Customer Sentiment Analysis - E-Commerce Sector

Shruti Bhanot E24092

Overview

The customer sentiment analysis employed VADER and machine learning models to classify reviews as positive, neutral, or negative. Analysis revealed predominantly positive feedback with higher ratings correlating with more positive sentiments. Logistic Regression was selected for its superior performance, offering actionable insights for improving customer satisfaction.

Objective

The primary objective of this analysis is to classify customer reviews into three sentiment categories: Positive, Neutral, and Negative. This classification helps in understanding customer feedback and aligning business strategies accordingly.

Assigned Task(s)

Customer Sentiment Analysis - E-Commerce Sector

Task Details

- Task 12 : The project involved analyzing customer sentiment using VADER and machine learning techniques. Data was preprocessed and reviews were classified into positive, neutral, or negative sentiments. Various models, including Logistic Regression, Random Forest, and XGBoost, were evaluated. Logistic Regression was chosen for its best performance, achieving high accuracy in sentiment classification. This analysis provided insights into customer feedback and satisfaction trends.
- Status : Completed
- Details : The task involved several key steps:
    1. <u>Data Preparation</u>: Cleaned and preprocessed review data, including text and sentiment labels.
    2. <u>Sentiment Analysis</u>: Applied VADER for initial sentiment classification.
    3. <u>Feature Extraction</u>: Used TF-IDF to vectorize review text for machine learning models.
    4. <u>Model Training</u>: Trained and evaluated Logistic Regression, Random Forest, and XGBoost classifiers.
    5. <u>Evaluation</u>: Assessed model performance using metrics like precision, recall, and F1-score.

6. Selection: Identified Logistic Regression as the most effective model based on accuracy and balanced performance.

Progress

- Accomplishments :
  - Successful Sentiment Classification: Accurately classified sentiment into positive, neutral, and negative categories using VADER, highlighting the overall positive sentiment in the dataset.
  - Feature Extraction Efficiency: Implemented TF-IDF to capture relevant features from text, enhancing model input quality.
  - Model Evaluation: Achieved high accuracy and robust performance with Logistic Regression, demonstrating its effectiveness in sentiment classification with a 96% accuracy rate.
  - Comprehensive Comparison: Compared multiple models (Logistic Regression, Random Forest, XGBoost) to determine the best-performing classifier, ensuring reliable and accurate sentiment analysis results.
- Metrics :

Accuracy Rates:

- Logistic Regression: 96% accuracy, with precision, recall, and f1-scores indicating strong performance across all sentiment classes.
- Random Forest: 92% accuracy, showing good performance but slightly lower compared to Logistic Regression.
- XGBoost: 94% accuracy, with balanced precision and recall, demonstrating effective classification but with some variance compared to Logistic Regression.

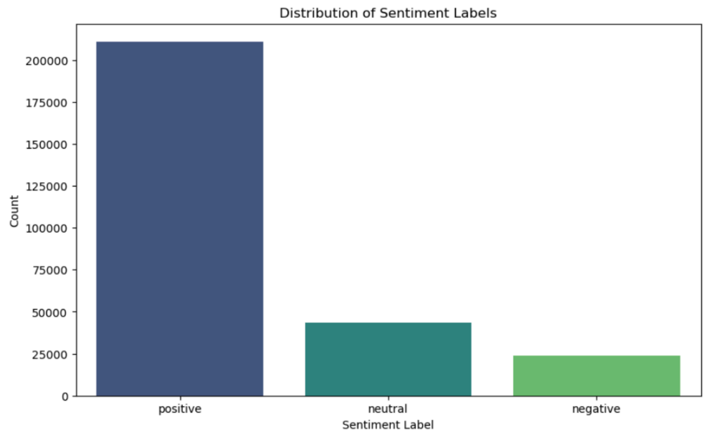
Classification Report Highlights:

- Logistic Regression: Highest overall f1-score of 0.95 for positive sentiment, reflecting excellent model performance.
- Random Forest: F1-score of 0.95 for positive sentiment, with lower performance for negative sentiment.
- XGBoost: Consistently high f1-scores for neutral and positive sentiments, demonstrating reliable classification.

