Project Title: Fraud Transaction Detection

Author: Sandeep Sharma

Email: s_sharma@ce.iitr.ac.in

GitHub: https://github.com/sandeeps9119

Objective

The goal of this project is to create a predictive model to detect fraudulent insurance claims for private motor products. The model will assess whether a customer's insurance claim is fraudulent.

Benefits

- **Proactive Fraud Detection:** Identifies potential frauds before they cause significant losses.
- **Enhanced Customer Insights:** Improves understanding of customer behavior.
- **Resource Optimization:** Streamlines resource management for fraud investigation.
- **Supports Manual Review:** Facilitates manual inspection when fraud is suspected.

Technologies

- **Machine Learning**

Domain

- **Banking and Insurance**

Project Difficulty Level

- **Intermediate**

Dataset

The dataset can be accessed from: Kaggle Fraud Detection Dataset

Data Sharing Agreement:

- **Sample File Names:** Fraudtrain.csv, Fraudtest.csv
- **Date Stamp Length:** 8 digits
- **Time Stamp Length:** 6 digits
- **Number of Columns:** 10 (example)
- **Column Names:** Transaction_ID, Trans_date, Trans_time, Customer_ID, Amount, Merchant, Category, Is_Fraud, Gender, Age
- **Column Data Types:** Integer, Date, Time, Integer, Float, String, String, Integer, String, Integer

Architecture

1. Data Validation and Transformation

Ensure data integrity and structure before further processing.

- **File Validation:** Check the integrity and structure of the fraudtrain and fraudtest datasets.
- **Column Verification:** Ensure both datasets have the correct number of columns and expected column names.
- **Data Type Validation:** Validate the data types of columns.
- **Missing Values:** Identify and appropriately handle missing values.

2. Data Insertion into Database

Insert validated data into the database for further analysis.

- **Database Connection:** Connect to the Cassandra database.
- **Table Creation:** Create `fraud_train_data` and `fraud_test_data` tables in Cassandra. Append new data without duplication if tables exist.
- **Data Insertion:** Insert validated datasets into respective tables.

3. Model Training

Prepare and use data for training machine learning models.

- **Data Extraction:** Extract data from Cassandra for model training.
- **Data Preprocessing:** Handle missing values, encode categorical variables, and scale numerical features.
- **Model Selection:** Train models such as SVM, Random Forest, and XGBoost. Optimize hyperparameters using cross-validation.
- **Model Evaluation:** Evaluate models using accuracy, precision, recall, and ROC-AUC score to select the best model.

