Market Basket Analysis for Retail Stores

Optimising Retail Strategies with Predictive Analytics and Machine Learning

Nicole Perez

Agenda

Introduction and Problem Statement

Problem Statement

How can we predict the likelihood of customers purchasing specific products based on their past behaviour using large-scale grocery data? The significant challenge in online grocery shopping is recommending products that a user is likely to purchase together. With a vast array of products available on platforms like Instacart, appropriately identifying product combinations can enhance user experience and increase sales.

Impact

This prediction can help Instacart personalize recommendations, increase customer satisfaction, and drive sales growth. Effective product recommendations can drive sales and customer satisfaction. This issue affects customers, businesses, and even supply chain dynamics, as understanding purchasing habits can lead to optimized inventory management and targeted marketing strategies.

Objective

To analyse purchasing patterns, build a predictive model for purchase likelihood, and understand customer behaviour. The primary goal of this project is to develop a machine learning model capable of predicting product co-occurrences, thereby facilitating personalized recommendations for users. This will be achieved through clustering techniques and decision tree implementations.

Data Collection and Preparation

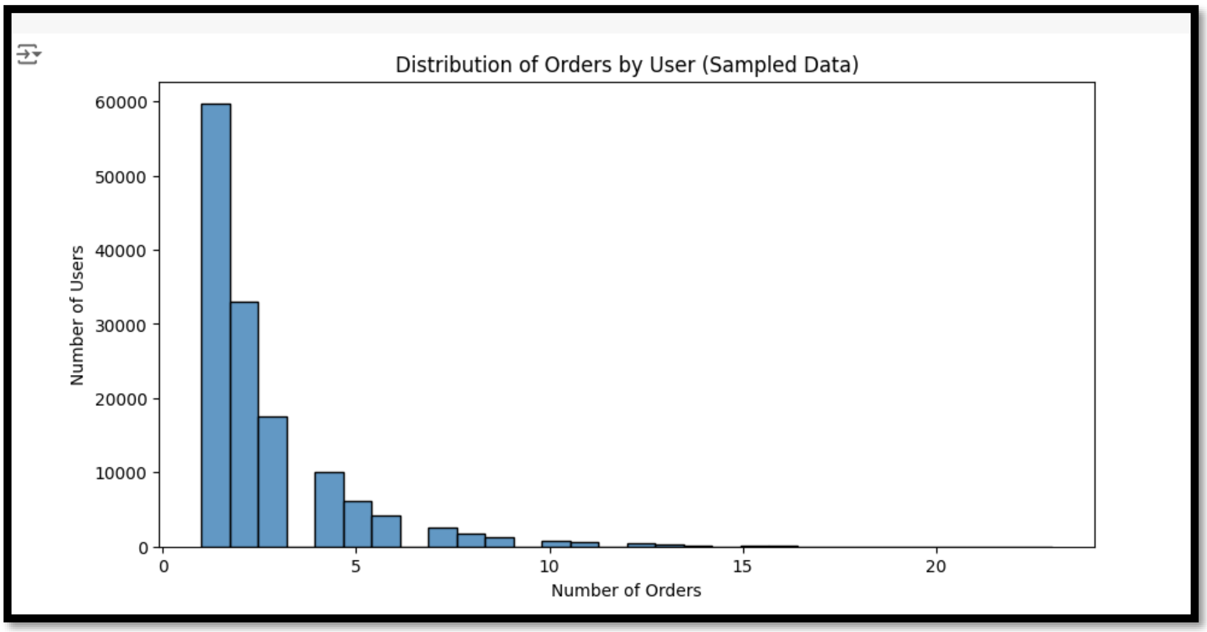
Key Preprocessing Steps

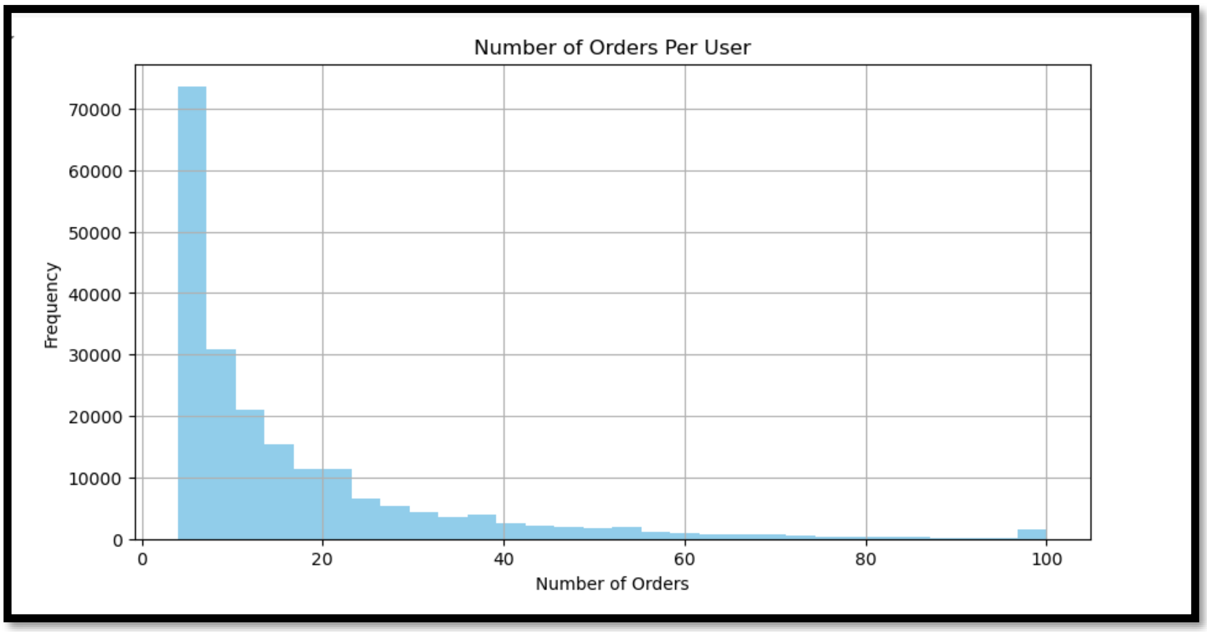
- Data types were optimized for efficient memory usage.
- Missing values were handled during the data merging process to ensure completeness.

Exploratory Data Analysis (EDA)

Orders Per User

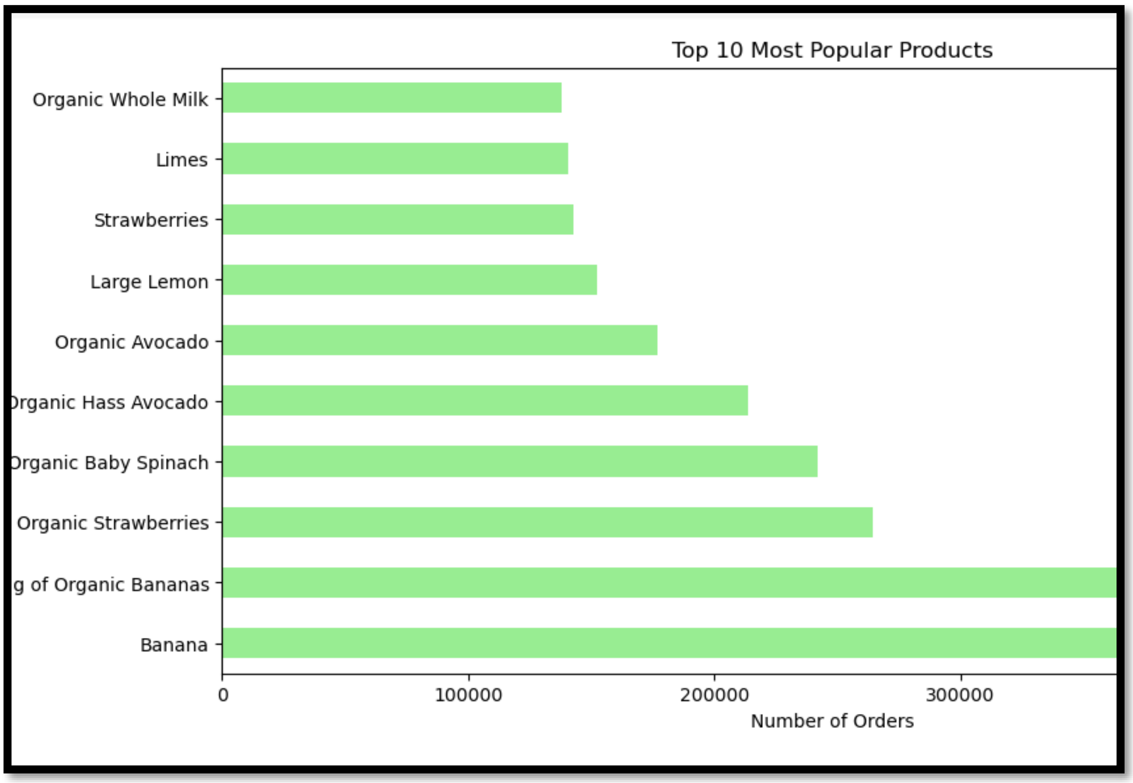
Include a visualization showing the distribution of orders per user.