Sentiment Distribution:

- Positive: 208,843 reviews classified as positive, indicating a predominant favorable sentiment.
- Neutral: 50,785 reviews, showing a considerable amount of neutral feedback.

- Negative: 18,471 reviews, representing a smaller portion of the dataset.



Feature Extraction:

- **TF-IDF Vectorization:** Utilized up to 10,000 features, improving model training and performance.

Challenges and Solutions

- Challenge Faced :
  - Data Inconsistencies: Encountered issues with column names and data types that required adjustment to ensure accurate processing and model training.
  - Model Class Imbalance: Faced challenges with class imbalances in the sentiment data, impacting the performance of some models, particularly for the negative sentiment class.
  - Algorithm Compatibility: Issues with XGBoost due to unexpected class labels, requiring adjustments to the class mapping to align with the model's expectations.
  - Performance Variability: Variability in model performance across different algorithms, necessitating careful evaluation and selection of the most suitable model based on accuracy and other metrics.
- Solutions Faced :
  - Data Inconsistencies: Standardized column names and data types; ensured accurate feature extraction through data integrity checks.
  - Model Class Imbalance: Implemented class weighting and stratified sampling; evaluated models on precision, recall, and F1-score for balanced performance.
  - Algorithm Compatibility: Aligned class mappings with model requirements; corrected dataset class definitions.
  - Performance Variability: Evaluated model performance using various metrics; selected the model with the highest accuracy and balanced metrics.

Next Step

- Upcoming Task:
  - Perform further Analysis based on given task
- Goals :
  - Complete the upcoming task and prepare ppt.

Conclusion

- Summary :
  - Project Overview: Analyzed review data to identify sentiment trends using VADER for sentiment polarity and distributions.
  - Models Trained: Logistic Regression, Random Forest, and XGBoost for classifying sentiments into positive, neutral, and negative categories.
  - Key Accomplishments: Achieved high accuracy and detailed performance metrics for each model.
  - Challenges Addressed: Resolved data inconsistencies and class imbalances through preprocessing and model adjustments.
  - Final Outcome: Logistic Regression emerged as the top-performing model based on precision, recall, and F1-score.
- Acknowledgments : Thank you for your time and attention

Code

In [1]:
```python
#Import Libraries
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
# Ignore warnings
import warnings
warnings.filterwarnings('ignore')
```

In [2]:
```python
#Load dataset
df = pd.read_csv("TeePublic_review.csv", encoding='ISO-8859-1')
df.head()
```

Out[2]:

	reviewer_id	store_location	latitude	longitude	date	month	year	title	review	review-label
0	0.0	US	37.090240	-95.712891	2023	6	2015 00:00:00	Great help with lost order	I had an order that was lost in transit. When ...	5
1	1.0	US	37.090240	-95.712891	2023	6	2024 00:00:00	I ordered the wrong size tee and hadï¿½ï¿½ï¿½	I ordered the wrong size tee and had difficult...	5
2	2.0	US	37.090240	-95.712891	2023	6	2017 00:00:00	These guys offer the best customerï¿½ï¿½ï¿½	These guys offer the best customer service in ...	5
3	3.0	US	37.090240	-95.712891	2023	6	2024 00:00:00	Good Stuff	Looked for an obscure phrase on a shirt. Teepu...	5
4	4.0	CA	56.130366	-106.346771	2023	6	2023	My order arrived in a good	My order arrived in a good timely	4

In [3]:
```python
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 278100 entries, 0 to 278099
Data columns (total 10 columns):
 #   Column          Non-Null Count   Dtype
---  ------          --------------   -----
 0   reviewer_id     278099 non-null  float64
 1   store_location  278100 non-null  object
 2   latitude        278100 non-null  float64
 3   longitude       278100 non-null  float64
 4   date            278100 non-null  int64
 5   month           278100 non-null  int64
 6   year            278100 non-null  object
 7   title           278088 non-null  object
 8   review          247597 non-null  object
 9   review-label    278100 non-null  int64
dtypes: float64(3), int64(3), object(4)
memory usage: 21.2+ MB
```

Data Cleaning and Transformation:

In [5]:
```python
# Finding duplicate rows (keeping all instances)
duplicate_rows = df[df.duplicated(keep=False)]

# Sorting the data by certain columns to see the duplicate rows next to each other
duplicate_rows_sorted = duplicate_rows.sort_values(by=['reviewer_id','store_location','latitude','longitude','date','month','year

# Displaying the first 10 records
duplicate_rows_sorted.head(10)
```

Out[5]:

reviewer_id	store_location	latitude	longitude	date	month	year	title	review	review-label

In [6]:
```python
# Check for missing values
print(df.isnull().sum())
```

```
reviewer_id           1
store_location        0
latitude              0
longitude             0
date                  0
month                 0
year                  0
title                12
review            30503
```

In [7]:
```python
# Fill missing values in 'title' with a placeholder
df['title'] = df['title'].fillna('No Title')
df.isnull().sum()
```

Out[7]:
```
reviewer_id          1
store_location       0
latitude             0
longitude            0
date                 0
month                0
year                 0
title                0
review           30503
review-label         0
dtype: int64
```

In [8]:
```python
# Check for non-integer values in the 'year' column
invalid_years = df[~df['year'].str.match(r'^\d{4}$', na=False)]
print(invalid_years[['year']])
```

```
                 year
0       2015 00:00:00
1       2024 00:00:00
2       2017 00:00:00
3       2024 00:00:00
4       2023 00:00:00
...               ...
278095  2027 00:00:00
```

In [9]:
```python
# Fill missing reviews with a placeholder
df['review'] = df['review'].fillna('No Review')

# Clean 'year' column
# Extract the year from date-time strings
df['year'] = df['year'].astype(str).str.split(' ', expand=True)[0]  # Split on space and take the first part (year)
df['year'] = pd.to_numeric(df['year'], errors='coerce').astype('Int64')  # Convert to numeric, coerce errors, convert to integer

# Handle remaining missing values in 'year' if necessary
df['year'] = df['year'].fillna(df['year'].mode()[0])  # Fill missing years with the most common year

# Convert 'review-label' to integer
df['review-label'] = pd.to_numeric(df['review-label'], errors='coerce').astype('Int64')

# Standardize text data
df['title'] = df['title'].str.lower().str.strip()  # Convert title to lowercase and strip whitespace
df['review'] = df['review'].str.lower().str.strip()  # Convert review to lowercase and strip whitespace

# Remove punctuation from review text
df['review'] = df['review'].str.replace('[^\w\s]', '', regex=True)

# Verify data consistency
# Check for valid review labels (1 to 5)
valid_labels = [1, 2, 3, 4, 5]
df = df[df['review-label'].isin(valid_labels)]

# Optionally reset the index
df = df.reset_index(drop=True)
```

In [11]:
```
# Investigate rows with missing 'reviewer_id'
missing_reviewer_id = df[df['reviewer_id'].isna()]
missing_reviewer_id
```

Out[11]:

	reviewer_id	store_location	latitude	longitude	date	month	year	title	review	review-label
278099	NaN	US	37.09024	-95.712891	2018	4	2027	not great quality	print of t shirt was blurry and appeared faded...	2

In [12]:
```
# Remove rows where 'reviewer_id' is missing
df = df.dropna(subset=['reviewer_id'])
# Save the cleaned dataset
df.to_csv('cleaned_dataset.csv', index=False)

print("Rows with missing 'reviewer_id' have been removed. Cleaned dataset saved.")
```

```
Rows with missing 'reviewer_id' have been removed. Cleaned dataset saved.
```

In [13]:
```
print(df.isnull().sum())
```

```
reviewer_id       0
store_location    0
latitude          0
longitude         0
date              0
month             0
year              0
title             0
review            0
```

In [15]:
```
from textblob import TextBlob

# Function to calculate sentiment polarity
def get_sentiment(text):
    analysis = TextBlob(text)
    return analysis.sentiment.polarity

# Apply the function to the 'review' column
df['sentiment_polarity'] = df['review'].apply(get_sentiment)

# Classify sentiment based on polarity
df['sentiment_label'] = df['sentiment_polarity'].apply(lambda x: 'positive' if x > 0 else ('negative' if x < 0 else 'neutral'))

# Check sentiment distribution
print(df['sentiment_label'].value_counts())
```

```
sentiment_label
positive    210946
neutral      43398
negative     23755
Name: count, dtype: int64
```