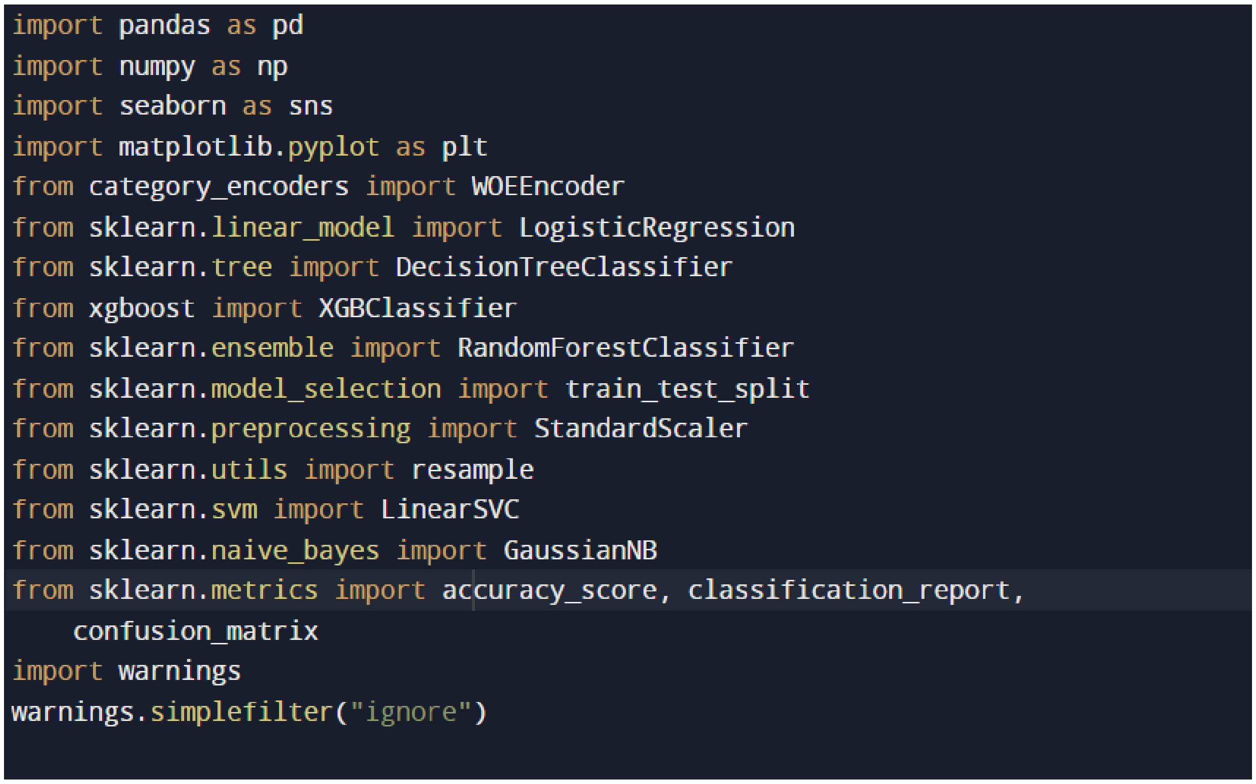
4. Prediction

Deploy trained models to predict fraud in new data.

- **Data Extraction:** Extract data from Cassandra for prediction.
- **Data Preprocessing:** Apply training preprocessing steps to the test dataset.
- **Model Prediction:** Use the selected model to predict fraud labels.
- **Performance Evaluation:** Evaluate model performance using accuracy, precision, recall, and ROC-AUC score.

