INFO 523: Data Mining & Discovery
Final Project by Group 8

shopping trends with instacart

Instacart Dataset (Introduction)

The Instacart dataset is a collection of anonymized grocery shopping data from real customers. It contains information about the products purchased, the order history, and the user demographics.

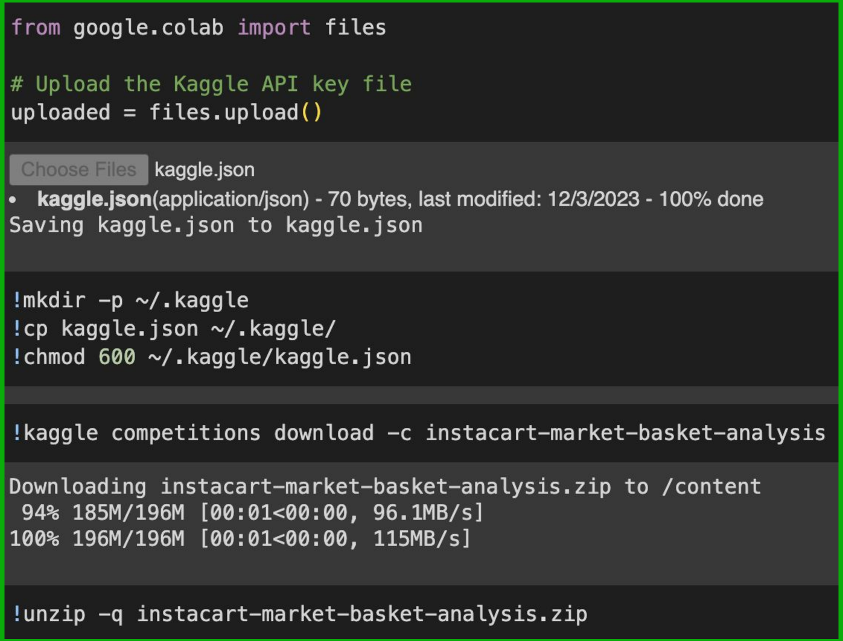
The dataset for this project is a relational set of files describing customers' orders over time. The goal of the project is to predict which products will be in a user's next order. The dataset is anonymized and contains a sample of over 3 million grocery orders from more than 200,000 Instacart users. For each user, we have between 4 and 100 of their orders, with the sequence of products purchased in each order. We also have the week and hour of day the order was placed, and a relative measure of time between orders.

File descriptions:

Each entity (customer, product, order, aisle, etc.) has an associated unique id.

Instacart Dataset Overview (ETL)

The instacart dataset used is taken from Instacart Market Basket Analysis dataset hosted by Instacrt and has been extracted using the Kaggle API



```
from google.colab import files

# Upload the Kaggle API key file
uploaded = files.upload()
```

Choose Files kaggle.json

- **kaggle.json**(application/json) - 70 bytes, last modified: 12/3/2023 - 100% done

```
Saving kaggle.json to kaggle.json
```

```
!mkdir -p ~/.kaggle
!cp kaggle.json ~/.kaggle/
!chmod 600 ~/.kaggle/kaggle.json
```

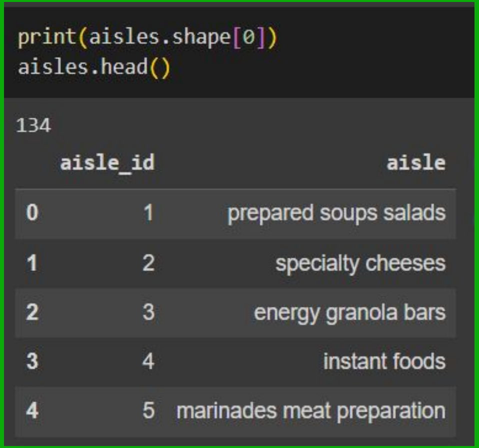
```
!kaggle competitions download -c instacart-market-basket-analysis
```

```
Downloading instacart-market-basket-analysis.zip to /content
 94% 185M/196M [00:01<00:00, 96.1MB/s]
100% 196M/196M [00:01<00:00, 115MB/s]
```

```
!unzip -q instacart-market-basket-analysis.zip
```

Instacart Dataset (Overview)

We have the following tables in the dataset, with the corresponding number of rows:

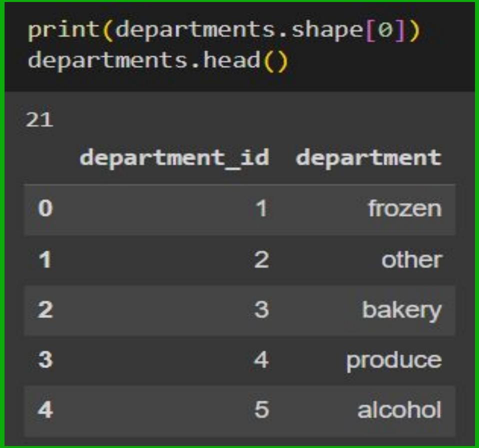


```
print(aisles.shape[0])
aisles.head()
```

```
134
```

	aisle_id	aisle
0	1	prepared soups salads
1	2	specialty cheeses
2	3	energy granola bars
3	4	instant foods
4	5	marinades meat preparation

ailes.csv

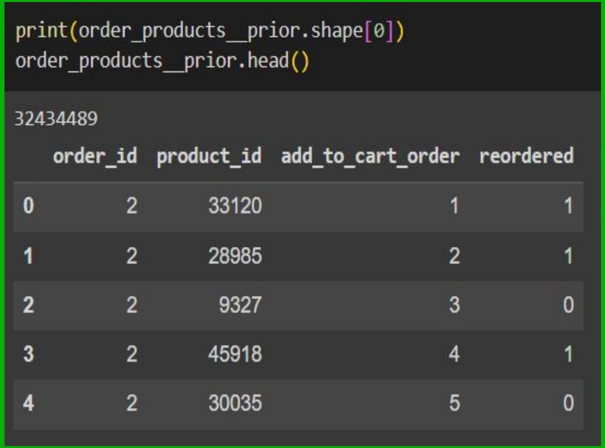


```
print(departments.shape[0])
departments.head()
```

```
21
```

	department_id	department
0	1	frozen
1	2	other
2	3	bakery
3	4	produce
4	5	alcohol

departments.csv

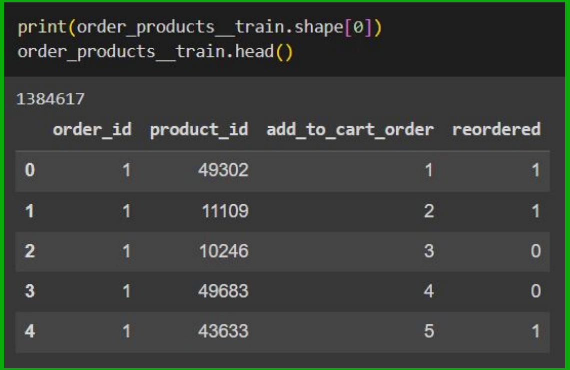


```
print(order_products__prior.shape[0])
order_products__prior.head()
```

```
32434489
```

	order_id	product_id	add_to_cart_order	reordered
0	2	33120	1	1
1	2	28985	2	1
2	2	9327	3	0
3	2	45918	4	1
4	2	30035	5	0

order_products_prior.csv

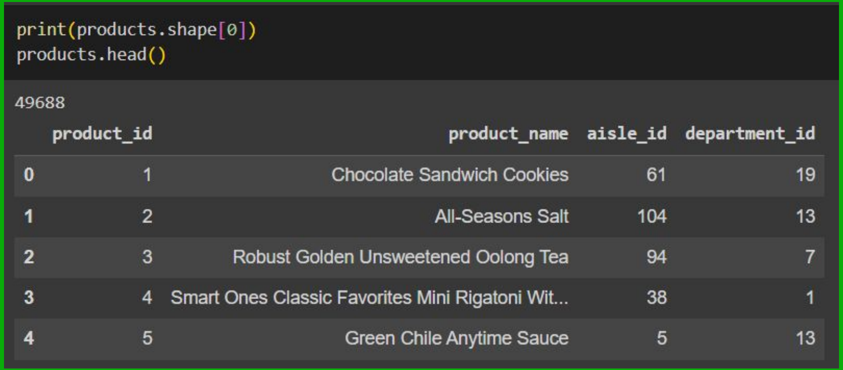


```
print(order_products__train.shape[0])
order_products__train.head()
```

```
1384617
```

	order_id	product_id	add_to_cart_order	reordered
0	1	49302	1	1
1	1	11109	2	1
2	1	10246	3	0
3	1	49683	4	0
4	1	43633	5	1

order_products_train.csv

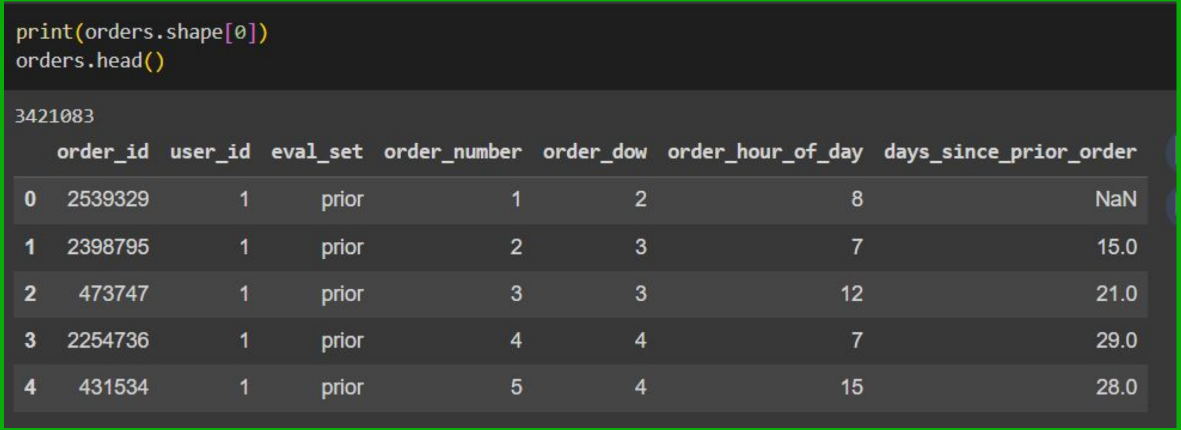


```
print(products.shape[0])
products.head()
```

```
49688
```

	product_id	product_name	aisle_id	department_id
0	1	Chocolate Sandwich Cookies	61	19
1	2	All-Seasons Salt	104	13
2	3	Robust Golden Unsweetened Oolong Tea	94	7
3	4	Smart Ones Classic Favorites Mini Rigatoni Wit...	38	1
4	5	Green Chile Anytime Sauce	5	13

products.csv



```
print(orders.shape[0])
orders.head()
```

```
3421083
```

	order_id	user_id	eval_set	order_number	order_dow	order_hour_of_day	days_since_prior_order
0	2539329	1	prior	1	2	8	NaN
1	2398795	1	prior	2	3	7	15.0
2	473747	1	prior	3	3	12	21.0
3	2254736	1	prior	4	4	7	29.0
4	431534	1	prior	5	4	15	28.0

orders.csv

Data Cleaning and Initial Analysis

➔ **Row Count Calculation**:

Row count calculation is the process of determining the number of rows or observations present in a dataset. In Python, for example, you can use the .shape attribute of a DataFrame to retrieve the count of rows (shape[0]). This count represents the total number of records or instances in the dataset.

