Mid-Report 1

Dissecting the InstaCart Dataset: Revealing Patterns in Customer Purchase Behavior

Team: The NaN Division

Member Name	Email
Christopher Davis	cmd7490@psu.edu
Maya Hohman	meh6853@psu.edu
Jordan Michael Lee Hunt	jmh9240@psu.edu
Sami Younis	szy5447@psu.edu

SQLite3 Database Setup

As mentioned in the Project Proposal, InstaCart's dataset includes 6 CSV tables:

Table	Row Count
aisles.csv	134
departments.csv	21
order_products__prior.csv	32,434,489
order_products__train.csv	1,384,617
orders.csv	3,421,083
products.csv	49,688

Given that these tables were not provided in true relational database format, we needed a reliable method for integrating the data. We decided to load the data into an improvised SQLite3 relational database, with the goal of using SQL queries for table joins and missing value checks.

Setting up the database required a multi-step process:

1. First, we studied the columns in each table to understand data contents and primary/foreign key relationships between tables. The data structure is illustrated in Figure 1.
2. We designed and ran a master script (**database_setup_master.py**) that runs the next three scripts in sequence:
   - merge_order_products.py
   - database_setup_skeleton.py
   - database_setup_import.py

3. The **merge_order_products.py** script vertically concatenates the contents of order_products__prior.csv and order_products__train.csv, into order_products.csv. The script adds an “eval_set” column with categorical values of “prior” or “train”, depending on the source csv file.
4. The **database_setup_skeleton.py** script sets up an empty SQLite3 database (**Instacart.db**) with the desired schema, i.e. tables, columns, data types, primary keys, foreign keys / references.
   This step was particularly complex, as we had to work around two major limitations of the SQLite3 platform:
   - Primary keys and foreign keys / relations must be declared simultaneously with table creation.
   - Foreign keys / relations cannot be declared until the referenced tables containing primary keys exist.
   The workaround for these limitations required careful construction of a sequence of table creation queries, creating the tables (and their corresponding keys/relations) in the following order:
   1) orders
   2) aisles
   3) departments
   4) products
   5) order_products
   To facilitate and error-proof this process, the script builds the necessary queries using schema information from a separate file, “table_definition.csv”, shown in Table 1.
   Once the tables, keys, and relations are in place, a second pass of queries adds the remaining columns for each table.
5. The **database_setup_import.py** script connects to the newly-created SQLite3 database (**instacart.db**) and builds queries to sequentially load data from each source CSV file into the appropriate database table, committing the changes.

Finding & Handling Missing Values

With the data loaded successfully into a relational database, we proceeded with missing value checks in each SQL table using a new script, **processing_database.py**. The only missing values present were in the orders table’s *days_since_prior_order* column, which had roughly 6% missing values.

In every case, the missing values correspond to order_number = 1 (See Figure 2). The omission of values is intentional, so rather than imputing a value we chose to add an *is_first_order*

flagging column in a subsequent script (**preprocessing_order.py)**.

Data Integration

As the source data is distributed across multiple tables, the **preprocessing_database.py** script executes a SQL query that joins the various tables together, exporting the joined records to **instacart_joined.csv**. This query intentionally filters out records where eval_set = 'test', as those orders do not provide corresponding basket items. Every row in this new CSV file represents a basket item within an order.

In comparing the row counts between order_products.csv and instacart_joined.csv, they have the same count (33.8M rows), with no records lost during the integration process.

To avoid rerunning the time-consuming population-level queries in **preprocessing_database.py**, we created a new script – **preprocessing_order.py.** This script randomly samples 5% of all *orders*, along with their corresponding basket items. The script can also be used to produce statistics and visualizations with the sampled data.

Statistics and Visualizations

Population-level counts of unique values for each table variable can be found in Table 2.

A random 5% sample of orders was selected, using the order_id key that links each order to its basket contents. These groupings are required to compute item counts, basket sizes, and to support subsequent frequent pattern mining.

Typical basket size varies as shown below, indicating severe right skew as the median is lower than the mean, and the maximum is an outlier:

```
mean         10.107991
std           7.527212
min           1.000000
25%           5.000000
50%           8.000000
75%          14.000000
max         115.000000
```

A histogram of the basket quantity distribution can be found in Figure 3. The heterogeneity indicates that customers alternate between small convenience purchases and large stock up events.

We observed that the average number of days between orders is 11.73 days (See Figure 4). The average order size is found to be 10.06 items per order with a 7.5 standard deviation (See Figure 5). One of the key findings is that the percentage of users with at least 1 reorder is 87%. This number leads us to believe that if a customer is to purchase once, there will be an 87% change they will create another purchase. Of the 87%, the average number of reorder purchases is 7.93, with a very high outlier of 193 max reorder by a single user.