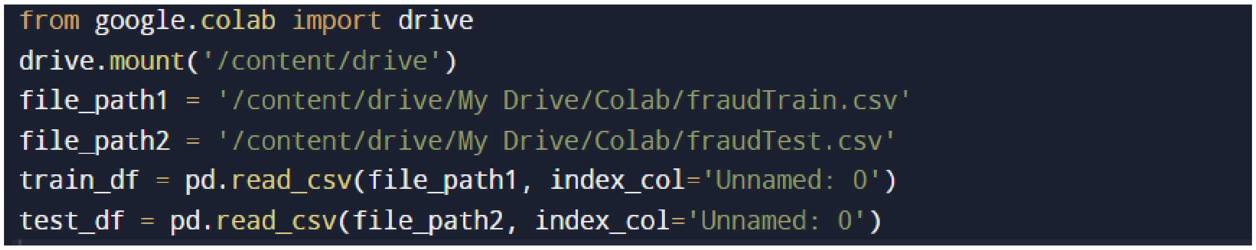
Code

1. Import Libraries



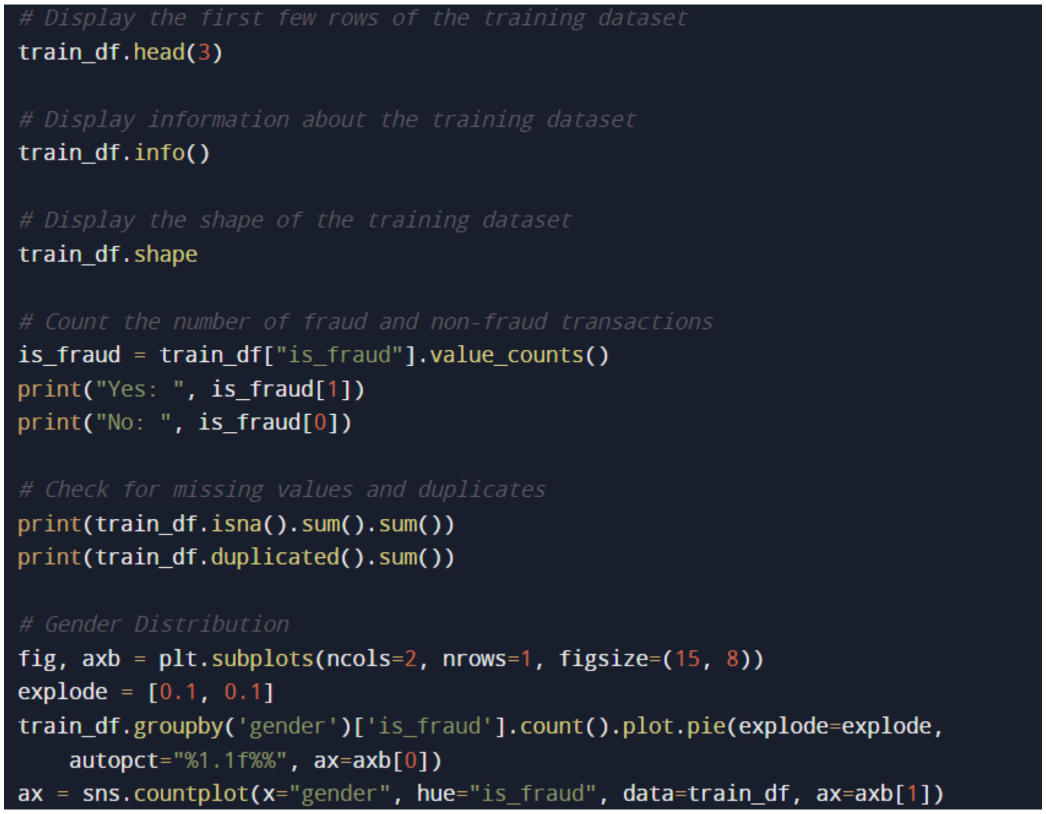
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from category_encoders import WOEEncoder
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from xgboost import XGBClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.utils import resample
from sklearn.svm import LinearSVC
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score, classification_report,
    confusion_matrix
import warnings
warnings.simplefilter("ignore")
```

2. Load Data

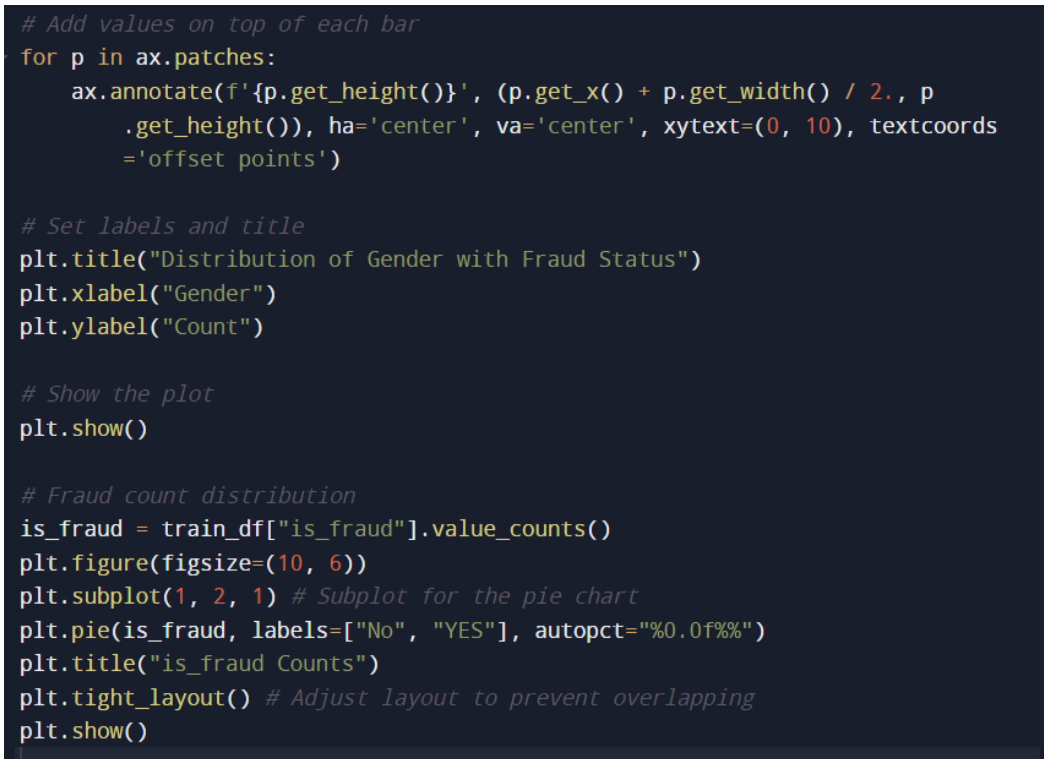