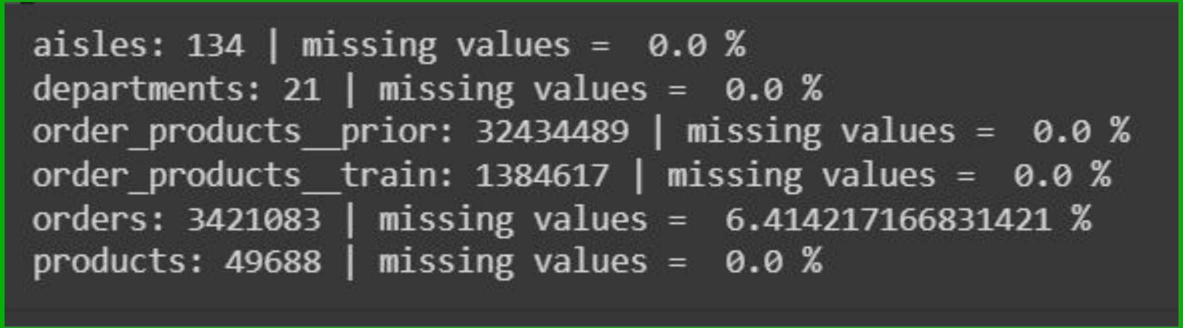
```
# To get the row count of a DataFrame called 'data':

## row_count = data.shape[0]
```

➔ **Missing Value Calculation:**

Missing value calculation involves determining the presence and quantity of missing or NaN (Not a Number) values within a dataset. This calculation often involves comparing the total count of rows against the count of non-missing or non-null values.

## missing_values_percentage = 100 * (total_rows - non_missing_rows) / non_missing_rows

For each dataframe (aisles, departments, etc.), the missing values are calculated by finding the percentage difference between the total number of rows and the number of rows after dropping the missing values (dropna() removes rows with any missing values). The percentage of missing values indicates the proportion of rows that contain at least one missing value compared to the total non-missing rows in the respective dataframe.



```
aisles: 134 | missing values =  0.0 %
departments: 21 | missing values =  0.0 %
order_products__prior: 32434489 | missing values =  0.0 %
order_products__train: 1384617 | missing values =  0.0 %
orders: 3421083 | missing values =  6.414217166831421 %
products: 49688 | missing values =  0.0 %
```

The above image shows the missing values in the dataset.

Exploratory Data Analysis

Exploratory data analysis (EDA) is an important step in the data mining process. It involves visualizing and summarizing the data to gain insights and identify patterns. In the context of the Instacart dataset, EDA can help us understand customer behavior, preferences, and shopping patterns. Some common techniques used in EDA include data visualization, summary statistics, and correlation analysis.

Data Visualization

Data visualization is a powerful tool for exploring and understanding complex datasets. It allows us to visually represent the data in the form of charts, graphs, and plots.

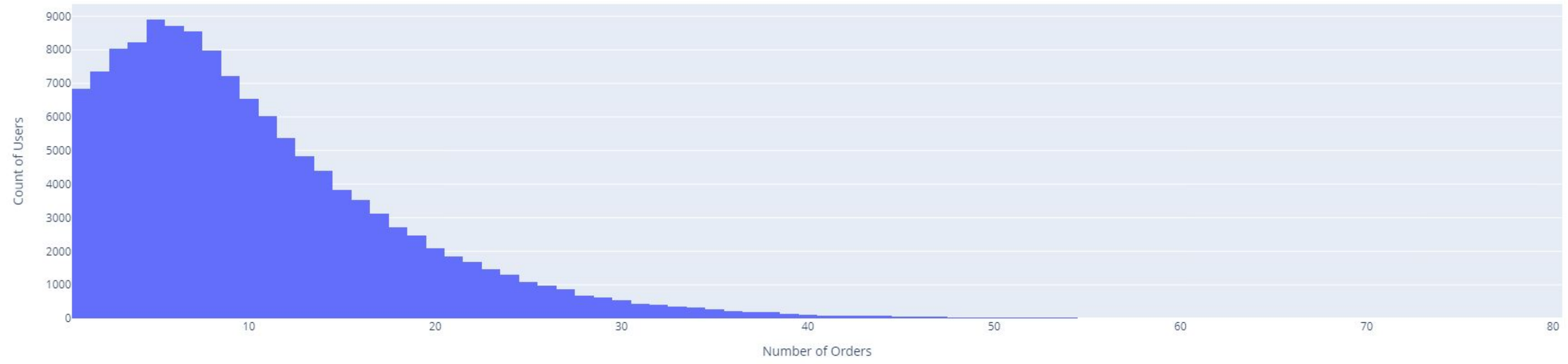
Summary Statistics

Summary statistics provide a concise summary of the main characteristics of a dataset. They include measures such as mean, median, mode, standard deviation, and range.

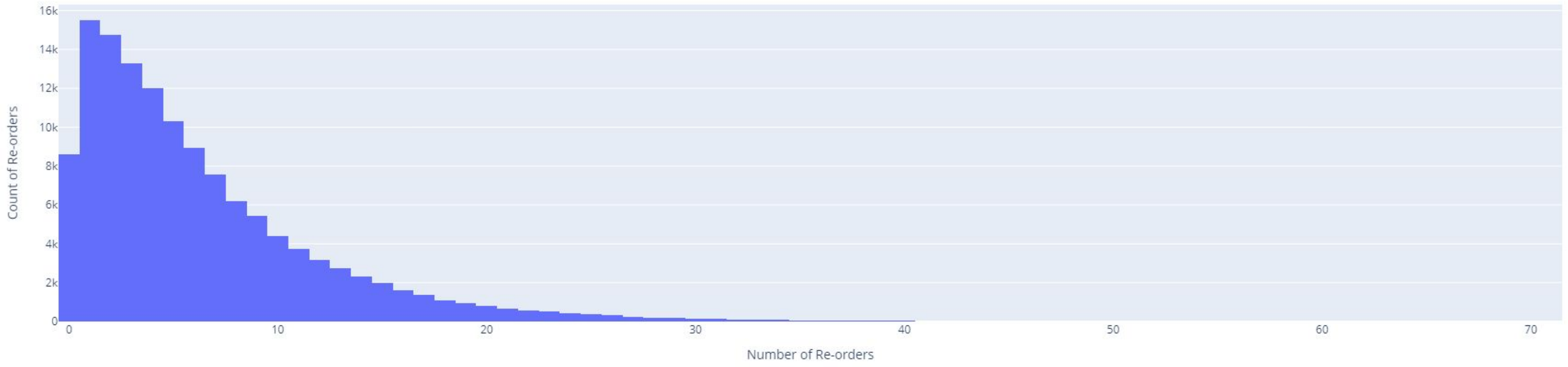
Correlation Analysis

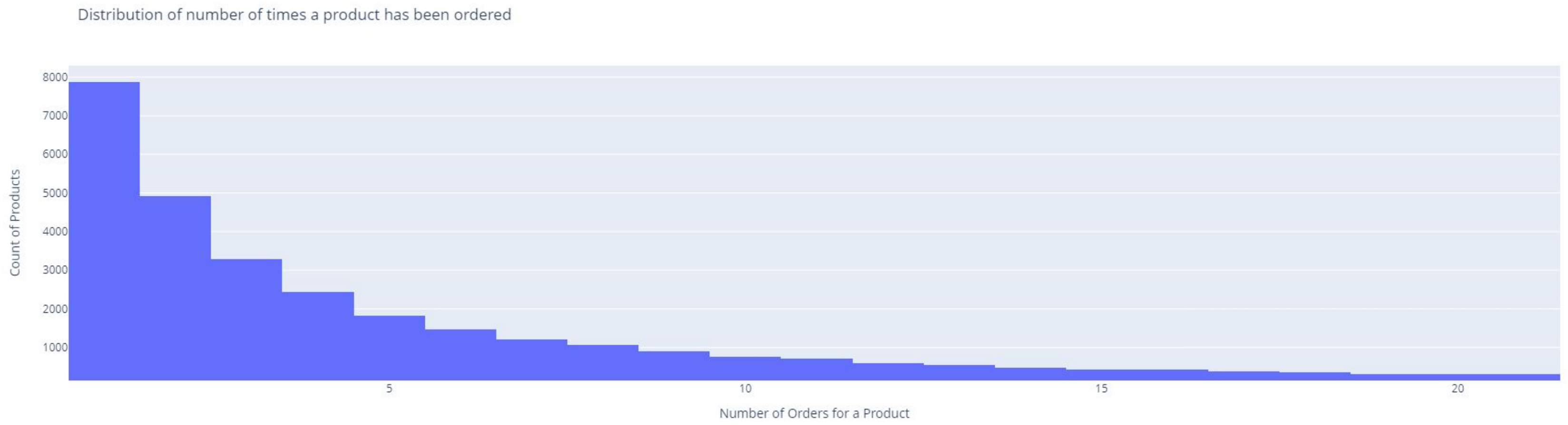
Correlation analysis helps us understand the relationship between different variables in the dataset. It measures the strength and direction of the linear relationship between two variables.