In [16]:
```
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem import WordNetLemmatizer
import re

nltk.download('punkt')
nltk.download('stopwords')
nltk.download('wordnet')
```

```
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\91932\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\91932\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package wordnet to
[nltk_data]     C:\Users\91932\AppData\Roaming\nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
```

Out[16]: True

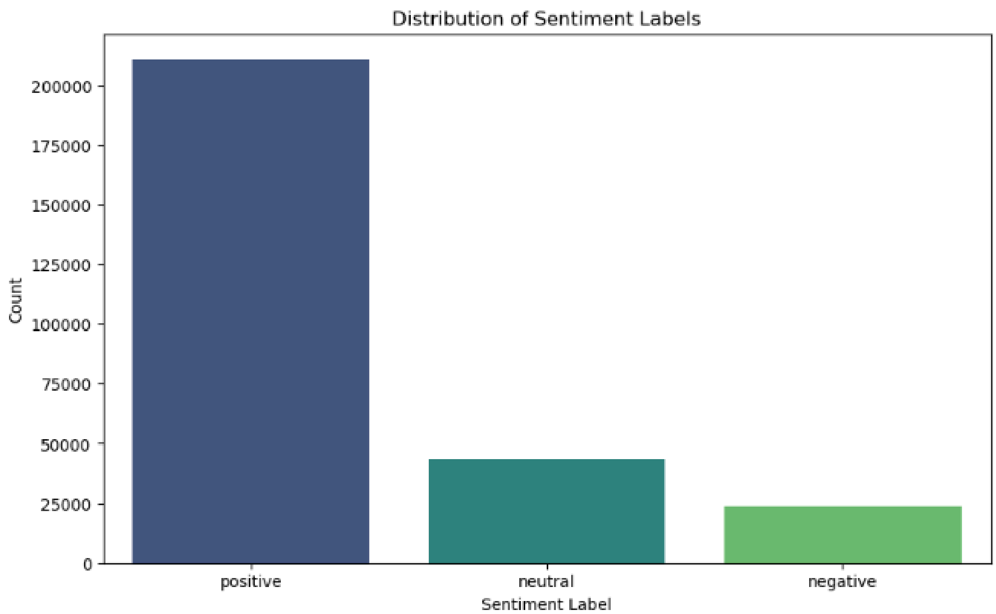
In [17]:
```
# Preprocessing function to clean text
def preprocess_text(text):
    text = re.sub(r'[^\w\s]', '', text)  # Remove punctuation
    tokens = word_tokenize(text.lower())  # Tokenize and convert to lowercase
    tokens = [word for word in tokens if word not in stopwords.words('english')]  # Remove stopwords
    lemmatizer = WordNetLemmatizer()  # Lemmatizer
    tokens = [lemmatizer.lemmatize(word) for word in tokens]  # Lemmatize words
    return ' '.join(tokens)

# Apply preprocessing to the 'review' column
df['cleaned_review'] = df['review'].apply(preprocess_text)
```

In [18]:
```
import re

# Function to clean text data
def clean_text(text):
    # Remove unwanted characters or encoding errors
    text = re.sub(r'[^A-Za-z0-9\s]+', '', text)  # Remove non-ASCII and special characters
    text = re.sub(r'\s+', ' ', text)  # Remove extra spaces
    text = text.strip()  # Remove leading and trailing whitespace
    return text

# Apply the cleaning function to the 'title' column
df['cleaned_title'] = df['title'].apply(clean_text)

# Display a few cleaned titles to verify the changes
print(df[['title', 'cleaned_title']].head(10))
```

```
                                            title  \
0                          great help with lost order
1  i ordered the wrong size tee and hadï¿½ï¿½ï¿½
2    these guys offer the best customerï¿½ï¿½ï¿½
3                                       good stuff
4     my order arrived in a good timelyï¿½ï¿½ï¿½
5                                  always top notch
6                                     recent review
7                               great communication
8                                           awesome
9     wonderful quality t-shirts for anï¿½ï¿½ï¿½

                           cleaned_title
0             great help with lost order
1  i ordered the wrong size tee and had
2    these guys offer the best customer
3                             good stuff
4     my order arrived in a good timely
5                       always top notch
6                          recent review
```