```
from google.colab import drive
drive.mount('/content/drive')
file_path1 = '/content/drive/My Drive/Colab/fraudTrain.csv'
file_path2 = '/content/drive/My Drive/Colab/fraudTest.csv'
train_df = pd.read_csv(file_path1, index_col='Unnamed: 0')
test_df = pd.read_csv(file_path2, index_col='Unnamed: 0')
```

3. Exploratory Data Analysis

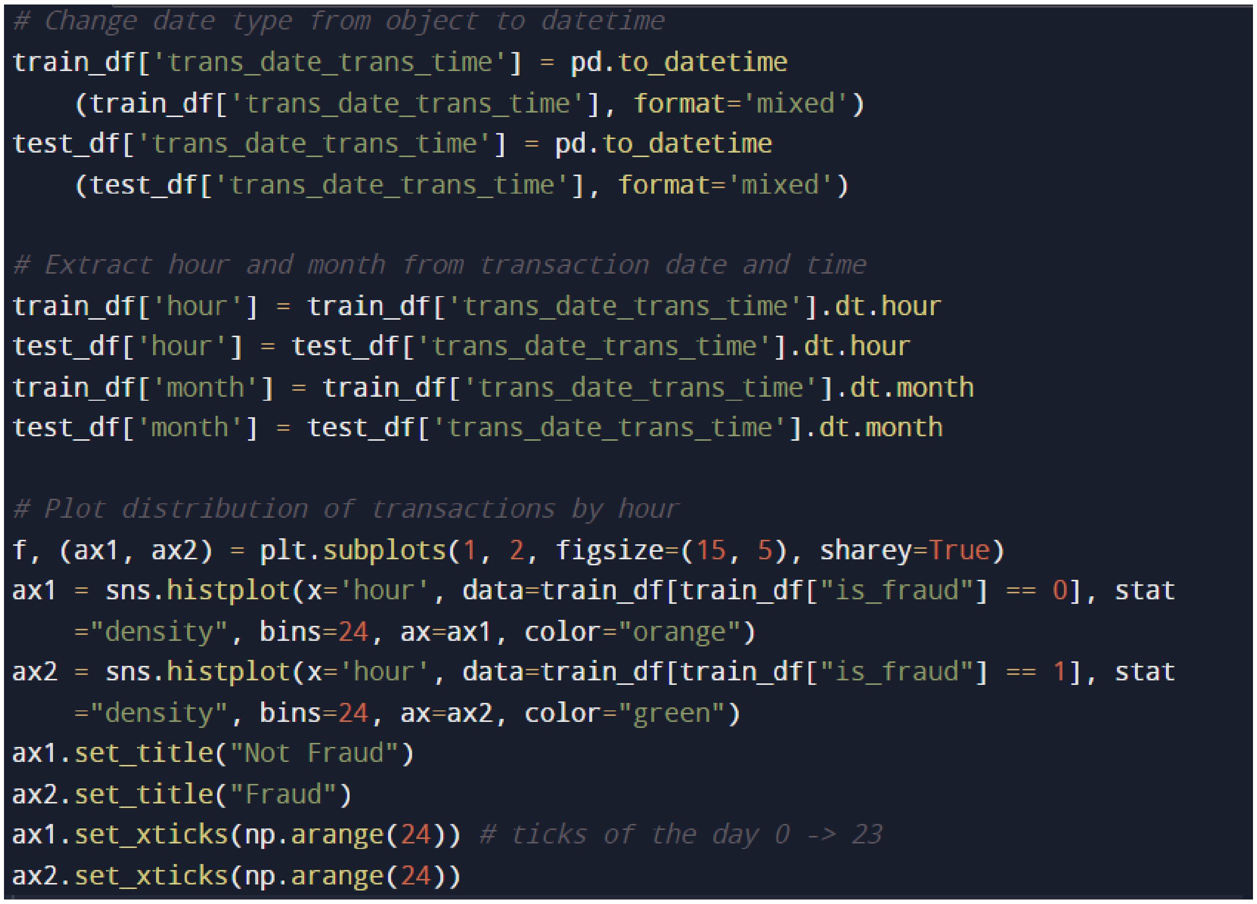


