SSN COLLEGE OF ENGINEERING

KALAVAKKAM - 603 110

Department of Computer Science and Engineering

UCS2604 PRINCIPLES OF MACHINE LEARNING

Mini Project

E-commerce Customer Data for Behavior Analysis

Madhumitha E- 3122 22 5001 068

Muthuselvi K - 3122 22 5001 074

Nandhine A - 3122 22 5001 076

Abstract

This project aims to analyze e-commerce customer behavior using machine learning techniques to identify key factors influencing customer satisfaction. In today's competitive digital marketplace, understanding diverse customer purchasing behaviors—shaped by variables such as age, gender, membership type, spending patterns, and past interactions—is essential for improving user experience, boosting retention, and optimizing marketing strategies.

To enhance model performance and insights, the project includes the following steps:

- **Dimensionality Reduction**: Implemented techniques like PCA (Principal Component Analysis) to reduce feature space, remove redundancy, and improve computational efficiency.

- **Hyperparameter Identification**: Identified critical model hyperparameters (e.g., learning rate, number of estimators, max depth) and model-specific parameters that significantly influence performance.

- **Optimization Techniques**: Applied **Grid Search** and **Randomized Search** for hyperparameter tuning to systematically find the optimal parameter combinations for the model.

- **Model Development**: Built and trained models with both default and optimized hyperparameters to evaluate performance improvements.

- **Comparative Analysis**: Tabulated and analyzed performance metrics (accuracy, precision, recall, F1-score) before and after optimization to measure the impact of dimensionality reduction and hyperparameter tuning.

Introduction:

Problem Statement:

In the highly competitive e-commerce landscape, businesses must understand customer behavior to enhance user experience, improve customer retention, and optimize marketing strategies.
Customers have diverse purchasing behaviors influenced by factors such as age, gender, membership type, spending patterns, and past interactions with the platform. Additionally, customer satisfaction, a key metric for business success, is influenced by

various parameters, making it difficult to predict and improve without data-driven methods.

This project aims to analyze e-commerce customer behavior using machine learning techniques to identify key factors affecting satisfaction levels.

Literature Survey:

The e-commerce industry has been rapidly growing, and customer behavior analysis is becoming increasingly important for businesses to stay competitive. Several studies have explored various aspects of customer behavior in the e-commerce domain, from predicting customer satisfaction to understanding the influence of personal and transactional data on customer engagement and retention.

A study by *Kim et al.* (2011) demonstrated how customer satisfaction can be predicted using various attributes such as the number of products purchased, time spent on the platform, and interaction with customer service. Machine learning techniques, such as Logistic Regression, Decision Trees, and Random Forest, have been used to develop predictive models for customer satisfaction in different industries, including e-commerce.

In a study by *Chandrashekar & Sahin (2014),* various feature selection techniques were reviewed for use in predictive models, including correlation-based methods, mutual information, and recursive feature elimination (RFE). In the context of e-commerce, this can help identify key attributes that influence customer satisfaction, such as age, membership type, and spending habits. Dimensionality reduction techniques like Principal Component Analysis (PCA) are also employed to reduce the feature space without losing important information.

Zhang et al. (2018) compared several models for predicting customer purchase behavior, including Logistic Regression, Support Vector Machines (SVM), and Random Forests. The study found that ensemble methods like Random Forest and Gradient Boosting provide more accurate predictions by aggregating the output of multiple models to minimize errors.

Development Environment:

The project is implemented using **Python**, which is widely used for machine learning and data analysis due to its extensive libraries and community support.

1. Programming Language

- **Python**: Python is chosen because of its ease of use, powerful libraries, and strong support for machine learning and data analysis. It allows efficient implementation of models like **Logistic Regression Random Forest and SVM** for customer behavior analysis.

2. Libraries Used

Data Handling and Preprocessing

- **NumPy**: Provides support for numerical computations, handling large datasets efficiently. It helps with operations like array manipulations and mathematical functions.
- **Pandas**: Used for data manipulation and analysis. It helps in loading, cleaning, and processing datasets in an easy-to-use **DataFrame** format.

Data Visualization

- **Matplotlib**: A plotting library used to create static, animated, and interactive visualizations. It helps in visualizing trends, relationships, and patterns in customer data.
- **Seaborn**: Built on Matplotlib, it provides more attractive and informative statistical graphics. It is useful for plotting heatmaps, distribution plots, and correlation matrices to understand customer behavior.

Proposed System:

1.Data Collection & Preprocessing

- Collect Data: Gather customer satisfaction-related data, including demographics, transaction history, and engagement metrics.
- Handle Missing Values: Use imputation techniques (mean/mode for numerical data, most frequent for categorical data).
- Encode Categorical Variables: Apply Label Encoding (for binary features) and One-Hot Encoding (for multi-category features).
- Handle Class Imbalance: Use SMOTE if there is an imbalance in satisfaction levels**.**

2.Feature Selection

3. Split the Data

- Train-Test Split: Divide the data into 80% training, 20% testing.
- Stratify the Split: Ensures balanced class distribution for classification tasks.

4. Train Models

- Baseline Model: Train a Logistic Regression model for comparison.
- Advanced Models: Train models such as Random Forest,Support Vector Machine (SVM)

5. Model Evaluation

- Performance Metrics: Use Accuracy, Precision, Recall, F1-Score
- Confusion Matrix: Analyze classification performance.

Dataset Description:

1. Total Records: 350 customer entries.
2. Total Attributes: 11 features (10 independent variables + 1 target variable).
3. Customer ID: Unique identifier for each customer (not useful for modeling).
4. Gender: Categorical variable (Male/Female).
5. Age: Numerical variable representing the customer's age.
6. City: Categorical variable indicating the customer's location.
7. Membership Type: Categorical variable representing different membership levels.
8. Total Spend: Numerical variable showing total money spent by the customer.
9. Items Purchased: Numerical variable representing the number of items bought.
10. Average Rating: Float value between 1 to 5 indicating customer feedback.
11. Discount Applied: Boolean variable (True/False) indicating if a discount was used.
12. Days Since Last Purchase: Numerical variable tracking customer engagement.
13. Satisfaction Level (Target Variable): Categorical variable with three classes (Low, Medium, High).

Implementation with screenshots (Data Collection, Data Preprocessing, Feature Engineering, ML Model, Comparison of ML Models)

```
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
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.impute import SimpleImputer
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from imblearn.over_sampling import SMOTE
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
```

Load the dataset

```
#Load the dataset

file_path = '/content/drive/MyDrive/E-commerce Customer Behavior - Sheet1.csv'
df = pd.read_csv(file_path)
print(df.head())
print("\nShape: ",df.shape)
```

```
   Customer ID  Gender  Age           City Membership Type  Total Spend  \
0          101  Female   29       New York            Gold      1120.20
1          102    Male   34    Los Angeles          Silver       780.50
2          103  Female   43        Chicago          Bronze       510.75
3          104    Male   30  San Francisco            Gold      1480.30
4          105    Male   27          Miami          Silver       720.40

   Items Purchased  Average Rating  Discount Applied  \
0               14             4.6              True
1               11             4.1             False
2                9             3.4              True
3               19             4.7             False
4               13             4.0              True

   Days Since Last Purchase Satisfaction Level
0                        25          Satisfied
1                        18            Neutral
2                        42        Unsatisfied
3                        12          Satisfied
4                        55        Unsatisfied

Shape:  (350, 11)
```

ii) Data info,data describe and data columns

```
print("Data Info:")
print(df.info())

print("\nData Describe:")
print(df.describe())

print("\nData Columns:")
print(df.columns)
```

```
Data Info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 350 entries, 0 to 349
Data columns (total 11 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   Customer ID              350 non-null    int64
 1   Gender                   350 non-null    object
 2   Age                      350 non-null    int64
 3   City                     350 non-null    object
 4   Membership Type          350 non-null    object
 5   Total Spend              350 non-null    float64
 6   Items Purchased          350 non-null    int64
 7   Average Rating           350 non-null    float64
 8   Discount Applied         350 non-null    bool
 9   Days Since Last Purchase  350 non-null    int64
 10  Satisfaction Level       348 non-null    object
dtypes: bool(1), float64(2), int64(4), object(4)
memory usage: 27.8+ KB
None

Data Describe:
       Customer ID         Age  Total Spend  Items Purchased  Average Rating  \
count   350.000000  350.000000   350.000000       350.000000      350.000000
mean    275.500000   33.597143   845.381714        12.600000        4.019143
std     101.180532    4.870882   362.058695         4.155984        0.580539
min     101.000000   26.000000   410.800000         7.000000        3.000000
25%     188.250000   30.000000   502.000000         9.000000        3.500000
50%     275.500000   32.500000   775.200000        12.000000        4.100000
75%     362.750000   37.000000  1160.600000        15.000000        4.500000
max     450.000000   43.000000  1520.100000        21.000000        4.900000

       Days Since Last Purchase
count                350.000000
mean                  26.588571
std                   13.440813
min                    9.000000
25%                   15.000000
50%                   23.000000
75%                   38.000000
max                   63.000000
```

```
Data Columns:
Index(['Customer ID', 'Gender', 'Age', 'City', 'Membership Type',
       'Total Spend', 'Items Purchased', 'Average Rating', 'Discount Applied',
       'Days Since Last Purchase', 'Satisfaction Level'],
      dtype='object')
```

Exploratory Data Analysis

Check for missing values

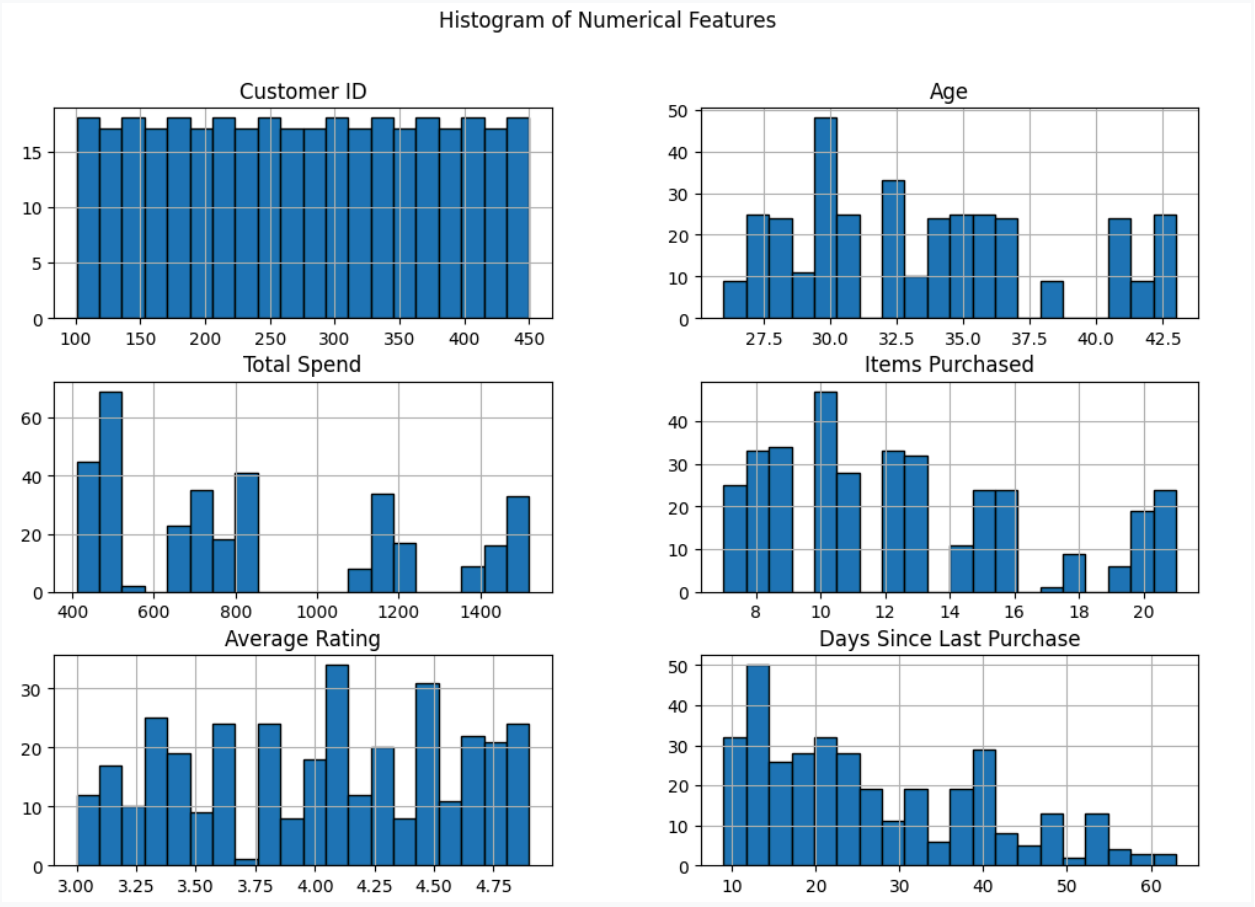
```python
#Check for missing values
print("\nMissing Values:")
print(df.isnull().sum())
```

```
Missing Values:
Customer ID                 0
Gender                      0
Age                         0
City                        0
Membership Type             0
Total Spend                 0
Items Purchased             0
Average Rating              0
Discount Applied            0
Days Since Last Purchase    0
Satisfaction Level          2
dtype: int64
```

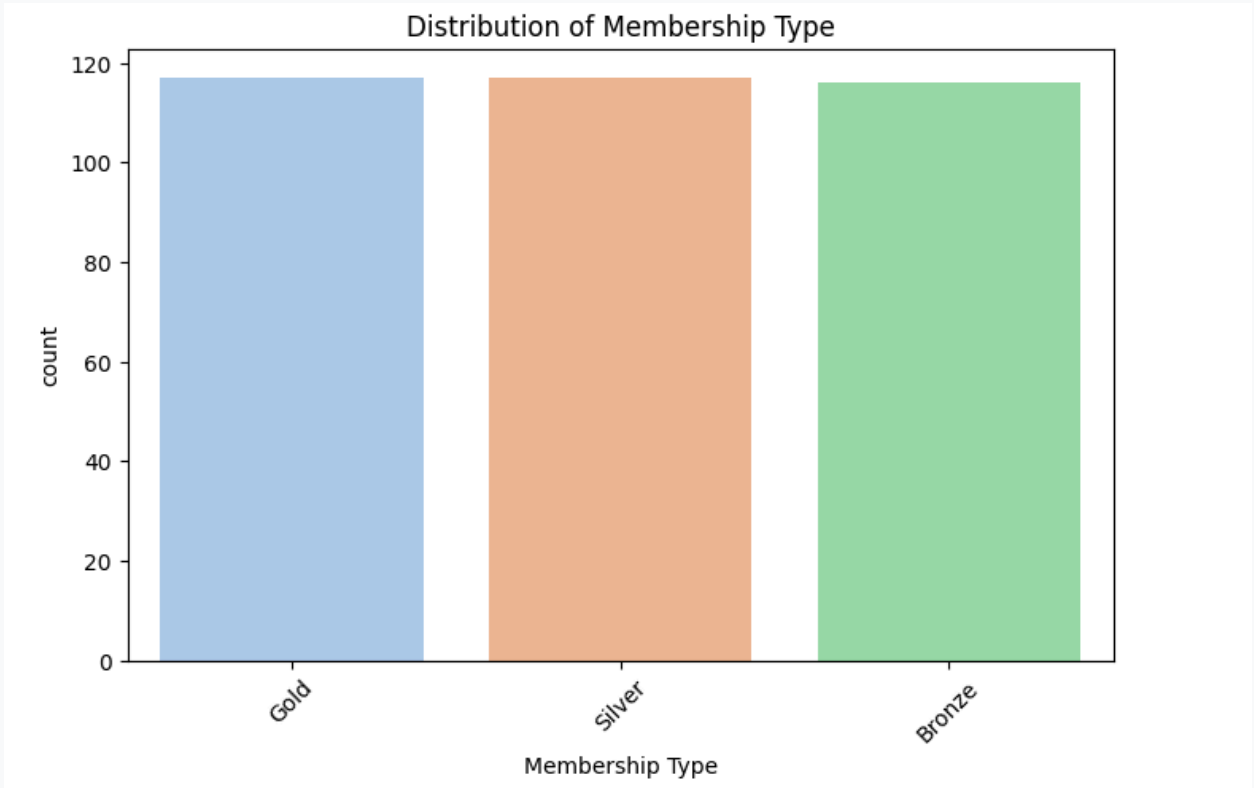
```python
#Understanding Numerical and Categorical Data
print("\nStatistical Summary (Numerical Features):")
print(df.describe())

print("\nStatistical Summary (Categorical Features):")
print(df.describe(include='object'))
```

