com
2. https://www.starmarket.com
3. https://www.walgreens.com
4. https://www.traderjoes.com
5. https://www.kaggle.com/




```python
def json_to_csv_traderjoes(path, list_of_jsons):

    list_of_dicts = []
    for file_ in list_of_jsons:
        # Opening JSON file
        prod_json = open(path + "//" +file_)

        # returns JSON object as a dictionary
        data = (json.load(prod_json))["data"]["products"]

        dict_ = {}

        product_ids = []
        product_title = []
        product_url = []
        price_list = []
        ingredient_link = []
        packet_size_list = []
        packet_size_unit_list = []
        availability_list = []

        for product in data["items"]:

            id = product.get("sku")
            product_ids.append(id)

            title = product.get("item_title")
            product_title.append(title)

            url = product.get("url_key")
            product_url.append(url)

            price = product.get("retail_price")
            price_list.append(price)

            try:...
            except:...
            packet_size = product["sales_size"]
            packet_size_list.append(packet_size)

            packet_size_unit = product["sales_uom_description"]
            packet_size_unit_list.append(packet_size_unit)

            availability = product["availability"]
            availability_list.append(availability)

        dict_.update({
            "product_ids": product_ids,
            "product_title" : product_title,
            "product_url" : product_url,
            "price" : price_list,
            "ingredient_link" : ingredient_link,
            "packet_size_list": packet_size_list,
            "packet_size_unit_list": packet_size_unit_list,
            "availability_list" : availability_list
        })

        list_of_dicts.append(dict_)
    return list_of_dicts
```

We fetched the URL responses and converted those JSONs into data frames and eventually into csv datafiles/tables. Above is the screenshot of sample function used to convert JSON into python dictionary with required keys.

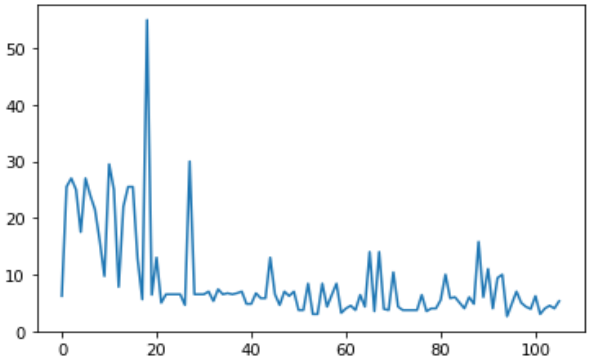
Audit Validity and Data Cleaning:

Checking count of null values in all columns

```
target_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 106 entries, 0 to 105
Data columns (total 17 columns):
 #   Column                          Non-Null Count  Dtype
---  ------                          --------------  -----
 0   Unnamed: 0                      106 non-null    int64
 1   product_ids                     106 non-null    int64
 2   product_title                   106 non-null    object
 3   product_url                     106 non-null    object
 4   vendor_ids                      106 non-null    object
 5   price                           106 non-null    float64
 6   eligibility_rules               106 non-null    object
 7   oos_all_store                   106 non-null    bool
 8   availability_shipping           106 non-null    object
 9   shipping_min_date               41 non-null     object
 10  shipping_max_date               41 non-null     object
 11  two_days_shipping_availability  41 non-null     object
 12  store_ids                       106 non-null    int64
 13  store_names                     106 non-null    object
 14  pickup_availability             106 non-null    object
 15  pickup_date                     90 non-null     object
 16  delivery_availability           106 non-null    object
dtypes: bool(1), float64(1), int64(3), object(12)
memory usage: 13.5+ KB
```

Checking anomalies in price column



Checking possibilities for null values and filling appropriate values

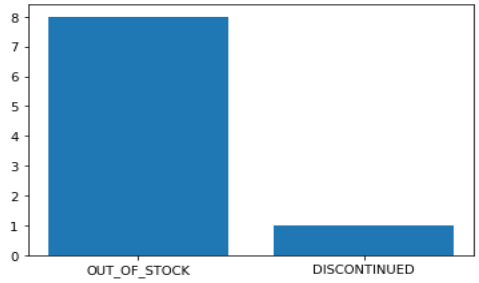
```
unique_statuses = list(target_data[target_data['shipping_min_date'].isna()]['availability_shipping'].unique())
records_with_no_dates = target_data[target_data['shipping_min_date'].isna()]['availability_shipping'].tolist()
records_with_no_dates

# Make a random dataset:
height = [records_with_no_dates.count(x) for x in unique_statuses]
bars = unique_statuses
y_pos = np.arange(len(bars))

# Create bars
plt.bar(y_pos, height)

# Create names on the x-axis
plt.xticks(y_pos, bars)

# Show graphic
plt.show()
```



Filling null values in rating field with average ratings

```
avg_ratings = walgreens_data['ratings_list'].mean()
walgreens_data[['ratings_list']] = walgreens_data[['ratings_list']].fillna(avg_ratings)
walgreens_data.info()
```

Conclusion

The final created database can be widely used by all types user to decide where to buy items from. This will help users to save their time as well hard-earned money.

Contributors

1. Vipul Rajderkar (rajderkar.v@northeastern.edu, 002700991)
2. Raj Sarode (sarode.r@northeastern.edu, 002762015)
3. Tanmay Zope (zope.t@northeastern.edu, 002767087)