```python
# Display the first few rows of the training dataset
train_df.head(3)

# Display information about the training dataset
train_df.info()

# Display the shape of the training dataset
train_df.shape

# Count the number of fraud and non-fraud transactions
is_fraud = train_df["is_fraud"].value_counts()
print("Yes: ", is_fraud[1])
print("No: ", is_fraud[0])

# Check for missing values and duplicates
print(train_df.isna().sum().sum())
print(train_df.duplicated().sum())

# Gender Distribution
fig, axb = plt.subplots(ncols=2, nrows=1, figsize=(15, 8))
explode = [0.1, 0.1]
train_df.groupby('gender')['is_fraud'].count().plot.pie(explode=explode,
    autopct="%1.1f%%", ax=axb[0])
ax = sns.countplot(x="gender", hue="is_fraud", data=train_df, ax=axb[1])
```



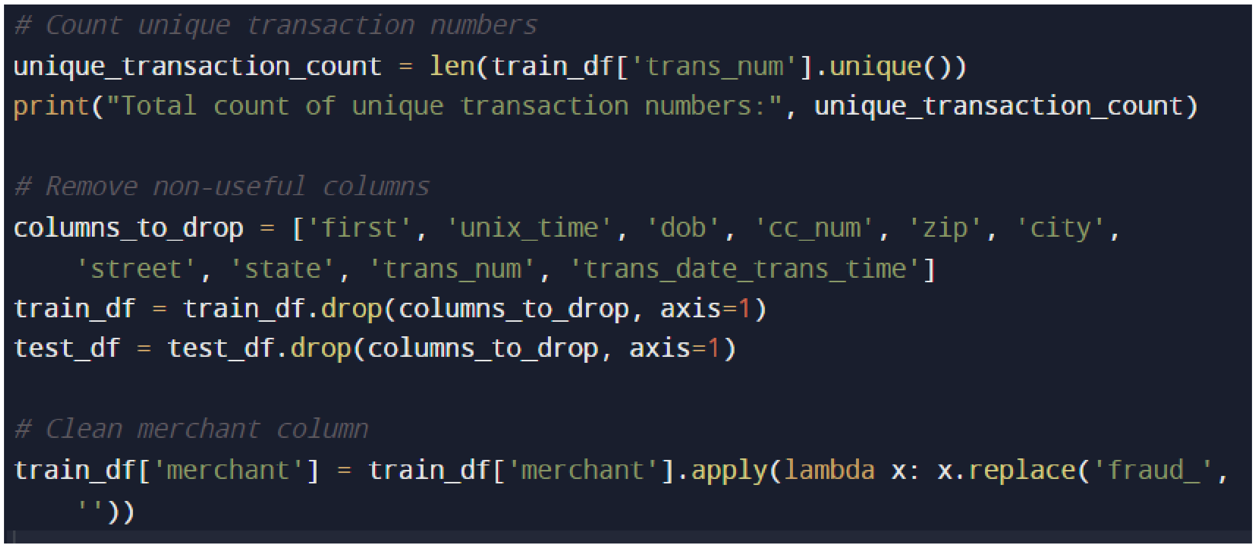
```python
# Add values on top of each bar
for p in ax.patches:
    ax.annotate(f'{p.get_height()}', (p.get_x() + p.get_width() / 2., p
        .get_height()), ha='center', va='center', xytext=(0, 10), textcoords
        ='offset points')