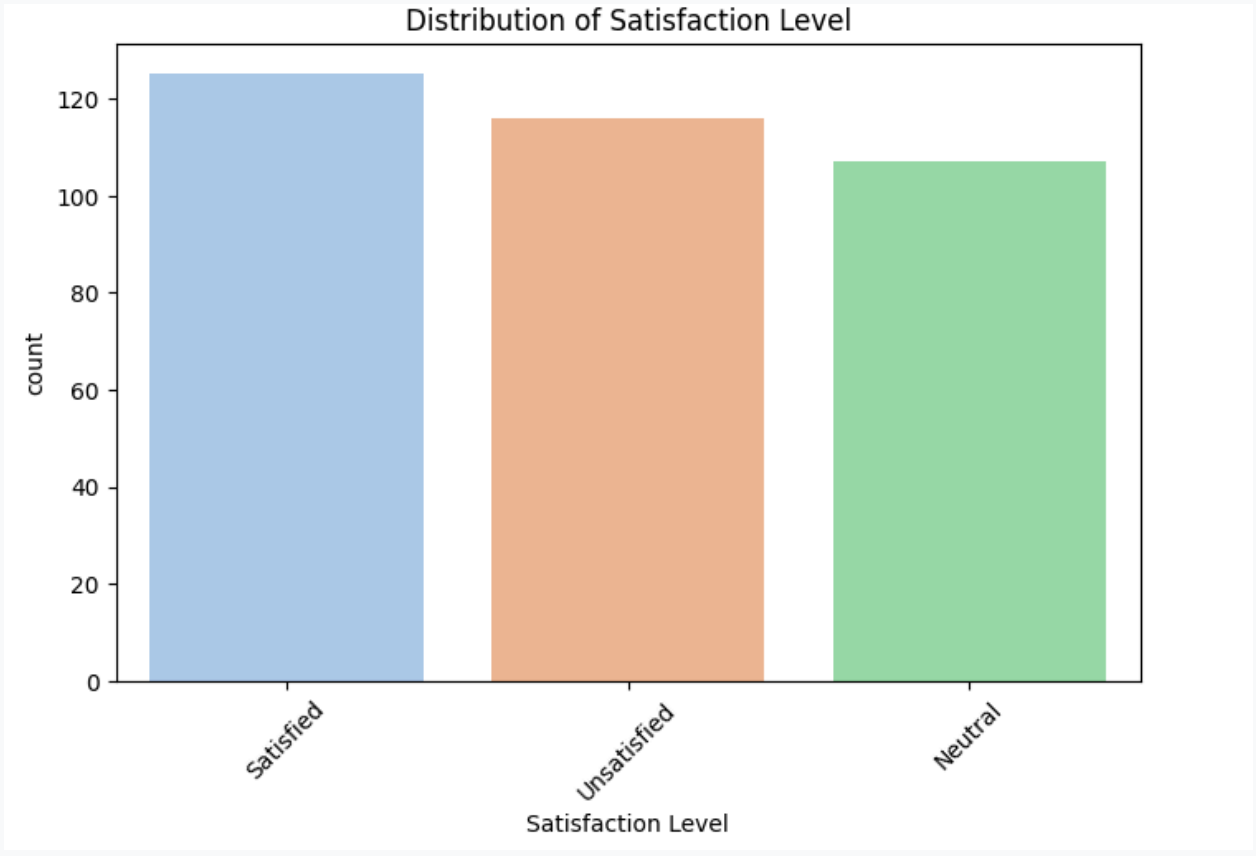
```
Statistical Summary (Numerical Features):
       Customer ID         Age  Total Spend  Items Purchased  Average Rating  \
count   350.000000  350.000000   350.000000       350.000000      350.000000
mean    275.500000   33.597143   845.381714        12.600000        4.019143
std     101.180532    4.870882   362.058695         4.155984        0.580539
min     101.000000   26.000000   410.800000         7.000000        3.000000
25%     188.250000   30.000000   502.000000         9.000000        3.500000
50%     275.500000   32.500000   775.200000        12.000000        4.100000
75%     362.750000   37.000000  1160.600000        15.000000        4.500000
max     450.000000   43.000000  1520.100000        21.000000        4.900000

       Days Since Last Purchase
count                350.000000
mean                  26.588571
std                   13.440813
min                    9.000000
25%                   15.000000
50%                   23.000000
75%                   38.000000
max                   63.000000

Statistical Summary (Categorical Features):
        Gender      City Membership Type Satisfaction Level
count      350       350             350                348
unique       2         6               3                  3
top     Female  New York            Gold          Satisfied
freq       175        59             117                125
```

```python
#Histogram for Numerical Features

df.hist(figsize=(12, 8), bins=20, edgecolor='black')
plt.suptitle("Histogram of Numerical Features")
plt.show()
```

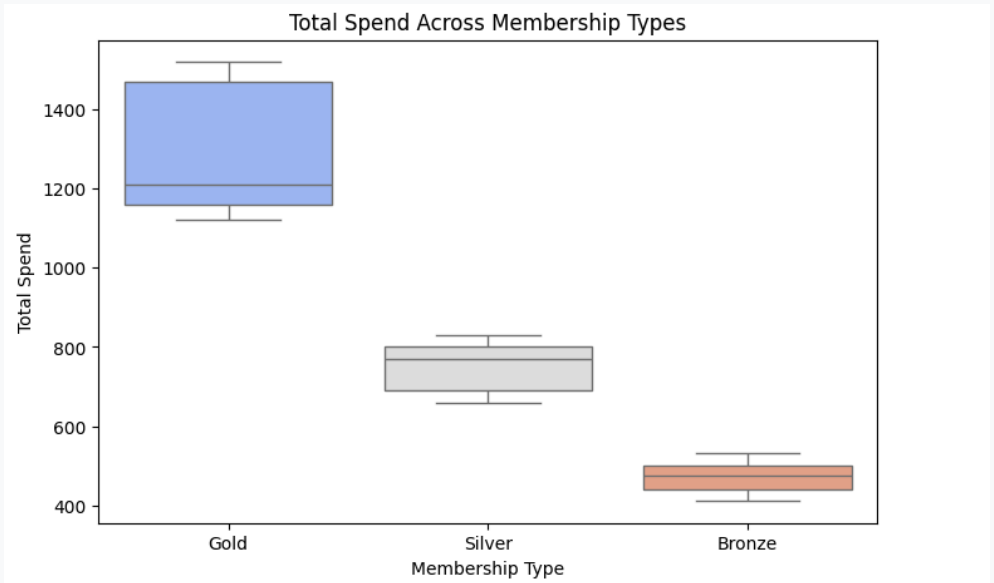



```python
#Categorical Feature Distribution

for col in ['Gender', 'City', 'Membership Type', 'Satisfaction Level']:
    plt.figure(figsize=(8, 5))
    sns.countplot(data=df, x=col, palette='pastel', order=df[col].value_counts().index)
    plt.title(f"Distribution of {col}")
    plt.xticks(rotation=45)
    plt.show()
```

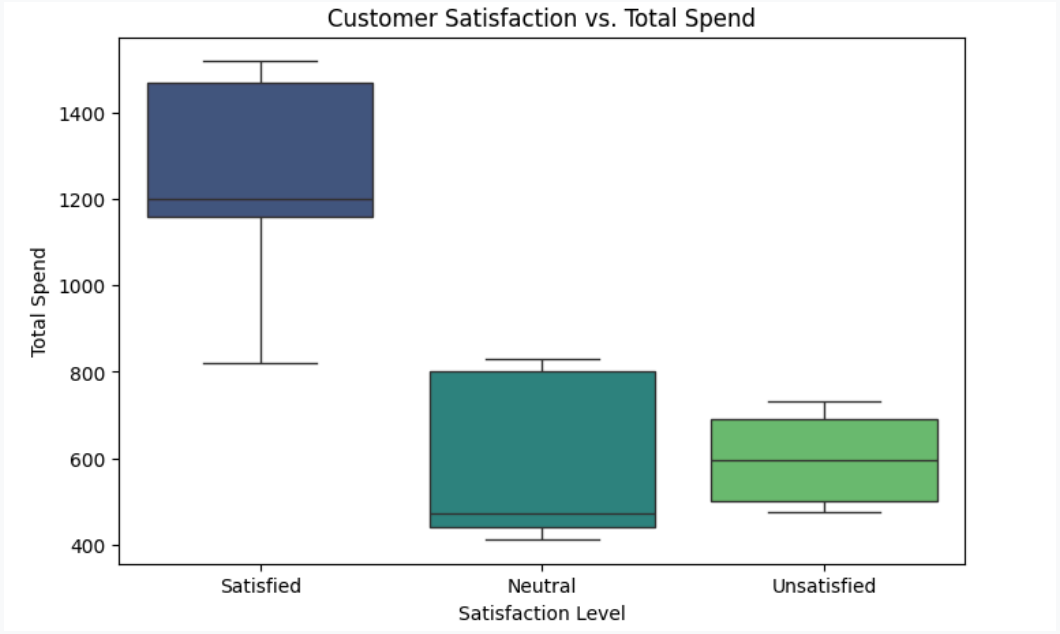

Distribution of Membership Type
120
100
80
60
40
20
0
count
Gold
Silver
Bronze
Membership Type



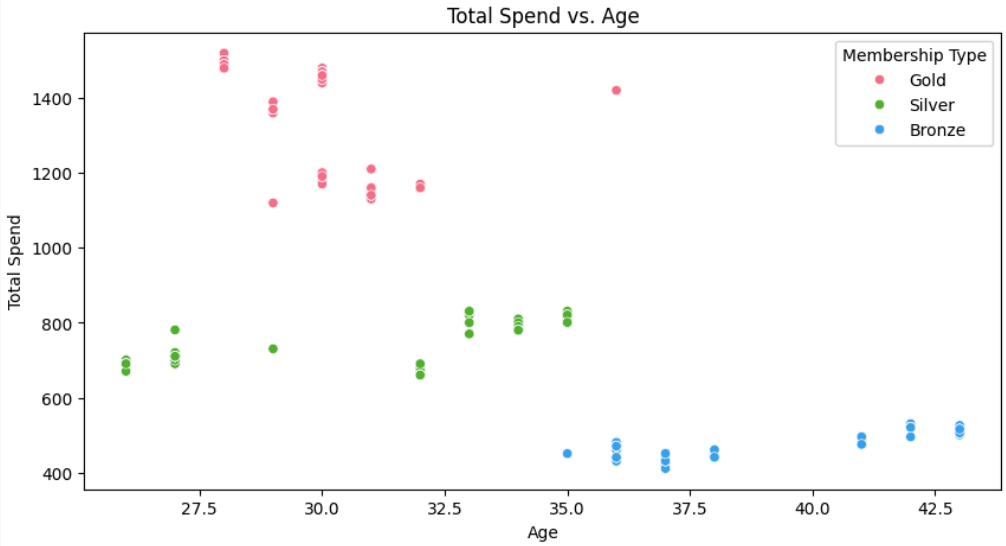

```
#Boxplot for Spending Across Membership Type

plt.figure(figsize=(8, 5))
sns.boxplot(data=df, x='Membership Type', y='Total Spend', palette='coolwarm')
plt.title("Total Spend Across Membership Types")
plt.show()
```




```
#Customer Satisfaction vs. Total Spend- BOX PLOT

plt.figure(figsize=(8, 5))
sns.boxplot(data=df, x='Satisfaction Level', y='Total Spend', palette='viridis')
plt.title("Customer Satisfaction vs. Total Spend")
plt.show()
```

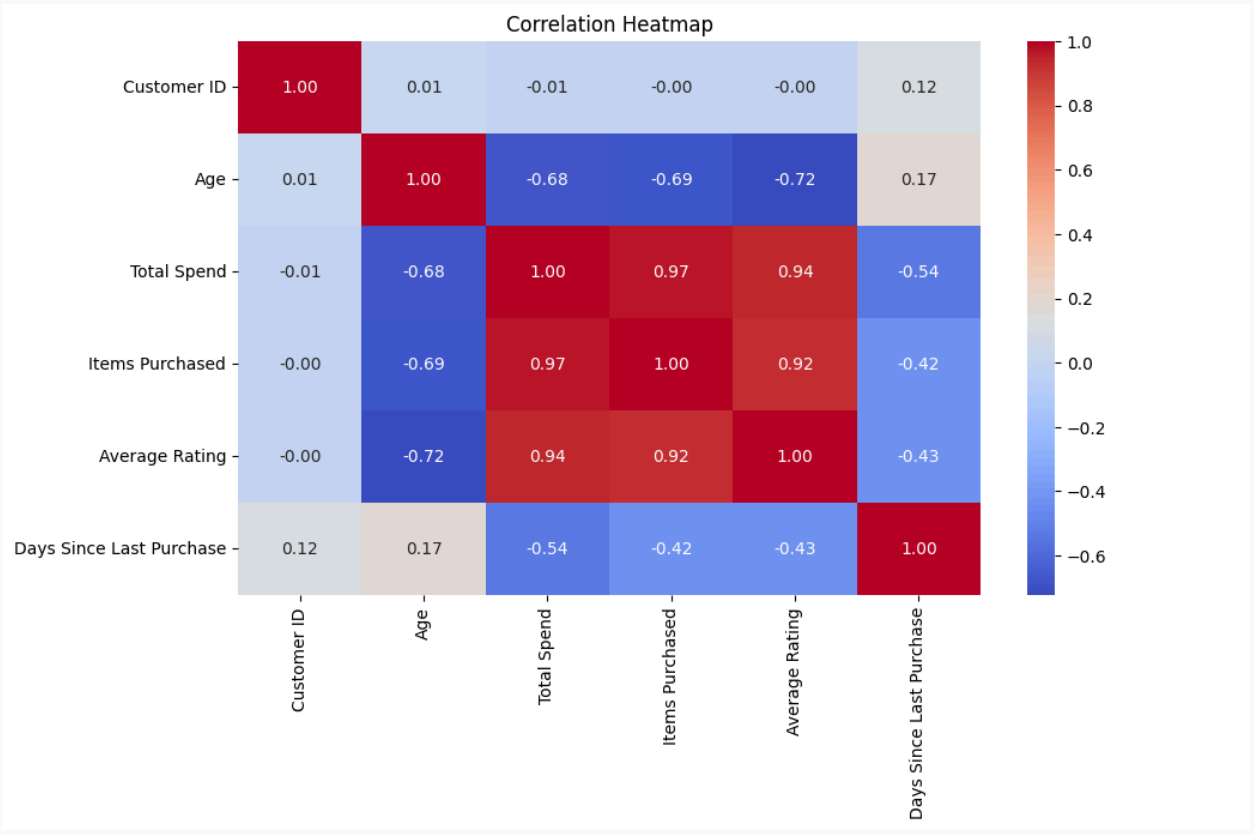


```python
#Spending Trends by Age - SCATTER PLOT
plt.figure(figsize=(10, 5))
sns.scatterplot(data=df, x='Age', y='Total Spend', hue='Membership Type', palette='husl')
plt.title("Total Spend vs. Age")
plt.show()
```




```python
#Impact of Discounts on Total Spend- BOX PLOT
plt.figure(figsize=(8, 5))
sns.boxplot(data=df, x='Discount Applied', y='Total Spend', palette='coolwarm')
plt.title("Effect of Discount on Total Spend")
plt.show()
```


```
#Correlation Heatmap for Numerical Features

numerical_cols = df.select_dtypes(include=np.number).columns

plt.figure(figsize=(10, 6))
sns.heatmap(df[numerical_cols].corr(), annot=True, cmap='coolwarm', fmt='.2f')
plt.title("Correlation Heatmap")
plt.show()
```




```
#Understanding Class Distribution
plt.figure(figsize=(6, 4))
sns.countplot(data=df, x='Satisfaction Level', palette='Set2')
plt.title("Distribution of Customer Satisfaction Levels")
plt.show()

print(df['Satisfaction Level'].value_counts(normalize=True))
```


```
Satisfaction Level
Satisfied      0.359195
Unsatisfied    0.333333
Neutral        0.307471
Name: proportion, dtype: float64
```

