**Price Prediction Model Explanation**

**Kernel 1: Importing Necessary Libraries**

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.model\_selection import train\_test\_split, GridSearchCV

from sklearn.metrics import mean\_squared\_error, mean\_absolute\_error

from sklearn.model\_selection import cross\_val\_score

from surprise import Dataset, Reader, SVD

from surprise.model\_selection import cross\_validate

from zipfile import ZipFile

from xgboost import XGBRegressor

import os

**Explanation**:

* This kernel imports a series of essential Python libraries needed for various tasks in the notebook:

1. **pandas** and **numpy**: For data handling and numerical computations.
2. **matplotlib.pyplot** and **seaborn**: For data visualization and plotting trends, distributions, and correlations.
3. **scikit-learn**: For splitting datasets, building models, tuning them with GridSearchCV, and calculating model evaluation metrics such as MSE (Mean Squared Error) and MAE (Mean Absolute Error).
4. **surprise**: A library for collaborative filtering used in recommendation systems (though not applied directly here).
5. **xgboost**: A machine learning library used for gradient boosting algorithms.
6. **zipfile**: For handling compressed files.
7. **os**: Used for interacting with the operating system (e.g., navigating file directories).

**Kernel 2: Loading and Merging Datasets**

# Load the dataset

orders = pd.read\_csv('/content/orders.csv')

order\_products\_prior = pd.read\_csv('/content/order\_products\_\_prior.csv')

products = pd.read\_csv('/content/products.csv')

aisles = pd.read\_csv('/content/aisles.csv')

departments = pd.read\_csv('/content/departments.csv')

# Merge product details into the orders

products = pd.merge(products, aisles, on='aisle\_id', how='left')

products = pd.merge(products, departments, on='department\_id', how='left')

order\_products = pd.merge(order\_products\_prior, orders, on='order\_id', how='left')

order\_products = pd.merge(order\_products, products, on='product\_id', how='left')

# Handle missing values

order\_products.fillna(0, inplace=True)

**Explanation**:

* **Loading Data**: The kernel loads several CSV files, each containing essential information for the grocery dataset:

1. orders.csv: Details of orders made by users.
2. order\_products\_prior.csv: The list of products from prior orders.
3. products.csv: Product information.
4. aisles.csv and departments.csv: Metadata that categorizes products into aisles and departments.

* **Merging Data**:

1. The kernel merges multiple datasets into a single comprehensive DataFrame called order\_products. This includes merging product details such as aisle and department to provide richer product information.

* **Handling Missing Values**:

1. Missing values in the order\_products DataFrame are filled with 0 to avoid errors in subsequent processing steps.

* **Output**:

1. The output of this kernel is a comprehensive dataset (order\_products) that combines information on orders, products, aisles, and departments. This DataFrame is ready for further processing and analysis, with missing values handled appropriately.

**Kernel 3: Feature Engineering**

# Feature Engineering: Create new features

order\_products['days\_since\_prior\_order'].fillna(0, inplace=True)

order\_products['user\_total\_orders'] = order\_products.groupby('user\_id')['order\_number'].transform('max')

order\_products['user\_avg\_days\_since\_prior'] = order\_products.groupby('user\_id')['days\_since\_prior\_order'].transform('mean')

order\_products['user\_avg\_order\_size'] = order\_products.groupby('user\_id')['add\_to\_cart\_order'].transform('mean')

# Reorder rate per product

product\_reorder\_rate = order\_products.groupby('product\_id')['reordered'].mean().reset\_index()

product\_reorder\_rate.columns = ['product\_id', 'product\_reorder\_rate']

order\_products = pd.merge(order\_products, product\_reorder\_rate, on='product\_id', how='left')

* **Explanation**:
* **Feature Engineering**:

1. **days\_since\_prior\_order**: Missing values are filled with 0.
2. **user\_total\_orders**: For each user, the total number of orders is computed.
3. **user\_avg\_days\_since\_prior**: Calculates the average number of days between orders for each user.
4. **user\_avg\_order\_size**: The average size of each order (based on how many items were added to the cart) is computed.

* **Reorder Rate**:

1. A new feature, product\_reorder\_rate, is calculated. This feature captures how often each product is reordered by averaging the reordered values for each product.
2. The product\_reorder\_rate is then merged into the main order\_products DataFrame to give additional information about each product.

* **Output**:

1. Several new features related to user behaviour (e.g., total orders, average order size) and product behaviour (e.g., reorder rate) have been added to the dataset. These features are important for improving the model's ability to predict reorder probabilities.