The relationship between basket size and days between orders appears to be weak. Instead, basket composition appears to depend on household needs and habitual preferences. See Figure 6.

The basket of items in a reorder purchase typically contains 58.9% items for repurchase and 41.1% of new items (See Figure 7). Depending on the sampling subset and computer used, these numbers can vary slightly – see measures below:

```
Overall reorder rate: 0.5883979803745952
mean          0.597452
std           0.338342
min           0.000000
25%           0.333333
50%           0.666667
75%           0.909091
max           1.000000
```

Highest order volume occurs on days “0” and “1”. These are likely weekend days, though not explicitly labeled as such.

Across all days, hours 9 through 16 are typically the peak timeframe for order volume.

0	29977
1	29208
2	23700
3	21887
4	21380
5	22467
6	22435

order_hour_of_day	
0	1148
1	627
2	384
3	267
4	280
5	516
6	1524
7	4654
8	8943
9	12925
10	14476
11	14105
12	13760
13	13908
14	14151
15	14053
16	13714
17	11416
18	9079
19	6994
20	5186
21	3875
22	3080
23	1989

Plots of temporal order volume data can be found in Figure 8, Figure 9, Figure 10, and Figure 11.

The most-frequently reordered items are milk, water, fresh fruits and eggs; all having a reorder rate >70%. This aligns with the top reordered departments being Dairy/Eggs, Beverages, and Produce. Charts of statistics for products, departments, and aisles are plotted in Figure 12, Figure 13, Figure 14, Figure 15, and Figure 16.

Top 5 Departments by Purchase Volume	**Top 5 Departments by Reorder Rate**
1. Produce	1. Dairy/Eggs – 66.67%
2. Dairy/Eggs	2. Beverages – 65.41%
3. Snacks	3. Produce – 64.97%
4. Beverages	4. Bakery – 62.77%
5. Frozen Foods	5. Deli – 60.92%

Based on the previous statistics, we can make some general assumptions of customer behavior. For example, if a customer purchases once, they will most likely purchase again. When they do place their next order, that are more likely than not to include an item from a previous order. Department-level groupings, however, are limited in the insights they can provide. This is where we begin to dive deeper into the data with frequent pattern mining.

Frequent Pattern Analysis

Limiting one-hot variables to unique department_ids, we could not run the Apriori algorithm on the entire dataset, due to excessive memory usage. We were however successful in running the FPGrowth algorithm.

Results and Support Sensitivity

At min_support = 0.05:

- ~300+ frequent itemsets
- Interpretable number of rules

At min_support = 0.03:

- More patterns, still manageable

At min_support = 0.01:

- ~32k rules

- Significant rule explosion
- Low signal-to-noise ratio

Lowering minimum support increases coverage but rapidly increases the number of patterns and reduces interpretability. Based on this, analysis below 0.03 was not pursued.

```
Number of frequent itemsets: 338
    support                itemsets
0  0.749009       frozenset({produce})
1  0.676833    frozenset({dairy eggs})
2  0.212404  frozenset({canned goods})
3  0.348315        frozenset({pantry})
4  0.274345        frozenset({bakery})
                antecedents                consequents  ...  confidence      lift
0    frozenset({dairy eggs})     frozenset({produce})  ...    0.812938  1.085351
1       frozenset({produce})  frozenset({dairy eggs})  ...    0.734601  1.085351
2  frozenset({canned goods})     frozenset({produce})  ...    0.878896  1.173412
3  frozenset({canned goods})  frozenset({dairy eggs})  ...    0.801984  1.184908
4  frozenset({canned goods})      frozenset({snacks})  ...    0.516968  1.194008

[5 rows x 5 columns]
Number of rules: 1613
```

When applying the Apriori algorithm to unique product_ids, we had to be mindful of hardware limitations. Even with a 5% orders subset of the source data, the high cardinality of the product_id column would require over 180GB of RAM with complete set of one-hot variables. To work around this issue, we had to limit sampling of order_ids even further. After some trial and error, 5000 seemed to be the hardware limit. The following output was created with randomized data with a random state (=42) for reproducibility.

```
Top 10 Association Rules (by Lift):
          antecedents_str         consequents_str  support  confidence      lift
0  Bag of Organic Bananas    Organic Hass Avocado   0.0208    0.165079  2.420519
1    Organic Hass Avocado  Bag of Organic Bananas   0.0208    0.304985  2.420519
2  Bag of Organic Bananas    Organic Strawberries   0.0212    0.168254  1.933954
3    Organic Strawberries  Bag of Organic Bananas   0.0212    0.243678  1.933954
```