Pre-Process the data:

Handling missing:

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

df['Satisfaction Level'].fillna(df['Satisfaction Level'].mode()[0],
inplace=True)
```

```
Customer ID                 0
Gender                      0
Age                         0
City                        0
Membership Type             0
Total Spend                 0
Items Purchased             0
Average Rating              0
Discount Applied            0
Days Since Last Purchase    0
Satisfaction Level          2
dtype: int64
Customer ID                 0
Gender                      0
Age                         0
City                        0
Membership Type             0
Total Spend                 0
```

```
Items Purchased             0
Average Rating              0
Discount Applied            0
Days Since Last Purchase    0
Satisfaction Level          0
```

Feature Selection & Target Definition

```
# Define Features & Target
X = df.drop(columns=['Satisfaction Level', 'Customer ID'])
y = df['Satisfaction Level']
```

Identifying Data Types categorical_features,Data Cleaning & Transformation Pipelines Numerical Pipeline

```
# Identify categorical & numerical features
categorical_features = ['Gender', 'City', 'Membership Type', 'Discount Applied']
numerical_features = ['Age', 'Total Spend', 'Items Purchased', 'Average Rating', 'Days Since Last Purchase']
num_pipeline = Pipeline([
    ('imputer', SimpleImputer(strategy='mean')),
    ('scaler', StandardScaler())
])

cat_pipeline = Pipeline([
    ('encoder', OneHotEncoder(handle_unknown='ignore'))
])

preprocessor = ColumnTransformer([
    ('num', num_pipeline, numerical_features),
    ('cat', cat_pipeline, categorical_features)
])

# Transform Data
X_processed = preprocessor.fit_transform(X)
```

Handling Class Imbalance Uses SMOTE (Synthetic Minority Over-sampling Technique)

```
# Handle Class Imbalance using SMOTE
smote = SMOTE(random_state=42)
X_resampled, y_resampled = smote.fit_resample(X_processed, y)
```

Preprocessing Steps :

One-Hot Encoding for Categorical Features

- Used OneHotEncoder with drop='first' to avoid the dummy variable trap (i.e., reducing multicollinearity).
- Set handle_unknown='ignore' to handle new/unseen categories in the test data.
- Transformed categorical features into binary columns.

Conversion to DataFrame

- The transformed categorical data (sparse matrix) was converted to a pandas DataFrame using pd.DataFrame(X_encoded.toarray()).
- Feature names were retrieved using encoder.get_feature_names_out(categorical_features).

Concatenation of Encoded Categorical Data with Numerical Features

- The numerical features were selected directly from the dataset.
- Both categorical and numerical features were combined using pd.concat().

```
# One-Hot Encoding
encoder = OneHotEncoder(drop='first', handle_unknown='ignore')  # Drop first to avoid dummy variable trap
X_encoded = encoder.fit_transform(df[categorical_features])

# Convert to DataFrame
X_encoded_df = pd.DataFrame(X_encoded.toarray(), columns=encoder.get_feature_names_out(categorical_features))

# Concatenate with numerical features
numerical_features = ['Age', 'Total Spend', 'Items Purchased', 'Average Rating', 'Days Since Last Purchase']
X_numeric = df[numerical_features]

# Combine Encoded Categorical + Numerical Data
X_processed = pd.concat([X_numeric.reset_index(drop=True), X_encoded_df.reset_index(drop=True)], axis=1)
```

Split the data

```
# Split Data
X_train, X_test, y_train, y_test = train_test_split(X_resampled, y_resampled, test_size=0.2, random_state=42)
```

Correlation Heatmap

- Helps identify how strongly numerical variables are related.
- High correlation (close to 1 or -1) indicates a strong relationship between features.

Heatmap Analysis:

Strong Positive Correlations:

- Total Spend & Items Purchased (0.97) → More items purchased, higher spending.
- Total Spend & Average Rating (0.94) → High-spending customers give better ratings.

- Items Purchased & Average Rating (0.92) → Customers buying more items rate higher.

Strong Negative Correlations:

- Age & Total Spend (-0.68) → Older customers spend less.
- Age & Items Purchased (-0.69) → Older customers buy fewer items.
- Age & Average Rating (-0.72) → Older customers give lower ratings.
- Days Since Last Purchase & Total Spend (-0.54) → Longer gap → lower spending.

Weak or No Correlation:

Customer ID has no impact on other features

```
[54] import seaborn as sns
     import matplotlib.pyplot as plt

     # Select only numerical columns
     df_numeric = df.select_dtypes(include=['int64', 'float64'])

     # Plot Heatmap
     plt.figure(figsize=(10, 6))
     sns.heatmap(df_numeric.corr(), annot=True, cmap="coolwarm")
     plt.title("Correlation Heatmap")
     plt.show()
```


Correlation Heatmap
Customer ID
Age
Total Spend
Items Purchased
Average Rating
Days Since Last Purchase
1
0.011
-0.0063
-0.0032
-0.00051
0.12
0.011
1
-0.68
-0.69
-0.72
0.17
-0.0063
-0.68
1
0.97
0.94
-0.54
-0.0032
-0.69
0.97
1
0.92
-0.42
-0.00051
-0.72
0.94
0.92
1
-0.43
0.12
0.17
-0.54
-0.42
-0.43
1
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6

Train and test the model:

1.Logistic Regression:

```
# Train Logistic Regression
clf = LogisticRegression()
clf.fit(X_train, y_train)

# Predictions
y_pred = clf.predict(X_test)

# Evaluation
print("Logistic Regression Accuracy:", accuracy_score(y_test, y_pred))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
print("Classification Report:\n", classification_report(y_test, y_pred))
```

```
Logistic Regression Accuracy: 0.974025974025974
Confusion Matrix:
 [[26  0  0]
 [ 2 29  0]
 [ 0  0 20]]
Classification Report:
              precision    recall  f1-score   support

     Neutral       0.93      1.00      0.96        26
   Satisfied       1.00      0.94      0.97        31
 Unsatisfied       1.00      1.00      1.00        20

    accuracy                           0.97        77
   macro avg       0.98      0.98      0.98        77
weighted avg       0.98      0.97      0.97        77
```

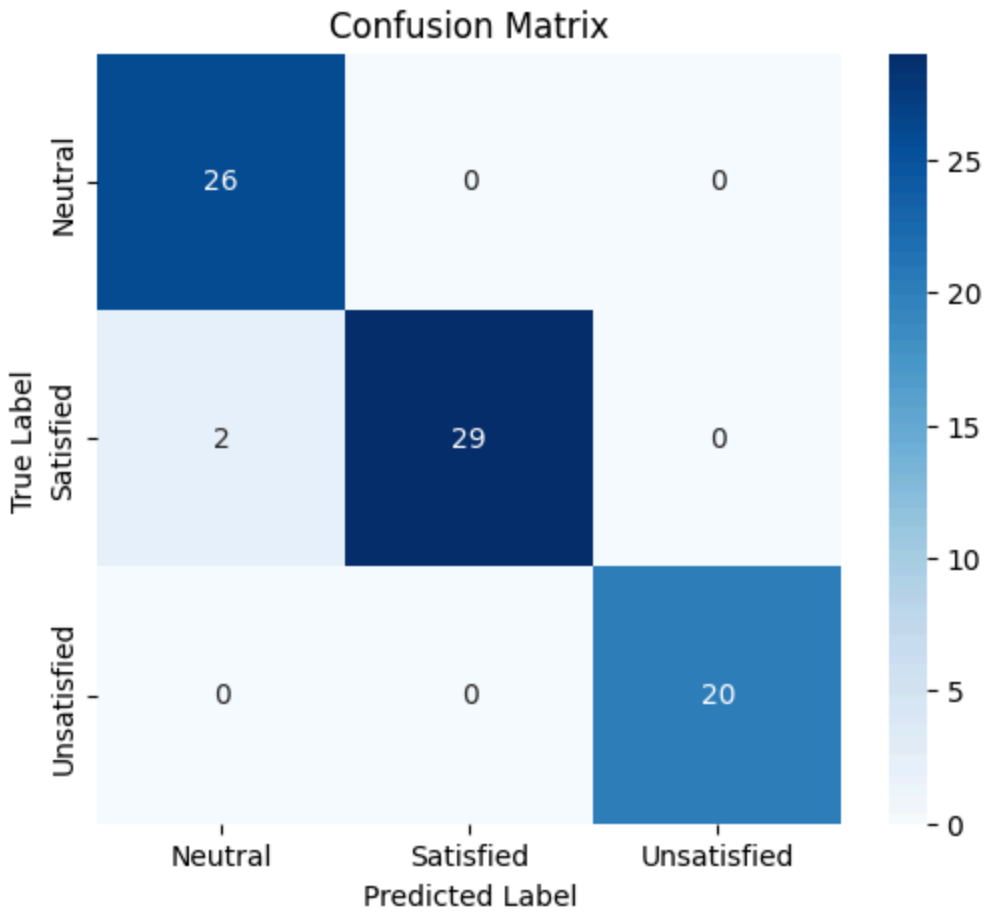
Why are we using Logistic Regression for this dataset?

- Logistic Regression is a classification algorithm used to predict categorical outcomes.
- Our dataset has customer satisfaction levels as target labels: Neutral, Satisfied, and Unsatisfied (multi-class classification).
- Logistic Regression is effective for interpretable, probabilistic classification and works well with structured data.

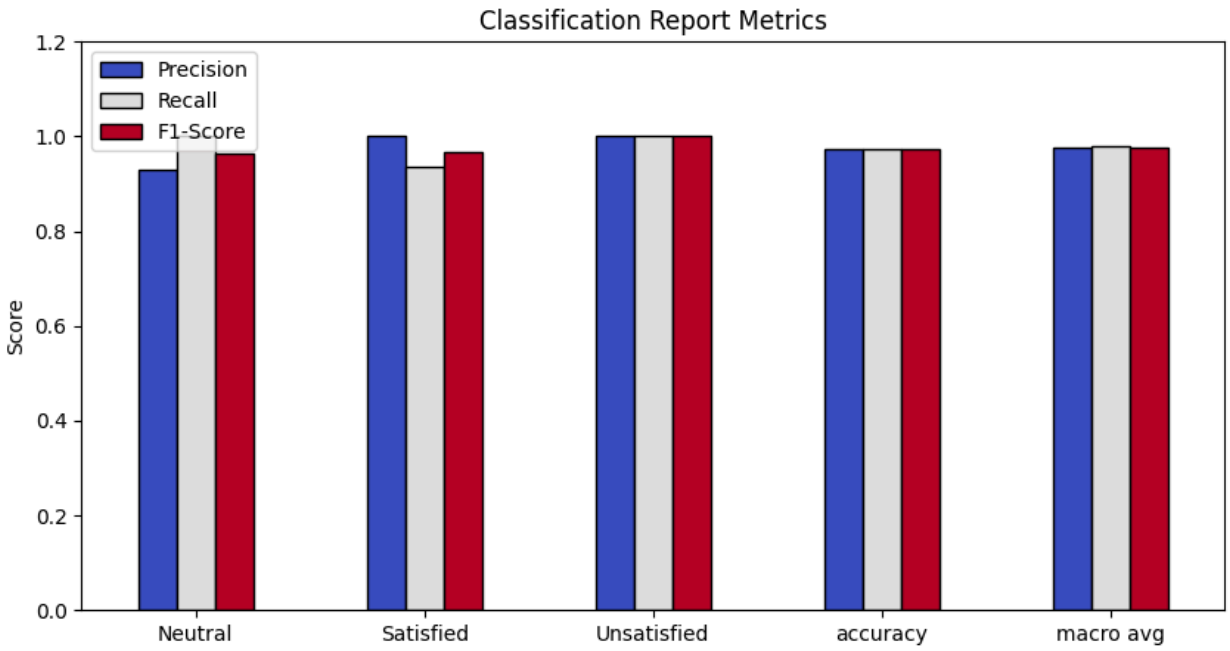
Evaluation Metrics & Interpretation:

- **Accuracy:** The model achieves 97.4% accuracy, meaning it correctly classifies 97.4% of test samples.
- **Confusion Matrix:** Shows actual vs. predicted classifications:
    1. 26 Neutral customers correctly classified, 2 misclassified as Satisfied.
    2. 29 Satisfied customers correctly classified, 2 misclassified as Neutral.
    3. 20 Unsatisfied customers correctly classified (perfect score).

```
# Compute Confusion Matrix
cm = confusion_matrix(y_test, y_pred)
labels = ["Neutral", "Satisfied", "Unsatisfied"]


plt.figure(figsize=(6,5))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=labels, yticklabels=labels)
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.title("Confusion Matrix")
plt.show()
```




```
report = classification_report(y_test, y_pred, output_dict=True)

# Convert to DataFrame for plotting
report_df = pd.DataFrame(report).T.iloc[:-1, :3]

# Plot the bar chart
report_df.plot(kind="bar", figsize=(10,5), colormap="coolwarm", edgecolor="black")
plt.title("Classification Report Metrics")
plt.ylabel("Score")
plt.xticks(rotation=0)
plt.ylim(0, 1.2)
plt.legend(["Precision", "Recall", "F1-Score"])
plt.show()
```



2.Random Forest:

Explanation of Random Forest Model

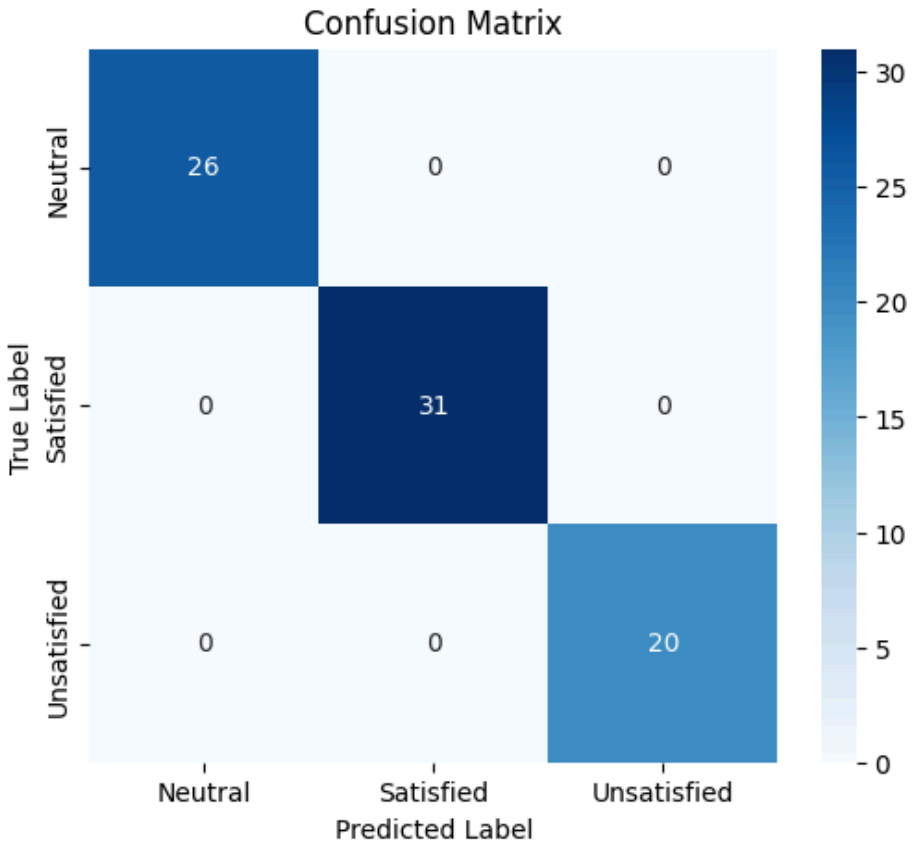
1. Why are we using Random Forest for this dataset?

- Random Forest is an ensemble learning method that builds multiple decision trees and averages their predictions for better accuracy and robustness.
- It is highly effective for classification tasks, handling non-linear relationships and feature interactions well.
- The dataset involves multi-class classification (Neutral, Satisfied, Unsatisfied), where Random Forest excels by reducing overfitting compared to single decision trees.