**Kernel 4: Decision Tree Model Training**

from sklearn.tree import DecisionTreeRegressor

from sklearn.metrics import mean\_squared\_error, mean\_absolute\_error

# Initialize the Decision Tree model

features = ['order\_dow', 'order\_hour\_of\_day', 'days\_since\_prior\_order',

'user\_total\_orders', 'user\_avg\_days\_since\_prior',

'user\_avg\_order\_size', 'product\_reorder\_rate']

X = order\_products[features]

y = order\_products['reordered']

# Split the data

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

dt\_model = DecisionTreeRegressor(max\_depth=6, random\_state=42)

# Train the model

dt\_model.fit(X\_train, y\_train)

# Make predictions

y\_pred = dt\_model.predict(X\_test)

# Evaluate the model

mse = mean\_squared\_error(y\_test, y\_pred)

mae = mean\_absolute\_error(y\_test, y\_pred)

print(f'MSE: {mse}')

print(f'MAE: {mae}')

* **Explanation**:

1. **Feature Selection**: Selected features like order\_dow (day of the week the order was placed), product\_reorder\_rate, and user statistics are used as predictors (X), and reordered is the target variable (y).
2. **Splitting Data**: The dataset is split into training (80%) and testing (20%) sets using train\_test\_split to evaluate model performance on unseen data.

* **Training and Predictions**:

1. A **Decision Tree Regressor** model is initialized with a maximum depth of 6 to prevent overfitting. It is trained using the training data (X\_train and y\_train).
2. Predictions (y\_pred) are made on the test data (X\_test).

* **Model Evaluation**:

1. The model's performance is evaluated using **Mean Squared Error (MSE)** and **Mean Absolute Error (MAE)**. These metrics measure the error between predicted and actual reorder probabilities.

**Output**:

MSE: 0.18893694275234443

MAE: 0.37791376293099094

* **MSE**: 0.1889 indicates that the average squared error between the predicted and actual reorder probabilities is approximately 0.189.
* **MAE**: 0.3779 shows the average magnitude of the errors in the model's predictions, with an average error of 0.378. These metrics suggest that while the model is somewhat accurate, there may be room for improvement.

**Kernel 5: Predictions and Evaluation**

for i in range(10): # Print the first 10 predictions for example

print(f'Predicted Price: {predicted\_prices[i]}, Actual Price: {actual\_prices[i]}')

**Explanation:**

1. In this kernel, after training the **Decision Tree Regressor**, we are making predictions of product prices using the test set (X\_test).
2. The **Predicted Price** represents the product price predicted by the trained model based on the features from the test data.
3. The **Actual Price** represents the true price of the product, as given in the test set (y\_test).

**Interpretation**:

1. **Predicted Price: 0.1041, Actual Price: 0**: The model predicted a price of **0.1041**, while the actual price is **0**. This shows that the model is predicting a low price, but there’s still some error compared to the actual price of 0.
2. **Predicted Price: 0.8539, Actual Price: 1**: The model predicted a price of **0.8539**, which is close to the actual price of **1**. This indicates the model is performing well in this case.
3. **Predicted Price: 0.7656, Actual Price: 1**: The model predicted a price of **0.7656**, which is slightly lower than the actual price of **1**.

**Kernel 6: Saving the Model**

import joblib

# Save the model to a file

joblib.dump(dt\_model, 'price\_model.pkl')

**Explanation**:

* **joblib.dump**: This method is used to save the trained Decision Tree model (dt\_model) to a file called price\_model.pkl.
* **Why Save the Model**: Saving the model allows it to be reused in the future without needing to retrain it. This is essential in real-world applications where models need to make predictions in real-time or be deployed in production environments.

**Output**:

['price\_model.pkl']

* This output confirms that the model has been successfully saved as a file named **price\_model.pkl**.

**Kernel 7: Saving the Model as 'price\_recommendation\_model.pkl'**

import joblib

# Save the model to a file

joblib.dump(dt\_model, 'price\_recommendation\_model.pkl')

**Explanation**:

* This kernel is similar to the previous one, but it saves the trained model under a different filename: **price\_recommendation\_model.pkl**. This could be useful if the same model is intended to be used for a specific purpose, such as price recommendations in addition to predictions.

**Output**:

['price\_recommendation\_model.pkl']

* This output confirms that the model has been saved under a different name, **price\_recommendation\_model.pkl**.