Data Exploration and Visualization:

In [20]:
```
# Plot the distribution of sentiment labels
plt.figure(figsize=(10, 6))
sns.countplot(x='sentiment_label', data=df, palette='viridis')
plt.title('Distribution of Sentiment Labels')
plt.xlabel('Sentiment Label')
plt.ylabel('Count')
plt.show()

# Analyze the relationship between sentiment labels and review ratings
plt.figure(figsize=(10, 6))
sns.boxplot(x='review-label', y='sentiment_polarity', data=df, palette='viridis')
plt.title('Sentiment Polarity by Review Ratings')
plt.xlabel('Review Rating')
plt.ylabel('Sentiment Polarity')
plt.show()
```

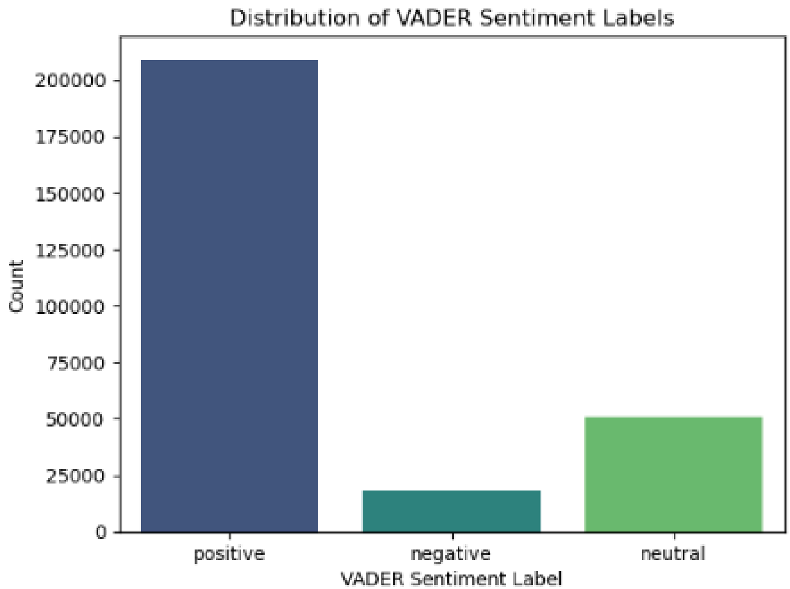


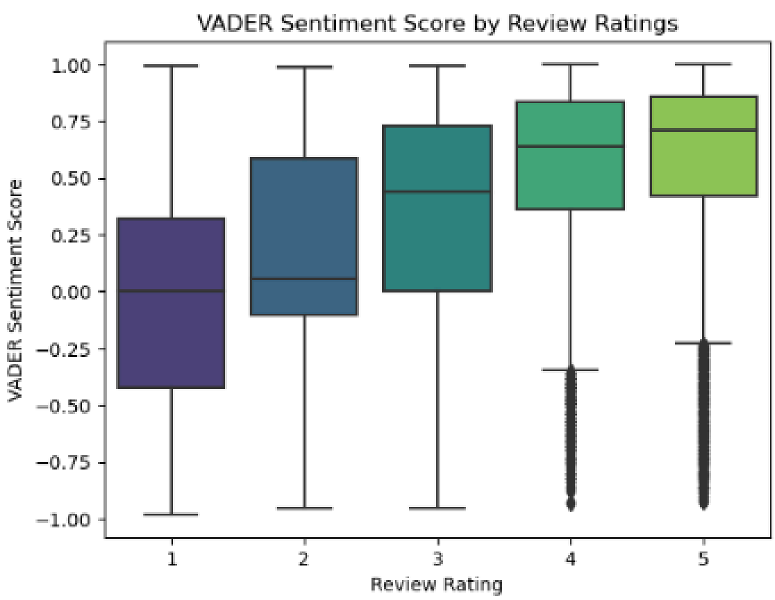
VADER Sentiment Analysis

In [21]:
```python
from nltk.sentiment.vader import SentimentIntensityAnalyzer
import nltk

# Download VADER lexicon
nltk.download('vader_lexicon')

# Initialize VADER sentiment analyzer
vader_analyzer = SentimentIntensityAnalyzer()

# Function to get sentiment scores
def vader_sentiment_scores(text):
    score = vader_analyzer.polarity_scores(text)
    return score['compound']

# Apply the VADER function to the 'cleaned_review' column
df['vader_sentiment_score'] = df['cleaned_review'].apply(vader_sentiment_scores)

# Classify VADER sentiment based on the compound score
df['vader_sentiment_label'] = df['vader_sentiment_score'].apply(
    lambda x: 'positive' if x > 0.05 else ('negative' if x < -0.05 else 'neutral')
)

# Check the distribution of VADER sentiment labels
print(df['vader_sentiment_label'].value_counts())

# Visualize VADER sentiment distribution
sns.countplot(data=df, x='vader_sentiment_label', palette='viridis')
plt.title('Distribution of VADER Sentiment Labels')
plt.xlabel('VADER Sentiment Label')
plt.ylabel('Count')
plt.show()

# Boxplot of VADER Sentiment Scores vs. Review Ratings
sns.boxplot(data=df, x='review-label', y='vader_sentiment_score', palette='viridis')
plt.title('VADER Sentiment Score by Review Ratings')
plt.xlabel('Review Rating')
plt.ylabel('VADER Sentiment Score')
plt.show()
```

```
[nltk_data] Downloading package vader_lexicon to
[nltk_data]     C:\Users\91932\AppData\Roaming\nltk_data...
[nltk_data]   Package vader_lexicon is already up-to-date!
```

```
vader_sentiment_label
positive    208843
neutral      50785
negative     18471
Name: count, dtype: int64
```





Feature Extraction Using TF-IDF

[22]:
```python
from sklearn.feature_extraction.text import TfidfVectorizer

# Initialize TF-IDF Vectorizer
tfidf_vectorizer = TfidfVectorizer(max_features=10000, ngram_range=(1, 2), stop_words='english')

