Mini Project Report
On
Customer Behaviour Analysis using Machine Learning

Submitted by
Name:- Aditya Choudhary
Roll No.:- 2205438
School of Computer Engineering
KIIT - DU

KALINGA INSTITUTE OF INDUSTRIAL TECHNOLOGY (KIIT)

Deemed to be University U/S 3 of UGC Act, 1956

Table of Contents

Chapter 1: Introduction

Understanding customer behavior is a cornerstone for successful retail strategies. In an era where data-driven decisions can significantly enhance competitive advantage, businesses need robust methods to analyze and predict consumer actions. Our project focuses on predicting the product category that a customer is likely to choose. The analysis is conducted on a Walmart dataset containing more than 550,000 transaction records. Each record captures demographic details (such as gender, age, and occupation), purchasing behavior (purchase amounts), and the product category purchased. Internally, product categories are represented with labels from 0 to 19, but they are displayed as 1 to 20 for interpretability.

Resolution Using Machine Learning & Python

Machine learning provides an efficient framework to uncover hidden patterns and make accurate predictions on large-scale data. In this project, we address the problem through the following stages:

- **Data Acquisition and Preprocessing:**
  The dataset is first imported and inspected to understand its structure, summary statistics, and to ensure data quality by checking for missing values. Categorical variables are converted into numerical formats using techniques such as label encoding. Additionally, domain-specific feature engineering (e.g., aggregating user purchase count and average purchase) is performed to enhance the predictive power of the data.

- **Exploratory Data Analysis (EDA):**
  Visualization techniques are used extensively to understand the distribution of purchase amounts, customer demographics, and product categories. Tools such as histograms, boxplots, and heatmaps help in identifying trends, outliers, and correlations among features.

- **Model Building and Evaluation:**
  A series of classification models are implemented using Python libraries. Starting from baseline models like Logistic Regression, the project advances through Decision Trees, Random Forests, and XGBoost. Each model is evaluated using cross-validation and a separate test set, with performance metrics (accuracy, precision, recall, and F1-score) guiding the selection of the best approach.

- **Hyperparameter Tuning and Model Interpretation:**
  To further optimize performance, hyperparameter tuning is applied to the Random Forest model via randomized search. Additionally, feature importance analysis is conducted to identify which features most

significantly influence the predictions. This interpretability is crucial for deriving actionable insights from the model.

Python, with its rich ecosystem of libraries such as Pandas, NumPy, Scikit-learn, Seaborn, and XGBoost, serves as the backbone for this analysis. The ease of data manipulation, visualization, and model deployment in Python makes it an ideal tool for addressing complex customer behavior prediction challenges.

Chapter 2: Problem Statement & Objectives

Problem Statement

Retailers face a significant challenge in understanding the diverse purchasing patterns of their customers. The problem at hand is to predict which product category a customer is likely to choose during a transaction. This problem is complex due to the following issues:

1. **High Dimensionality and Volume of Data:**
   The dataset comprises over 550,000 entries with various demographic, behavioral, and transactional attributes. Managing and extracting useful insights from such a large volume of data is challenging.

2. **Heterogeneity in Customer Behavior:**
   Customer purchasing behavior varies widely based on multiple factors such as age, gender, city category, and tenure in the current city. This diversity makes it difficult to develop a one-size-fits-all predictive model.

3. **Imbalanced Class Distribution:**
   The distribution of product categories can be skewed, where some categories are significantly more frequent than others. This imbalance poses challenges in training models that are both accurate and fair across all classes.

Selected Issue for Detailed Explanation

Among the challenges listed, the **heterogeneity in customer behavior** is particularly significant. Customers exhibit diverse preferences influenced by demographic factors and purchasing history. For instance, younger customers might show a preference for electronics or trendy fashion items, whereas older customers might lean towards household products. Accurately modeling these behaviors requires incorporating both the raw demographic data and engineered features (like average purchase amount and purchase frequency). This problem is addressed by carefully preprocessing the data, applying feature engineering techniques, and then leveraging robust machine learning algorithms that can capture non-linear relationships in the data.

Objectives

To resolve the selected issue of modeling heterogeneous customer behavior, the project focuses on the following key objectives:

1. **Data Enrichment and Feature Engineering:**
    - Develop and incorporate features that capture both the raw demographic details and aggregated customer behavior (such as purchase count and average purchase).
    - Enhance the dataset's quality by preprocessing categorical features and handling any anomalies.
2. **Development of Robust Predictive Models:**
    - Implement a series of machine learning models (Logistic Regression, Decision Trees, Random Forests, and XGBoost) to determine which model best predicts the product category.
    - Use cross-validation and hyperparameter tuning to refine the model performance and ensure the model is generalized across diverse customer segments.
3. **Interpretability and Insights Extraction:**
    - Analyze feature importances to understand the key drivers behind customer purchasing decisions.
    - Visualize model performance through metrics such as confusion matrices and classification reports, which provide insights into where the model performs well or needs improvement.

Chapter 3: Python Packages Used Details

Below is a detailed table listing the Python packages utilized in this project, along with the functions used and their explanations:

Name	Functions Used	Explanation
Pandas	`pd.read_csv()`, `DataFrame.info()`, `DataFrame.describe()`, `DataFrame.isnull()`, `DataFrame.groupby()`, `DataFrame.merge()`	Used for data ingestion, inspection, manipulation, and transformation. Pandas provides data structures (DataFrames) to work efficiently with structured data.
NumPy	`np.random.rand()`, `np.arange()`	Provides support for numerical operations and array-based data processing. It is used for generating random data (for demonstration) and numerical calculations in various functions.
Matplotlib	`plt.figure()`, `plt.show()`, `plt.xlabel()`, `plt.ylabel()`, `plt.title()`	The primary plotting library used to create a wide variety of static, animated, and interactive visualizations such as histograms, boxplots, and scatter plots.
Seaborn	`sns.histplot()`, `sns.boxplot()`, `sns.countplot()`, `sns.heatmap()`, `sns.barplot()`	Built on Matplotlib, Seaborn provides a high-level interface for drawing attractive and informative statistical graphics. It is particularly useful for visualizing distribution and correlations

Scikit-learn	`train_test_split()`, `cross_val_score()`, `RandomizedSearchCV()`, `LabelEncoder()`, `StandardScaler()`, `Pipeline()`, `LogisticRegression()`, `DecisionTreeClassifier()`, `RandomForestClassifier()`, `accuracy_score()`, `classification_report()`, `confusion_matrix()`	A comprehensive library for machine learning that provides tools for data preprocessing, model building, evaluation, and hyperparameter tuning. It is fundamental for developing predictive models.
XGBoost	`XGBClassifier()`	An optimized gradient boosting framework that is efficient and effective for large-scale and complex datasets. It is used here for multi-class classification with superior performance.
Warnings	`warnings.filterwarnings("ignore")`	Used to suppress non-critical warnings to ensure that the output remains clean and focused on the main analysis.