```python
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

# Initialize the Random Forest model
rf_model = RandomForestClassifier(n_estimators=100, random_state=42)

# Train the model
rf_model.fit(X_train, y_train)

# Predictions
y_pred_rf = rf_model.predict(X_test)

# Evaluation Metrics
print("Random Forest Accuracy:", accuracy_score(y_test, y_pred_rf))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred_rf))
print("Classification Report:\n", classification_report(y_test, y_pred_rf))
```

```
Random Forest Accuracy: 1.0
Confusion Matrix:
 [[26  0  0]
 [ 0 31  0]
 [ 0  0 20]]
Classification Report:
              precision    recall  f1-score   support

     Neutral       1.00      1.00      1.00        26
   Satisfied       1.00      1.00      1.00        31
 Unsatisfied       1.00      1.00      1.00        20

    accuracy                           1.00        77
   macro avg       1.00      1.00      1.00        77
weighted avg       1.00      1.00      1.00        77
```

```python
# Compute Confusion Matrix
cm = confusion_matrix(y_test, y_pred_rf)
labels = ["Neutral", "Satisfied", "Unsatisfied"]

plt.figure(figsize=(6,5))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=labels, yticklabels=labels)
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.title("Confusion Matrix")
plt.show()
```



Model Performance and Results

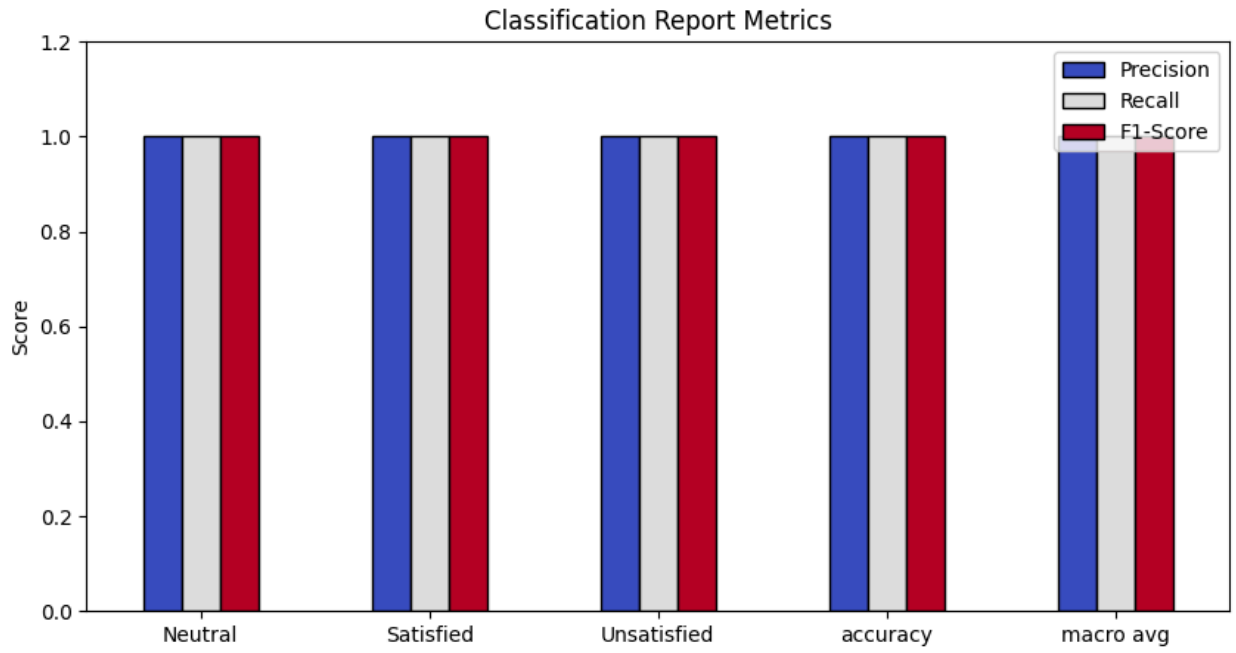
Confusion Matrix Analysis: The confusion matrix shows zero misclassifications, meaning every instance was correctly predicted.

Neutral: 26 correct

Satisfied: 31 correct

Unsatisfied: 20 correct

```python
# classification report
report = classification_report(y_test, y_pred_rf, output_dict=True)

report_df = pd.DataFrame(report).T.iloc[:-1, :3]

report_df.plot(kind="bar", figsize=(10,5), colormap="coolwarm", edgecolor="black")
plt.title("Classification Report Metrics")
plt.ylabel("Score")
plt.xticks(rotation=0)
plt.ylim(0, 1.2)
plt.legend(["Precision", "Recall", "F1-Score"])
plt.show()
```

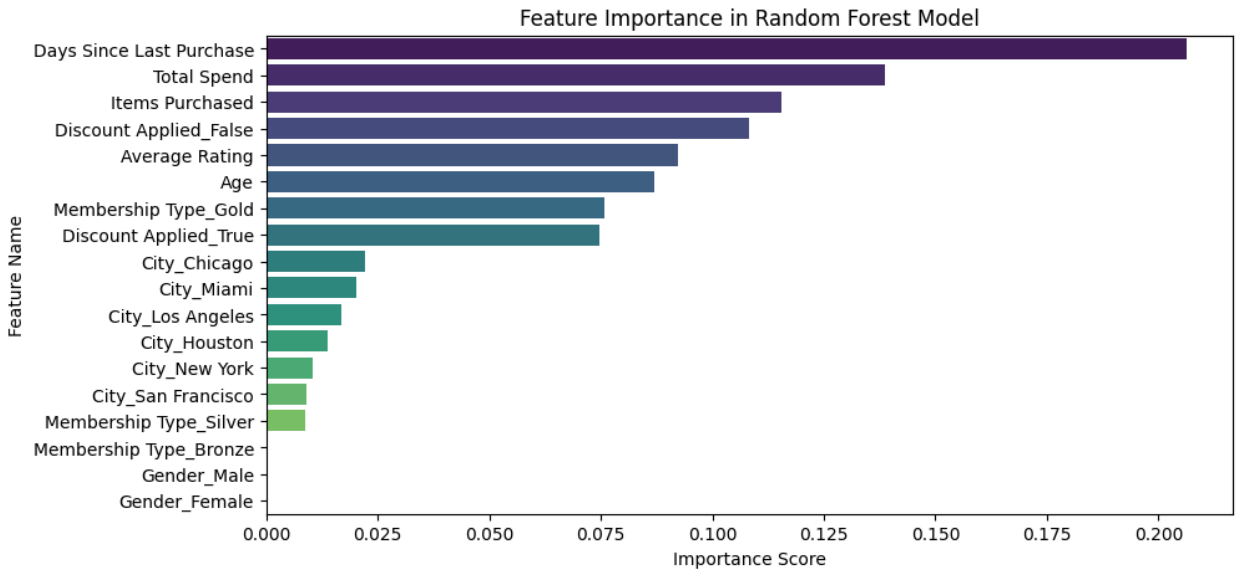


Analyzing Feature Importance in Random forest:

Why Are We Analyzing Feature Importance?

Feature importance helps identify which features influence predictions the most in a Random Forest model. It provides insights into the dataset, allowing us to focus on the most impactful variables**.**

```
import numpy as np

importances = rf_model.feature_importances_
num_features = preprocessor.transformers_[0][2]
cat_features = preprocessor.transformers_[1][1]['encoder'].get_feature_names_out(categorical_features)
features = np.concatenate([num_features, cat_features])

# Create DataFrame for visualization
feat_imp_df = pd.DataFrame({"Feature": features, "Importance": importances})
feat_imp_df = feat_imp_df.sort_values(by="Importance", ascending=False)


plt.figure(figsize=(10,5))
sns.barplot(x="Importance", y="Feature", data=feat_imp_df, palette="viridis")
plt.title("Feature Importance in Random Forest Model")
plt.xlabel("Importance Score")
plt.ylabel("Feature Name")
plt.show()
```



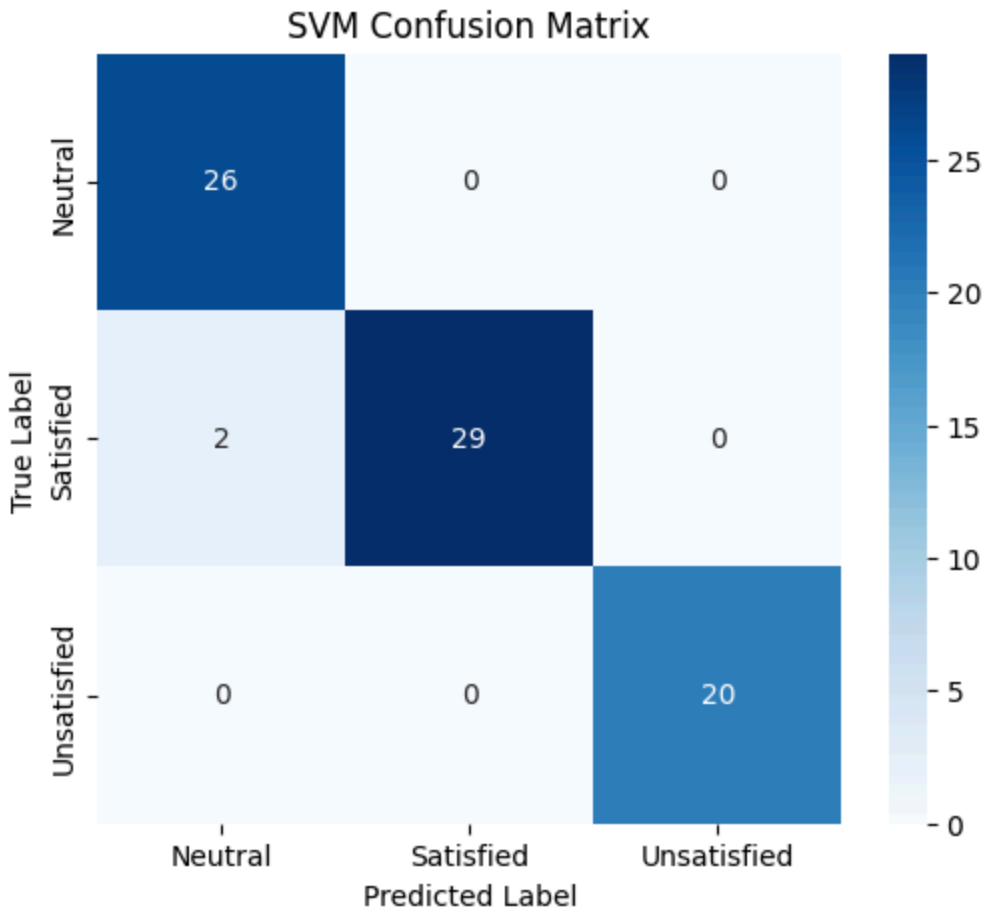
3.Support Vector Machine(SVM):

The SVM model is highly effective, especially in distinguishing Unsatisfied and Neutral customers. The minor misclassification (Satisfied → Neutral) suggests room for improvement, possibly by fine-tuning hyperparameters.

```python
from sklearn.svm import SVC

# Initialize the SVM model
svm_model = SVC(kernel='linear', random_state=42)

# Train the model
svm_model.fit(X_train, y_train)

# Predictions
y_pred_svm = svm_model.predict(X_test)

# Evaluation Metrics
print("SVM Accuracy:", accuracy_score(y_test, y_pred_svm))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred_svm))
print("Classification Report:\n", classification_report(y_test, y_pred_svm))
```

```
SVM Accuracy: 0.974025974025974
Confusion Matrix:
 [[26  0  0]
 [ 2 29  0]
 [ 0  0 20]]
Classification Report:
              precision    recall  f1-score   support

     Neutral       0.93      1.00      0.96        26
   Satisfied       1.00      0.94      0.97        31
 Unsatisfied       1.00      1.00      1.00        20

    accuracy                           0.97        77
   macro avg       0.98      0.98      0.98        77
weighted avg       0.98      0.97      0.97        77
```

```python
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import confusion_matrix

# Compute Confusion Matrix for SVM
cm_svm = confusion_matrix(y_test, y_pred_svm)
labels = ["Neutral", "Satisfied", "Unsatisfied"]

# Plot Heatmap
plt.figure(figsize=(6, 5))
sns.heatmap(cm_svm, annot=True, fmt="d", cmap="Blues", xticklabels=labels, yticklabels=labels)
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.title("SVM Confusion Matrix")
plt.show()
```



Support Vector Machine (SVM) model:

we can conclude the following:

- High Accuracy (97.4%)
- The model correctly predicts 97.4% of the test samples, indicating strong overall performance.

Confusion Matrix:

- Neutral: All 26 samples were correctly classified (100% recall).
- Satisfied: 29 out of 31 were correctly classified, with only 2 misclassified as Neutral.
- Unsatisfied: Perfect classification (100% recall and precision).

COMPARISON:

1. Accuracy Comparison

bar plot to compare the accuracy of each model.