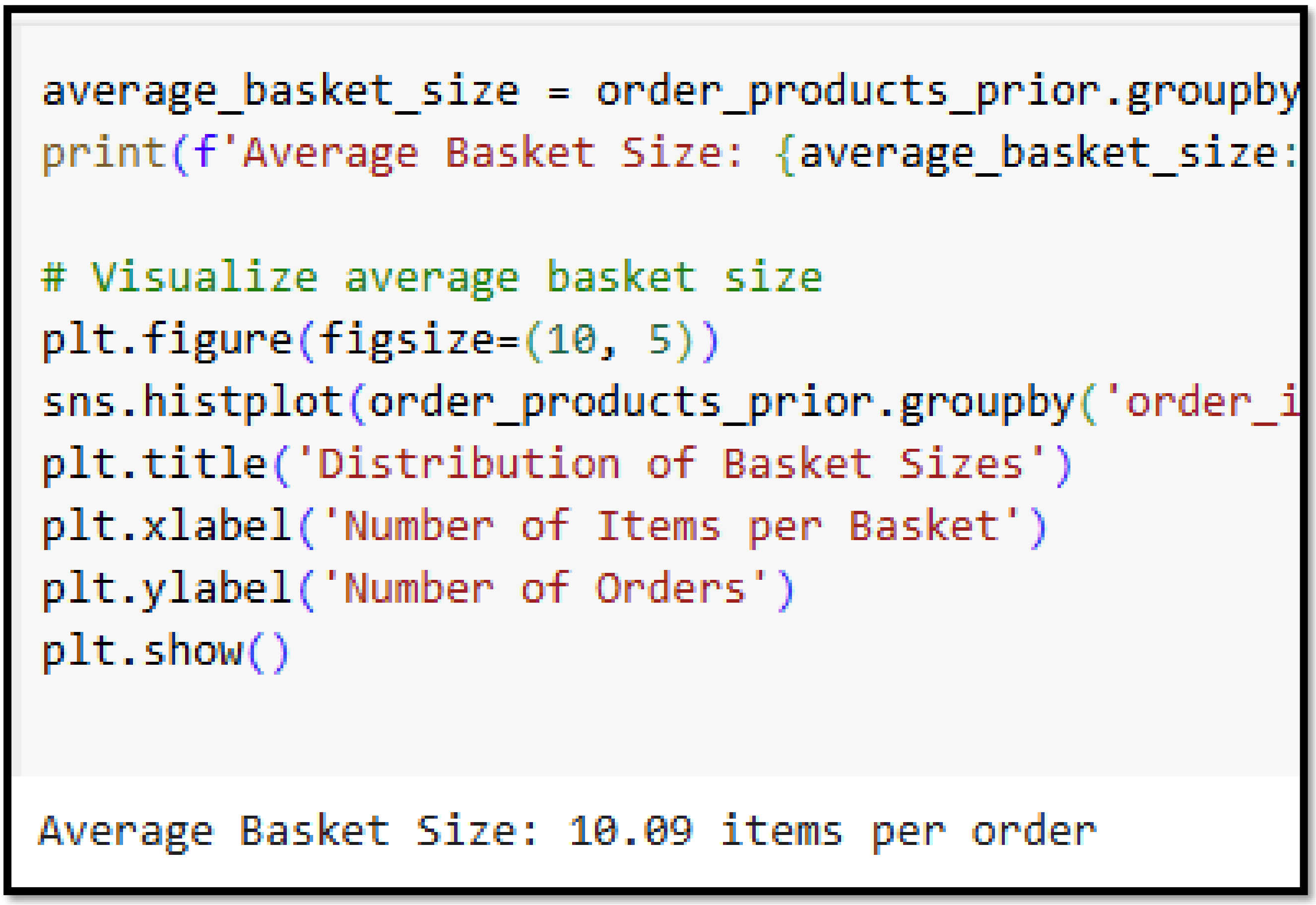
Top 10 Products

Include a bar chart or similar visualization showing the top 10 most frequently purchased products.



Distribution of Basket Size

Include a histogram or similar visualisation showing the distribution of basket sizes.



```
average_basket_size = order_products_prior.groupby
print(f'Average Basket Size: {average_basket_size:

# Visualize average basket size
plt.figure(figsize=(10, 5))
sns.histplot(order_products_prior.groupby('order_i
plt.title('Distribution of Basket Sizes')
plt.xlabel('Number of Items per Basket')
plt.ylabel('Number of Orders')
plt.show()
```

```
Average Basket Size: 10.09 items per order
```

Modelling Approach

Possible Models

- **Association Rule Learning:** The Apriori algorithm finds interesting relationships and patterns in buying behaviour by identifying sets of items frequently purchased together.
- **Clustering (K-Means):** K-Means groups users based on their purchase behaviour, allowing for segmentation of customers. This can help tailor marketing efforts or inventory management.
- **Classification (Decision Trees):** Decision Trees utilize past purchasing behaviour to predict whether a customer will purchase a particular product.

Model Selection

K-Means was selected for clustering due to its simplicity and efficiency in handling large datasets, while the Decision Tree was chosen for its interpretability and effectiveness in classification tasks.

Pros and Cons

- **K-Means:**
  - Pros: Simple and fast.
  - Cons: Choosing the right number of clusters can be subjective.
- **Decision Trees:**
  - Pros: Easy to interpret and visualize.
  - Cons: Prone to overfitting without tuning.

Error Analysis

Challenge

Our analysis relies on the Instacart Grocery Dataset, which can be massive in scale.

Limitation

Due to RAM constraints (probing local environment with 8GB), full in-memory processing of the dataset is unfeasible.

Explaining the RAM Limitation

Use visuals or graphics to illustrate how much memory is consumed with example data sizes. For large datasets, the data structures can exceed available RAM, leading to performance degradation or failure to run the model.