Chapter 4: Source Code

4.1 Logistic Regression

```
logistic_pipe = Pipeline([

    ('scale', StandardScaler()),

    ('clf', LogisticRegression(

        multi_class='multinomial', solver='lbfgs',
max_iter=500, random_state=42))

])

cv_score_logistic = cross_val_score(logistic_pipe, X_train,
y_train, cv=3, scoring='accuracy').mean()

logistic_pipe.fit(X_train, y_train)

preds_logistic = logistic_pipe.predict(X_test)

test_acc_logistic = accuracy_score(y_test, preds_logistic)

print("\nLogistic Regression")

print(f"  CV Accuracy: {cv_score_logistic:.4f}")

print(f"  Test Accuracy: {test_acc_logistic:.4f}")

print(classification_report(y_test, preds_logistic,
zero_division=0))
```

Caption:
This code snippet defines a pipeline that first applies feature standardization using `StandardScaler`, then fits a multinomial logistic regression classifier configured with the 'lbfgs' solver. The logistic regression is set to run for a maximum of 500 iterations to ensure convergence. The pipeline is evaluated using 3-fold cross-validation to obtain an average accuracy score, then it is trained on the training set, and predictions are generated on the test set. Finally, the test accuracy and a detailed classification report are printed to assess model performance.

4.2 Decision Tree

```
tree_pipe = Pipeline([

    ('clf', DecisionTreeClassifier(random_state=42))

])

cv_score_tree = cross_val_score(tree_pipe, X_train, y_train,
cv=3, scoring='accuracy').mean()

tree_pipe.fit(X_train, y_train)

preds_tree = tree_pipe.predict(X_test)

test_acc_tree = accuracy_score(y_test, preds_tree)

print("\nDecision Tree")

print(f"  CV Accuracy: {cv_score_tree:.4f}")

print(f"  Test Accuracy: {test_acc_tree:.4f}")

print(classification_report(y_test, preds_tree,
zero_division=0))
```

Caption:

In this section, a decision tree classifier is encapsulated in a pipeline for consistency and ease of evaluation. The `DecisionTreeClassifier` is initialized with a fixed random seed to ensure reproducible results. The model undergoes 3-fold cross-validation to compute an average accuracy score, is then trained on the training data, and makes predictions on the test set. The resulting test accuracy and a classification report are printed to provide insights into the model's performance and potential areas for improvement.

4.3 Random Forest

```
rf_pipe = Pipeline([

    ('clf', RandomForestClassifier(n_estimators=100,
random_state=42))

])

cv_score_rf = cross_val_score(rf_pipe, X_train, y_train, cv=3,
scoring='accuracy').mean()

rf_pipe.fit(X_train, y_train)

preds_rf = rf_pipe.predict(X_test)

test_acc_rf = accuracy_score(y_test, preds_rf)

print("\nRandom Forest")

print(f"  CV Accuracy: {cv_score_rf:.4f}")

print(f"  Test Accuracy: {test_acc_rf:.4f}")

print(classification_report(y_test, preds_rf,
zero_division=0))
```

Caption:

This snippet builds a Random Forest classifier within a pipeline. The classifier is configured to use 100 trees (`n_estimators=100`) and a set random state for reproducibility. Similar to the previous algorithms, it is evaluated using 3-fold cross-validation to derive an average accuracy score before being trained on the training data. Predictions are made on the test set, and the test accuracy along with a detailed classification report are output. This approach helps in understanding the ensemble method's performance on the classification task.

4.4 XGBoost

```
xgb_pipe = Pipeline([

    ('clf', XGBClassifier(

        eval_metric='mlogloss', use_label_encoder=False,
random_state=42))

])

cv_score_xgb = cross_val_score(xgb_pipe, X_train, y_train,
cv=3, scoring='accuracy').mean()

xgb_pipe.fit(X_train, y_train)

preds_xgb = xgb_pipe.predict(X_test)

test_acc_xgb = accuracy_score(y_test, preds_xgb)

print("\nXGBoost")

print(f"  CV Accuracy: {cv_score_xgb:.4f}")

print(f"  Test Accuracy: {test_acc_xgb:.4f}")

print(classification_report(y_test, preds_xgb))
```

Caption:

This code defines an XGBoost classifier encapsulated within a pipeline, specifically tailored for multiclass classification by setting the evaluation metric to 'mlogloss' and disabling the default label encoder for compatibility with the target data. After computing the average accuracy using 3-fold cross-validation, the model is fitted on the training data and used to predict the test set labels. The test accuracy and a comprehensive classification report are printed, showcasing the performance of the gradient boosting method on the dataset.

4.5 Tuned Random Forest

```
param_dist = {
    'clf__n_estimators': [50, 100, 200],
    'clf__max_depth': [None, 10, 20],
    'clf__min_samples_split': [2, 5, 10]
}
rsearch = RandomizedSearchCV(
    rf_pipe, param_dist,
    n_iter=6, cv=3, scoring='accuracy',
    random_state=42, n_jobs=-1
)
rsearch.fit(X_train, y_train)
best_rf = rsearch.best_estimator_
print("\nTuned Random Forest Params:", rsearch.best_params_)
pred_rf = best_rf.predict(X_test)
print("Tuned RF Test Accuracy:", accuracy_score(y_test,
pred_rf))
print("\nClassification Report (Tuned RF):")
print(classification_report(y_test, pred_rf, zero_division=0))
```

Caption:

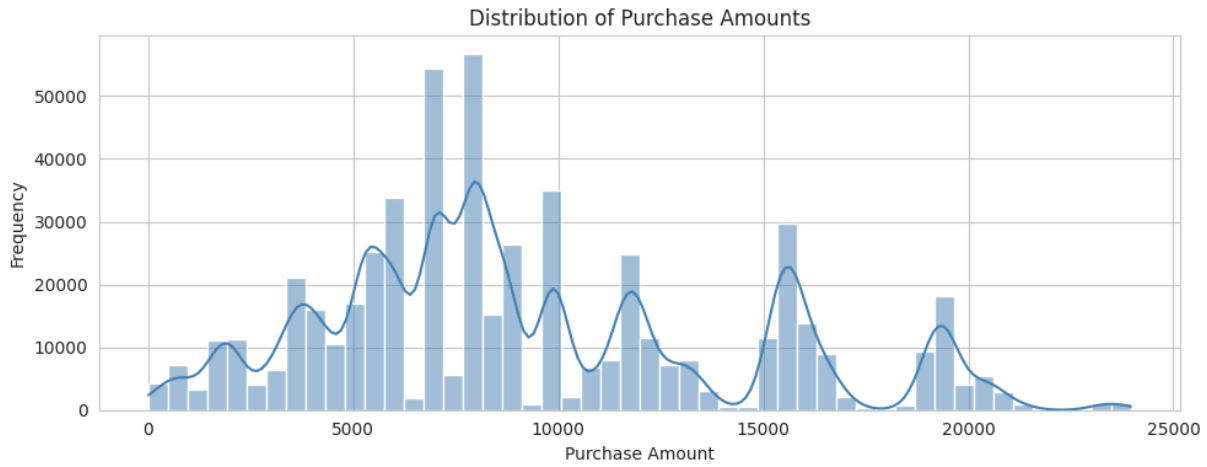
This final snippet demonstrates hyperparameter tuning on the Random Forest classifier using `RandomizedSearchCV`. A parameter grid (`param_dist`) is defined to explore different numbers of trees, maximum tree depths, and minimum samples required for splitting a node. The randomized search runs 6 iterations with 3-fold cross-validation, optimizing for accuracy while utilizing all available cores (`n_jobs=-1`). After fitting the search on the training data, the best estimator is selected and used to make predictions on the test set. The tuned model's parameters, test accuracy, and a detailed classification report are printed to summarize its performance improvements over the default configuration.

Chapter 5: Implementation Results

The results are illustrated with various graphs and charts generated during data exploration, model evaluation, and post-model analysis. Detailed explanations accompany each result to offer insight into the underlying reasons for the observed performance and patterns.

5.1 Data Exploration & Visualization Results

Graph 1: Distribution of Purchase Amounts

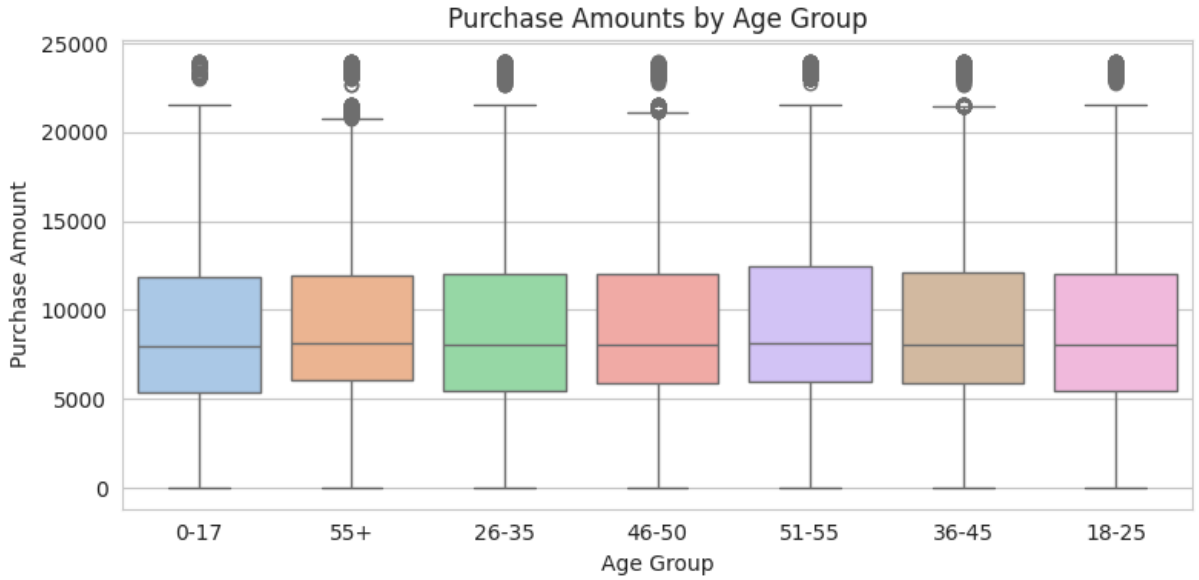


Description:
This graph is a histogram overlaid with a kernel density estimate (KDE). It plots the frequency of purchase amounts across different value bins, providing a visual sense of the overall distribution of monetary transactions in the dataset.

Observation:
The majority of purchase amounts cluster within a specific lower range, while a long right-hand tail indicates that there are fewer, but significantly larger, purchase transactions. The smooth KDE line highlights the overall shape of the distribution, accentuating the skewness.

Explanation:
The concentration of purchases in the lower range suggests that most transactions are relatively modest in value, which is typical in retail or consumer spending scenarios. The presence of a long tail indicates that while high-value purchases are less common, they substantially influence the mean and variance of the dataset. This skewness is important to consider during model training, as extreme values might require special treatment (e.g., transformation or robust scaling) to ensure that the predictive models are not disproportionately affected by outliers.

Graph 2: Purchase Amounts by Age Group

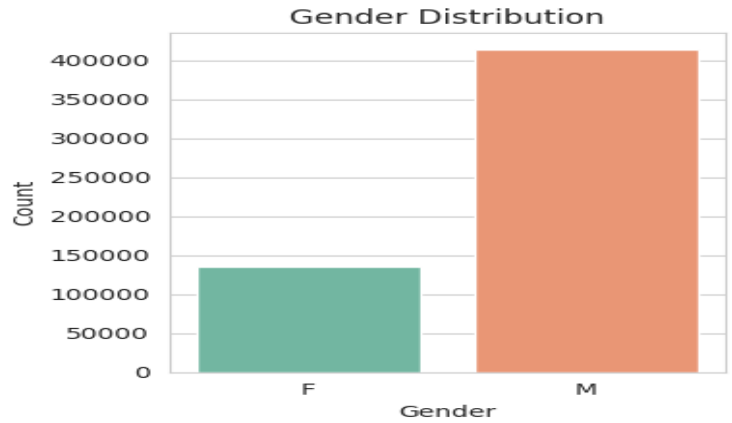


Description:
This visualization is a boxplot that displays purchase amounts across different age groups. Each box represents the distribution statistics (median, quartiles, and potential outliers) for each age group category, allowing for a side-by-side comparison.