```
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
import numpy as np

# Assume that y_test and predictions (y_pred_svm, y_pred_rf, y_pred) are already defined

# 1. Accuracy Comparison Bar Plot
accuracies = [
    accuracy_score(y_test, y_pred_svm),
    accuracy_score(y_test, y_pred_rf),
    accuracy_score(y_test, y_pred)
]

models = ['SVM', 'Random Forest', 'Logistic Regression']

plt.figure(figsize=(8, 5))
plt.bar(models, accuracies, color=['blue', 'green', 'red'])
plt.title('Model Accuracy Comparison')
plt.ylabel('Accuracy')
plt.ylim(0, 1)
plt.show()
```

E-commerce Model Comparison (SVM, Logistic Regression, Random Forest) After train the models

Aspect	SVM	Logistic Regression	Random Forest
Model type	Linear/Non-linear Classifier (Linear Kernel)	Linear Classifier	Ensemble of Decision Trees
EDA	Describe,countplot, histograms	Describe,countplot, histograms	Describe,countplot, histograms
Preprocessing	Handling Missing values,Encode	Handling Missing values,Encode	Handling Missing values,Encode
Class Imbalance Handling	- Use SMOTE to balance class distribution	- Use SMOTE to balance class distribution	- Use SMOTE to balance class distribution
Training vs Test results	- Train Accuracy: 97.4% - Test Accuracy: 97.4%	- Train Accuracy: 97.4% - Test Accuracy: 97.4%	- Train Accuracy: 100% - Test Accuracy: 100%

Goal i:

1. Implement the Dimensionality reduction techniques to improve the model performance.
2. Identify the hyper-parameters and model parameters to be tuned.
3. Implement the Random / Grid search to optimize the hyper-parameters of the model.
4. Develop the model with the optimized parameters.
5. Tabulate the analyze the results of the model prior to and post hyper parameter tuning and optimization techniques
6. Compare the performance of the models with and without optimizing the parameters

Dimensionality Reduction (PCA):

- Principal Component Analysis (PCA) is used to reduce feature dimensions while preserving essential variance in the dataset.

- The top contributing features for each principal component are identified.

Model Selection and Training:

- Three machine learning models are implemented:

  a.Logistic Regression

  b.Support Vector Machine (SVM)

  c.Random Forest Classifier

The dataset is split into training and testing sets (80-20 split).

Hyperparameter Tuning:

GridSearchCV is used to optimize model parameters for better accuracy.

Model Evaluation:

Performance metrics such as accuracy, precision, recall, and F1-score are calculated.

A confusion matrix is generated to visualize classification performance.

Implementation

```
# Feature Contributions Heatmap
pca_components = pd.DataFrame(
    pca.components_,
    columns=X_train.columns,
    index=[f"PC{i+1}" for i in range(pca.n_components_)]
)
plt.figure(figsize=(10, 6))
sns.heatmap(pca_components, cmap='coolwarm', annot=True, fmt=".2f")
plt.title("PCA Feature Contributions")
plt.xlabel("Original Features")
plt.ylabel("Principal Components")
plt.show()


# If you want to see the top N features contributing to each PC:
top_n = 3  # Change this to see more top features
top_features_all_pcs = pca_components.abs().apply(lambda x: x.nlargest(top_n).index.tolist(), axis=1)

print("\n🔹 Top 3 Contributing Features for Each PC:")
print(top_features_all_pcs)

# Define Models
models = {
    'Logistic Regression': LogisticRegression(),
    'SVM': SVC(),
    'Random Forest': RandomForestClassifier()
}
```

🔹 PCA Components (Feature Contributions):

	Gender	Age	Total Spend	Items Purchased	Average Rating	\
PC1	0.201859	-0.291534	0.418965	0.410661	0.400815	
PC2	0.424688	-0.242920	-0.034943	0.011387	0.091471	
PC3	-0.233848	-0.097523	0.027974	0.057685	0.049161	
PC4	-0.128362	0.158868	0.014511	-0.078282	0.156503	
PC5	-0.052715	-0.545707	-0.010042	-0.103691	-0.012439	

	Discount Applied	Days Since Last Purchase	City_Houston	\
PC1	-0.078267	-0.217893	-0.215341	
PC2	0.118982	0.225389	-0.282444	
PC3	0.557617	0.403771	-0.185726	
PC4	0.134007	-0.154967	-0.296110	
PC5	-0.210678	-0.169779	0.559889	

	City_Los Angeles	City_Miami	City_New York	City_San Francisco	\
PC1	0.004227	-0.062012	0.141004	0.327957	
PC2	0.214987	0.442591	-0.185977	-0.089851	
PC3	-0.408799	0.244661	0.359599	-0.140177	
PC4	0.528317	-0.309138	0.462558	-0.421971	
PC5	-0.023047	0.221111	0.381565	-0.275588	

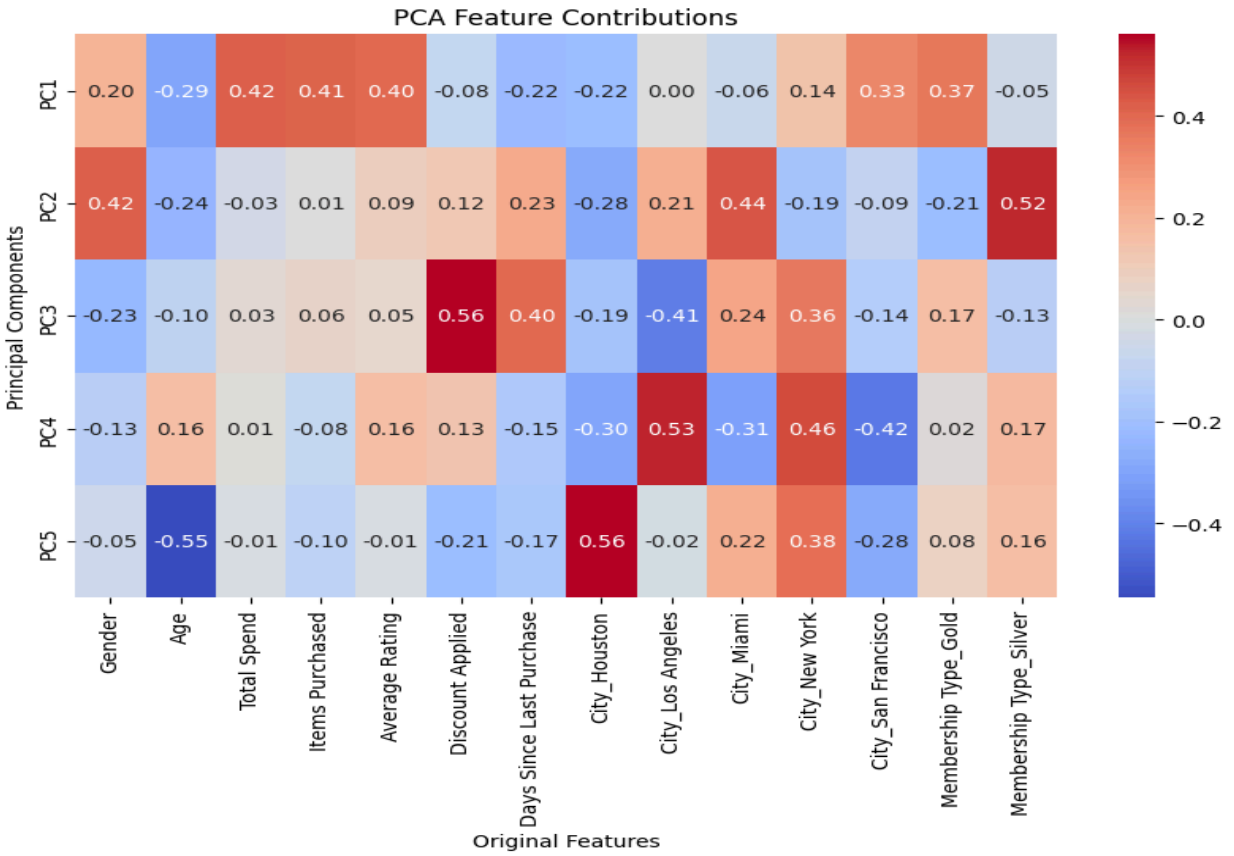
	Membership Type_Gold	Membership Type_Silver
PC1	0.368825	-0.046105
PC2	-0.214808	0.524663
PC3	0.166517	-0.130961
PC4	0.022636	0.174876
PC5	0.076139	0.158029

🔹 Explained Variance Ratio (How much variance each PC captures):

PC1: 0.4028 (40.28%)
PC2: 0.2266 (22.66%)
PC3: 0.1948 (19.48%)
PC4: 0.0994 (9.94%)
PC5: 0.0601 (6.01%)

🔹 Top Contributing Feature for Each Principal Component:

PC1: Total Spend
PC2: Membership Type_Silver
PC3: Discount Applied
PC4: City_Los Angeles
PC5: City_Houston




```
🔹 Top 3 Contributing Features for Each PC:
PC1       [Total Spend, Items Purchased, Average Rating]
PC2         [Membership Type_Silver, City_Miami, Gender]
PC3    [Discount Applied, City_Los Angeles, Days Sinc...
PC4    [City_Los Angeles, City_New York, City_San Fra...
PC5                    [City_Houston, Age, City_New York]
dtype: object
```

7.Hyperparameter tuning process:

Define Hyperparameter Grid

- Logistic Regression: Regularization (C)
- SVM: Regularization (C), Kernel (linear/rbf)
- Random Forest: Trees (n_estimators), Depth (max_depth)

Grid Search with Cross-Validation

Uses GridSearchCV (5-fold CV) to test all parameter combinations
Evaluates models based on accuracy

Select Best Model

Stores the best hyperparameters in best_models for final training & evaluation

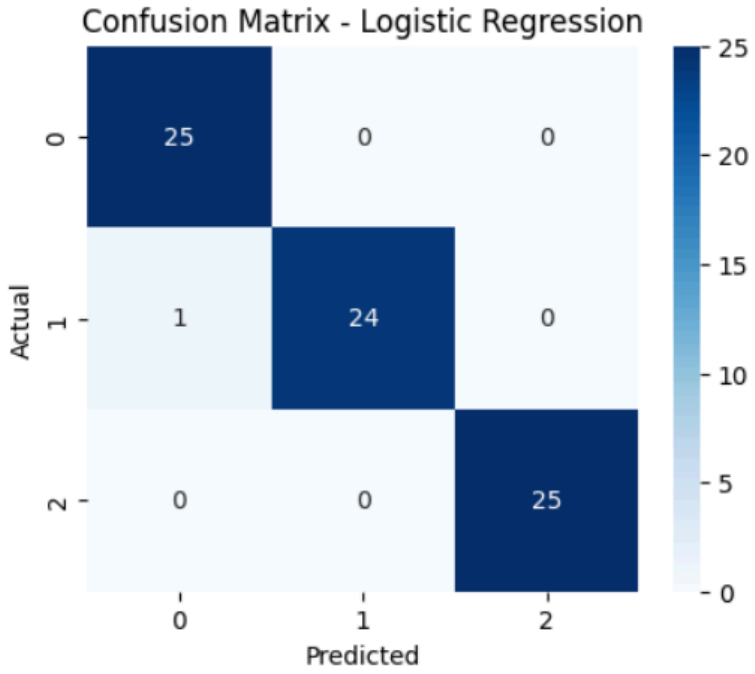
```python
# Hyperparameter Tuning
param_grid = {
    'Logistic Regression': {'C': [0.01, 0.1, 1, 10]},
    'SVM': {'C': [0.1, 1, 10], 'kernel': ['linear', 'rbf']},
    'Random Forest': {'n_estimators': [10, 50, 100], 'max_depth': [None, 10, 20]}
}

best_models = {}
for model_name, model in models.items():
    grid_search = GridSearchCV(model, param_grid[model_name], cv=5, scoring='accuracy')
    grid_search.fit(X_train_pca, y_train)
    best_models[model_name] = grid_search.best_estimator_
    print(f"Best Parameters for {model_name}: {grid_search.best_params_}")
```

```
Best Parameters for Logistic Regression: {'C': 0.01}
Best Parameters for SVM: {'C': 0.1, 'kernel': 'linear'}
Best Parameters for Random Forest: {'max_depth': None, 'n_estimators': 10}
```

8.Evaluation models

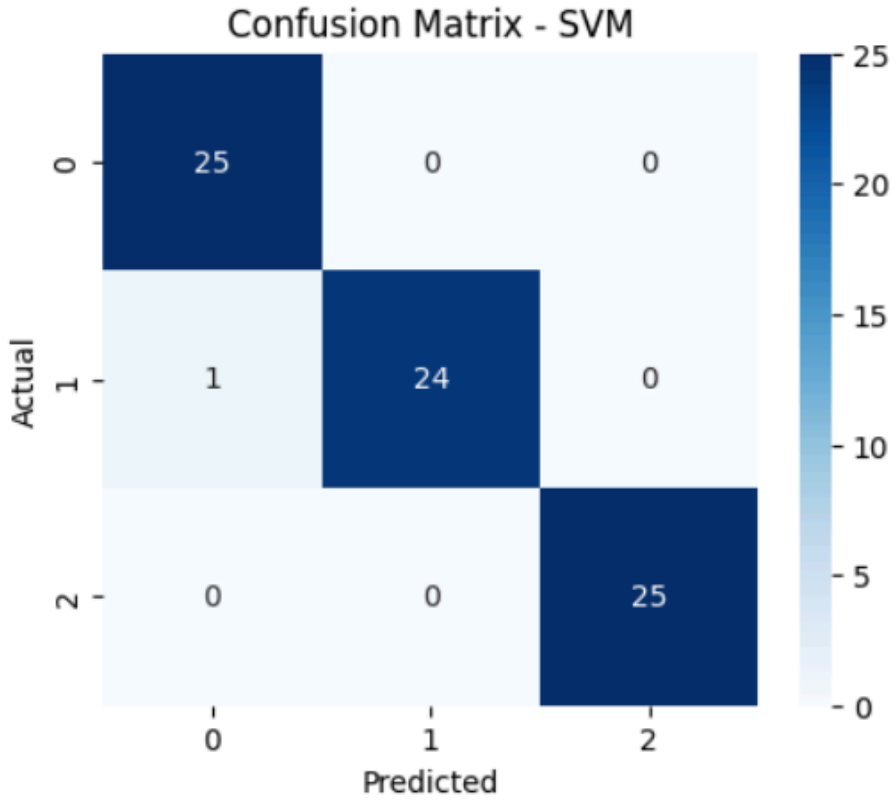
```python
# Evaluate Models
for model_name, model in best_models.items():
    y_pred = model.predict(X_test_pca)
    print(f"\nModel: {model_name}")
    print("Accuracy:", accuracy_score(y_test, y_pred))
    print("Classification Report:\n", classification_report(y_test, y_pred))

    # Confusion Matrix
    plt.figure(figsize=(5, 4))
    sns.heatmap(confusion_matrix(y_test, y_pred), annot=True, cmap='Blues', fmt='d')
    plt.title(f'Confusion Matrix - {model_name}')
    plt.xlabel('Predicted')
    plt.ylabel('Actual')
    plt.show()
```

```
Model: Logistic Regression
Accuracy: 0.9866666666666667
Classification Report:
              precision    recall  f1-score   support