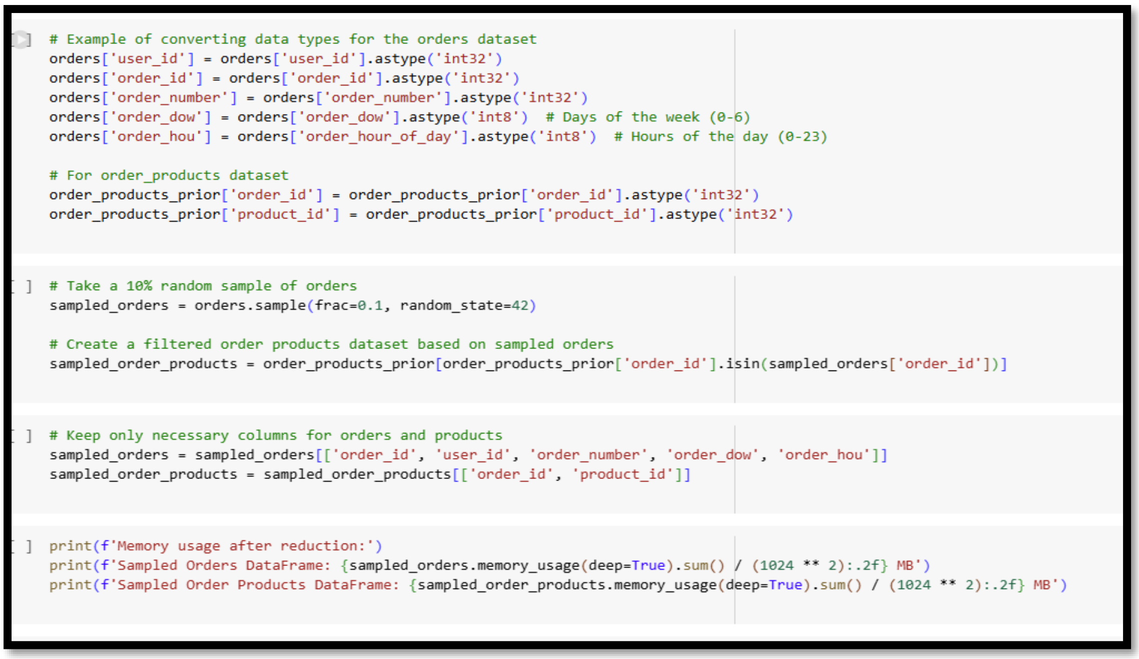
Error Summary

Due to the excessive size of the Instacart dataset, the model encountered significant memory issues, leading to incomplete data loading and processing failures. This resulted in the inability to run the model effectively, as the available RAM could not handle the dataset's volume.

Visualize Examples

Since the model could not be fully executed, typical errors such as misclassifications or incorrect predictions could not be visualized. However, common issues in such scenarios often include:

- Data Loading Failures: Incomplete data leading to missing features.
- Memory Overflows: Crashes during data preprocessing or model training.