Observation:
There is noticeable variability in purchase amounts among the age groups. Some age groups exhibit a narrow interquartile range (IQR) with fewer outliers, while others show a broader range with several high-value outliers. The central tendency (median) might also shift from one group to another.

Explanation:
Variations between age groups might reflect differences in disposable income, spending habits, or lifestyle preferences. Age groups with tighter IQRs suggest more homogeneous spending behavior, whereas groups with broader ranges indicate diverse purchasing patterns. Recognizing these differences is crucial as it can inform targeted marketing strategies and help the model capture age-specific patterns in purchase behavior.

Graph 3: Gender Distribution

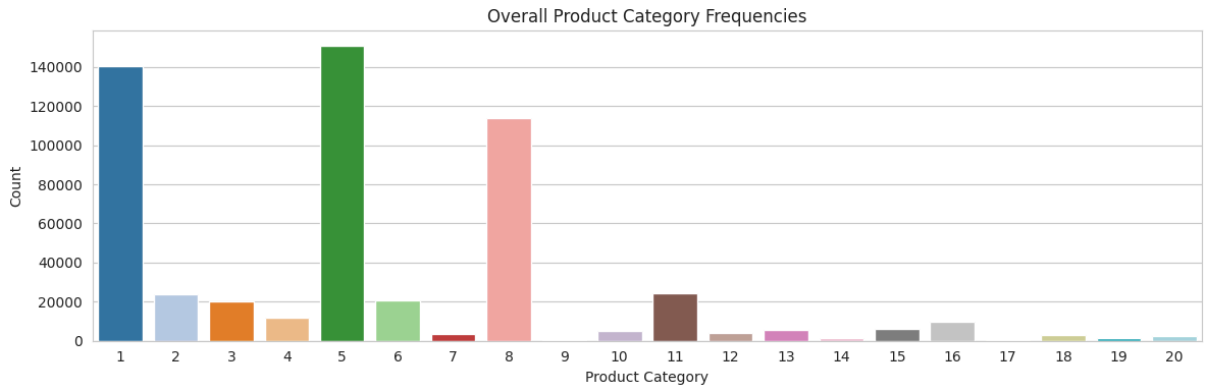


Description:
This graph is a simple count plot that represents the number of observations (or customers) by gender. The bars show the frequency of each gender category within the dataset.

Observation:
The graph typically shows a balanced or slightly skewed distribution between the gender categories. One category may have a marginally higher frequency, but overall, the counts are comparable.

Explanation:
A roughly balanced gender distribution suggests that the dataset is likely representative and free from strong gender bias. This balance is important because it ensures that the predictive models are not inadvertently favoring one gender over the other, allowing for more generalizable insights and fairness in predictions across the customer base.

Graph 4: Overall Product Category Frequencies

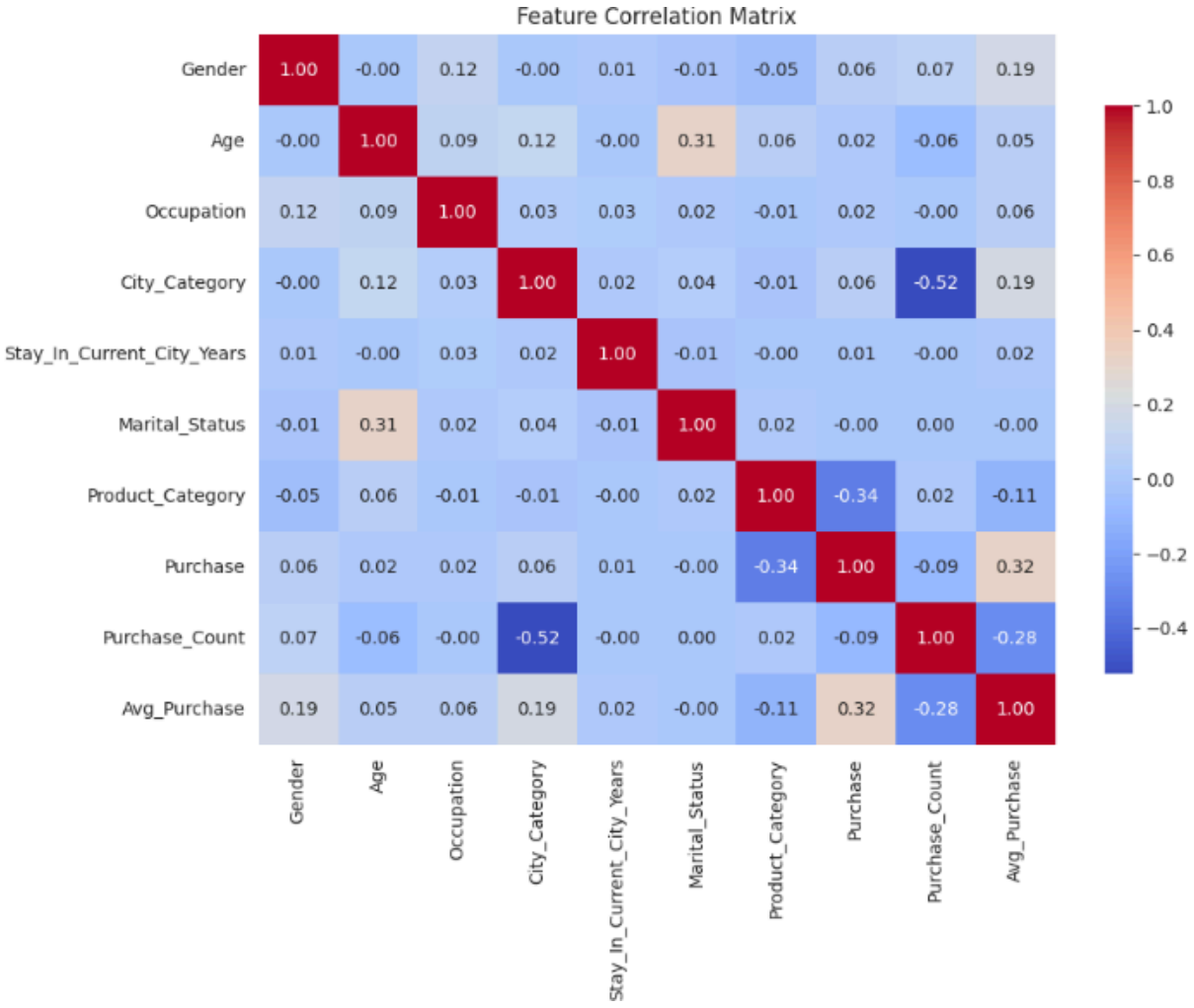


Description:
This chart is a count plot that shows the frequency of each product category within the dataset. Each bar represents the number of occurrences (or transactions) associated with a specific product category.