Distribution of number of orders placed by users

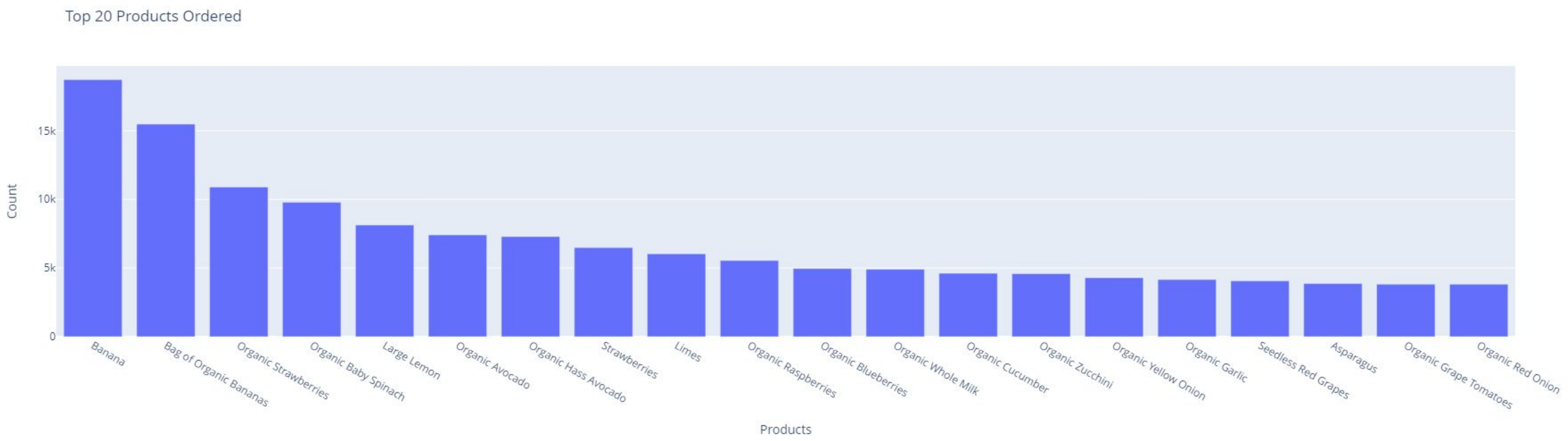


Distribution of number of products re-ordered

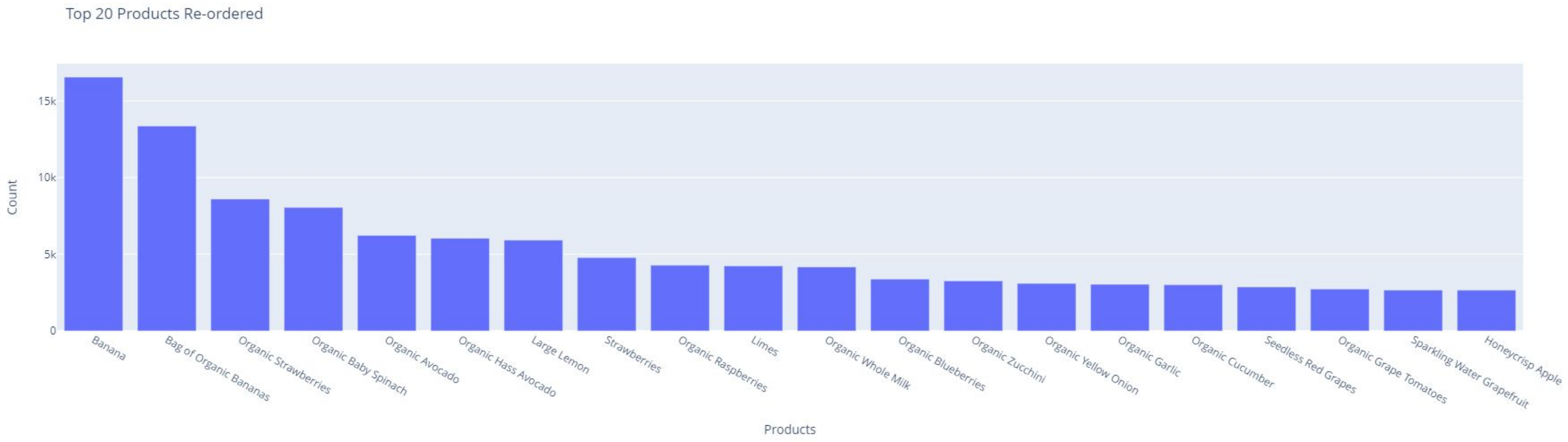




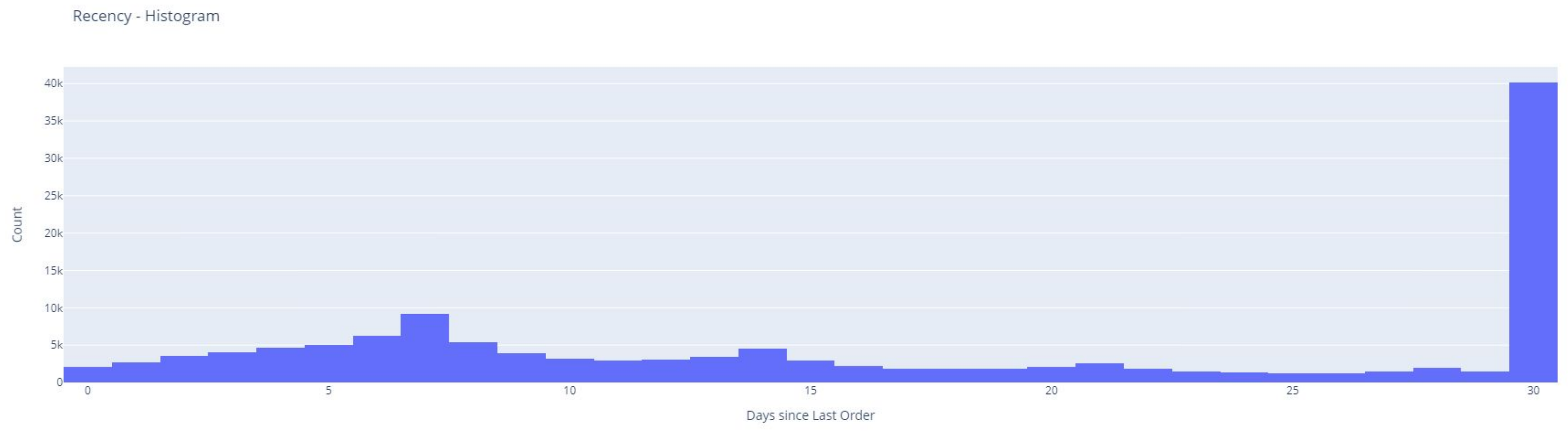
Distribution of number of times a product has been ordered
Count of Products
8000
7000
6000
5000
4000
3000
2000
1000
5
10
15
20
Number of Orders for a Product

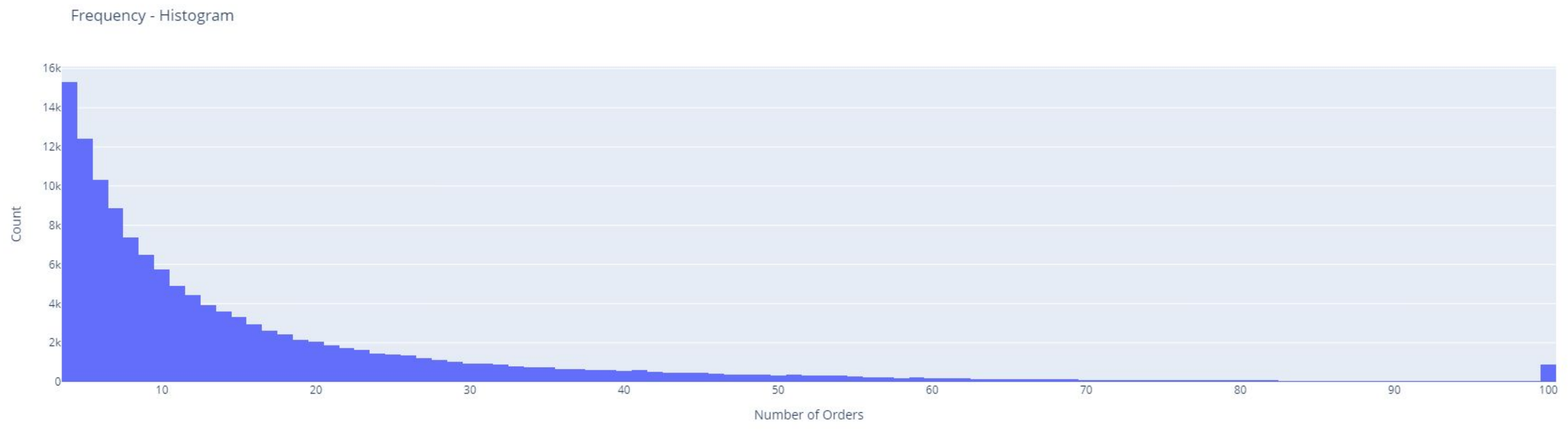


Top 20 Products Ordered
Count
15k
10k
5k
0
Banana
Bag of Organic Bananas
Organic Strawberries
Organic Baby Spinach
Large Lemon
Organic Avocado
Organic Hass Avocado
Strawberries
Limes
Organic Raspberries
Organic Blueberries
Organic Whole Milk
Organic Cucumber
Organic Zucchini
Organic Yellow Onion
Organic Garlic
Seedless Red Grapes
Asparagus
Organic Grape Tomatoes
Organic Red Onion
Products

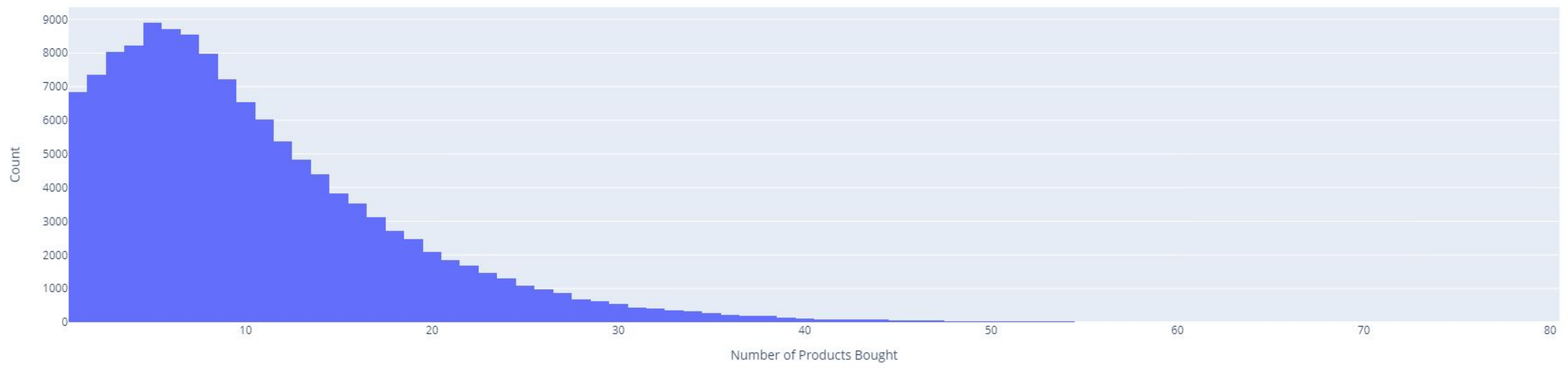


Top 20 Products Re-ordered
Count
15k
10k
5k
0
Banana
Bag of Organic Bananas
Organic Strawberries
Organic Baby Spinach
Organic Avocado
Organic Hass Avocado
Large Lemon
Strawberries
Organic Raspberries
Limes
Organic Whole Milk
Organic Blueberries
Organic Zucchini
Organic Yellow Onion
Organic Garlic
Organic Cucumber
Seedless Red Grapes
Organic Grape Tomatoes
Sparkling Water Grapefruit
Honeycrisp Apple
Products


Recency - Histogram
Count
40k
35k
30k
25k
20k
15k
10k
5k
0
0
5
10
15
20
25
30
Days since Last Order


Frequency - Histogram
Count
16k
14k
12k
10k
8k
6k
4k
2k
0
10
20
30
40
50
60
70
80
90
100
Number of Orders

Monetary - Histogram



Data Preprocessing

Data preprocessing is an essential step in data mining and involves cleaning and transforming raw data to prepare it for analysis. Some techniques used in this dataset preprocessing include:

Handling Missing Values:

One important aspect of data preprocessing is handling missing values. This can be done by either removing the rows or columns with missing values, or by imputing the missing values with appropriate values.

Removing Duplicates:

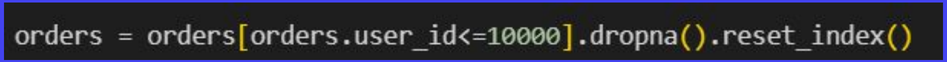
Another technique in data preprocessing is removing duplicate entries. Duplicate data can skew analysis results and introduce biases, so it is important to identify and remove them before proceeding with further analysis.

Scaling:

Scaling in the context of data mining refers to the process of adjusting the range or distribution of numeric variables or features within a dataset. It's a crucial preprocessing step that aims to bring all features to a similar scale or range.