# Set labels and title
plt.title("Distribution of Gender with Fraud Status")
plt.xlabel("Gender")
plt.ylabel("Count")

# Show the plot
plt.show()

# Fraud count distribution
is_fraud = train_df["is_fraud"].value_counts()
plt.figure(figsize=(10, 6))
plt.subplot(1, 2, 1) # Subplot for the pie chart
plt.pie(is_fraud, labels=["No", "YES"], autopct="%0.0f%%")
plt.title("is_fraud Counts")
plt.tight_layout() # Adjust layout to prevent overlapping
plt.show()
```

4. Feature Engineering



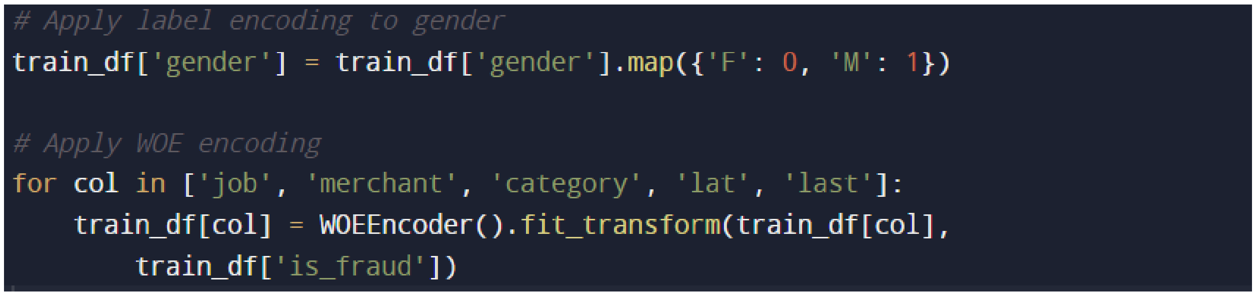
```python
# Change date type from object to datetime
train_df['trans_date_trans_time'] = pd.to_datetime
    (train_df['trans_date_trans_time'], format='mixed')
test_df['trans_date_trans_time'] = pd.to_datetime
    (test_df['trans_date_trans_time'], format='mixed')

# Extract hour and month from transaction date and time
train_df['hour'] = train_df['trans_date_trans_time'].dt.hour
test_df['hour'] = test_df['trans_date_trans_time'].dt.hour
train_df['month'] = train_df['trans_date_trans_time'].dt.month
test_df['month'] = test_df['trans_date_trans_time'].dt.month

# Plot distribution of transactions by hour
f, (ax1, ax2) = plt.subplots(1, 2, figsize=(15, 5), sharey=True)
ax1 = sns.histplot(x='hour', data=train_df[train_df["is_fraud"] == 0], stat
    ="density", bins=24, ax=ax1, color="orange")
ax2 = sns.histplot(x='hour', data=train_df[train_df["is_fraud"] == 1], stat
    ="density", bins=24, ax=ax2, color="green")
ax1.set_title("Not Fraud")
ax2.set_title("Fraud")
ax1.set_xticks(np.arange(24)) # ticks of the day 0 -> 23
ax2.set_xticks(np.arange(24))
```

5. Data Pre-processing



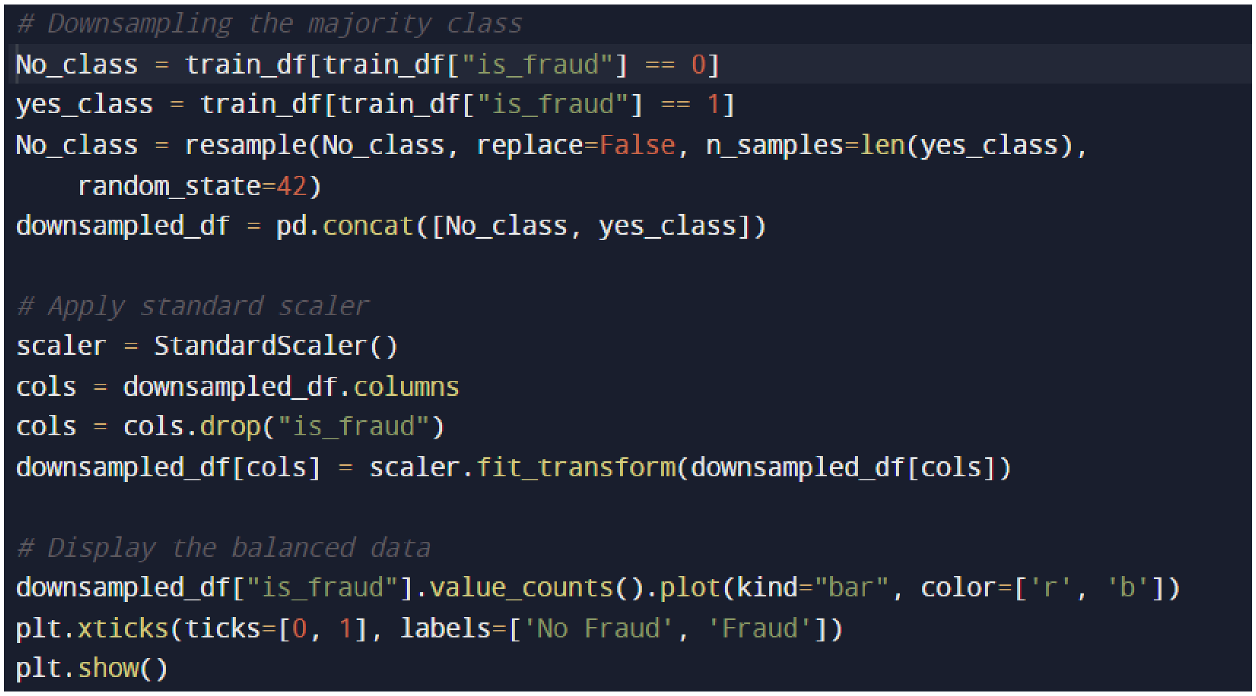
```python
# Count unique transaction numbers
unique_transaction_count = len(train_df['trans_num'].unique())
print("Total count of unique transaction numbers:", unique_transaction_count)