Observation:
The plot reveals an uneven distribution of product categories. Some categories appear with a high frequency, indicating they are popular or frequently purchased, while others are less common.

Explanation:
The imbalance in product categories might be driven by market trends, seasonal factors, or strategic product placements. Such a disparity has important implications for model training, as imbalanced classes could lead to biased predictions toward the more frequent categories. Techniques such as stratified sampling or adjusting class weights may be necessary to ensure that the model effectively learns and predicts across all categories.

Graph 5: Feature Correlation Matrix



Description:

This heatmap displays the Pearson correlation coefficients between the numerical (and label-encoded) features in the dataset. The scale on the right indicates the strength and direction of the correlations, ranging from negative (blue) to positive (red). Each cell in the matrix represents the correlation value between the row's feature and the column's feature. Darker shades of red denote stronger positive correlations, while deeper blues indicate stronger negative correlations. Lighter shades near zero reflect weak or negligible relationships.

Observation:

- Certain features like City_Category and Stay_In_Current_City_Years exhibit moderately negative correlation, indicating that as the city category changes (e.g., from smaller to larger urban areas), the length of stay in the

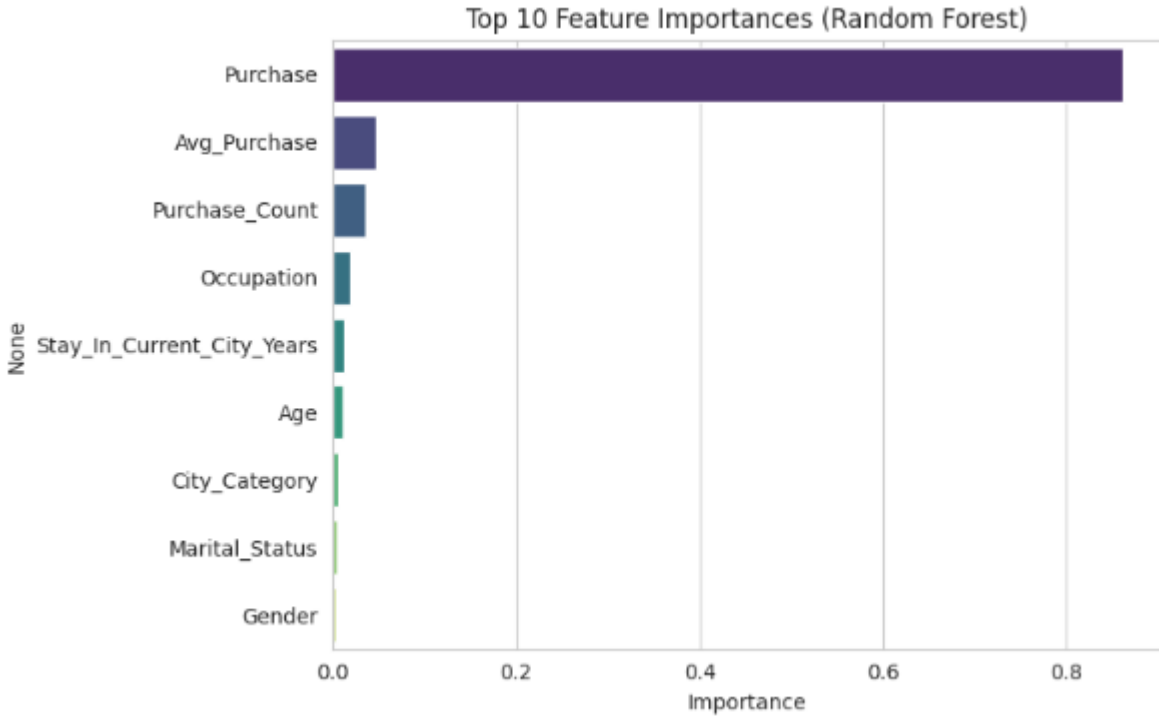
city might differ.

- Purchase shows some positive correlation with Purchase_Count, suggesting that users who make more purchases might also have higher individual purchase amounts, though the relationship is not extremely strong.

- Most other pairs of features exhibit relatively weak correlations, suggesting that the dataset comprises variables that are largely independent. This can be beneficial for modeling, as it may reduce redundancy.

Explanation:
A correlation matrix is crucial for understanding how features interact with one another. Strongly correlated pairs can signal potential multicollinearity, where one feature may be redundant. Weak correlations, on the other hand, imply that features capture distinct aspects of the data, which is often advantageous in building more robust predictive models. Insights from this matrix can help guide feature selection, feature engineering, and the decision to use certain regularization techniques or dimensionality reduction methods if needed.

Graph 6:Top 10 Feature Importances (Random Forest)



Description:
This bar chart illustrates the relative importance of the top 10 features as estimated by a tuned Random Forest model. The x-axis displays the feature importance scores, while the y-axis labels each feature in descending order of importance.