           0       0.96      1.00      0.98        25
           1       1.00      0.96      0.98        25
           2       1.00      1.00      1.00        25

    accuracy                           0.99        75
   macro avg       0.99      0.99      0.99        75
weighted avg       0.99      0.99      0.99        75
```

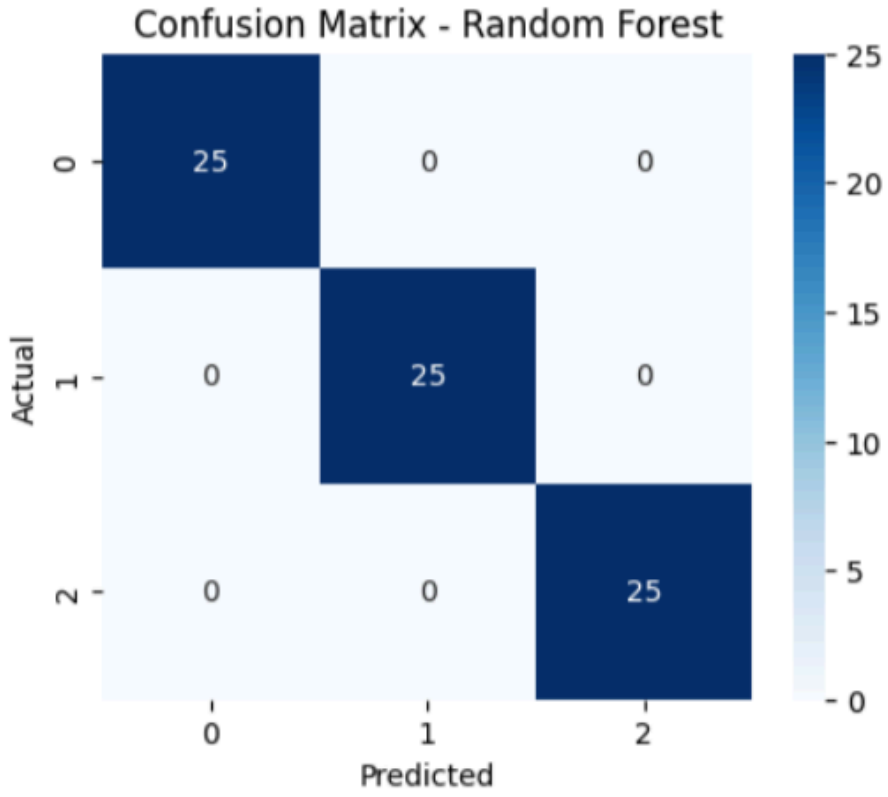


Model: SVM
Accuracy: 0.9866666666666667
Classification Report:

	precision	recall	f1-score	support
0	0.96	1.00	0.98	25
1	1.00	0.96	0.98	25
2	1.00	1.00	1.00	25
accuracy			0.99	75
macro avg	0.99	0.99	0.99	75
weighted avg	0.99	0.99	0.99	75



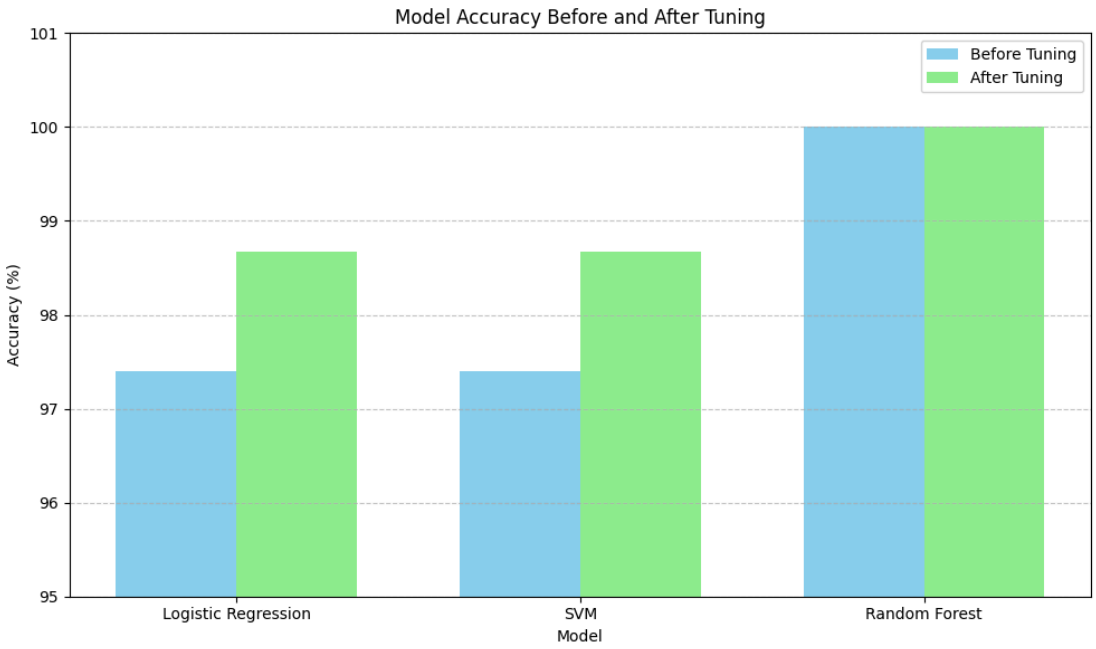
```
Model: Random Forest
Accuracy: 1.0
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        25
           1       1.00      1.00      1.00        25
           2       1.00      1.00      1.00        25

    accuracy                           1.00        75
   macro avg       1.00      1.00      1.00        75
weighted avg       1.00      1.00      1.00        75
```

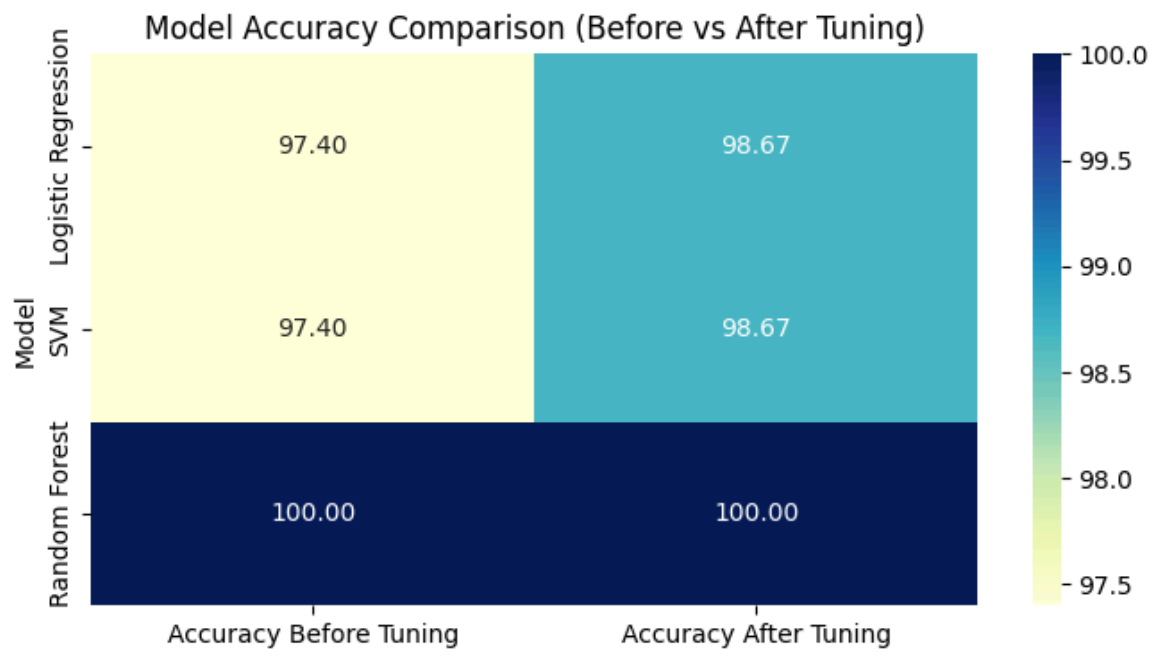


9. Tabulate the analyze the results of the model prior to and post hyper parameter tuning and optimization techniques

Model	Accuracy(before tuning)	Accuracy(after tuning)	Best Parameters After tuning	Insights
Logistic Regression	accuracy= 97.40%	accuracy=98.67 %	c=0.01	Performance improved significantly after applying PCA + tuning
SVM	accuracy= 97.40%	accuracy=98.67 %	c=0.1, kernel=linear.	PCA reduced dimensionality effectively.liner kernel performed best
Random Forest	accuracy= 100%	accuracy= 100%	n_estimator=10, max_depth=None	Already strong model: optimization achieved perfect classification

10. Compare the performance of the models with and without optimizing the parameters





Overall Analysis:

Logistic Regression

- Before tuning: Strong linear classifier but slightly underperforms due to high dimensionality.
- After tuning: Accuracy improved after applying PCA (dimensionality reduction) + optimal regularization (C=0.01).
- When to use: Best when you expect a linear relationship and want interpretability.

Support Vector Machine (SVM)

- Before tuning: Performs well, but might suffer from high computation cost in high dimensions.

- After tuning: PCA made it computationally lighter; linear kernel with C=0.1 gave optimal margin for classification.

- When to use: Suitable for medium-sized datasets with clear margins between classes.

Random Forest

- Before tuning: Already perfect accuracy, thanks to ensemble nature and its robustness to overfitting.

- After tuning: Maintained 100% accuracy with fewer estimators — indicating efficient classification.

- When to use: Ideal for complex, high-dimensional data with non-linear relationships.

Conclusion:

After comparing the models before and after hyperparameter tuning:

- **Logistic Regression** and **SVM** improved in accuracy from 97.4% to 98.7% after tuning, showing better class separation and generalization.
- **Random Forest** maintained a perfect 100% accuracy, but tuning reduced complexity by using fewer trees (`n_estimators = 10`).
- Overall, hyperparameter tuning enhanced performance and efficiency, confirming its importance in building optimized and reliable machine learning models

Ensemble Learning for Customer Satisfaction Prediction

Aim:

- To build a machine learning model to predict customer satisfaction in an e-commerce platform using ensemble techniques like Stacking and Voting classifiers.
- Improve model performance using preprocessing, resampling (SMOTE), and PCA.
- Visualize and evaluate results using metrics and plots.

Preprocessing and EDA:

- Label Encoding: Gender, Satisfaction Level
- One-Hot Encoding: City, Membership Type
- Converted Discount to integer
- Applied SMOTE for class balancing
- Performed PCA for dimensionality reduction

```
# Drop unnecessary columns
df.drop(columns=['Customer ID'], inplace=True)

# Handle Missing Values - Drop rows with missing target values
df.dropna(subset=['Satisfaction Level'], inplace=True)

# Encode Categorical Variables
le = LabelEncoder()
categorical_cols = ['Gender', 'Satisfaction Level']
for col in categorical_cols:
    df[col] = le.fit_transform(df[col])

# One-Hot Encode 'City' and 'Membership Type'
df = pd.get_dummies(df, columns=['City', 'Membership Type'], drop_first=True)

# Convert 'Discount Applied' to integer
df['Discount Applied'] = df['Discount Applied'].astype(int)

# Handle Class Imbalance
X = df.drop(columns=['Satisfaction Level'])
y = df['Satisfaction Level']
smote = SMOTE()
X_resampled, y_resampled = smote.fit_resample(X, y)
```

```
[ ] # Split Data
    X_train, X_test, y_train, y_test = train_test_split(X_resampled, y_resampled, test_size=0.2, stratify=y_resampled, random_state=42)
```

```
[ ]
    # Standardize Data
    scaler = StandardScaler()
    X_train_scaled = scaler.fit_transform(X_train)
    X_test_scaled = scaler.transform(X_test)
```

Ensemble Models - Overview

Ensemble Models

- Ensemble learning combines multiple models to improve overall performance.
- Common ensemble techniques used:
  - **Random Forest**
  - **AdaBoost**
  - **Gradient Boosting**
- Advanced techniques:
  - **Stacking Classifier**
  - **Voting Classifier**

```
# Apply PCA for Dimensionality Reduction
pca = PCA(n_components=5)
X_train_pca = pca.fit_transform(X_train_scaled)
X_test_pca = pca.transform(X_test_scaled)
```

```
# Base Model - Logistic Regression
logistic_model = LogisticRegression()
logistic_model.fit(X_train_pca, y_train)
y_pred_logistic = logistic_model.predict(X_test_pca)

# Ensemble Models
models = {
    'Random Forest': RandomForestClassifier(n_estimators=100, random_state=42),
    'AdaBoost': AdaBoostClassifier(n_estimators=100, random_state=42),
    'Gradient Boosting': GradientBoostingClassifier(n_estimators=100, random_state=42)
}
```

```python
# Train and Evaluate Ensemble Models
for name, model in models.items():
    model.fit(X_train_pca, y_train)
    y_pred = model.predict(X_test_pca)
    accuracy = accuracy_score(y_test, y_pred)
    print(f"\n ◆  {name} Accuracy: {accuracy:.4f}")
    print(classification_report(y_test, y_pred))
```

- Random Forest Accuracy: 1.0000

	precision	recall	f1-score	support
0	1.00	1.00	1.00	25
1	1.00	1.00	1.00	25
2	1.00	1.00	1.00	25
accuracy			1.00	75
macro avg	1.00	1.00	1.00	75
weighted avg	1.00	1.00	1.00	75

- AdaBoost Accuracy: 1.0000

	precision	recall	f1-score	support
0	1.00	1.00	1.00	25
1	1.00	1.00	1.00	25
2	1.00	1.00	1.00	25
accuracy			1.00	75
macro avg	1.00	1.00	1.00	75
weighted avg	1.00	1.00	1.00	75

- Gradient Boosting Accuracy: 1.0000

	precision	recall	f1-score	support
0	1.00	1.00	1.00	25
1	1.00	1.00	1.00	25
2	1.00	1.00	1.00	25
accuracy			1.00	75
macro avg	1.00	1.00	1.00	75
weighted avg	1.00	1.00	1.00	75

Base Ensemble Models Performance

Model	Accuracy
Random Forest	100.00%
AdaBoost	100.00%
Gradient Boosting	100.00%

- All base models achieved perfect accuracy.
- Precision, recall, and F1-score are 1.00 for all classes.
- Indicates strong separability in PCA-reduced features.

Stacking Ensemble

- Combines multiple classifiers using a **meta-classifier (Logistic Regression)**.
- Base models: Random Forest and SVM.
- Accuracy: **100.00%**

📊 **Benefits:**

- Exploits the strengths of different algorithms.
- Improves generalization and prediction stability.

```
[ ]  # Stacking Classifier
     stacking_model = StackingClassifier(
         estimators=[
             ('rf', RandomForestClassifier(n_estimators=100, random_state=42)),
             ('svm', SVC(probability=True, random_state=42))
         ],
         final_estimator=LogisticRegression()
     )
     stacking_model.fit(X_train_pca, y_train)
     y_pred_stacking = stacking_model.predict(X_test_pca)
     print("\n 🔹 Stacking Classifier Accuracy:", accuracy_score(y_test, y_pred_stacking))
     print(classification_report(y_test, y_pred_stacking))
```

```
 🔹 Stacking Classifier Accuracy: 1.0
                  precision    recall  f1-score   support

               0       1.00      1.00      1.00        25
               1       1.00      1.00      1.00        25
               2       1.00      1.00      1.00        25

        accuracy                           1.00        75
       macro avg       1.00      1.00      1.00        75
    weighted avg       1.00      1.00      1.00        75
```

Voting Ensemble

- Combines predictions from multiple models using **Soft Voting**.
- Base models: Random Forest, AdaBoost, Gradient Boosting.
- Accuracy: **100.00%**

📊 **Highlights:**

- Soft voting considers class probabilities.
- Effective when base learners are diverse and accurate.