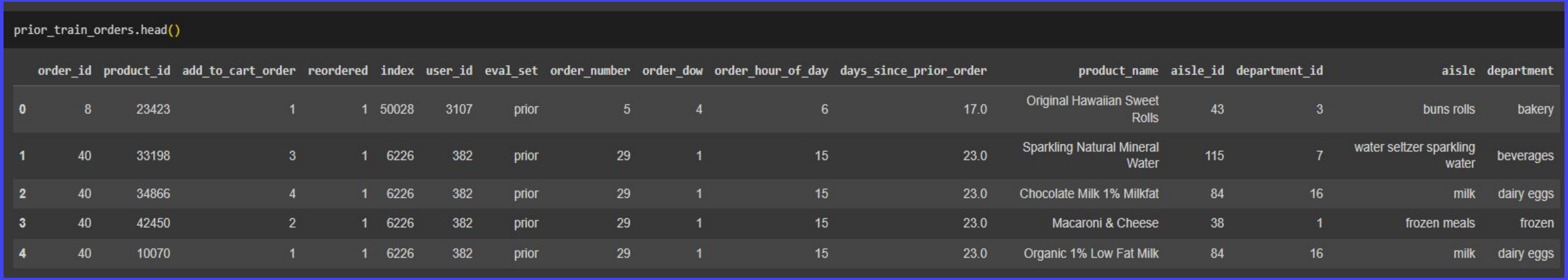
PreProcessing Steps:

- Sampling orders table: We consider the first 10000 users for processing convenience.



```
orders = orders[orders.user_id<=10000].dropna().reset_index()
```

- Merging all dataset into a master dataframe:

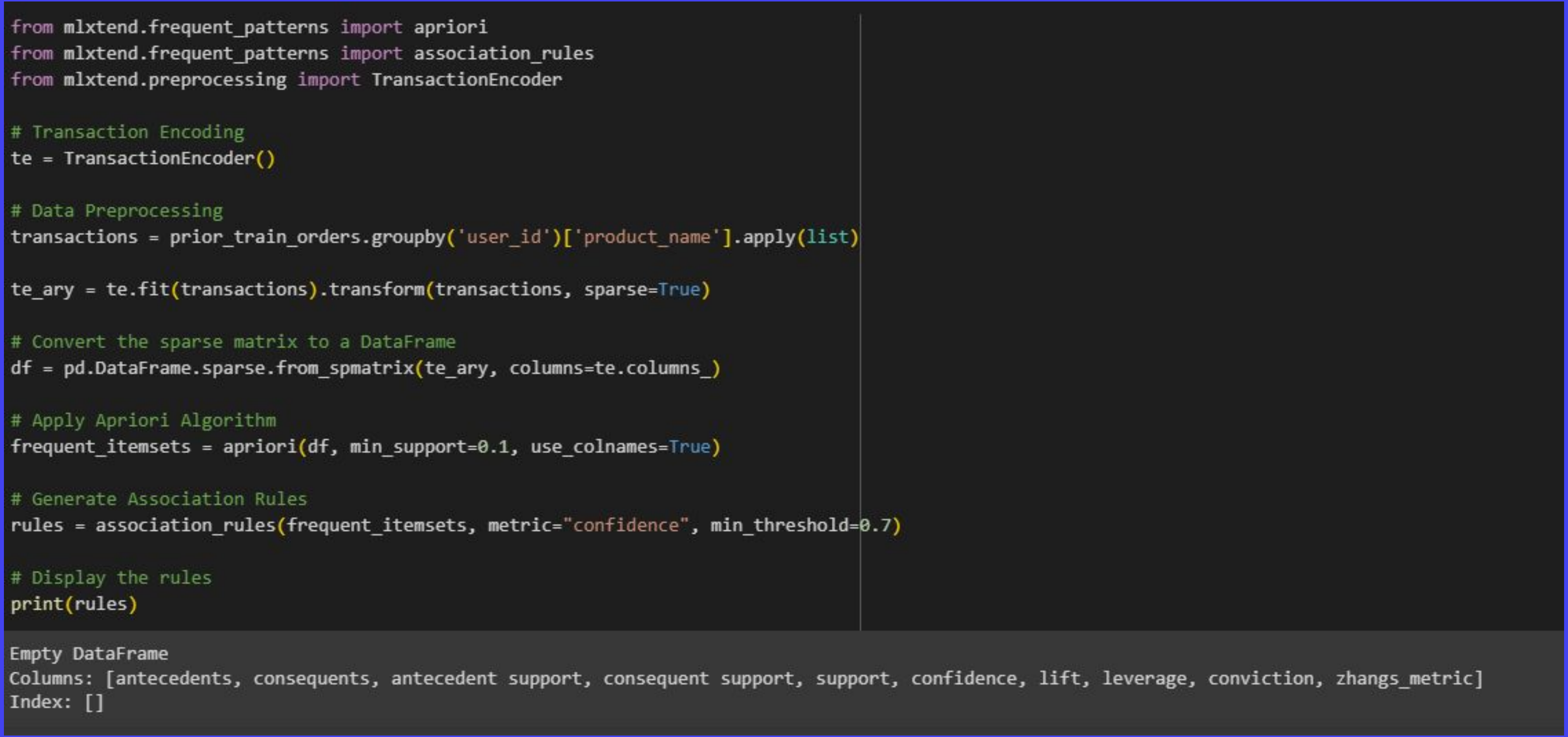


```
prior_train_orders.head()
```

	order_id	product_id	add_to_cart_order	reordered	index	user_id	eval_set	order_number	order_dow	order_hour_of_day	days_since_prior_order	product_name	aisle_id	department_id	aisle	department
0	8	23423	1	1	50028	3107	prior	5	4	6	17.0	Original Hawaiian Sweet Rolls	43	3	buns rolls	bakery
1	40	33198	3	1	6226	382	prior	29	1	15	23.0	Sparkling Natural Mineral Water	115	7	water seltzer sparkling water	beverages
2	40	34866	4	1	6226	382	prior	29	1	15	23.0	Chocolate Milk 1% Milkfat	84	16	milk	dairy eggs
3	40	42450	2	1	6226	382	prior	29	1	15	23.0	Macaroni & Cheese	38	1	frozen meals	frozen
4	40	10070	1	1	6226	382	prior	29	1	15	23.0	Organic 1% Low Fat Milk	84	16	milk	dairy eggs

Association Rule Mining

- Preprocessing of the data is done the same as earlier in clustering and then we applied Apriori Algorithm on the master table directly. But it does not works so we tried processing data.



```
from mlxtend.frequent_patterns import apriori
from mlxtend.frequent_patterns import association_rules
from mlxtend.preprocessing import TransactionEncoder

# Transaction Encoding
te = TransactionEncoder()

# Data Preprocessing
transactions = prior_train_orders.groupby('user_id')['product_name'].apply(list)

te_ary = te.fit(transactions).transform(transactions, sparse=True)

# Convert the sparse matrix to a DataFrame
df = pd.DataFrame.sparse.from_spmatrix(te_ary, columns=te.columns_)

# Apply Apriori Algorithm
frequent_itemsets = apriori(df, min_support=0.1, use_colnames=True)

# Generate Association Rules
rules = association_rules(frequent_itemsets, metric="confidence", min_threshold=0.7)

# Display the rules
print(rules)
```

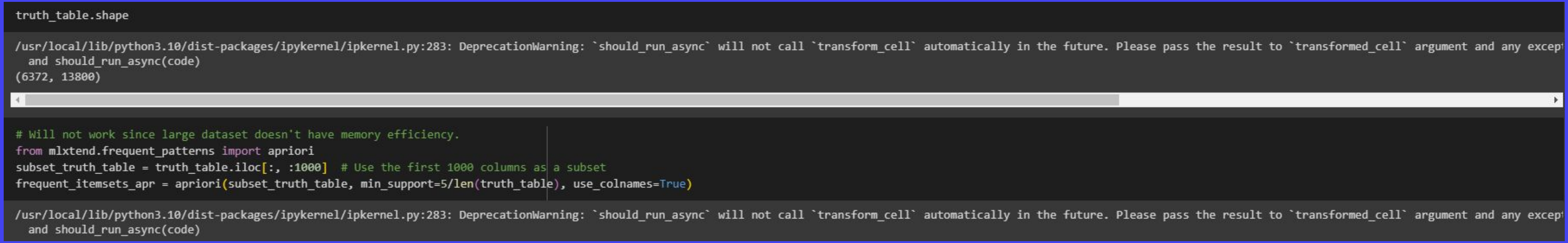
```
Empty DataFrame
Columns: [antecedents, consequents, antecedent support, consequent support, support, confidence, lift, leverage, conviction, zhangs_metric]
Index: []
```

Steps taken for processing data

The following steps are taken for preparation of data:

- Initializing an empty list called 'pivots'.
- Looping through 'small_train_split':
    - Reshaping the DataFrame by setting 'order_id' as the index and 'product_id' as columns using the pivot function.
    - Converting missing values to a sparse matrix format.
    - Appending the sparse matrix to the 'pivots' list.
- Memory optimization by deleting unnecessary variables ('pvt').
- Creating a sorted list of unique product IDs from the 'product_id' column in 'small_train'

Generating Frequent Itemsets:



```
truth_table.shape
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarning: `should_run_async` will not call `transform_cell` automatically in the future. Please pass the result to `transformed_cell` argument and any excep
  and should_run_async(code)
(6372, 13800)
```

```
# Will not work since large dataset doesn't have memory efficiency.
from mlxtend.frequent_patterns import apriori
subset_truth_table = truth_table.iloc[:, :1000]  # Use the first 1000 columns as a subset
frequent_itemsets_apr = apriori(subset_truth_table, min_support=5/len(truth_table), use_colnames=True)
```

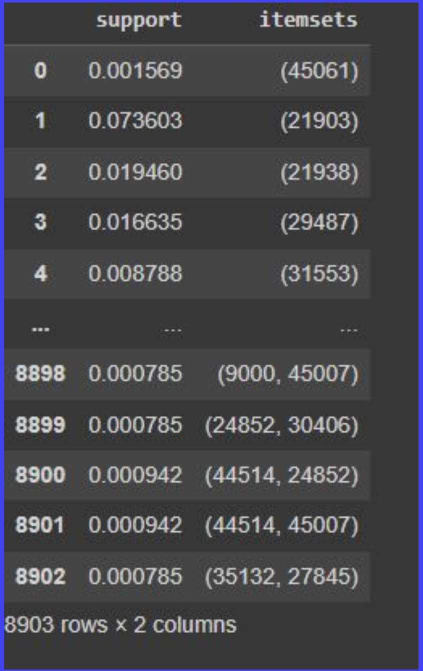
```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarning: `should_run_async` will not call `transform_cell` automatically in the future. Please pass the result to `transformed_cell` argument and any excep
  and should_run_async(code)
```

- Here, the association rules for the frequent itemsets subset generated by Apriori is an empty dataframe - since there are no itemsets which fulfill the minimum support threshold value to generate rules that can give us a rule based association.
- If we use a larger dataset (which we have) - Apriori crashes the session pretty consistently due to the lack of computational resources (which is expensive) and lack of memory needed for the operation to be performed. So we proceed with another algorithm.

Applying FP Growth

- FP Growth, which stands for "Frequent Pattern Growth," is a data mining algorithm used for finding frequent itemsets in a transactional database. It's an alternative to the Apriori algorithm and is particularly efficient for large datasets. Since the dataset we're utilizing is a huge dataset with millions of rows, FP Growth is preferred to Apriori algorithm.