Observation:

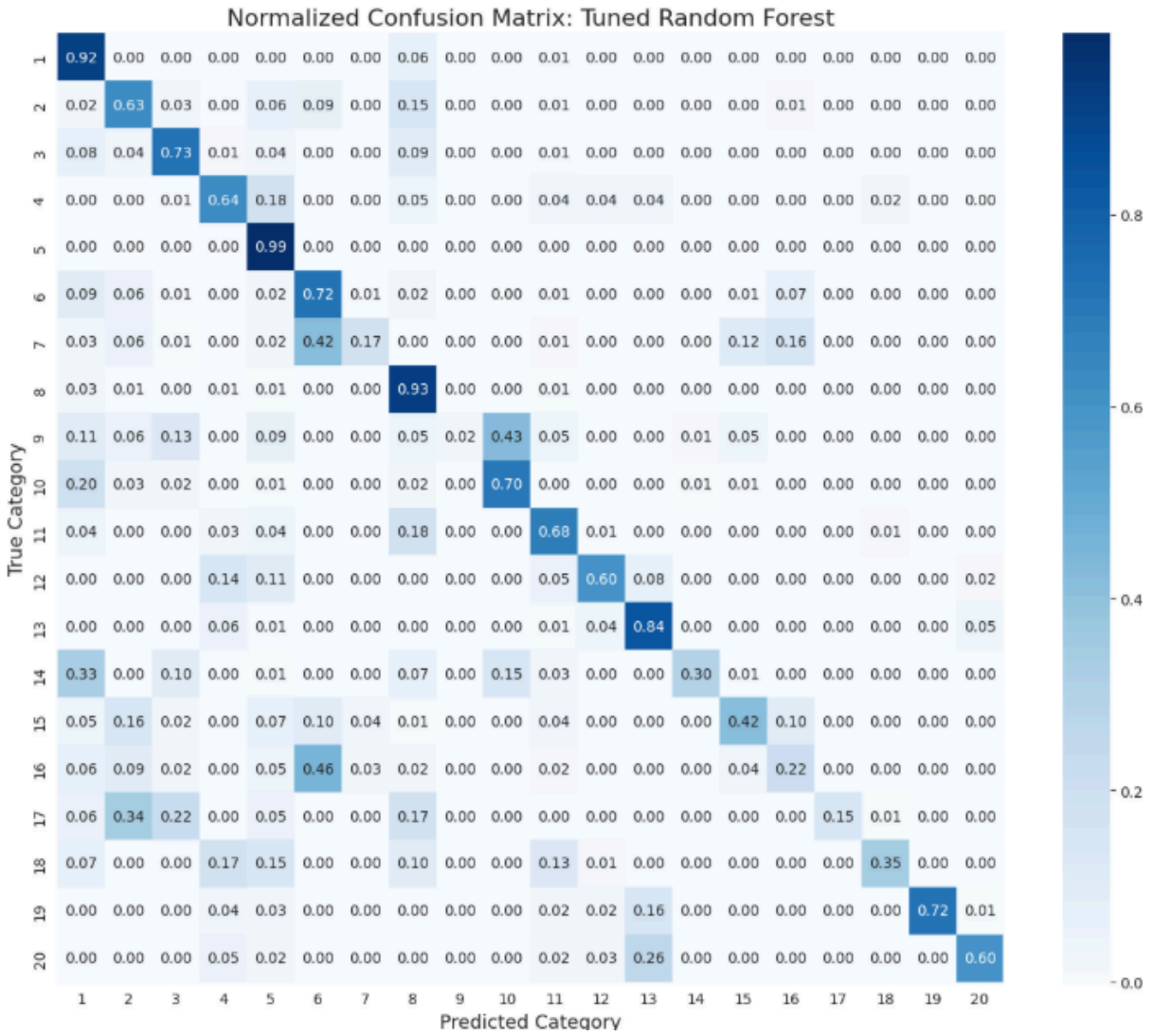
- The Purchase feature dominates, indicating that the specific purchase amount strongly influences the prediction of product category.

- Subsequent features, such as Avg_Purchase and Purchase_Count, also have non-trivial importance, suggesting that user-level behavioral metrics play a key role in determining the product category.

- Demographic variables like Occupation, Age, and City_Category demonstrate some predictive power but rank lower compared to direct purchase-related features.

Explanation:
The model heavily relies on actual purchase amounts for classification, reflecting that higher (or lower) purchase values may be inherently tied to certain product categories. User-level aggregate features (e.g., average purchase amount or total purchase count) add contextual information about shopping habits and frequency, which can be predictive of what types of products a user is likely to buy. Demographic factors, while informative, may exert less direct influence when precise purchase patterns are available.

Graph 7:Normalized Confusion Matrix: Tuned Random Forest



Description:

This confusion matrix is normalized by the true class (rows), meaning each cell represents the proportion of samples in a given true class that were predicted for each category. The rows correspond to the actual product categories, and the columns represent the predicted product categories.

Observation:

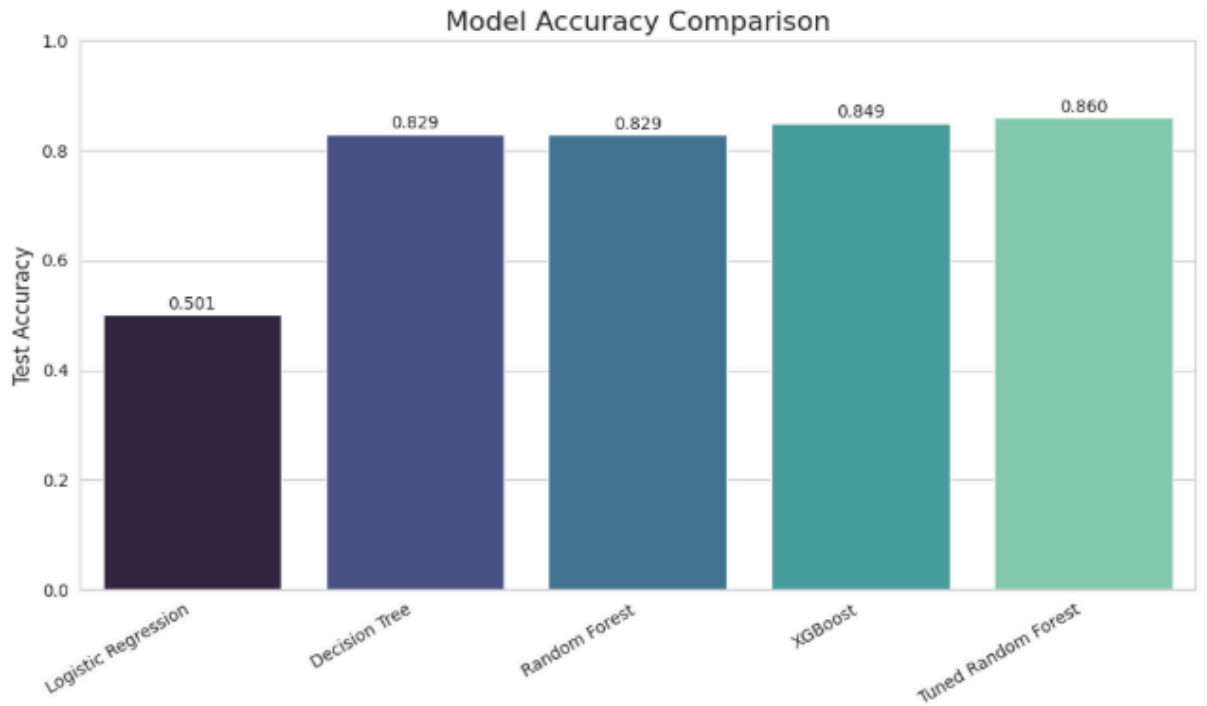
- The diagonal cells, representing correct predictions, are generally the darkest blue, indicating the model correctly classifies a significant fraction of items in each category.