For FRGrowth, the same support and confidence thresholds were used.

```
Top 10 FP-Growth Association Rules (by Lift):
          antecedents_str         consequents_str  support  confidence      lift
3  Bag of Organic Bananas    Organic Hass Avocado   0.0208    0.165079  2.420519
2    Organic Hass Avocado  Bag of Organic Bananas   0.0208    0.304985  2.420519
0  Bag of Organic Bananas    Organic Strawberries   0.0212    0.168254  1.933954
1    Organic Strawberries  Bag of Organic Bananas   0.0212    0.243678  1.933954
```

When comparing the two outputs, they are virtually the same. This was expected as both algorithms identify the same mathematically valid itemset, differing only in computational

strategy. With this, we can identify a relationship between *Bag of Organic Bananas* <-> *Organic Hass Avacado* and *Organic Strawberries* <-> *Bag of Organic Bananas*.

Figures / Tables

Figure 1

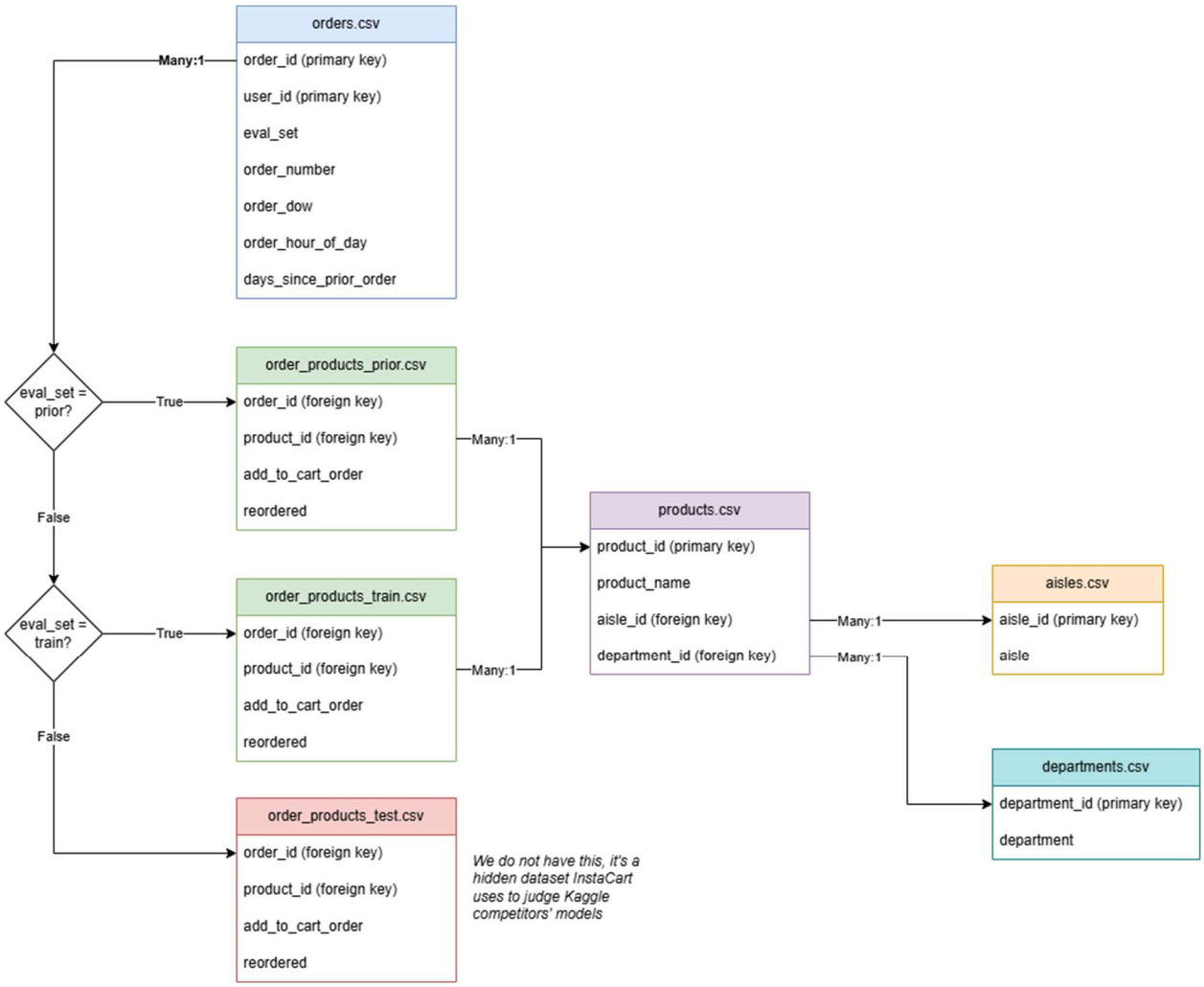


Table 1

table	column	data_type	primary_key	foreign_key	foreign_key_table
aisles	aisle_id	INT	TRUE	FALSE	N/A
aisles	aisle	TEXT	FALSE	FALSE	N/A
departments	department_id	INT	TRUE	FALSE	N/A
departments	department	TEXT	FALSE	FALSE	N/A
order_products	order_products_id	INT	TRUE	FALSE	N/A
order_products	order_id	INT	FALSE	TRUE	orders
order_products	product_id	INT	FALSE	TRUE	products
order_products	add_to_cart_order	INT	FALSE	FALSE	N/A
order_products	reordered	INT	FALSE	FALSE	N/A
orders	order_id	INT	TRUE	FALSE	N/A
orders	user_id	INT	FALSE	FALSE	N/A
orders	eval_set	TEXT	FALSE	FALSE	N/A
orders	order_number	INT	FALSE	FALSE	N/A
orders	order_dow	INT	FALSE	FALSE	N/A
orders	order_hour_of_day	INT	FALSE	FALSE	N/A
orders	days_since_prior_order	FLOAT	FALSE	FALSE	N/A
products	product_id	INT	TRUE	FALSE	N/A
products	product_name	TEXT	FALSE	FALSE	N/A
products	aisle_id	INT	FALSE	TRUE	aisles
products	department_id	INT	FALSE	TRUE	departments

Figure 2

Table 2

Table	Column	Unique Values
orders	order_id	3421083
	user_id	206209
	eval_set	3
	order_number	100
	order_dow	7
	order_hour_of_day	24
	days_since_prior_order	31
order_products	order_id	3346083
	product_id	49685
	add_to_cart_order	145
	reordered	2
	eval_set	2
products	product_id	49688
	product_name	49688
	aisle_id	134
	department_id	21
aisles	aisle_id	134
	aisle	134
departments	department_id	21
	department	21