- fp growth: This is a function from the mlxtend library, commonly used for frequent pattern mining using the FP-Growth algorithm.



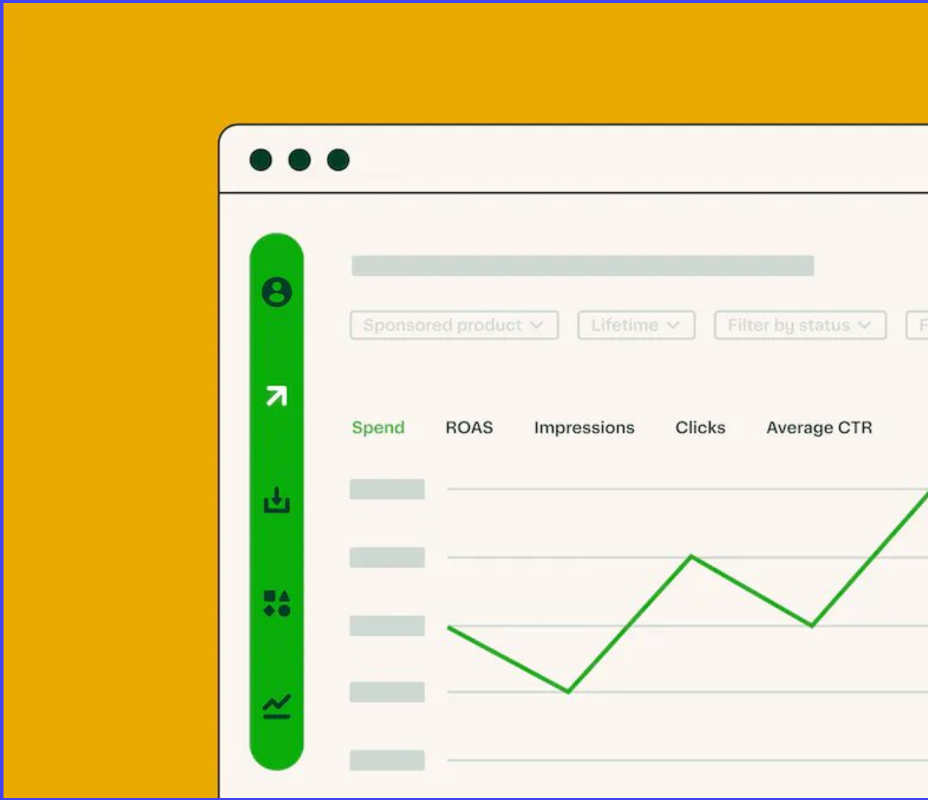
	support	itemsets
0	0.001569	(45061)
1	0.073603	(21903)
2	0.019460	(21938)
3	0.016635	(29487)
4	0.008788	(31553)
...	...	...
8898	0.000785	(9000, 45007)
8899	0.000785	(24852, 30406)
8900	0.000942	(44514, 24852)
8901	0.000942	(44514, 45007)
8902	0.000785	(35132, 27845)

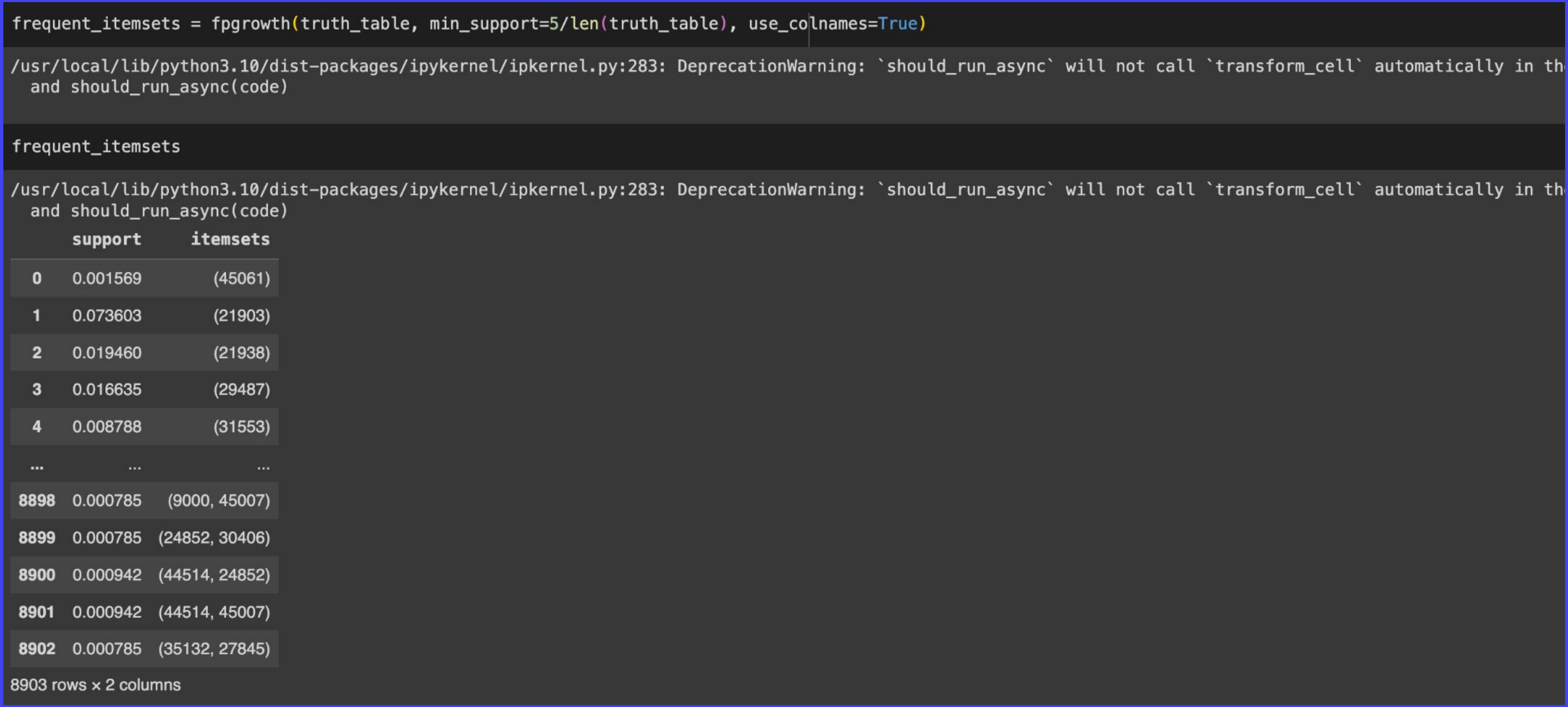
8903 rows × 2 columns

Why using FP-Growth?

The choice of the FP-Growth algorithm and the minimum support threshold are crucial parameters in association rule mining. FP-Growth is particularly efficient in handling large datasets compared to traditional Apriori-based approaches.

The minimum support threshold is a tuning parameter that influences the number and specificity of the discovered patterns; a lower threshold will yield more patterns, but they may be less significant. Adjusting these parameters depends on the characteristics of the dataset and the goals of the analysis.





```
frequent_itemsets = fpgrowth(truth_table, min_support=5/len(truth_table), use_colnames=True)
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarning: `should_run_async` will not call `transform_cell` automatically in th
  and should_run_async(code)
```

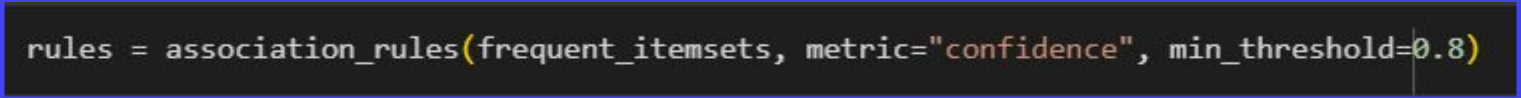
```
frequent_itemsets
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarning: `should_run_async` will not call `transform_cell` automatically in th
  and should_run_async(code)
```

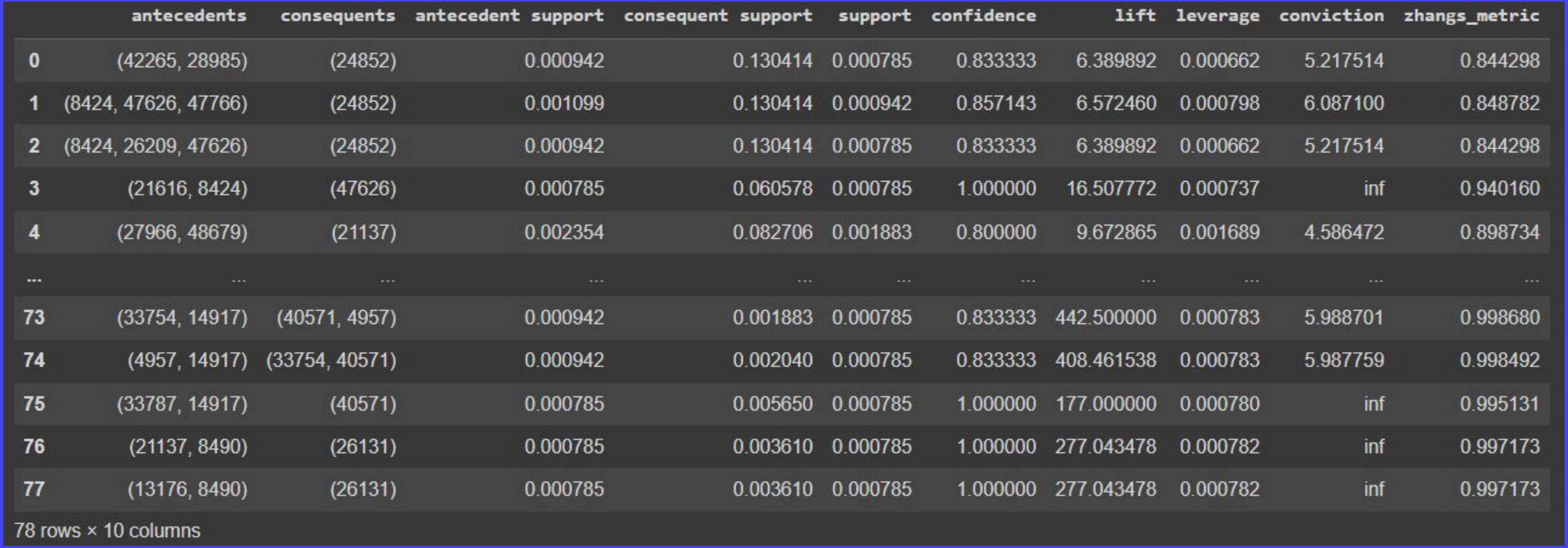
	support	itemsets
0	0.001569	(45061)
1	0.073603	(21903)
2	0.019460	(21938)
3	0.016635	(29487)
4	0.008788	(31553)
...	...	...
8898	0.000785	(9000, 45007)
8899	0.000785	(24852, 30406)
8900	0.000942	(44514, 24852)
8901	0.000942	(44514, 45007)
8902	0.000785	(35132, 27845)