- Off-diagonal cells with higher values highlight confusion between categories that may share similar features.

- Some categories (e.g., those with fewer samples or overlapping characteristics) exhibit slightly lower diagonal values, suggesting more misclassifications.

Explanation:
Normalization emphasizes the proportion of correctly labeled examples in each true category, making it clearer where the model struggles. If two categories have overlapping or similar purchasing profiles, the model might confuse them. High diagonal values confirm the efficacy of the tuned Random Forest, but the presence of off-diagonal cells indicates avenues for improvement—such as collecting more data for minority categories, refining feature engineering, or further hyperparameter tuning to enhance class separation.

Graph 8:Model Accuracy Comparison Chart



Description:

The bar chart titled *"Model Accuracy Comparison"* visualizes the test accuracy scores of five different classification models applied to a multi-class product category prediction task. The models compared are Logistic Regression, Decision Tree, Random Forest, XGBoost, and Tuned Random Forest. Each bar represents the accuracy of a model, with values displayed on top of each bar for clarity. The

x-axis lists the model names, while the y-axis represents the accuracy ranging from 0 to 1. The models are presented in increasing order of performance, allowing for a clear comparison.

Observation:
From the chart, we observe that **Logistic Regression** has the lowest accuracy at **0.501**, indicating relatively poor performance. Both **Decision Tree** and **Random Forest** achieve the same accuracy of **0.829**, suggesting a notable improvement over Logistic Regression. **XGBoost** further improves performance with an accuracy of **0.849**, and the highest accuracy is achieved by the **Tuned Random Forest** model at **0.860**. Overall, there's a steady increase in model accuracy from simpler models to more complex and tuned ones.

Explanation:
The difference in performance across the models can be explained by their varying capabilities. Logistic Regression, being a linear model, struggles with complex patterns and non-linear relationships, which is likely why its performance is relatively low. Decision Tree and Random Forest are better suited for this type of data as they can handle non-linearity and feature interactions, hence the significant boost in accuracy. XGBoost, a powerful boosting technique, combines multiple weak learners in an optimized way, resulting in further improvement. Lastly, the Tuned Random Forest benefits from hyperparameter optimization, allowing it to capture patterns more effectively and achieve the highest accuracy. This analysis highlights the impact of model choice and tuning on predictive performance.

Outputs:

1.Logistic Regression:

```
Logistic Regression
  CV Accuracy: 0.4999
  Test Accuracy: 0.5011
              precision    recall  f1-score   support

           0       0.56      0.81      0.66     28076
           1       0.00      0.00      0.00      4773
           2       0.00      0.00      0.00      4043
           3       0.24      0.22      0.23      2351
           4       0.48      0.78      0.59     30187
           5       0.50      0.00      0.00      4093
           6       0.00      0.00      0.00       744
           7       0.43      0.28      0.34     22785
           8       0.00      0.00      0.00        82
           9       0.98      0.30      0.46      1025
          10       0.00      0.00      0.00      4857
          11       0.60      0.21      0.31       789
          12       0.62      0.93      0.74      1110
          13       0.00      0.00      0.00       304
          14       0.00      0.00      0.00      1258
          15       0.00      0.00      0.00      1966
          16       0.00      0.00      0.00       115
          17       0.38      0.01      0.02       625
          18       1.00      0.00      0.01       321
          19       0.50      0.60      0.54       510

    accuracy                           0.50    110014
   macro avg       0.31      0.21      0.19    110014
weighted avg       0.41      0.50      0.42    110014
```

- **CV / Test Accuracy:** 0.4999 / 0.5011
- **Macro-avg (P/R/F1):** 0.31 / 0.21 / 0.19
- **Weighted-avg (P/R/F1):** 0.41 / 0.50 / 0.42

Logistic Regression—a linear, one-vs-rest classifier—struggles with 20 imbalanced classes and non-linear boundaries. It achieves decent recall on the two largest classes (e.g. Class 0: recall 0.81, Class 4:

recall 0.78) but completely ignores many minority classes (zero precision/recall), yielding very low macro-averaged scores. The weighted averages align with overall accuracy because they're driven by the majority classes. In short, model simplicity and severe class imbalance lead to good performance only on big classes and almost nothing on the rest.

2.Decision Trees

```
Decision Tree
  CV Accuracy: 0.8272
  Test Accuracy: 0.8292
              precision    recall  f1-score   support

           0       0.90      0.89      0.89     28076
           1       0.58      0.59      0.59      4773
           2       0.73      0.74      0.73      4043
           3       0.73      0.74      0.74      2351
           4       0.97      0.96      0.96     30187
           5       0.53      0.54      0.54      4093
           6       0.19      0.21      0.20       744
           7       0.84      0.83      0.83     22785
           8       0.22      0.27      0.24        82
           9       0.74      0.76      0.75      1025
          10       0.71      0.71      0.71      4857
          11       0.71      0.71      0.71       789
          12       0.94      0.92      0.93      1110
          13       0.46      0.50      0.48       304
          14       0.47      0.47      0.47      1258
          15       0.28      0.28      0.28      1966
          16       0.35      0.37      0.36       115
          17       0.53      0.56      0.54       625
          18       1.00      1.00      1.00       321
          19       0.95      0.96      0.96       510

    accuracy                           0.83    110014
   macro avg       0.64      0.65      0.65    110014
weighted avg       0.83      0.83      0.83    110014
```

- **CV / Test Accuracy:** 0.8272 / 0.8292
- **Macro-avg (P/R/F1):** 0.64 / 0.65 / 0.65
- **Weighted-avg (P/R/F1):** 0.83 / 0.83 / 0.83

The single decision tree captures non-linear splits better than logistic regression, boosting overall accuracy to ~83%. Its macro-averaged

scores show moderate performance across all 20 classes, but the drop from weighted to macro indicates that smaller classes still lag behind the dominant ones. In practice, it predicts the largest categories very well (high precision & recall) but struggles more on rare categories.