Figure 3

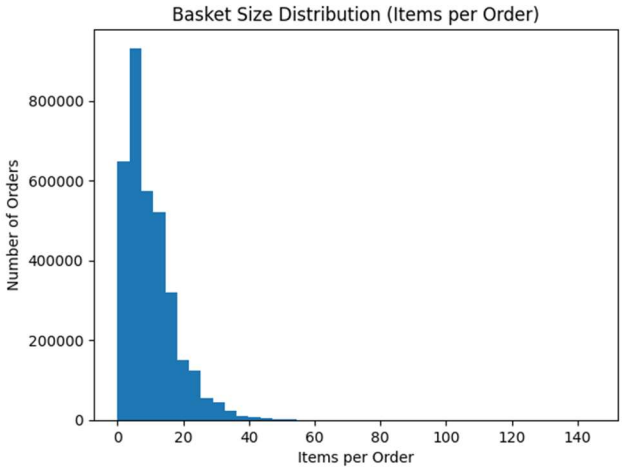


Figure 4

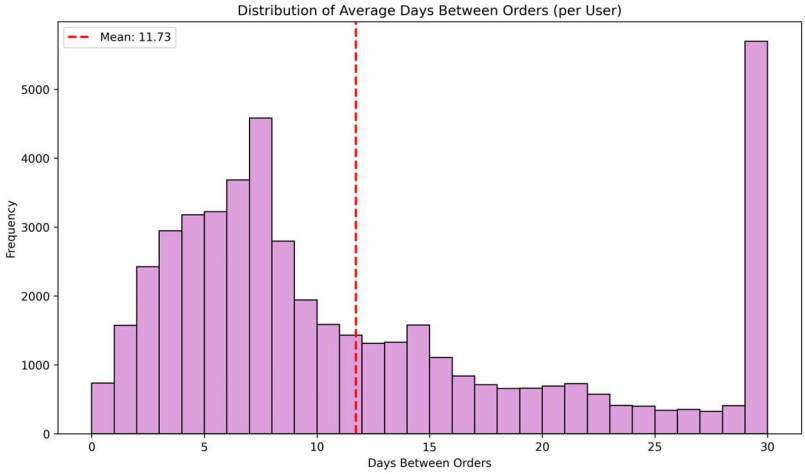


Figure 5

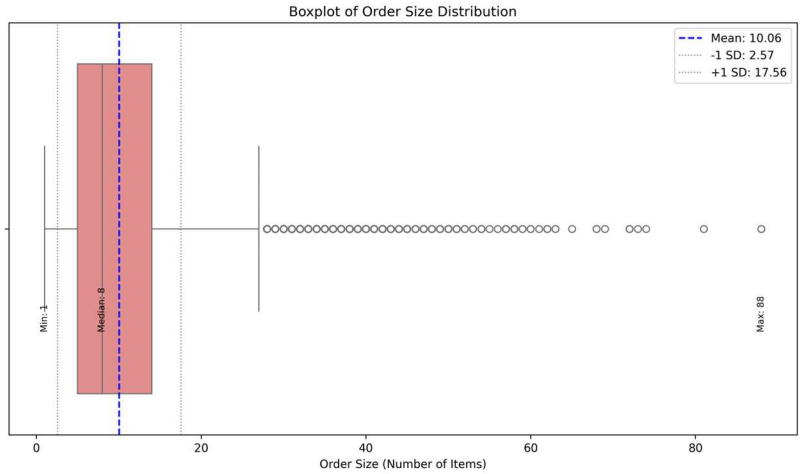


Figure 6

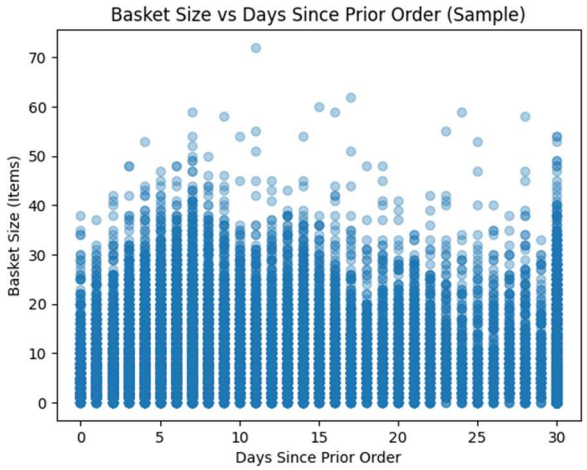


