Wal-Mart: Demand Forecasting and Inventory Management

Problem Statement:

Wal-Mart needed to improve its demand forecasting to manage its large-scale inventory more efficiently. Poor forecasting led to stock outs, overstock, and suboptimal inventory management, which affected the bottom line and customer satisfaction.

Step 1:Setting the research goal

The research goal for this project is to develop an effective demand forecasting and inventory management system for Walmart using machine learning techniques. By analyzing historical sales data, the objective is to improve the accuracy of demand predictions, thereby optimizing stock levels and minimizing both stock-outs and overstock. The project begins by retrieving and preparing the data, ensuring categorical variables are properly encoded for model training. Data exploration aims to uncover trends and relationships between factors such as user demographics, product categories, and purchasing behavior. A Random Forest regression model is chosen to forecast demand based on these factors. Finally, the results are evaluated and presented, with the goal of refining the inventory management process and enhancing customer satisfaction.

Step 2:Data Retrieval

Data retrieval refers to the process of accessing and loading data from a specified source, such as a database, file, or external system, to use in analysis or modeling. In this project, data retrieval involves importing the Walmart sales dataset into a Python environment using libraries like pandas,matplotlib etc.

Importing libraries and dataset

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
df = pd.read_csv('walmart.csv')
```

```
df.head()
```

	User_ID	Product_ID	Gender	Age	Occupation	City_Category	Stay_In_Current_City_
0	1000001	P00069042	F	0-17	10	A	
1	1000001	P00248942	F	0-17	10	A	
2	1000001	P00087842	F	0-17	10	A	

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 550068 entries, 0 to 550067
Data columns (total 10 columns):
 #   Column                      Non-Null Count   Dtype
---  ------                      --------------   -----
 0   User_ID                     550068 non-null  int64
 1   Product_ID                  550068 non-null  int64
 2   Gender                      550068 non-null  int64
 3   Age                         550068 non-null  int64
 4   Occupation                  550068 non-null  int64
 5   City_Category               550068 non-null  int64
 6   Stay_In_Current_City_Years  550068 non-null  int64
 7   Marital_Status              550068 non-null  int64
 8   Product_Category            550068 non-null  int64
 9   Purchase                    550068 non-null  int64
dtypes: int64(10)
memory usage: 42.0 MB
```

Summary Statistics:

describe() provides statistical summaries such as mean, standard deviation, and percentiles for numerical features. This helps in understanding the distribution and range of values.

```
df.describe()
```

	User_ID	Product_ID	Gender	Age	Occupation	City_
count	550068.000000	550068.000000	550068.000000	550068.000000	550068.000000	5500
mean	2948.888392	1707.473323	0.753105	2.496430	8.076707	
std	1685.407072	1012.201109	0.431205	1.353632	6.522660	
min	0.000000	0.000000	0.000000	0.000000	0.000000	
25%	1471.000000	930.000000	1.000000	2.000000	2.000000	
50%	2995.000000	1666.000000	1.000000	2.000000	7.000000	
75%	4365.000000	2550.000000	1.000000	3.000000	14.000000	
max	5890.000000	3630.000000	1.000000	6.000000	20.000000	

```
df.dtypes
```

```
User_ID                         int64
Product_ID                     object
Gender                         object
Age                            object
Occupation                      int64
City_Category                  object
Stay_In_Current_City_Years     object
Marital_Status                  int64
Product_Category                int64
Purchase                        int64
dtype: object
```

Step 3:Data preparation

Check for missing values

Data preparation includes handling missing values by filling or removing them.The isnull().sum() method checks for any missing values in the dataset. Missing values can skew the results, so it's crucial to address them. In this case, rows with missing values are dropped using dropna(), though other imputation techniques could also be used.

```
df.isnull().sum()
```

```
User_ID                        0
Product_ID                     0
Gender                         0
Age                            0
Occupation                     0
City_Category                  0
```

```
     Stay_In_Current_City_Years    0
     Marital_Status                0
     Product_Category              0
     Purchase                      0
     dtype: int64
```

Convert categorial columns

Many machine learning algorithms require numerical input.Encoding categorical variables such as Gender, Age, Product_ID, and City_Category into numeric formats using label encoding.