8903 rows × 2 columns

- The association_rules function generates rules from frequent itemsets (obtained via FP-Growth) with a minimum confidence of 80%. It evaluates rules based on confidence, measuring reliability by comparing combined item support to the antecedent support.
- Resulting rules contain columns like antecedents, consequents, support, confidence, lift, leverage, and conviction, offering insights into the strength and significance of associations within the dataset.



```
rules = association_rules(frequent_itemsets, metric="confidence", min_threshold=0.8)
```



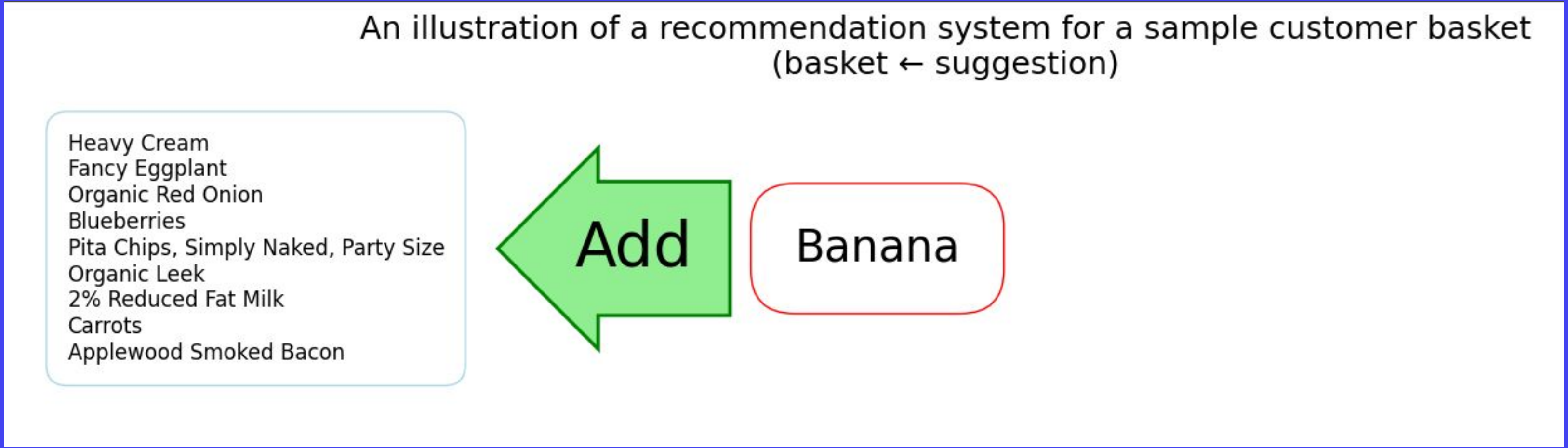
	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction	zhangs_metric
0	(42265, 28985)	(24852)	0.000942	0.130414	0.000785	0.833333	6.389892	0.000662	5.217514	0.844298
1	(8424, 47626, 47766)	(24852)	0.001099	0.130414	0.000942	0.857143	6.572460	0.000798	6.087100	0.848782
2	(8424, 26209, 47626)	(24852)	0.000942	0.130414	0.000785	0.833333	6.389892	0.000662	5.217514	0.844298
3	(21616, 8424)	(47626)	0.000785	0.060578	0.000785	1.000000	16.507772	0.000737	inf	0.940160
4	(27966, 48679)	(21137)	0.002354	0.082706	0.001883	0.800000	9.672865	0.001689	4.586472	0.898734
...	...	...	...	...	...	...	...	...	...	...
73	(33754, 14917)	(40571, 4957)	0.000942	0.001883	0.000785	0.833333	442.500000	0.000783	5.988701	0.998680
74	(4957, 14917)	(33754, 40571)	0.000942	0.002040	0.000785	0.833333	408.461538	0.000783	5.987759	0.998492
75	(33787, 14917)	(40571)	0.000785	0.005650	0.000785	1.000000	177.000000	0.000780	inf	0.995131
76	(21137, 8490)	(26131)	0.000785	0.003610	0.000785	1.000000	277.043478	0.000782	inf	0.997173
77	(13176, 8490)	(26131)	0.000785	0.003610	0.000785	1.000000	277.043478	0.000782	inf	0.997173

78 rows × 10 columns

The antecedents shows product id of the previous purchases and consequents shows what next product the customer will be likely to purchase. Followed by the support, confidence, lift and leverage Measures the difference between the observed frequency of the itemset and the frequency that would be expected if the items were independent.

Market Basket Analysis

Market basket analysis is a specific application of association rule mining that focuses on analyzing the items that are frequently purchased together. It can be used to understand customer behavior and make recommendations.



Clustering - RFM Analysis

Picking up important details about users' orders and condenses them into a table that gives a quick overview of how active users are (how often they order), how recent their last order was, and how diverse their shopping habits are (how many different products they buy).

R (Recency) - Days since last order

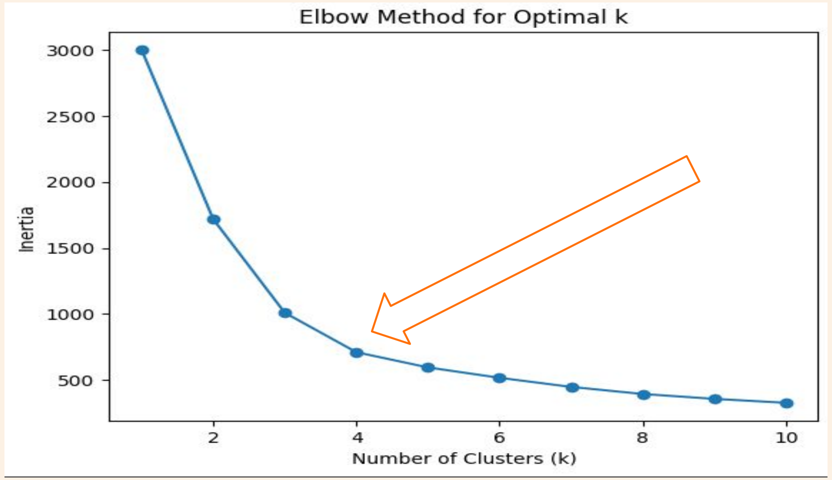
F (Frequency) - Number of orders

M (Monetary) - Number of products purchased

We start by scaling specific columns (Recency, Frequency, Monetary) in the rfm_df DataFrame using StandardScaler from scikit-learn, which will likely be used in subsequent clustering or analysis tasks.

Clustering - How many clusters?

Elbow method - Identifies the point on the plot where the rate of decrease in inertia (sum of squared distances within clusters) slows down, indicating a point of diminishing returns in improving clustering performance as the number of clusters increases. The optimal number of clusters can be chosen based on the location of this "elbow" in the plot.



Clustering Techniques

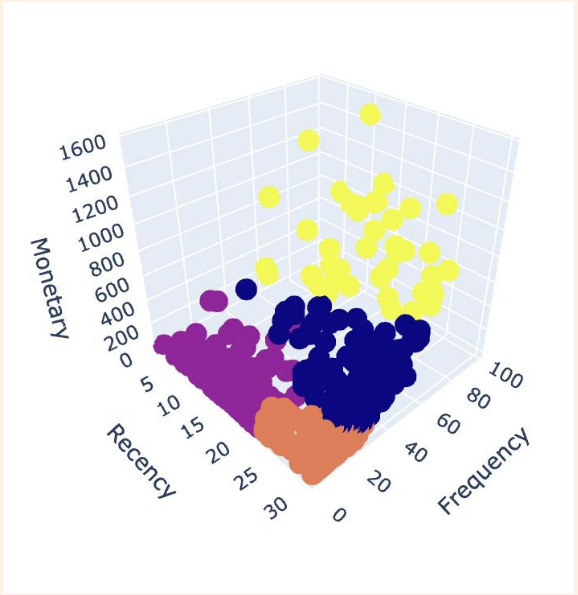
K-means Clustering:

Algorithm: K-means partitions the dataset into k clusters by minimizing the sum of squared distances between data points and the centroids of their assigned clusters.

Sensitivity to Outliers: K-means is sensitive to outliers because it minimizes the sum of squared distances, making it susceptible to being pulled towards outliers.

Shape of Clusters: K-means assumes that clusters are spherical and equally sized, which may not always be the case in real-world data.

Silhouette Score (K-Means): 0.57



Clustering Techniques

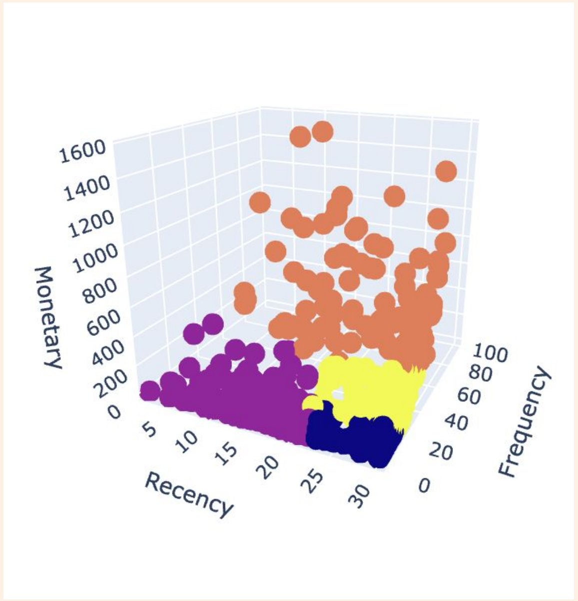
K-medoids Clustering:

Algorithm: K-medoids, a variant of k-means, uses actual data points (medoids) as cluster representatives. It is implemented using the Partitioning Around Medoids (PAM) algorithm. This makes it more robust to outliers than k-means.

Robustness to Outliers: K-medoids is less sensitive to outliers compared to k-means because it uses medoids (actual data points) as representatives, making it more robust.

Suitability for RFM Analysis: K-medoids can be suitable for RFM analysis as it may handle outliers in the monetary dimension better than k-means.

Silhouette Score (PAM): 0.46



Clustering Techniques

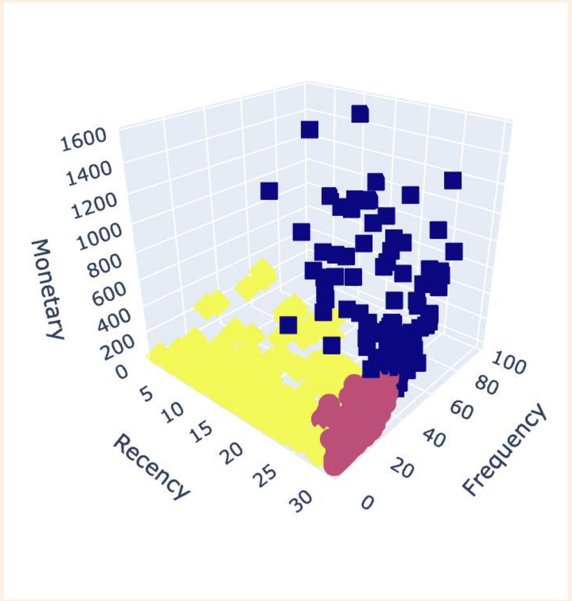
Hierarchical Clustering:

Algorithm: Hierarchical clustering builds a tree of clusters, either agglomerative (bottom-up) or divisively (top-down), based on the similarity between data points.