Figure 7

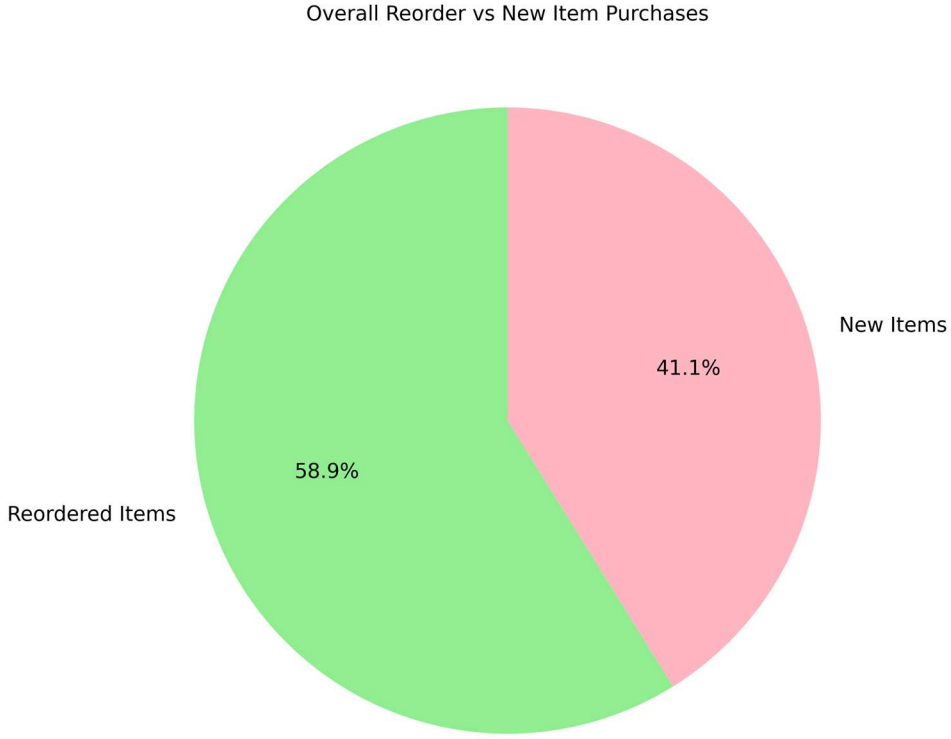


Figure 8

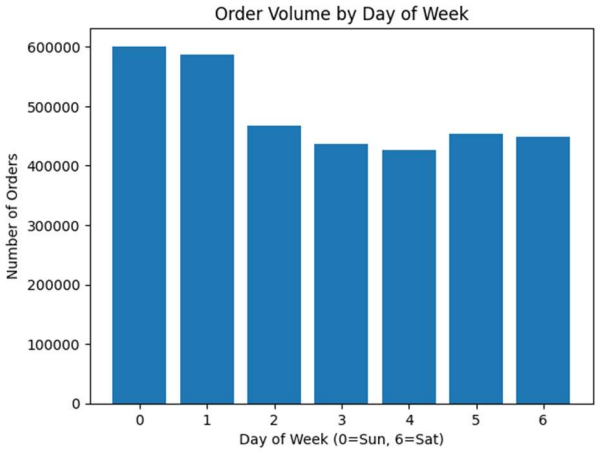


Figure 9

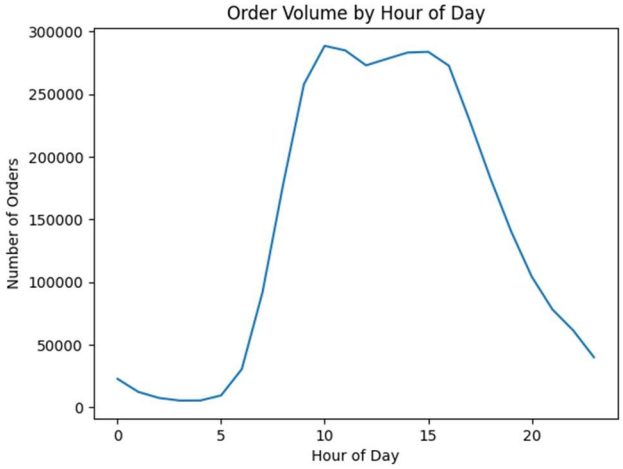


Figure 10