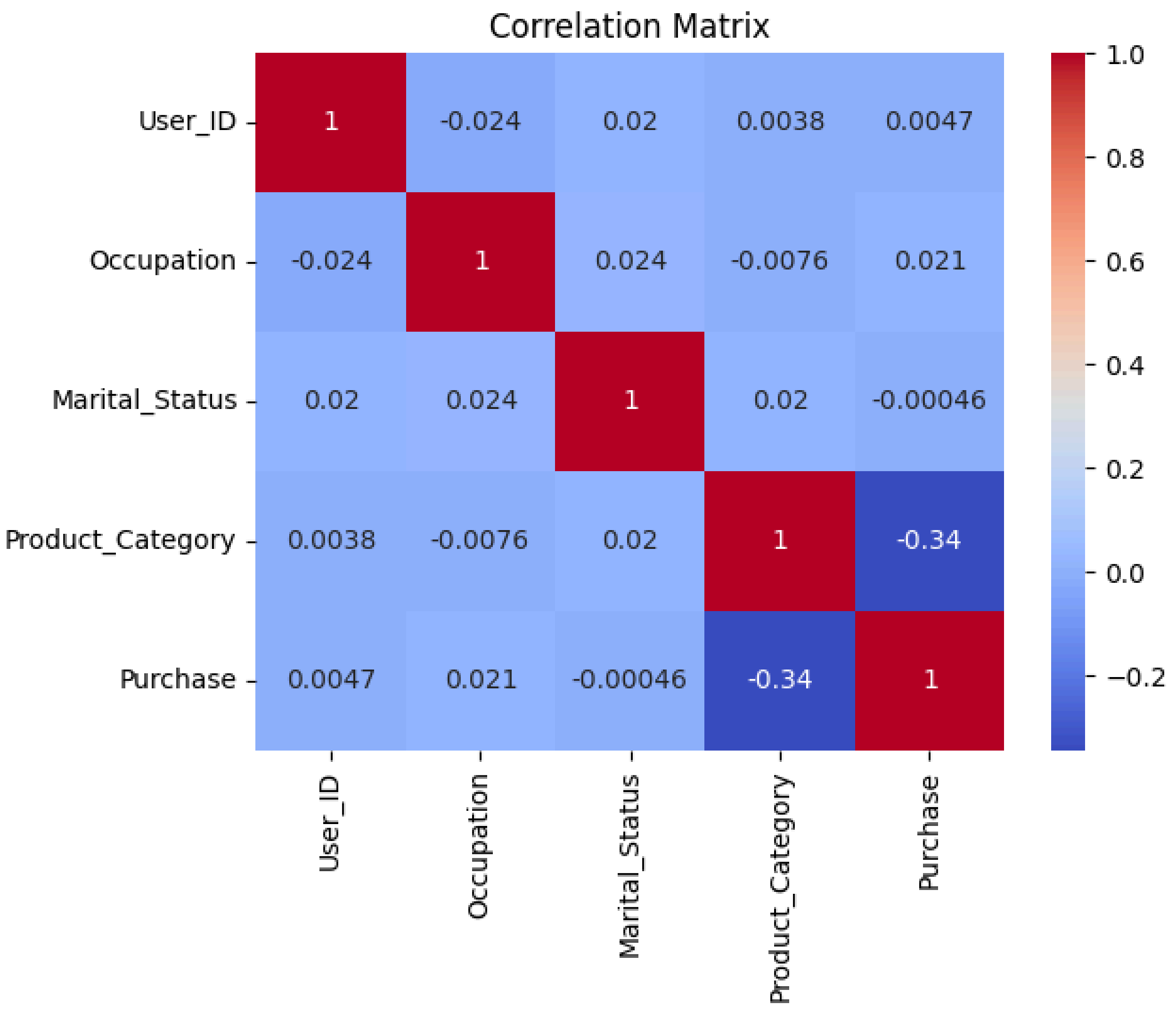
```
from sklearn.preprocessing import LabelEncoder
# Encode categorical columns (Product_ID, User_ID, etc.)
label_encoder = LabelEncoder()


# Apply label encoding to non-numeric columns
df['Product_ID'] = label_encoder.fit_transform(df['Product_ID'])
df['User_ID'] = label_encoder.fit_transform(df['User_ID'])
df['Stay_In_Current_City_Years'] = label_encoder.fit_transform(df['Stay_In_Current_City_Y


# Encoding the rest (Gender, Age, City_Category) as before
df['Gender'] = label_encoder.fit_transform(df['Gender'])
df['Age'] = label_encoder.fit_transform(df['Age'])
df['City_Category'] = label_encoder.fit_transform(df['City_Category'])
```

Step 4:Data Exploration

Exploring the dataset to find relationships between the features and the target variable.