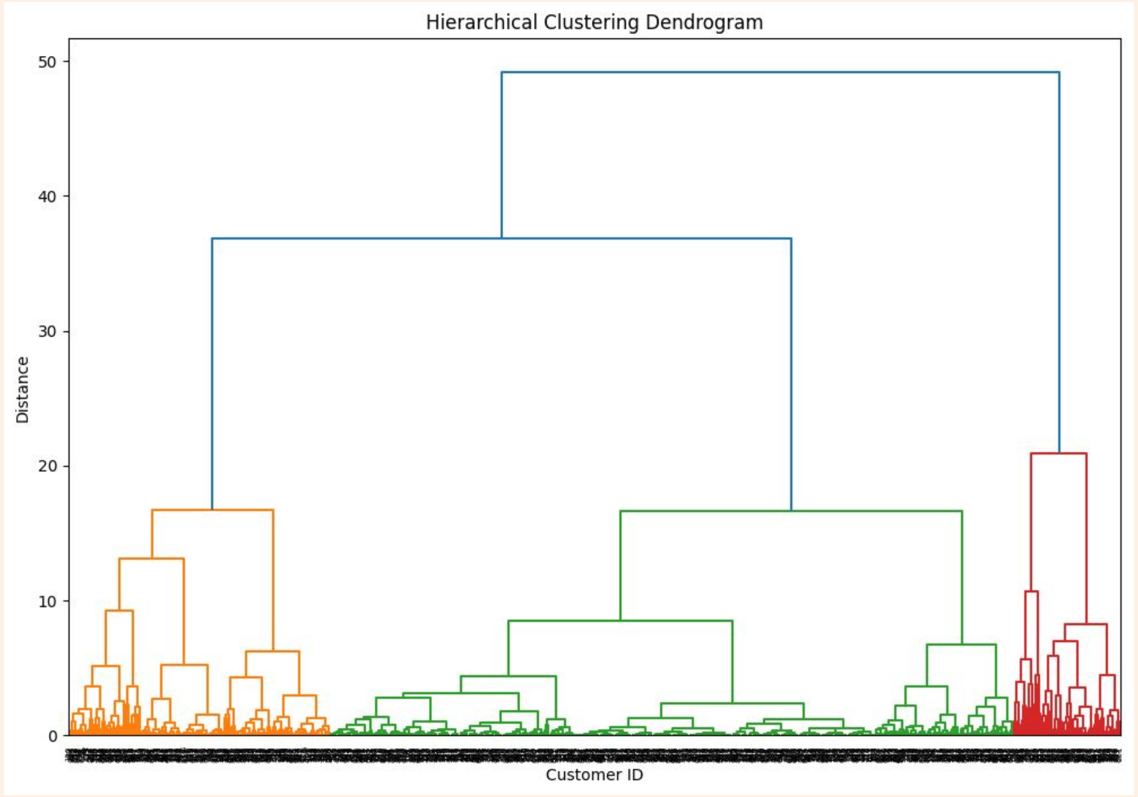
Interpretability: Hierarchical clustering provides a visual representation of the clustering structure through dendrograms, making it easier to interpret the hierarchy of clusters.

Flexibility: Hierarchical clustering doesn't require specifying the number of clusters beforehand, which can be an advantage. It allows you to explore different levels of granularity.

Silhouette Score (PAM): 0.53

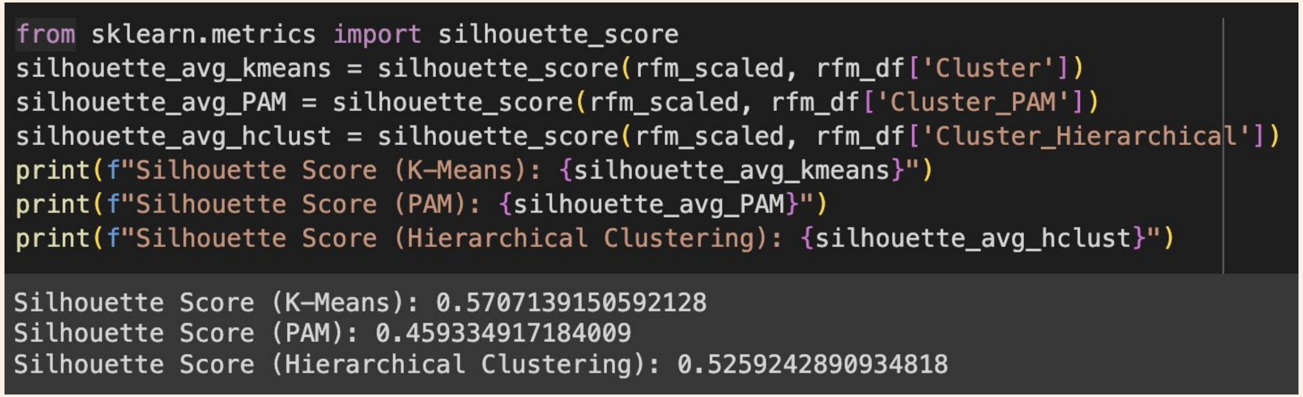


Hierarchical clustering was used to group customers based on their buying patterns. The dendrogram shows how similar or different customers are in their purchases. It helps identify clusters of customers with similar behaviors, enabling businesses to personalize marketing and better understand their customer base for tailored strategies and improved satisfaction.



Performance Evaluation for Clustering

The silhouette scores for K-Means, PAM (Partitioning Around Medoids), and Hierarchical Clustering provide insights into how well each method formed clusters. Higher scores indicate better-defined clusters. Comparing these scores helps determine which method best captures distinct customer segments for targeted marketing strategies or personalized experiences.



```python
from sklearn.metrics import silhouette_score
silhouette_avg_kmeans = silhouette_score(rfm_scaled, rfm_df['Cluster'])
silhouette_avg_PAM = silhouette_score(rfm_scaled, rfm_df['Cluster_PAM'])
silhouette_avg_hclust = silhouette_score(rfm_scaled, rfm_df['Cluster_Hierarchical'])
print(f"Silhouette Score (K-Means): {silhouette_avg_kmeans}")
print(f"Silhouette Score (PAM): {silhouette_avg_PAM}")
print(f"Silhouette Score (Hierarchical Clustering): {silhouette_avg_hclust}")
```

```
Silhouette Score (K-Means): 0.5707139150592128
Silhouette Score (PAM): 0.459334917184009
Silhouette Score (Hierarchical Clustering): 0.5259242890934818
```

Classification - Re-order Predictions

Learns important details about users' past orders and shopping patterns to predict what the user is likely to re-order in their next purchase. These recommendations can be delivered to the users' for a fast and smooth shopping experience.

Features:

- Categorical
  - user_id (ID of the user)
  - product_id (ID of the product)
  - aisle_id (broader category of product)
  - department_id (broader category of aisle)
- Numerical
  - add_to_cart_order (Priority of item being bought)

Target:

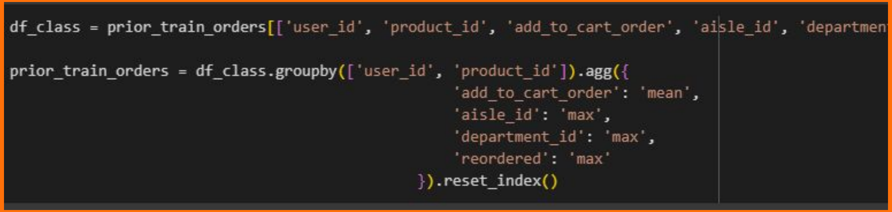
- reordered (whether a user re-orders a certain product)

Classification - Pre-processing

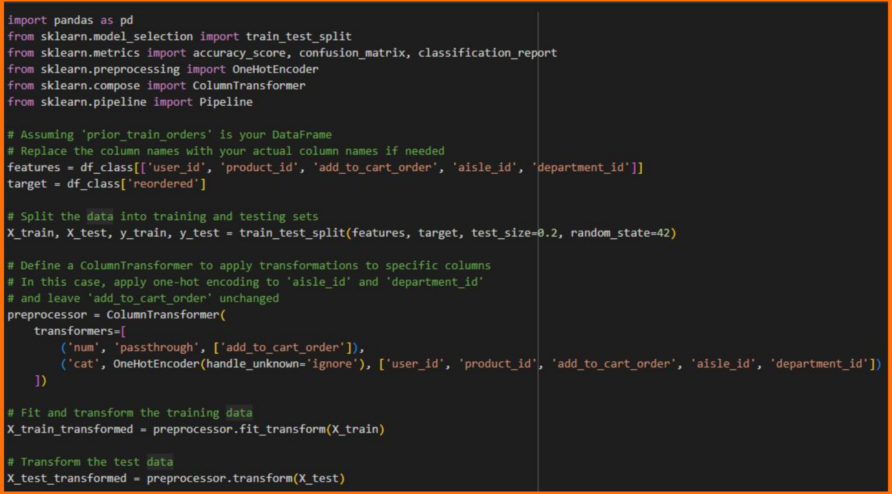
While grouping by user-product pairs - We take the mean of ‘add_to_cart_order’ to get a general priority of a product for a user and take the max value of ‘reordered’ to find how many times has the user re-ordered a product. The aisle and department correspond uniquely to the product.

We then pick the 1000 of these customers’ shopping data and divide our it between test and train. This code is replicable to incorporate more customers.

We also set up a data transformation pipeline that handles categorical features by using one-hot encoding.



```python
df_class = prior_train_orders[['user_id', 'product_id', 'add_to_cart_order', 'aisle_id', 'departmen

prior_train_orders = df_class.groupby(['user_id', 'product_id']).agg({
                                        'add_to_cart_order': 'mean',
                                        'aisle_id': 'max',
                                        'department_id': 'max',
                                        'reordered': 'max'
                                    }).reset_index()
```



```python
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline

# Assuming 'prior_train_orders' is your DataFrame
# Replace the column names with your actual column names if needed
features = df_class[['user_id', 'product_id', 'add_to_cart_order', 'aisle_id', 'department_id']]
target = df_class['reordered']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(features, target, test_size=0.2, random_state=42)

# Define a ColumnTransformer to apply transformations to specific columns
# In this case, apply one-hot encoding to 'aisle_id' and 'department_id'
# and leave 'add_to_cart_order' unchanged
preprocessor = ColumnTransformer(
    transformers=[
        ('num', 'passthrough', ['add_to_cart_order']),
        ('cat', OneHotEncoder(handle_unknown='ignore'), ['user_id', 'product_id', 'add_to_cart_order', 'aisle_id', 'department_id'])
    ])

# Fit and transform the training data
X_train_transformed = preprocessor.fit_transform(X_train)

# Transform the test data
X_test_transformed = preprocessor.transform(X_test)
```

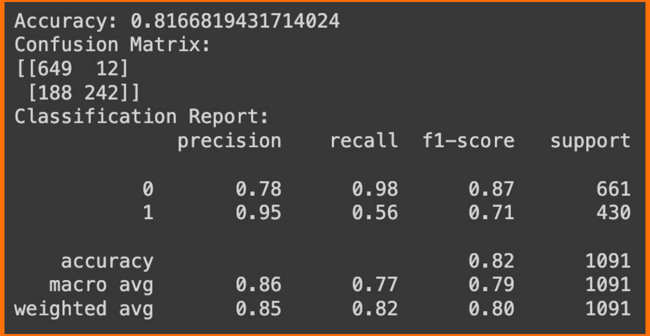
Classification Techniques

Support Vector Machines (SVM):

Algorithm: SVM is a supervised learning algorithm that separates data points into different classes by finding the hyperplane that maximizes the margin between classes.