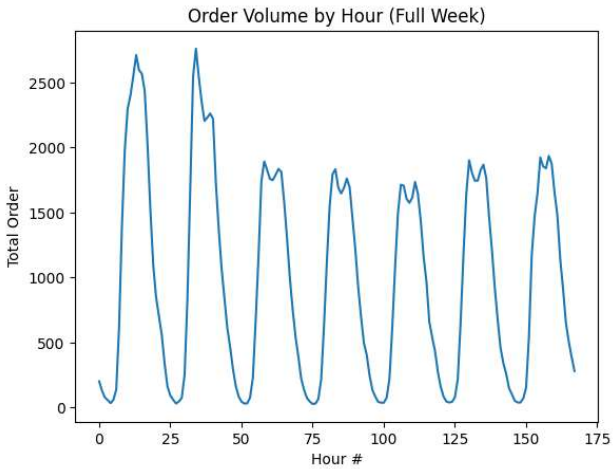


Figure 11

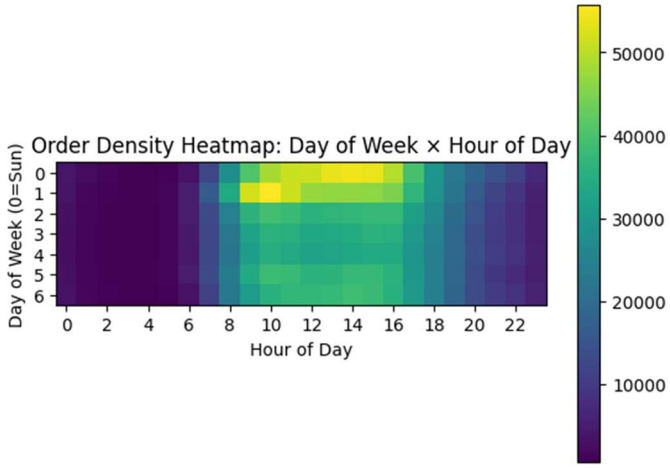


Figure 12

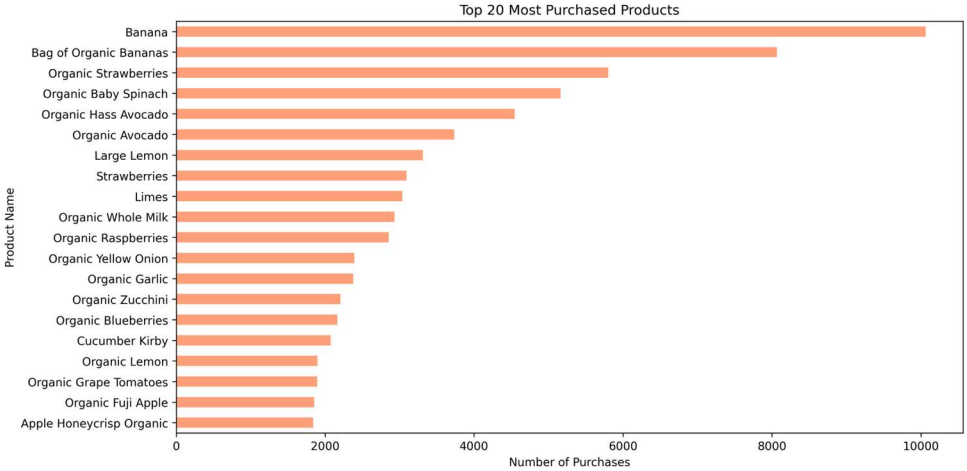


Figure 13

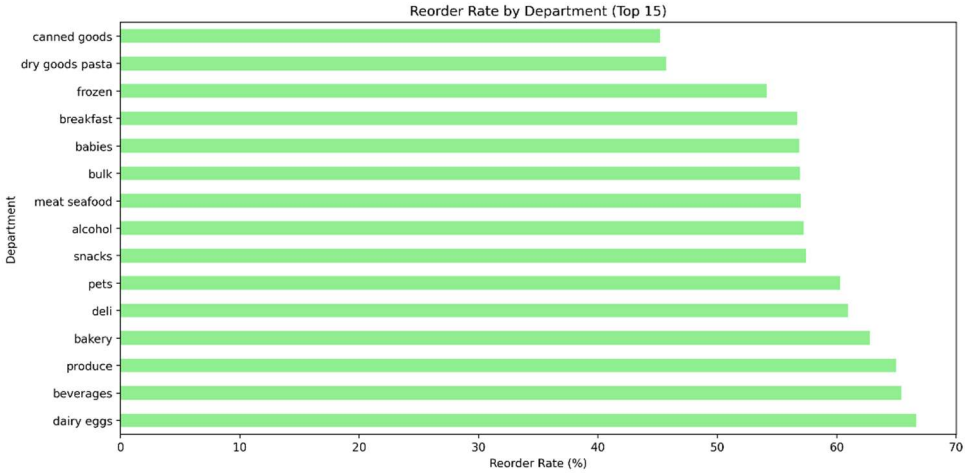