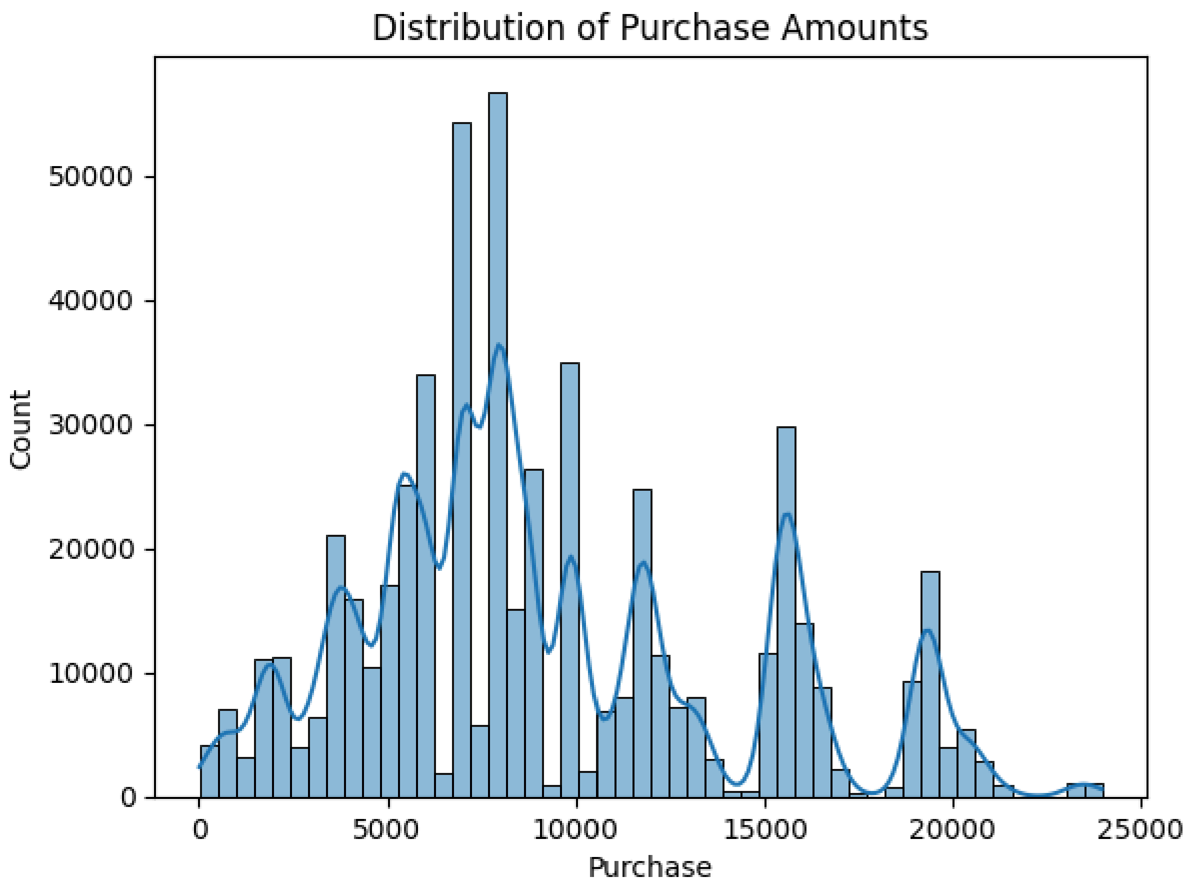
Checking Correlations

```
 numeric_columns = df.select_dtypes(include=['int64', 'float64']).columns
# Correlation matrix for numeric columns
sns.heatmap(df[numeric_columns].corr(), annot=True, cmap='coolwarm')
plt.title('Correlation Matrix')
plt.show()
```