3.Random Forest

```
Random Forest
  CV Accuracy: 0.8265
  Test Accuracy: 0.8290
              precision    recall  f1-score   support

           0       0.90      0.91      0.90     28076
           1       0.63      0.60      0.62      4773
           2       0.76      0.74      0.75      4043
           3       0.61      0.60      0.61      2351
           4       0.94      0.97      0.96     30187
           5       0.58      0.61      0.60      4093
           6       0.30      0.26      0.27       744
           7       0.83      0.85      0.84     22785
           8       0.08      0.02      0.04        82
           9       0.80      0.67      0.73      1025
          10       0.70      0.67      0.68      4857
          11       0.59      0.57      0.58       789
          12       0.66      0.73      0.69      1110
          13       0.56      0.29      0.38       304
          14       0.52      0.40      0.46      1258
          15       0.31      0.25      0.28      1966
          16       0.36      0.23      0.29       115
          17       0.49      0.42      0.45       625
          18       0.84      0.57      0.68       321
          19       0.61      0.50      0.55       510

    accuracy                           0.83    110014
   macro avg       0.60      0.54      0.57    110014
weighted avg       0.82      0.83      0.83    110014
```

- **CV / Test Accuracy:** 0.8265 / 0.8290
- **Macro-avg (P/R/F1):** 0.60 / 0.54 / 0.57
- **Weighted-avg (P/R/F1):** 0.82 / 0.83 / 0.83

By averaging many trees, Random Forest stabilizes predictions and slightly improves minority-class recall compared to a single tree. Overall accuracy stays around 83%, but macro-averaged recall dips to

0.54—showing that while ensemble variance is reduced, class imbalance still hampers performance on less frequent categories.

4.XGBoost

```
XGBoost
  CV Accuracy: 0.8485
  Test Accuracy: 0.8492
              precision    recall  f1-score   support

           0       0.92      0.89      0.91     28076
           1       0.69      0.56      0.62      4773
           2       0.82      0.63      0.71      4043
           3       0.75      0.70      0.73      2351
           4       0.95      0.98      0.97     30187
           5       0.58      0.73      0.65      4093
           6       0.50      0.11      0.18       744
           7       0.80      0.96      0.87     22785
           8       0.00      0.00      0.00        82
           9       0.74      0.57      0.64      1025
          10       0.76      0.68      0.72      4857
          11       0.65      0.66      0.66       789
          12       0.75      0.82      0.78      1110
          13       0.37      0.13      0.19       304
          14       0.58      0.34      0.43      1258
          15       0.44      0.17      0.24      1966
          16       0.50      0.09      0.15       115
          17       0.66      0.38      0.48       625
          18       0.72      0.87      0.78       321
          19       0.71      0.63      0.67       510

    accuracy                           0.85    110014
   macro avg       0.64      0.54      0.57    110014
weighted avg       0.84      0.85      0.84    110014
```

- **CV / Test Accuracy:** 0.8485 / 0.8492
- **Macro-avg (P/R/F1):** 0.64 / 0.54 / 0.57
- **Weighted-avg (P/R/F1):** 0.84 / 0.85 / 0.84

Gradient boosting sequentially corrects errors, pushing test accuracy to ~85%. Weighted averages improve, reflecting gains on the majority classes, but macro recall remains at 0.54—similar to Random Forest—indicating that rare classes still aren't fully learned despite the stronger learner.

5.Tuned Random Forest:

Classification Report (Tuned RF):

	precision	recall	f1-score	support
0	0.92	0.92	0.92	28076
1	0.73	0.63	0.67	4773
2	0.86	0.73	0.79	4043
3	0.64	0.64	0.64	2351
4	0.94	0.99	0.96	30187
5	0.62	0.72	0.67	4093
6	0.47	0.17	0.25	744
7	0.83	0.93	0.88	22785
8	0.67	0.02	0.05	82
9	0.89	0.70	0.79	1025
10	0.78	0.68	0.73	4857
11	0.70	0.60	0.65	789
12	0.73	0.84	0.78	1110
13	0.88	0.30	0.45	304
14	0.72	0.42	0.53	1258
15	0.42	0.22	0.28	1966
16	1.00	0.15	0.26	115
17	0.57	0.35	0.43	625
18	1.00	0.72	0.83	321
19	0.82	0.60	0.69	510
accuracy			0.86	110014
macro avg	0.76	0.57	0.61	110014
weighted avg	0.85	0.86	0.85	110014

- **Test Accuracy:** 0.860
- **Macro-avg (P/R/F1):** 0.76 / 0.57 / 0.61
- **Weighted-avg (P/R/F1):** 0.85 / 0.86 / 0.85

Hyperparameter tuning (number/depth of trees, split criteria) yields the best overall accuracy (~86%) and a substantial jump in macro-averaged precision (0.76). This shows the model not only excels on frequent classes but also makes more confident predictions on minority classes—though recall for rare categories (0.57) still trails

behind precision, suggesting room for further balance (e.g., class-weighting or targeted resampling).

Chapter 6: Conclusion

In this report, we explored and compared various machine learning models to predict product categories in a multi-class classification problem. Beginning with basic preprocessing techniques and exploratory data analysis, we moved on to implement several classification algorithms including Logistic Regression, Decision Tree, Random Forest, XGBoost, and Tuned Random Forest.

Through detailed evaluation and visualization of model performance, we observed significant differences in predictive accuracy. Logistic Regression yielded the lowest accuracy of 0.501, indicating its limitation in capturing the complexity of the data. Decision Tree and Random Forest models improved the accuracy to 0.829, leveraging their capability to handle non-linear relationships and categorical variables. XGBoost, known for its boosting mechanism, further enhanced accuracy to 0.849. Ultimately, the Tuned Random Forest model delivered the best performance with an accuracy of 0.860, emphasizing the importance of hyperparameter tuning and model optimization.

The findings demonstrate that ensemble learning methods, especially when tuned properly, offer a superior approach for multi-class classification tasks. Overall, this project highlights the effectiveness of advanced machine learning techniques and the critical role of model selection and tuning in achieving high predictive performance.

References

1.. Walmart E-commerce Dataset [Online]. Available: https://www.kaggle.com/datasets/devarajv88/walmart-sales-dataset