# Fit and transform the 'review' column to create the TF-IDF matrix
X = tfidf_vectorizer.fit_transform(df['review'].astype(str))

# Extract the sentiment label as the target variable
y = df['vader_sentiment_label'].map({'positive': 2, 'neutral': 1, 'negative': 0})
```

Train Machine Learning Models

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier
from sklearn.metrics import classification_report, accuracy_score, confusion_matrix

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize the models
log_reg = LogisticRegression(max_iter=200)
rf_clf = RandomForestClassifier(n_estimators=100, random_state=42)
xgb_clf = XGBClassifier(use_label_encoder=False, eval_metric='mlogloss')

# Train the Logistic Regression model
log_reg.fit(X_train, y_train)
y_pred_log_reg = log_reg.predict(X_test)

# Train the Random Forest model
rf_clf.fit(X_train, y_train)
y_pred_rf = rf_clf.predict(X_test)

# Train the XGBoost model
xgb_clf.fit(X_train, y_train)
y_pred_xgb = xgb_clf.predict(X_test)

# Evaluate the models
print("Logistic Regression Classification Report:\n", classification_report(y_test, y_pred_log_reg))
print("Random Forest Classification Report:\n", classification_report(y_test, y_pred_rf))
print("XGBoost Classification Report:\n", classification_report(y_test, y_pred_xgb))
```

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

           0       0.86      0.68      0.76      3679
           1       0.94      0.96      0.95     10073
           2       0.97      0.98      0.98     41868

    accuracy                           0.96     55620
   macro avg       0.92      0.87      0.89     55620
weighted avg       0.96      0.96      0.96     55620

Random Forest Classification Report:
              precision    recall  f1-score   support

           0       0.88      0.37      0.52      3679
           1       0.95      0.82      0.88     10073
           2       0.91      0.99      0.95     41868

    accuracy                           0.92     55620
   macro avg       0.91      0.73      0.78     55620
weighted avg       0.92      0.92      0.91     55620

XGBoost Classification Report:
              precision    recall  f1-score   support

           0       0.83      0.59      0.69      3679
           1       0.88      0.95      0.92     10073
           2       0.97      0.97      0.97     41868

    accuracy                           0.94     55620
   macro avg       0.89      0.84      0.86     55620
weighted avg       0.94      0.94      0.94     55620
```