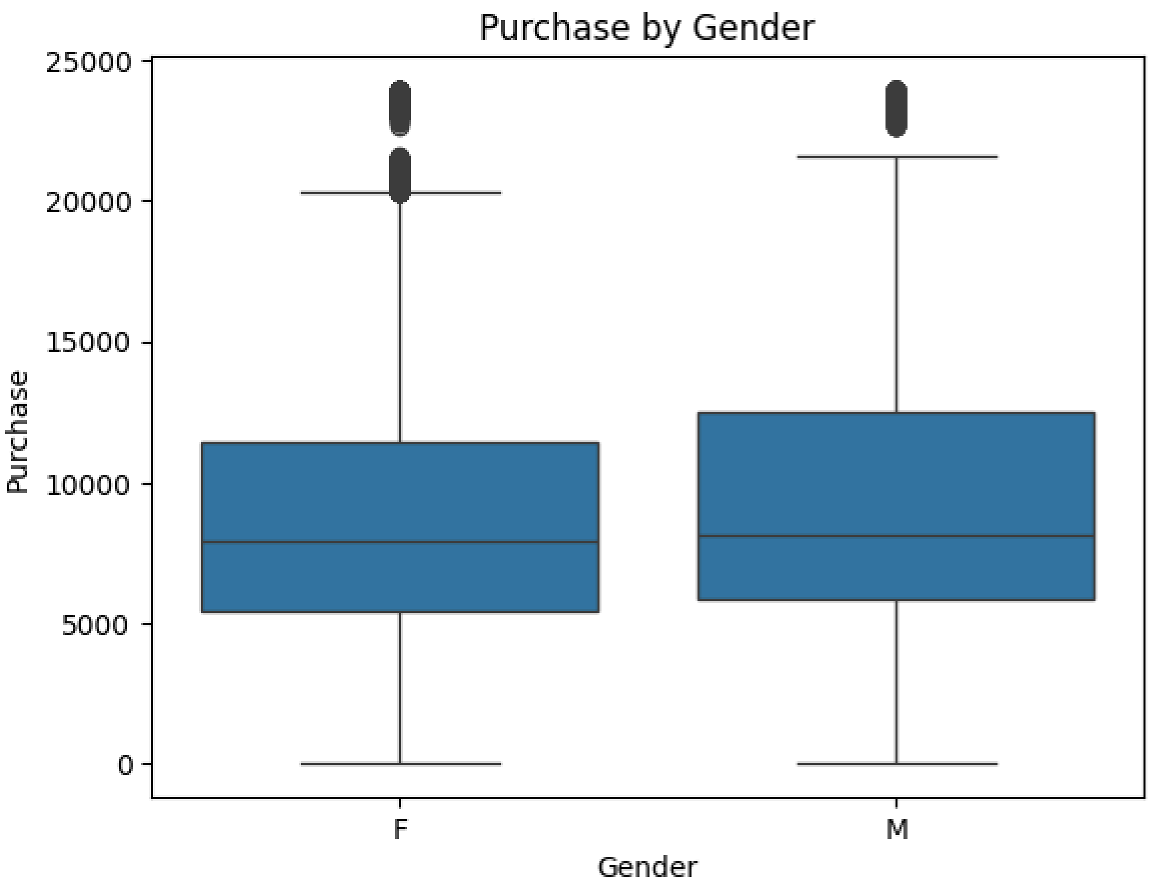


	User_ID	Occupation	Marital_Status	Product_Category	Purchase
User_ID	1	-0.024	0.02	0.0038	0.0047
Occupation	-0.024	1	0.024	-0.0076	0.021
Marital_Status	0.02	0.024	1	0.02	-0.00046
Product_Category	0.0038	-0.0076	0.02	1	-0.34
Purchase	0.0047	0.021	-0.00046	-0.34	1

```
# Distribution of purchase values plt.figure(figsize=(8, 6))
sns.histplot(df['Purchase'], bins=50, kde=True)
plt.title('Distribution of Purchase Amounts')
plt.show()
```



```
# Boxplot for Purchase by Gender
sns.boxplot(x='Gender', y='Purchase', data=df)
plt.title('Purchase by Gender')
plt.show()
```



Grouping data

Grouping data by Product_Category to find average purchases involves aggregating the data to compute the mean purchase amount for each category of products. This technique helps in summarizing the dataset by focusing on specific categories of interest.

```
# Grouping data by Product_Category to find average purchases
avg_purchase_by_category = df.groupby('Product_Category')['Purchase'].mean()
print(avg_purchase_by_category)
```

```
Product_Category
1     13606.218596
2     11251.935384
3     10096.705734
4      2329.659491
5      6240.088178
6     15838.478550
7     16365.689600
8      7498.958078
9     15537.375610
10    19675.570927
11     4685.268456
12     1350.859894
13      722.400613
14    13141.625739
15    14780.451828
16    14766.037037
```

```
17    10170.759516
18     2972.864320
19       37.041797
20      370.481176
Name: Purchase, dtype: float64
```