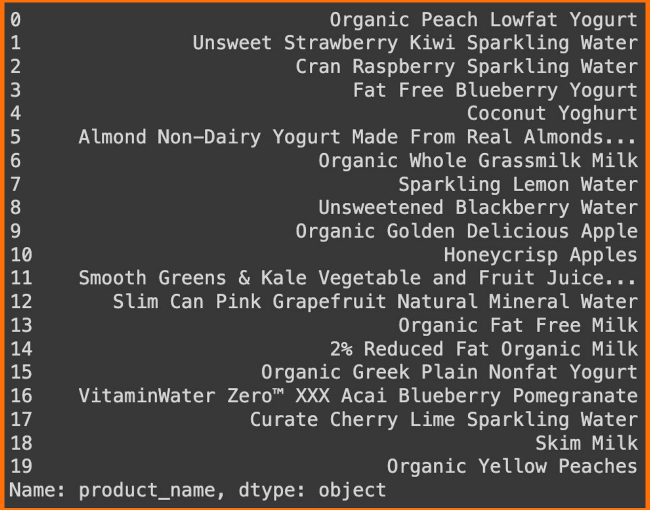
Non-linearity: SVM can handle non-linear decision boundaries through the use of kernel functions.

Interpretability: SVMs can be less interpretable compared to logistic regression, especially when using complex kernels.



```
Accuracy: 0.8166819431714024
Confusion Matrix:
[[649  12]
 [188 242]]
Classification Report:
              precision    recall  f1-score   support

           0       0.78      0.98      0.87       661
           1       0.95      0.56      0.71       430

    accuracy                           0.82      1091
   macro avg       0.86      0.77      0.79      1091
weighted avg       0.85      0.82      0.80      1091
```

Top 20 Product Recommendations:



```
0                              Organic Peach Lowfat Yogurt
1                  Unsweet Strawberry Kiwi Sparkling Water
2                          Cran Raspberry Sparkling Water
3                               Fat Free Blueberry Yogurt
4                                          Coconut Yoghurt
5      Almond Non-Dairy Yogurt Made From Real Almonds...
6                              Organic Whole Grassmilk Milk
7                                     Sparkling Lemon Water
8                              Unsweetened Blackberry Water
9                           Organic Golden Delicious Apple
10                                        Honeycrisp Apples
11     Smooth Greens & Kale Vegetable and Fruit Juice...
12        Slim Can Pink Grapefruit Natural Mineral Water
13                                   Organic Fat Free Milk
14                              2% Reduced Fat Organic Milk
15                         Organic Greek Plain Nonfat Yogurt
16     VitaminWater Zero™ XXX Acai Blueberry Pomegranate
17                        Curate Cherry Lime Sparkling Water
18                                               Skim Milk
19                                    Organic Yellow Peaches
Name: product_name, dtype: object
```

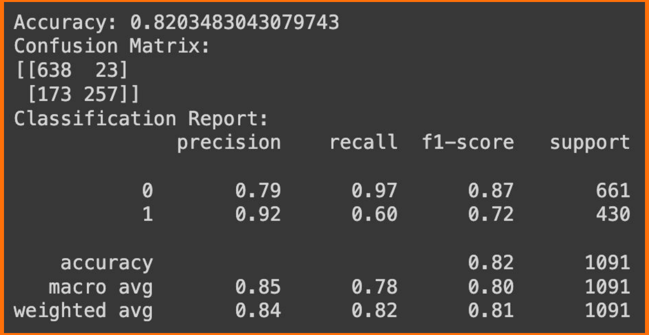
Classification Techniques

Logistic Regression:

Algorithm: Logistic Regression models the probability of a binary outcome using the logistic function. It's a simple and widely used algorithm for binary classification.

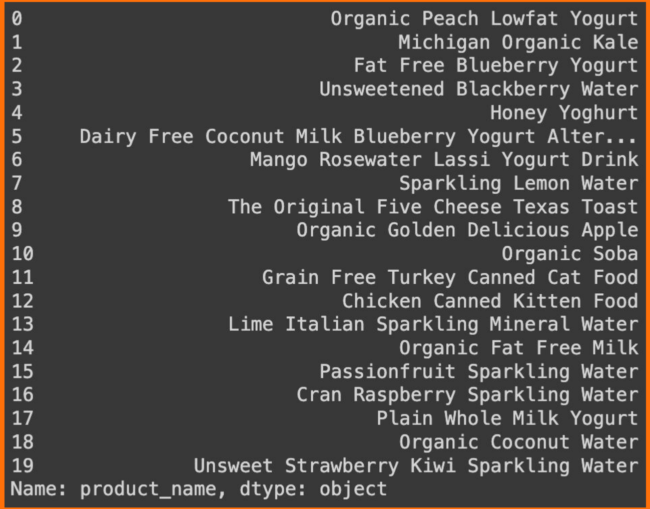
Interpretability: Logistic Regression provides interpretable coefficients that indicate the impact of each feature on the log-odds of the predicted outcome.

Assumption: Assumes a linear relationship between features and the log-odds of the outcome.



```
Accuracy: 0.8203483043079743
Confusion Matrix:
[[638  23]
 [173 257]]
Classification Report:
              precision    recall  f1-score   support

           0       0.79      0.97      0.87       661
           1       0.92      0.60      0.72       430

    accuracy                           0.82      1091
   macro avg       0.85      0.78      0.80      1091
weighted avg       0.84      0.82      0.81      1091
```

Top 20 Product Recommendations:



```
0                               Organic Peach Lowfat Yogurt
1                                     Michigan Organic Kale
2                                  Fat Free Blueberry Yogurt
3                               Unsweetened Blackberry Water
4                                             Honey Yoghurt
5     Dairy Free Coconut Milk Blueberry Yogurt Alter...
6                             Mango Rosewater Lassi Yogurt Drink
7                                       Sparkling Lemon Water
8                          The Original Five Cheese Texas Toast
9                               Organic Golden Delicious Apple
10                                             Organic Soba
11                            Grain Free Turkey Canned Cat Food
12                                  Chicken Canned Kitten Food
13                          Lime Italian Sparkling Mineral Water
14                                      Organic Fat Free Milk
15                                Passionfruit Sparkling Water
16                               Cran Raspberry Sparkling Water
17                                   Plain Whole Milk Yogurt
18                                      Organic Coconut Water
19                     Unsweet Strawberry Kiwi Sparkling Water
Name: product_name, dtype: object
```

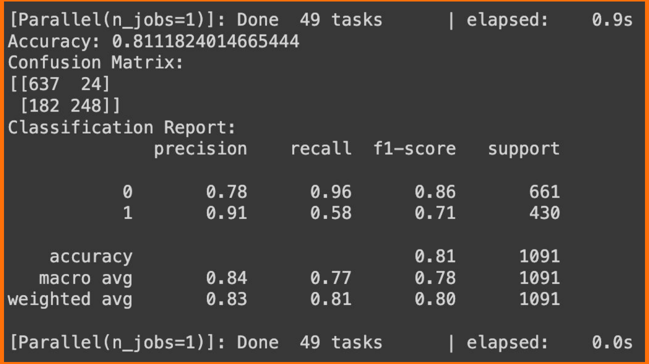
Classification Techniques

Random Forest:

Algorithm: Random Forest is an ensemble learning method that builds multiple decision trees and combines their predictions to improve overall performance.

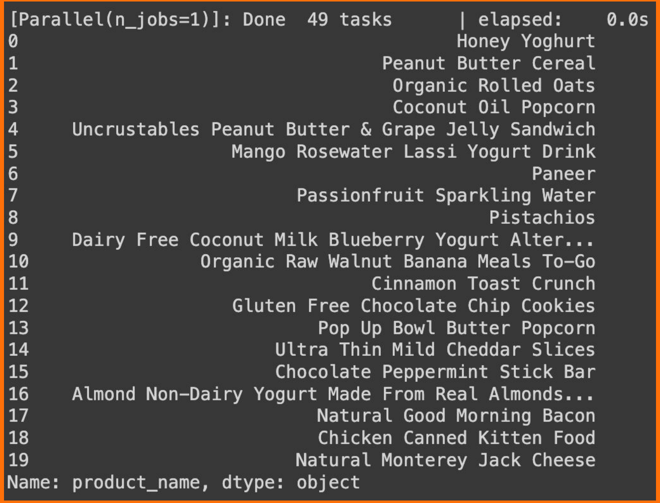
Robustness: Random Forest is robust to overfitting and can handle a large number of features. It also provides feature importance scores.

Non-linearity: Random Forest can capture nonlinear relationships between features and the target variable.



```
[Parallel(n_jobs=1)]: Done  49 tasks      | elapsed:    0.9s
Accuracy: 0.8111824014665444
Confusion Matrix:
[[637  24]
 [182 248]]
Classification Report:
              precision    recall  f1-score   support

           0       0.78      0.96      0.86       661
           1       0.91      0.58      0.71       430

    accuracy                           0.81      1091
   macro avg       0.84      0.77      0.78      1091
weighted avg       0.83      0.81      0.80      1091

[Parallel(n_jobs=1)]: Done  49 tasks      | elapsed:    0.0s
```

Top 20 Product Recommendations:



```
[Parallel(n_jobs=1)]: Done  49 tasks      | elapsed:    0.0s
0                                          Honey Yoghurt
1                                   Peanut Butter Cereal
2                                    Organic Rolled Oats
3                                    Coconut Oil Popcorn
4      Uncrustables Peanut Butter & Grape Jelly Sandwich
5                          Mango Rosewater Lassi Yogurt Drink
6                                                 Paneer
7                               Passionfruit Sparkling Water
8                                             Pistachios
9      Dairy Free Coconut Milk Blueberry Yogurt Alter...
10                  Organic Raw Walnut Banana Meals To-Go
11                                  Cinnamon Toast Crunch
12                          Gluten Free Chocolate Chip Cookies
13                                Pop Up Bowl Butter Popcorn
14                              Ultra Thin Mild Cheddar Slices
15                              Chocolate Peppermint Stick Bar
16     Almond Non-Dairy Yogurt Made From Real Almonds...
17                                Natural Good Morning Bacon
18                                Chicken Canned Kitten Food
19                               Natural Monterey Jack Cheese
Name: product_name, dtype: object
```

Conclusion and Future Work

- After conducting data mining analysis on the Instacart dataset, several key findings have emerged.
- Firstly, the analysis revealed that certain products are frequently purchased together, indicating potential cross-selling opportunities for the company.
- Secondly, customer segmentation based on purchasing behavior can help improve targeted marketing efforts and personalize the shopping experience.
- Lastly, the analysis also highlighted the importance of product availability and pricing in influencing customer purchasing decisions.
- Moving forward, there are several avenues for future research in this field.
- One potential direction is to explore the impact of external factors such as seasonality and promotions on customer behavior and purchasing patterns.
- Additionally, investigating the effectiveness of different marketing strategies and campaigns in driving customer engagement and loyalty would provide valuable insights for Instacart and other e-commerce platforms.
- Furthermore, incorporating sentiment analysis of customer reviews and feedback could help uncover sentiment trends and identify areas for improvement in the shopping experience.

We appreciate your time and attention

Thank You!

Any questions to add?