# Remove non-useful columns
columns_to_drop = ['first', 'unix_time', 'dob', 'cc_num', 'zip', 'city',
    'street', 'state', 'trans_num', 'trans_date_trans_time']
train_df = train_df.drop(columns_to_drop, axis=1)
test_df = test_df.drop(columns_to_drop, axis=1)

# Clean merchant column
train_df['merchant'] = train_df['merchant'].apply(lambda x: x.replace('fraud_',
    ''))
```

6. Data Encoding



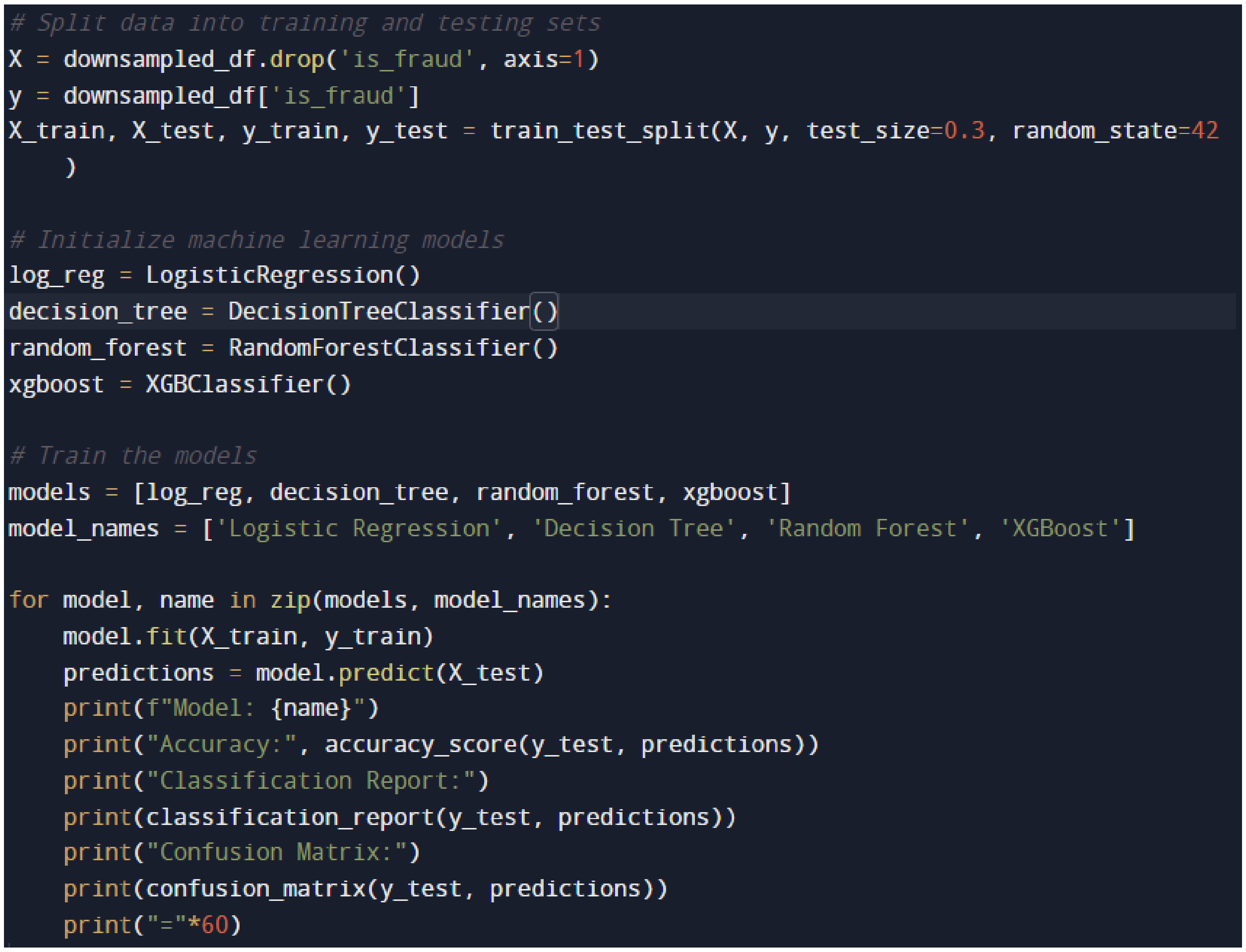
```
# Apply label encoding to gender
train_df['gender'] = train_df['gender'].map({'F': 0, 'M': 1})