Step 5:Data Modeling

Using a basic machine learning model for demand forecasting. We will use Random Forest for demonstration.

```
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error
```

Define features and target variable

In a machine learning task, two critical components must be defined before training a model: features (X) and the target variable (y).

```
X = df[['User_ID', 'Product_ID', 'Gender', 'Age', 'Occupation', 'City_Category', 'Stay_In
y = df['Purchase']
```

Split the data into training and testing sets

```
# Train-test split
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

Initializing the model

The RandomForestRegressor is initialized with n_estimators=100, meaning the model will construct 100 decision trees as part of the forest.random_state=42 is a seed used to ensure reproducibility of the results. This ensures that every time the model is run with the same data, the output will be consistent.

```
model = RandomForestRegressor(n_estimators=100, random_state=42)
```

Why Random forest

Using a Random Forest model for Walmart's demand forecasting and inventory management problem is a strategic choice for several key reasons:

1)Ability to Handle High-Dimensional Data:Random Forest can effectively handle datasets with many features, including a mix of categorical and continuous variables like User_ID, Product_ID, Gender, and Purchase.

2)Handling Non-Linear Relationships:In retail demand forecasting, the relationship between features (e.g., customer demographics, product categories) and the target variable (purchase amount) is often non-linear.

3)Resilience to Overfitting:Since the Random Forest model creates an ensemble of multiple decision trees, each trained on a random subset of data, it reduces the risk of overfitting. In large and noisy datasets like Walmart's, this is important for creating a model that generalizes well to unseen data.

4)Scalability and Efficiency:Random Forests can scale well with large datasets, making them suitable for Walmart's extensive sales data. While they may be more computationally expensive than simpler models, they provide higher accuracy, which is critical for inventory and demand forecasting.

Fit the model on the training data

The fit() method trains the model using the training data (X_train and y_train). Here, the Random Forest builds decision trees by splitting the data multiple times based on feature values and learns patterns in the data.

```
model.fit(X_train, y_train)
```

```
▾        RandomForestRegressor
RandomForestRegressor(random_state=42)
```

Prediction of test set result

After training, the model uses the predict() function to make predictions on the unseen test data (X_test). The output (y_pred) contains the model's predicted values for each sample in the test set.

```
y_pred = model.predict(X_test)
```

Evaluate the model

The mean_squared_error() function calculates how well the model's predictions (y_pred) match the actual target values (y_test). MSE measures the average squared difference between actual and predicted values.

```
# Evaluate the model
mse = mean_squared_error(y_test, y_pred)
print(f"Mean Squared Error: {mse}")
```