Figure 14

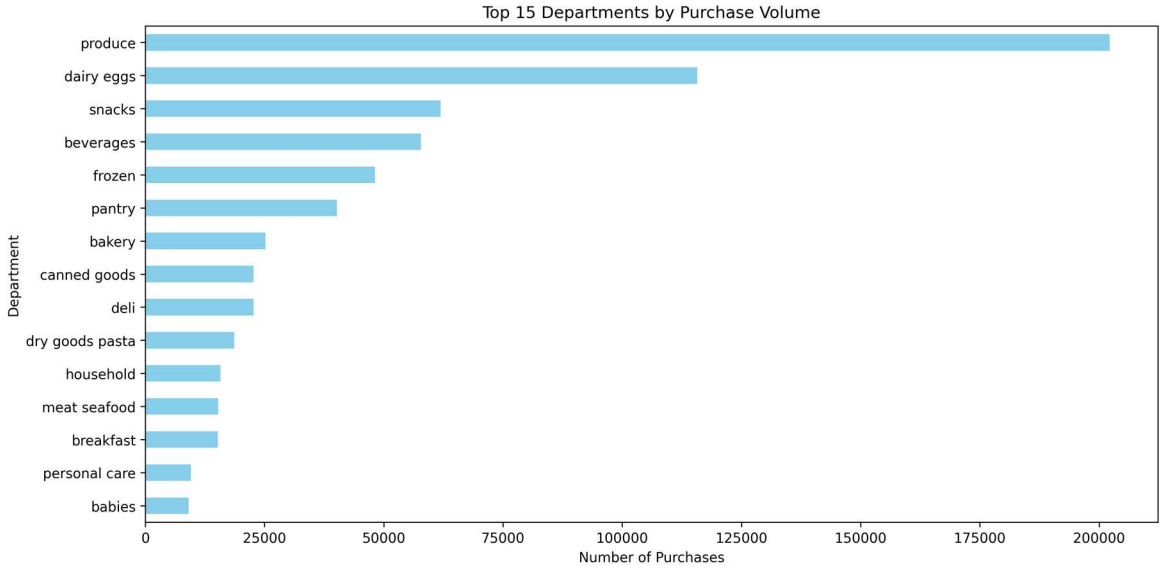


Figure 15

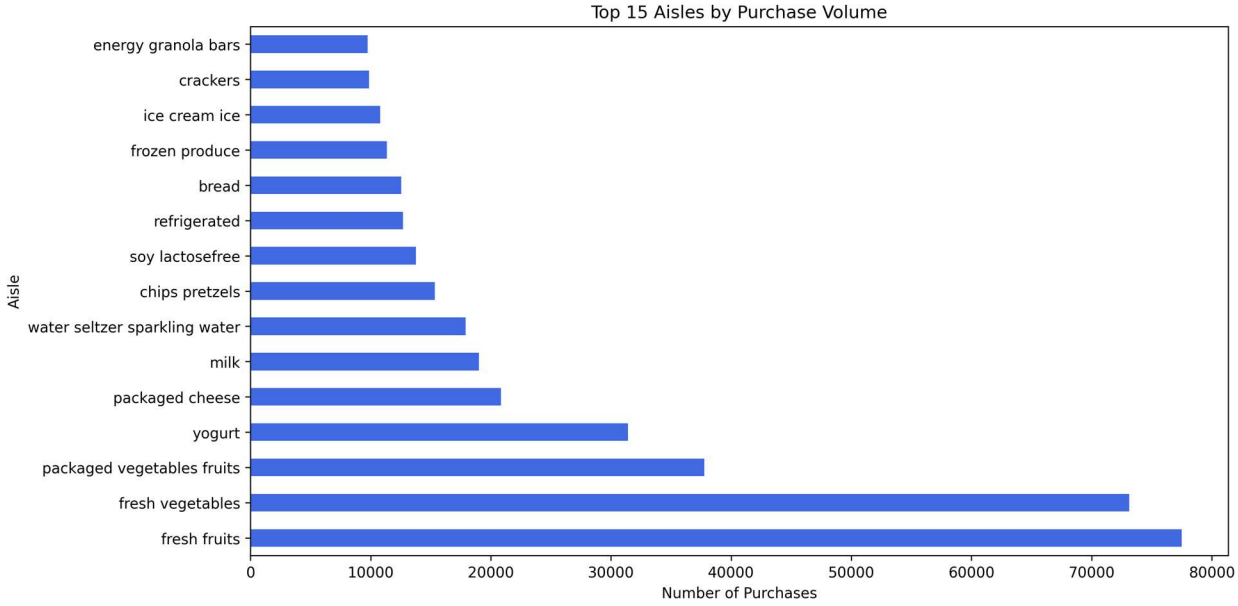
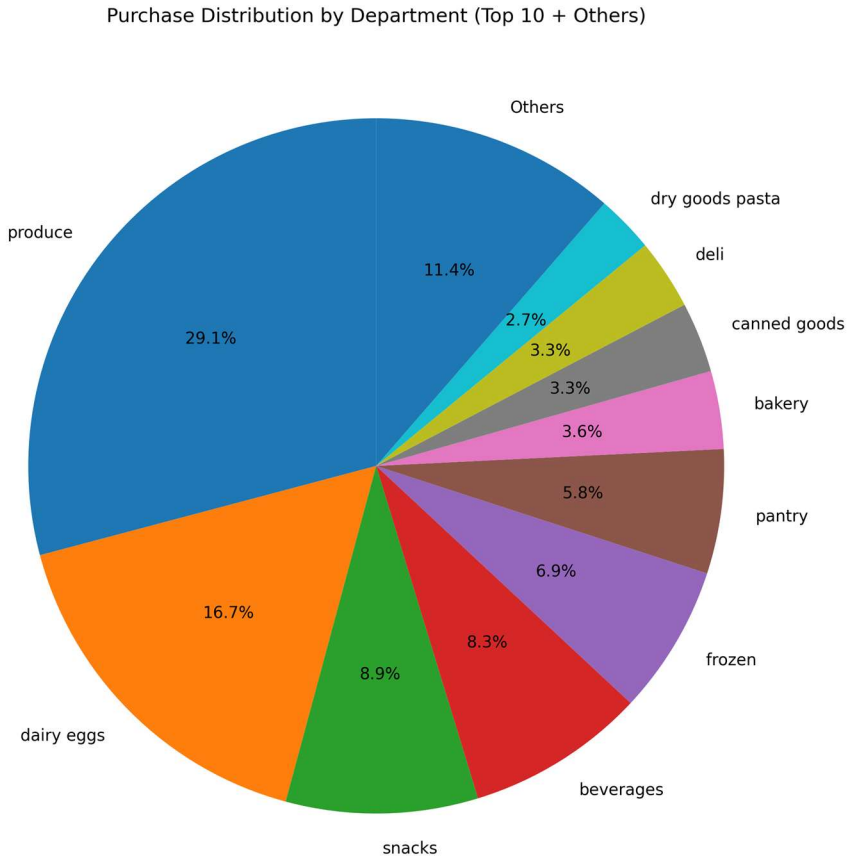


Figure 16


Purchase Distribution by Department (Top 10 + Others)
Others
dry goods pasta
deli
canned goods
bakery
pantry
frozen
beverages
snacks
dairy eggs
produce
11.4%
2.7%
3.3%
3.3%
3.6%
5.8%
6.9%
8.3%
8.9%
16.7%
29.1%