# Apply WOE encoding
for col in ['job', 'merchant', 'category', 'lat', 'last']:
    train_df[col] = WOEEncoder().fit_transform(train_df[col],
        train_df['is_fraud'])
```

7. Down-Sampling and Scaling



```
# Downsampling the majority class
No_class = train_df[train_df["is_fraud"] == 0]
yes_class = train_df[train_df["is_fraud"] == 1]
No_class = resample(No_class, replace=False, n_samples=len(yes_class),
    random_state=42)
downsampled_df = pd.concat([No_class, yes_class])

# Apply standard scaler
scaler = StandardScaler()
cols = downsampled_df.columns
cols = cols.drop("is_fraud")
downsampled_df[cols] = scaler.fit_transform(downsampled_df[cols])

# Display the balanced data
downsampled_df["is_fraud"].value_counts().plot(kind="bar", color=['r', 'b'])
plt.xticks(ticks=[0, 1], labels=['No Fraud', 'Fraud'])
plt.show()
```

8. Model Building



```python
# Split data into training and testing sets
X = downsampled_df.drop('is_fraud', axis=1)
y = downsampled_df['is_fraud']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42
    )

# Initialize machine learning models
log_reg = LogisticRegression()
decision_tree = DecisionTreeClassifier()
random_forest = RandomForestClassifier()
xgboost = XGBClassifier()

# Train the models
models = [log_reg, decision_tree, random_forest, xgboost]
model_names = ['Logistic Regression', 'Decision Tree', 'Random Forest', 'XGBoost']

for model, name in zip(models, model_names):
    model.fit(X_train, y_train)
    predictions = model.predict(X_test)
    print(f"Model: {name}")
    print("Accuracy:", accuracy_score(y_test, predictions))
    print("Classification Report:")
    print(classification_report(y_test, predictions))
    print("Confusion Matrix:")
    print(confusion_matrix(y_test, predictions))
    print("="*60)
```

Conclusion

This project illustrates the process of developing a fraud transaction detection model using various machine learning algorithms. Through comprehensive data preprocessing and feature engineering, the models were optimized for accuracy and efficiency. The evaluation results provided insights into the effectiveness of each model in identifying fraudulent transactions.