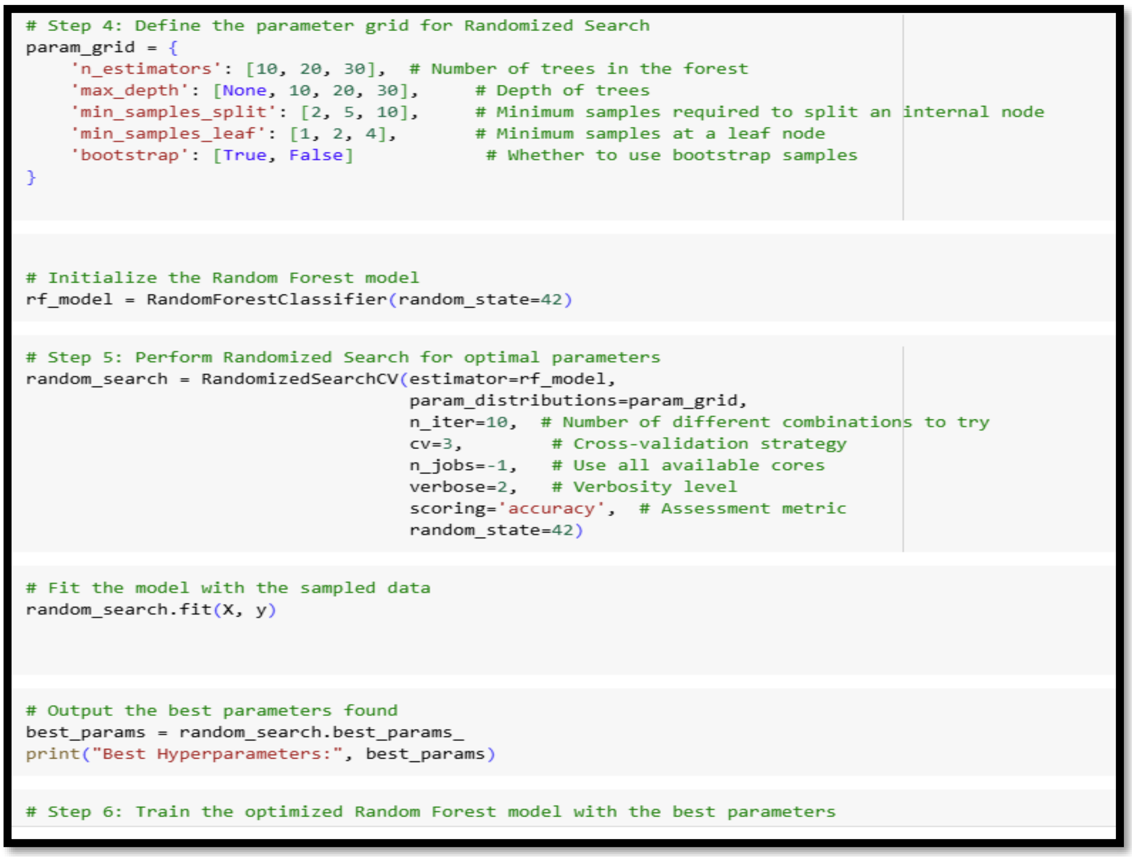


```python
# Example of converting data types for the orders dataset
orders['user_id'] = orders['user_id'].astype('int32')
orders['order_id'] = orders['order_id'].astype('int32')
orders['order_number'] = orders['order_number'].astype('int32')
orders['order_dow'] = orders['order_dow'].astype('int8')  # Days of the week (0-6)
orders['order_hou'] = orders['order_hour_of_day'].astype('int8')  # Hours of the day (0-23)

# For order_products dataset
order_products_prior['order_id'] = order_products_prior['order_id'].astype('int32')
order_products_prior['product_id'] = order_products_prior['product_id'].astype('int32')
```

```python
# Take a 10% random sample of orders
sampled_orders = orders.sample(frac=0.1, random_state=42)

# Create a filtered order products dataset based on sampled orders
sampled_order_products = order_products_prior[order_products_prior['order_id'].isin(sampled_orders['order_id'])]
```

```python
# Keep only necessary columns for orders and products
sampled_orders = sampled_orders[['order_id', 'user_id', 'order_number', 'order_dow', 'order_hou']]
sampled_order_products = sampled_order_products[['order_id', 'product_id']]
```

```python
print(f'Memory usage after reduction:')
print(f'Sampled Orders DataFrame: {sampled_orders.memory_usage(deep=True).sum() / (1024 ** 2):.2f} MB')
print(f'Sampled Order Products DataFrame: {sampled_order_products.memory_usage(deep=True).sum() / (1024 ** 2):.2f} MB')
```



```python
# Step 4: Define the parameter grid for Randomized Search
param_grid = {
    'n_estimators': [10, 20, 30],  # Number of trees in the forest
    'max_depth': [None, 10, 20, 30],      # Depth of trees
    'min_samples_split': [2, 5, 10],      # Minimum samples required to split an internal node
    'min_samples_leaf': [1, 2, 4],        # Minimum samples at a leaf node
    'bootstrap': [True, False]             # Whether to use bootstrap samples
}
```