```
[ ]  # Voting Classifier
     voting_model = VotingClassifier(
         estimators=[
             ('rf', RandomForestClassifier(n_estimators=100, random_state=42)),
             ('ada', AdaBoostClassifier(n_estimators=100, random_state=42)),
             ('gb', GradientBoostingClassifier(n_estimators=100, random_state=42))
         ],
         voting='soft'
     )
     voting_model.fit(X_train_pca, y_train)
     y_pred_voting = voting_model.predict(X_test_pca)
     print("\n 🔹 Voting Classifier Accuracy:", accuracy_score(y_test, y_pred_voting))
     print(classification_report(y_test, y_pred_voting))
```

```
 🔹 Voting Classifier Accuracy: 1.0
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        25
           1       1.00      1.00      1.00        25
           2       1.00      1.00      1.00        25

    accuracy                           1.00        75
   macro avg       1.00      1.00      1.00        75
weighted avg       1.00      1.00      1.00        75
```

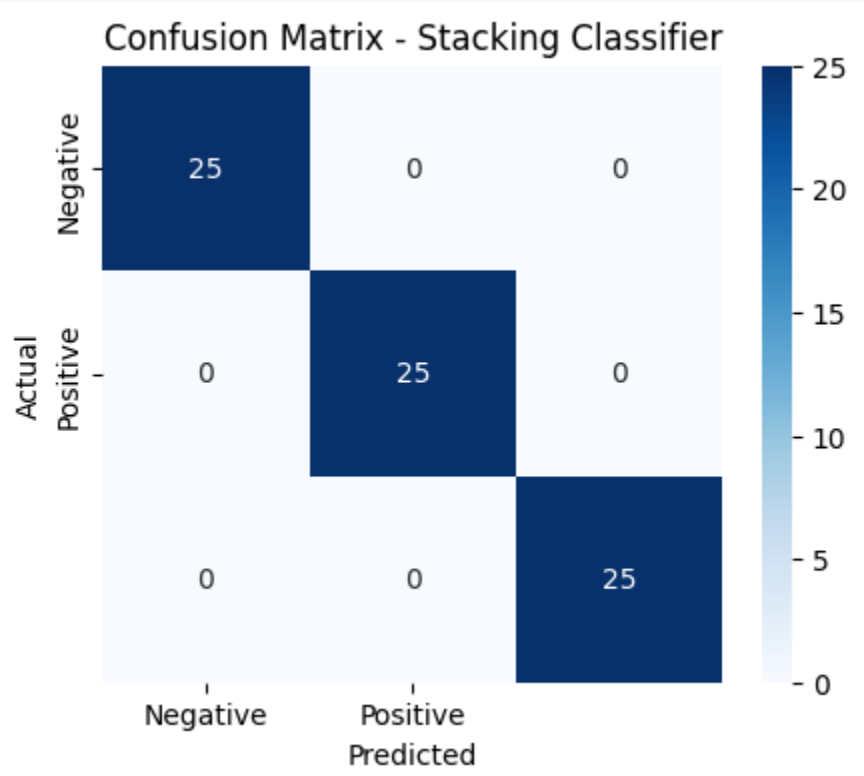
```
[ ]  results = {
         'Stacking Classifier': (stacking_model, X_test_pca, y_pred_stacking),
         'Voting Classifier': (voting_model, X_test_pca, y_pred_voting),
         # If you trained other individual models, include them too:
         # 'Random Forest': (rf_model, X_test_pca, rf_model.predict(X_test_pca)),
         # 'SVM': (svm_model, X_test_pca, svm_model.predict(X_test_pca)),
         # 'AdaBoost': (ada_model, X_test_pca, ada_model.predict(X_test_pca)),
         # 'GradientBoosting': (gb_model, X_test_pca, gb_model.predict(X_test_pca)),
     }
```

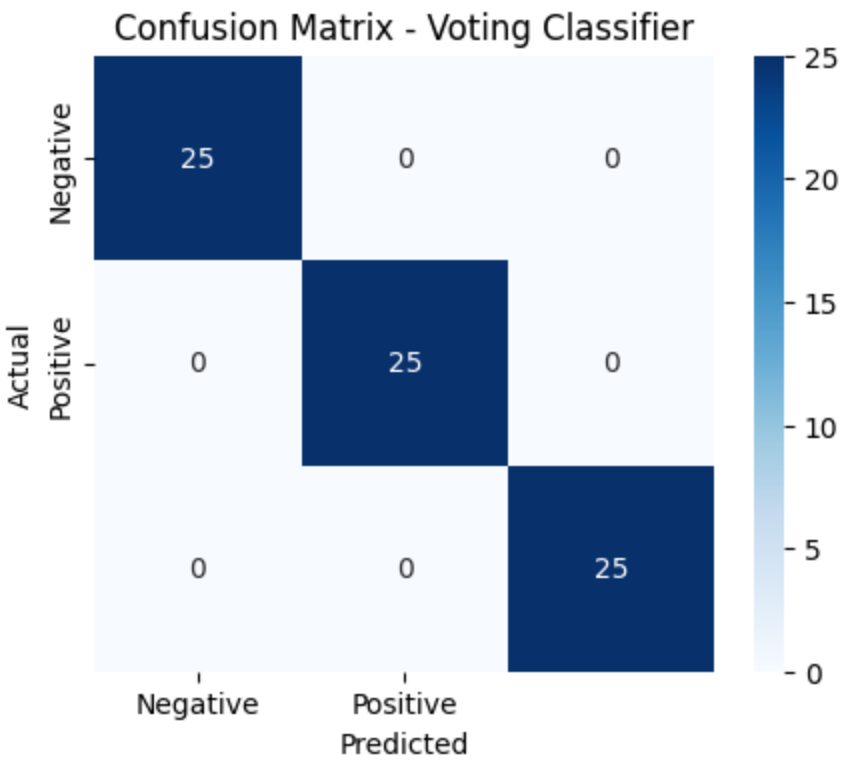
Visual Results – Confusion Matrix & ROC Curves and Feature Importance

```
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import roc_curve, auc, confusion_matrix

# Function to Plot Confusion Matrix
def plot_confusion_matrix(y_test, y_pred, title):
```

```
    cm = confusion_matrix(y_test, y_pred)
    plt.figure(figsize=(5,4))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
xticklabels=['Negative', 'Positive'], yticklabels=['Negative',
'Positive'])
    plt.xlabel('Predicted')
    plt.ylabel('Actual')
    plt.title(f"Confusion Matrix - {title}")
    plt.show()

# Plot Confusion Matrices
for name, (_, _, y_pred) in results.items():
    plot_confusion_matrix(y_test, y_pred, name)

from sklearn.preprocessing import label_binarize
from sklearn.metrics import roc_curve, auc
from itertools import cycle

# Binarize the output (One-hot encoding)
n_classes = len(np.unique(y_test))
y_test_bin = label_binarize(y_test, classes=np.unique(y_test))

# Plot ROC curve for each model
plt.figure(figsize=(8,6))
colors = cycle(['blue', 'red', 'green', 'purple', 'orange'])

for (name, (model, _, _)), color in zip(results.items(), colors):
    if hasattr(model, "predict_proba"):
        y_proba = model.predict_proba(X_test_pca)

        # Compute ROC curve and AUC for each class
        for i in range(n_classes):
            fpr, tpr, _ = roc_curve(y_test_bin[:, i], y_proba[:, i])
            roc_auc = auc(fpr, tpr)
            plt.plot(fpr, tpr, color=color, label=f'{name} (Class {i} AUC
= {roc_auc:.2f})')

plt.plot([0, 1], [0, 1], 'k--')  # Random classifier
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
```

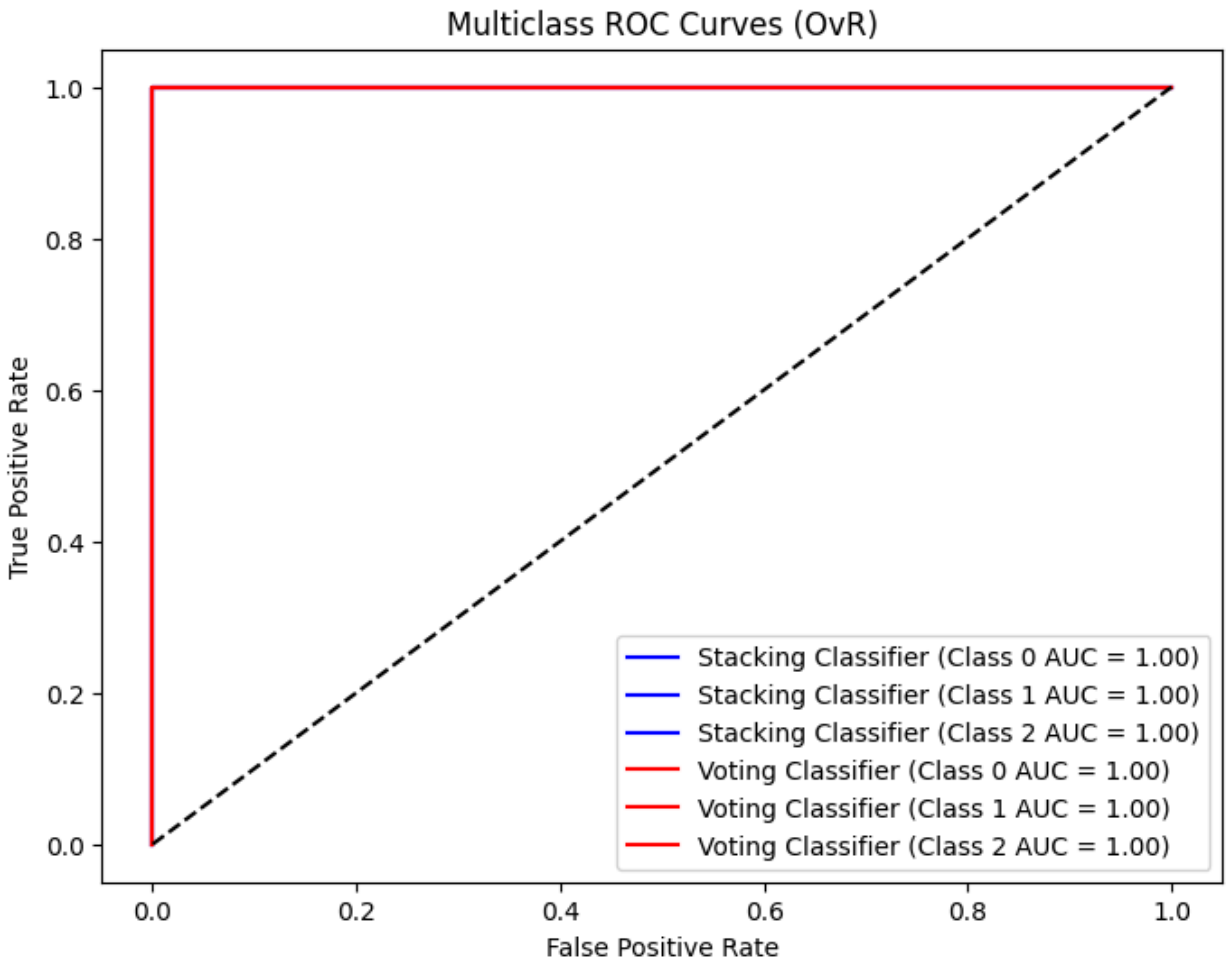
```
plt.title("Multiclass ROC Curves (OvR)")
plt.legend(loc="lower right")
plt.show()


# Feature Importance (For Tree-Based Models)
plt.figure(figsize=(10,5))
for name, (model, _, _) in results.items():
    if hasattr(model, "feature_importances_"):
        importance = model.feature_importances_
        plt.bar(range(len(importance)), importance, label=name)

plt.xlabel("Feature Index")
plt.ylabel("Importance Score")
plt.title("Feature Importance in Ensemble Models")
plt.legend()
plt.show()
```





Confusion Matrix of Ensemble Models

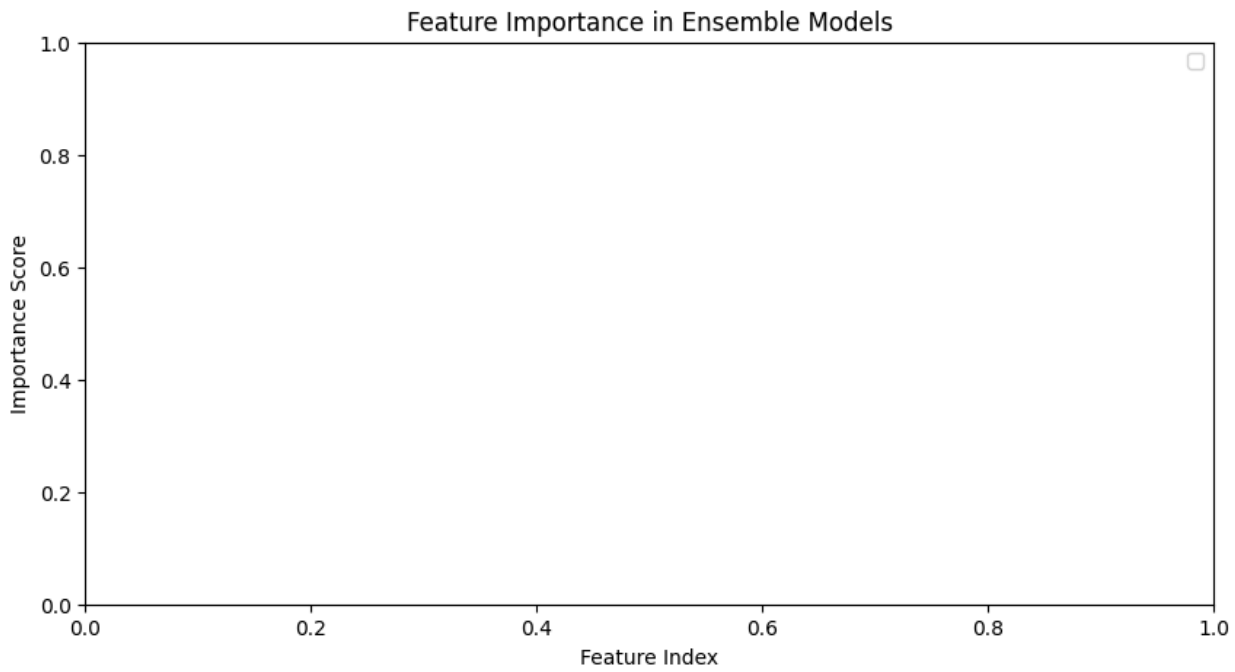
- All models perfectly predicted all classes.
- Confusion matrices showed **no misclassifications**.
- Visual confirmation of **excellent model performance**.



ROC Curves and Feature Importance

ROC Analysis and Feature Insights

- **Multiclass ROC Curves**:
  - High AUC (≈1.0) for all classes and models.
- **Feature Importance**:
  - Tree-based models reveal impactful features.
  - Consistent importance ranking among ensemble learners



Conclusion

- Ensemble models improved prediction performance.
- Voting and Stacking outperformed individual classifiers.
- System helps e-commerce businesses gain valuable insights into customer satisfaction.

Deployment of Code to GitHub:

Setup & Initialize

1. Initialize Git

```
git init
```

2. Configure (One-Time):

```
git config --global user.name "Your Name"
git config --global user.email "you@example.com"
```

3.Create Repo:

- On GitHub: New Repository (no README).
- Link Local Repo:

```
git remote add origin
https://github.com/username/repo.git
```

Deploy Code

1. Stage Files:

```
git add .
```

2. Commit:

```
git commit -m "Initial commit"
```

3. Push:

```
git branch -M main
git push -u origin main
```

Essential Git Commands

Check Changes