```
Mean Squared Error: 7629657.261921952
```

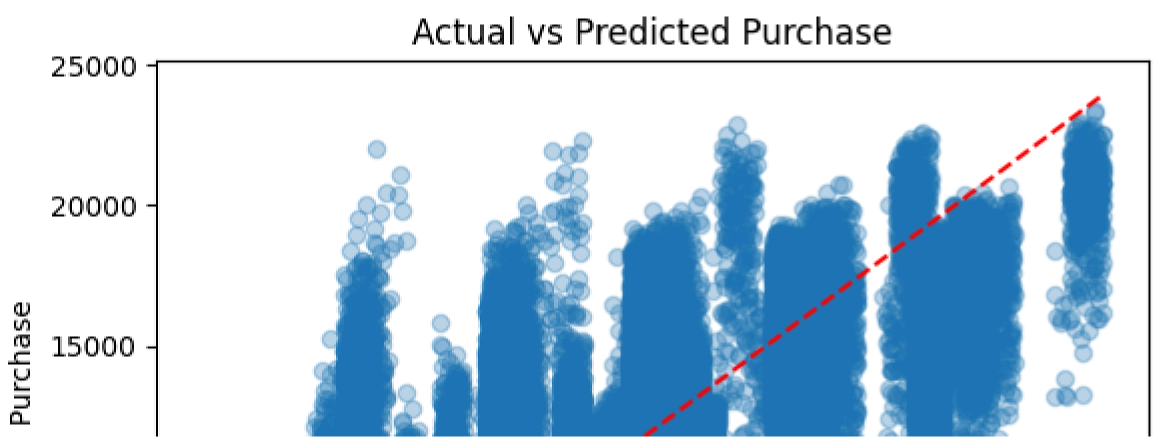
Step 6:Data Presentation and Automation

Presenting the model's results

Scatter plot of actual vs predicted values

he scatter plot of actual vs. predicted values is a useful visualization for evaluating the performance of a regression model. In this plot, the actual purchase amounts (y_test) are plotted on the x-axis, and the predicted purchase amounts (y_pred) are plotted on the y-axis. The plot also includes a red dashed line (y = x), which represents the ideal scenario where the predicted values match the actual values perfectly.

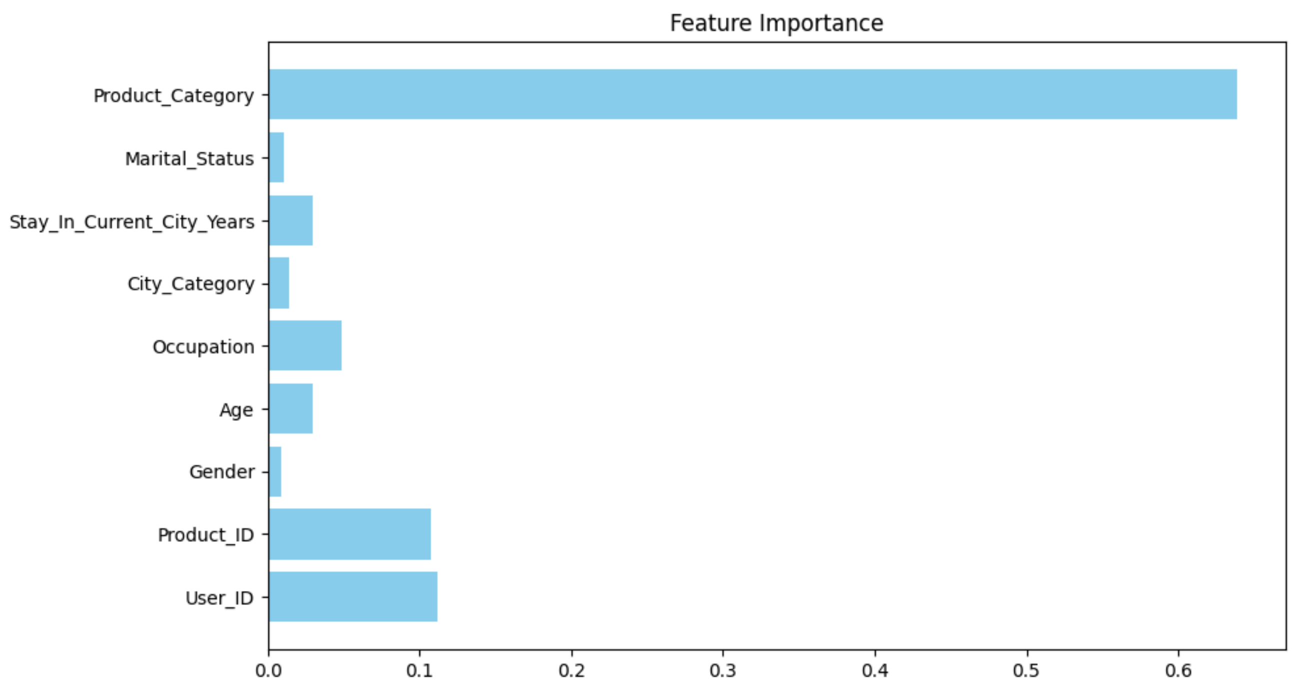
```
import numpy as np
plt.scatter(y_test, y_pred, alpha=0.3)
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'r--')
plt.xlabel('Actual Purchase')
plt.ylabel('Predicted Purchase')
plt.title('Actual vs Predicted Purchase')
plt.show()
```



Feature Importance

Feature importance is a score that reflects how much each feature contributes to the model's prediction. The Random Forest model computes this importance by analyzing how often and how effectively a feature is used to split the data in decision trees.

```
# Display the feature importance
feature_importances = model.feature_importances_
features = X.columns


plt.figure(figsize=(10, 6))
plt.barh(features, feature_importances, color='skyblue')
plt.title('Feature Importance')
plt.show()
```


Feature Importance
Product_Category
Marital_Status
Stay_In_Current_City_Years
City_Category
Occupation
Age
Gender
Product_ID
User_ID