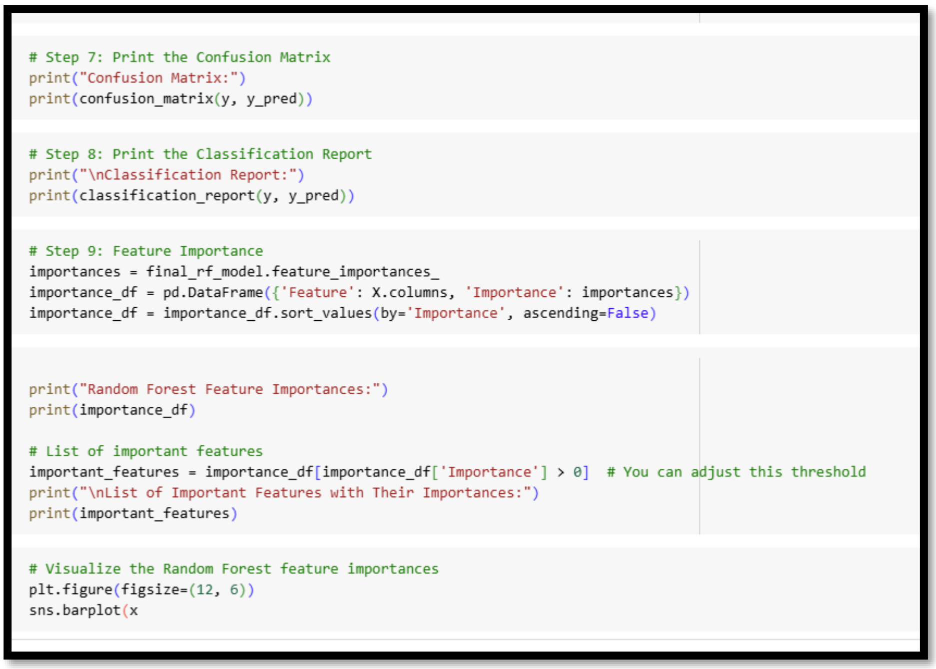
```python
# Initialize the Random Forest model
rf_model = RandomForestClassifier(random_state=42)
```

```python
# Step 5: Perform Randomized Search for optimal parameters
random_search = RandomizedSearchCV(estimator=rf_model,
                                   param_distributions=param_grid,
                                   n_iter=10,  # Number of different combinations to try
                                   cv=3,        # Cross-validation strategy
                                   n_jobs=-1,   # Use all available cores
                                   verbose=2,   # Verbosity level
                                   scoring='accuracy',  # Assessment metric
                                   random_state=42)
```

```python
# Fit the model with the sampled data
random_search.fit(X, y)
```

```python
# Output the best parameters found
best_params = random_search.best_params_
print("Best Hyperparameters:", best_params)
```

```python
# Step 6: Train the optimized Random Forest model with the best parameters
```



```
# Step 7: Print the Confusion Matrix
print("Confusion Matrix:")
print(confusion_matrix(y, y_pred))

# Step 8: Print the Classification Report
print("\nClassification Report:")
print(classification_report(y, y_pred))

# Step 9: Feature Importance
importances = final_rf_model.feature_importances_
importance_df = pd.DataFrame({'Feature': X.columns, 'Importance': importances})
importance_df = importance_df.sort_values(by='Importance', ascending=False)


print("Random Forest Feature Importances:")
print(importance_df)

# List of important features
important_features = importance_df[importance_df['Importance'] > 0]  # You can adjust this threshold
print("\nList of Important Features with Their Importances:")
print(important_features)

# Visualize the Random Forest feature importances
plt.figure(figsize=(12, 6))
sns.barplot(x
```

Explainability and Interpretability

Interpretability Techniques

Given the model did not run successfully, interpretability techniques like Apriori Algorithm, Clustering through K-Means, Decision Trees, and logistic regression could not be applied. Typically, these techniques would help in understanding feature importance and model decision-making processes.

Examples

Without successful model execution, specific examples of correct and incorrect predictions are unavailable. Normally, these examples would illustrate how the model interprets various features and makes decisions.

Results and Impact

Summary Table

As the model could not be executed, a summary table of performance metrics is not available. Ideally, this table would include metrics such as accuracy, precision, recall, and F1-score.

Impact Explanation

The inability to run the model due to memory constraints highlights the importance of efficient data handling and preprocessing. Addressing these issues is crucial for achieving the project's goals of improving retail decision-making and customer insights.

Deployment and Practical Applications

Deployment Potential

Given the current challenges with data loading and memory constraints, deploying the model in its current state is not feasible. Future deployment would require:

- **Optimized Data Handling:** Efficient data preprocessing to reduce memory usage.
- **Scalable Infrastructure:** Utilizing cloud-based solutions or distributed computing to handle large datasets.

Future Work and Improvements

Improvements

To overcome the current challenges, the following improvements are suggested:

- **Data Sampling:** Use a representative subset of the dataset to reduce memory load.
- **Incremental Learning:** Implement techniques that allow the model to learn from data in smaller batches.
- **Enhanced Infrastructure:** Leverage cloud computing resources to handle large-scale data processing.
- **Alternative Models:** Explore models that are more memory-efficient or designed for large datasets.

References:

- https://www.kaggle.com/c/instacart-market-basket-analysis
- https://www.instacart.com/
- https://www.ibm.com/topics/apriori-algorithm#:~:text=The%20Apriori%20algorithm%20is%20an,items%20called%20itemsets%20in%20data.
- https://colab.google/
- https://docs.anaconda.com/anaconda/pkg-docs/