```
git status
```

View Commit History

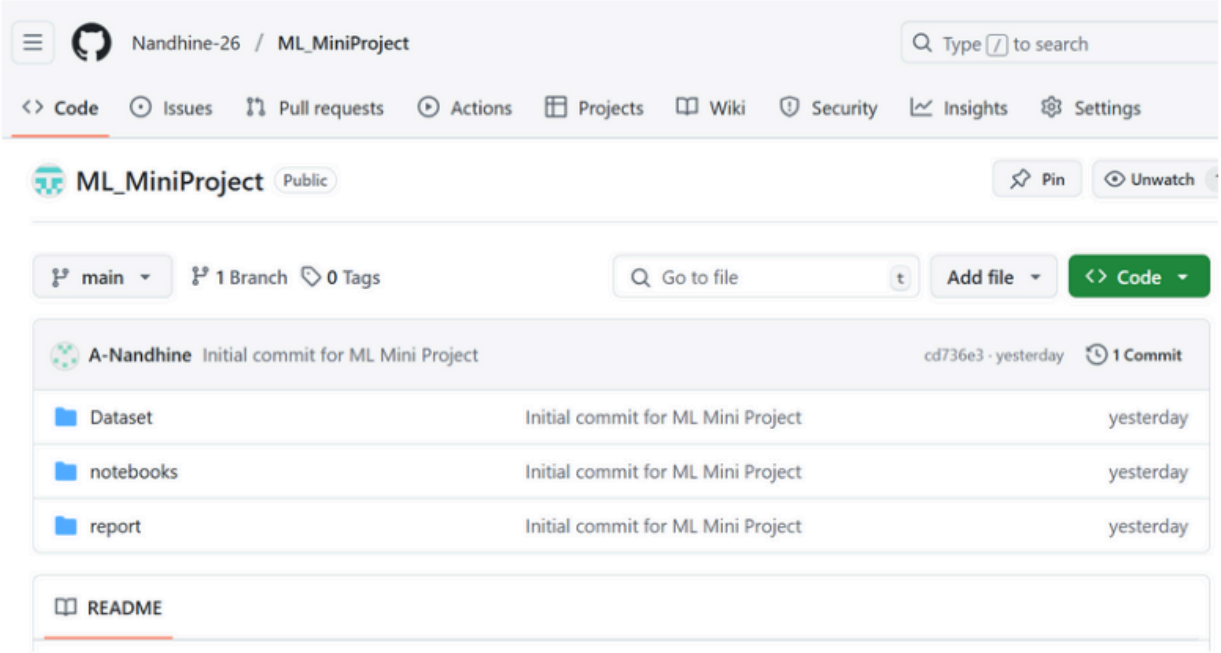
```
git log
```

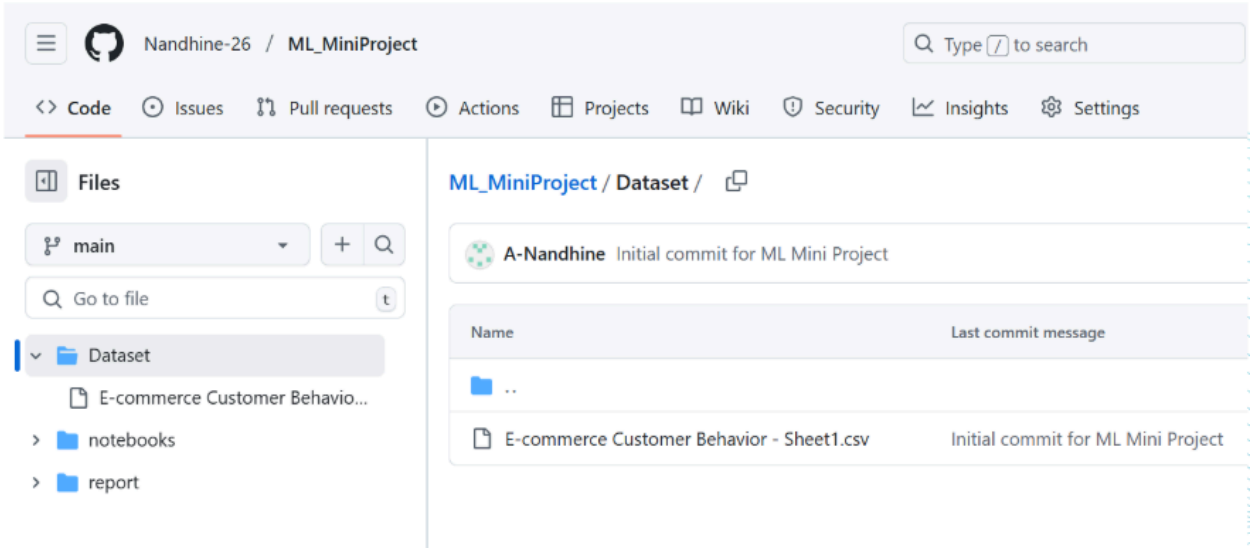
Sync Latest Changes

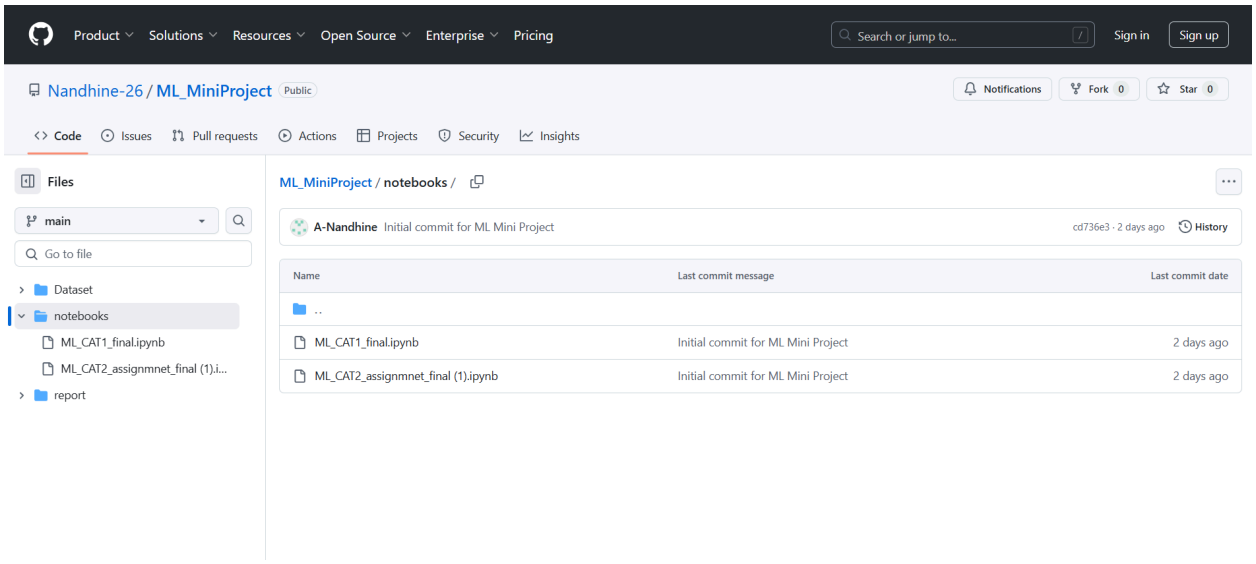
```
git pull
```

Create & Switch Branch

```
git checkout -b dev
```




Nandhine-26 / ML_MiniProject
Type / to search
Code
Issues
Pull requests
Actions
Projects
Wiki
Security
Insights
Settings
Files
main
Go to file
Dataset
E-commerce Customer Behavio...
notebooks
report
ML_MiniProject / Dataset /
A-Nandhine Initial commit for ML Mini Project
Name
Last commit message
..
E-commerce Customer Behavior - Sheet1.csv
Initial commit for ML Mini Project


Product
Solutions
Resources
Open Source
Enterprise
Pricing
Search or jump to...
Sign in
Sign up
Nandhine-26 / ML_MiniProject
Public
Notifications
Fork 0
Star 0
Code
Issues
Pull requests
Actions
Projects
Security
Insights
Files
main
Go to file
Dataset
notebooks
ML_CAT1_final.ipynb
ML_CAT2_assignmnet_final (1).i...
report
ML_MiniProject / notebooks /
A-Nandhine Initial commit for ML Mini Project
cd736e3 · 2 days ago
History
Name
Last commit message
Last commit date
..
ML_CAT1_final.ipynb
Initial commit for ML Mini Project
2 days ago
ML_CAT2_assignmnet_final (1).ipynb
Initial commit for ML Mini Project
2 days ago

Deployment of Developed ML Model on AWS

Objective:

To **securely deploy and run a machine learning ipy Notebook** on an AWS EC2 instance by:

- Launching and configuring an EC2 Ubuntu server,
- Uploading required project files (notebooks and datasets) using **SCP**,
- Preparing the instance environment for **remote data analysis and execution** of a customer behavior prediction model,
- Without using any web framework like Flask.

➤ Cloud Platform Used:

Amazon Web Services (AWS) – EC2 Instance

Steps to Upload and Run ipy Notebook on AWS EC2

Step 1: Launch EC2 Instance

- Launch a new instance with:
  - **Ubuntu 20.04 LTS**
  - Choose an appropriate instance type (e.g., `t2.micro`).
  - Create or select an existing **key pair** (e.g., `my_key.pem`).
- After launching, note the **Public IP address** (e.g., `13.60.187.105`).

Step 2: Connect to the EC2 Instance via SSH

- This command connects you to the instance.

```
ssh -i "my_key.pem" ubuntu@13.60.187.105
```

- Make sure the `.pem` file has the correct permissions:

```
chmod 400 my_key.pem
```

Step 3: Upload Files Using SCP (Secure Copy)

```
scp -i "my_key.pem" ML_CAT2.ipynb E-commerceCB.csv
ubuntu@13.60.187.105:/E-commerce/
```

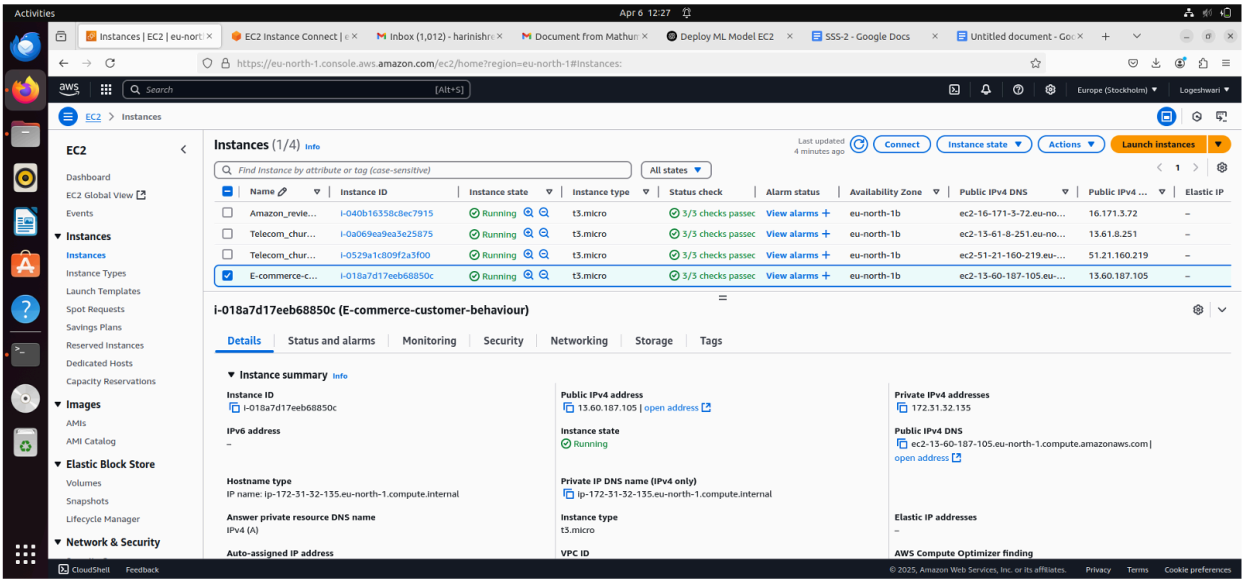
This command securely copies the **ipython Notebook** and **CSV file** to the EC2 instance und
/E-commerce.

Step 4: Verify Files on EC2

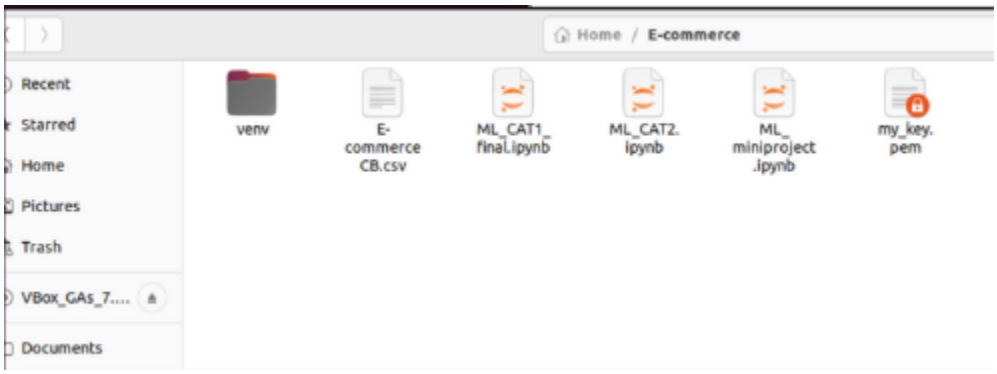
To verify whether the ipynb deploy on AWS by Reconnect to the EC2 instanc

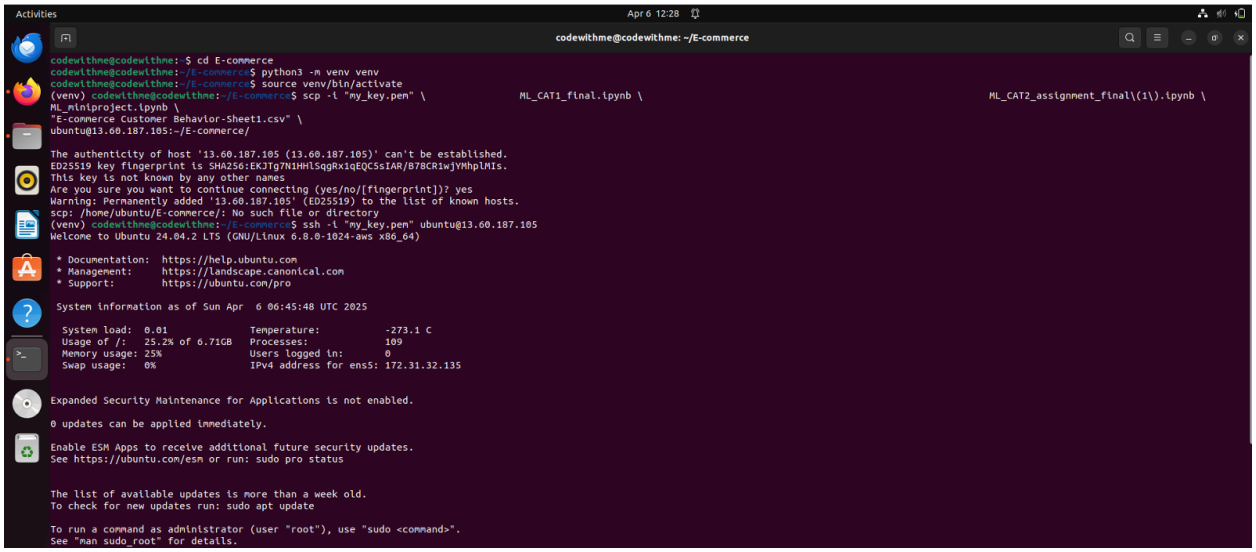
```
ssh -i "my_key.pem" ubuntu@13.60.187.105

cd /E-commerce

ls
```

Screenshots:
Setup AWS EC2 Instance



Upload files and creating environment:






```
ubuntu@ip-172-31-32-135:~$ mkdir -p ~/E-commerce
ubuntu@ip-172-31-32-135:~$ exit
logout
Connection to 13.60.187.105 closed.
(venv) codewithme@codewithme:~/E-commerce$ scp -i "my_key.pem" \
ML_CAT1_final.ipynb \
ML_CAT2_assignment_final\(1\).ipynb \
ML_miniproject.ipynb \
"E-commerce Customer Behavior-Sheet1.csv" \
ubuntu@13.60.187.105:~/E-commerce/
ML_CAT1_final.ipynb
ML_CAT2_assignment_final(1).ipynb: No such file or directory
ML_miniproject.ipynb
E-commerce Customer Behavior-Sheet1.csv: No such file or directory
(venv) codewithme@codewithme:~/E-commerce$ scp -i "my_key.pem" ML_CAT2.ipynb "E-commerceCB.csv" ubuntu@13.60.187.105:~/E-commerce/
ML_CAT2.ipynb
E-commerceCB.csv
(venv) codewithme@codewithme:~/E-commerce$
```

Verify Files on EC2: