King Fahd University of Petroleum and Minerals
Information and Computer Science Department
ICS474: Big Data Analytics

Project: Potential Fraudulent Predictor for Credit Card Transactions

Sunday Nov. 17, 2024

Instructor: Dr. Muzammil Behzad

Student Information		
Name	**ID#**	**Sec**
Omar Alromih	202019400	02
Mohammed Aljamili	202024300	
Mohammed Alothman	202019600	01

Contents

Objective 3

Part 1: Data Understanding and Exploration 3

Dataset Overview 3

Feature Description....4

Dataset Structure 6

Missing Values and Duplicates 6

Statistical Summary 7

Data Distribution 8

Correlation Analysis....14

Outlier Detection....16

Part 2: Data Preprocessing 17

Handling Missing Data....17

Encoding Categorical Variables 17

Feature Scaling 19

Feature Selection 20

Part 3: Modeling....21

Algorithm Selection....21

Data Splitting....21

Model Training 22

Model Evaluation....23

Performance Analysis....24

Model Improvement....25

Validation 26

Final Model Selection 28

Part 4: Visualization....30

Data Distribution 30

Feature Importance 32

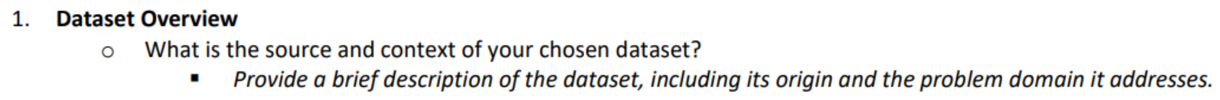
Model Performance Across Features....33

Objective

The objective of this project is to develop a predictive model that identifies potential fraudulent transactions in credit card usage by leveraging a combination of advanced machine learning algorithms. By analyzing key transaction-related features within a comprehensive dataset, this project aims to uncover significant patterns and anomalies indicative of fraud, thereby enhancing the accuracy and reliability of fraud detection systems. The analysis will prioritize precision and interpretability to support practical applications in financial security and risk management.

Part 1: Data Understanding and Exploration

Dataset Overview

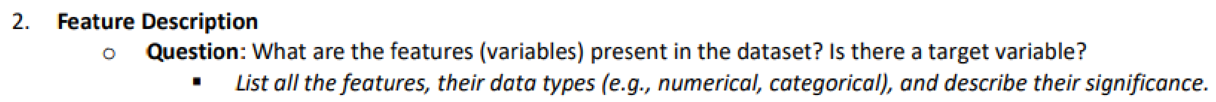


1. **Dataset Overview**
   - What is the source and context of your chosen dataset?
     - *Provide a brief description of the dataset, including its origin and the problem domain it addresses.*

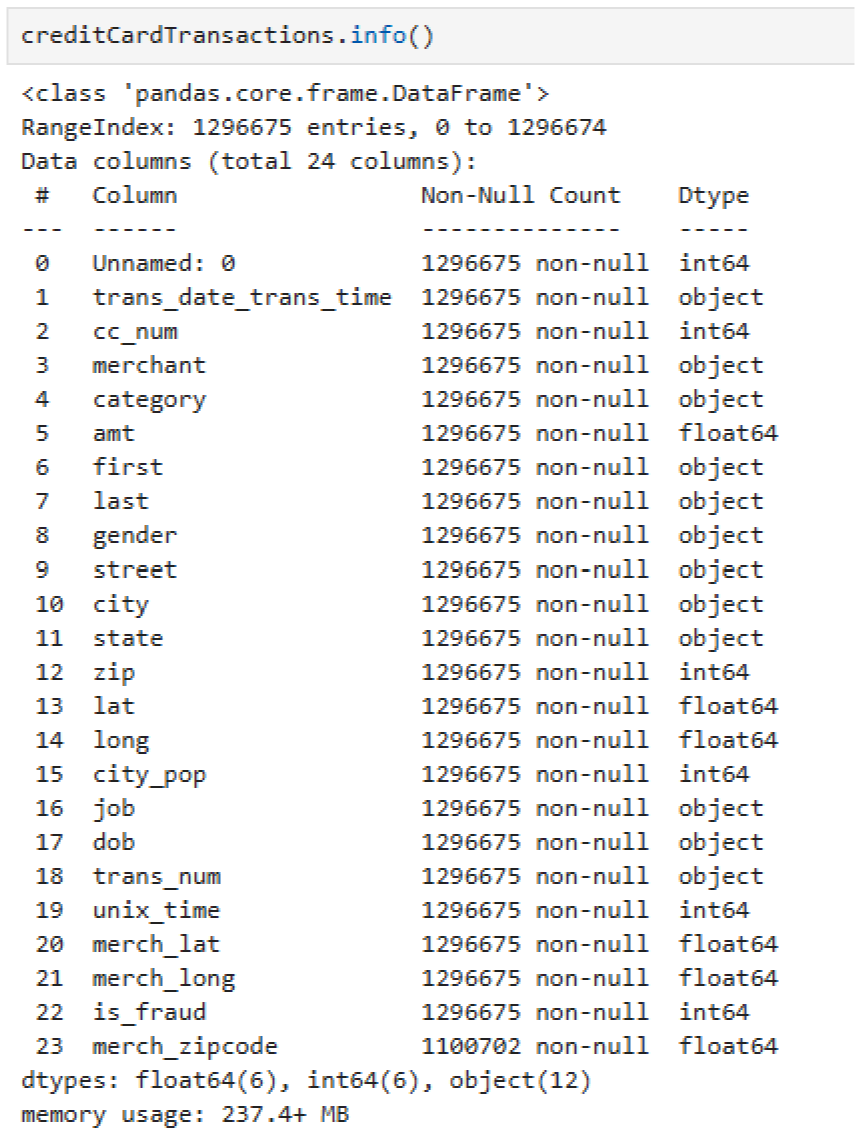
This dataset comprises transaction data with various features, including transaction amounts, dates of birth, and details about merchants and customers. The primary objective is to classify each transaction as either "fraudulent" or "non-fraudulent," which aids in identifying and mitigating fraud in financial systems.

The domain addressed by this dataset is financial fraud detection. In the finance sector, identifying fraudulent transactions is essential to prevent losses and uphold customer trust. By analyzing this dataset, data scientists and machine learning practitioners can create models that accurately predict and flag potentially fraudulent transactions.

Feature Description



2. **Feature Description**
   - **Question**: What are the features (variables) present in the dataset? Is there a target variable?
     - *List all the features, their data types (e.g., numerical, categorical), and describe their significance.*



```
creditCardTransactions.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1296675 entries, 0 to 1296674
Data columns (total 24 columns):
 #   Column                 Non-Null Count    Dtype  
---  ------                 --------------    -----  
 0   Unnamed: 0             1296675 non-null  int64  
 1   trans_date_trans_time  1296675 non-null  object 
 2   cc_num                 1296675 non-null  int64  
 3   merchant               1296675 non-null  object 
 4   category               1296675 non-null  object 
 5   amt                    1296675 non-null  float64
 6   first                  1296675 non-null  object 
 7   last                   1296675 non-null  object 
 8   gender                 1296675 non-null  object 
 9   street                 1296675 non-null  object 
 10  city                   1296675 non-null  object 
 11  state                  1296675 non-null  object 
 12  zip                    1296675 non-null  int64  
 13  lat                    1296675 non-null  float64
 14  long                   1296675 non-null  float64
 15  city_pop               1296675 non-null  int64  
 16  job                    1296675 non-null  object 
 17  dob                    1296675 non-null  object 
 18  trans_num              1296675 non-null  object 
 19  unix_time              1296675 non-null  int64  
 20  merch_lat              1296675 non-null  float64
 21  merch_long             1296675 non-null  float64
 22  is_fraud               1296675 non-null  int64  
 23  merch_zipcode          1100702 non-null  float64
dtypes: float64(6), int64(6), object(12)
memory usage: 237.4+ MB
```

The **Credit Card Transactions Dataset** includes the following features and their significance:

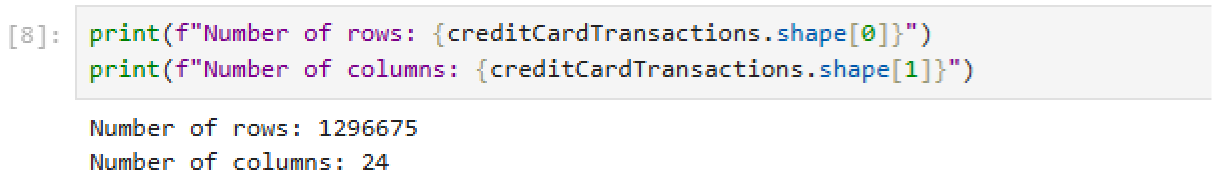
Column Name	Data Type	Description
trans_date_trans_time	object	The timestamp of the transaction, used for time-based analysis.
cc_num	int64	The credit card number (anonymized), essential for tracking transactions.

Column Name	Data Type	Description
merchant	object	The name of the merchant where the transaction occurred, useful for analyzing merchant activity.
category	object	The category of the transaction (e.g., retail, food), important for understanding spending habits.
amt	float64	The transaction amount, a critical feature for identifying high-risk transactions.
first	object	The first name of the cardholder, useful for demographic analysis.
last	object	The last name of the cardholder, also useful for demographic analysis.
gender	object	The gender of the cardholder, which can help in analyzing spending patterns by demographic.
street	object	The street address of the cardholder, helpful for geographical analysis.
city	object	The city of the cardholder, critical for location-based analyses.
state	object	The state of the cardholder, important for geographic segmentation.
zip	int64	The ZIP code for the cardholder's address, useful for detailed geographical insights.
lat	float64	The latitude of the transaction location, important for mapping and location analysis.
long	float64	The longitude of the transaction location, important for mapping and location analysis.
city_pop	int64	The population of the city, relevant for understanding transaction context.
job	object	The job title of the cardholder, useful for analyzing spending behaviors across occupations.
dob	object	The date of birth of the cardholder, necessary for demographic analysis and age segmentation.
trans_num	object	A unique transaction number, useful for tracking individual transactions.

Column Name	Data Type	Description
unix_time	int64	The transaction time in Unix timestamp format, useful for time-based analyses.
merch_lat	float64	The latitude of the merchant's location, important for location analysis.
merch_long	float64	The longitude of the merchant's location, important for location analysis.
is_fraud	int64	The target variable indicating whether the transaction is fraudulent (1) or not (0).
merch_zipcode	float64	The ZIP code of the merchant's location; it has some missing values, which may affect analysis.

Dataset Structure

3. **Dataset Structure**
   - **Question**: What is the size and structure of the dataset?
     - *Mention the number of rows and columns, and any hierarchical structure if applicable.*



```
[8]: print(f"Number of rows: {creditCardTransactions.shape[0]}")
     print(f"Number of columns: {creditCardTransactions.shape[1]}")

     Number of rows: 1296675
     Number of columns: 24
```

Missing Values and Duplicates

4. **Missing Values and Duplicates**
   - **Question**: Are there missing values or duplicates in the dataset?
     - *Identify any missing or duplicate entries and discuss how they might affect your analysis.*

```
[ ]: missing_values = creditCardTransactions.isnull().sum()
     print(missing_values)
```

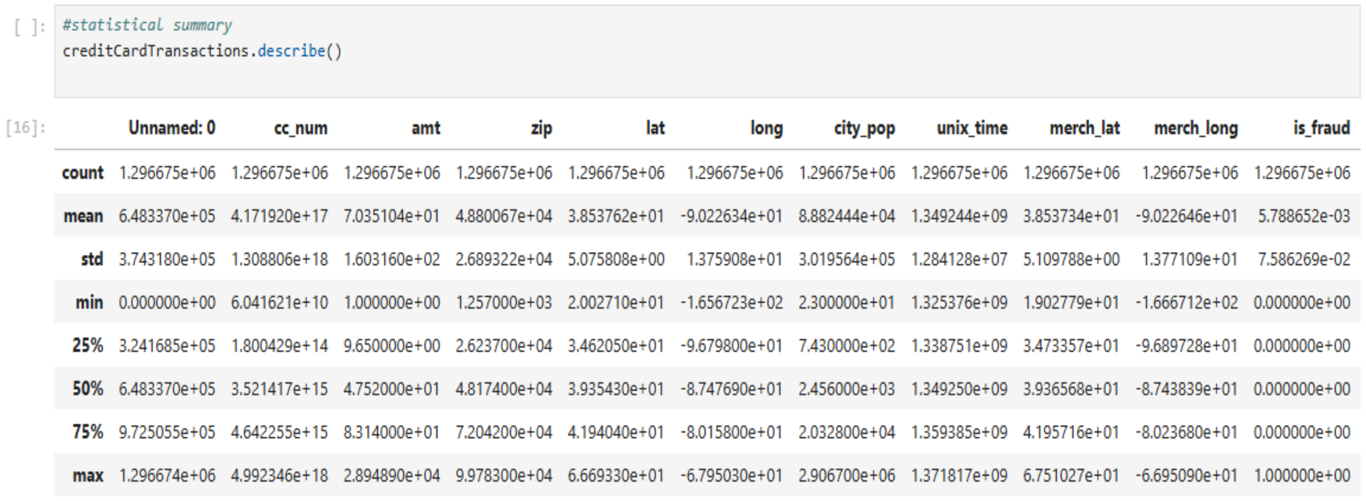
```
Unnamed: 0                    0
trans_date_trans_time         0
cc_num                        0
merchant                      0
category                      0
amt                           0
first                         0
last                          0
gender                        0
street                        0
city                          0
state                         0
zip                           0
lat                           0
long                          0
city_pop                      0
job                           0
dob                           0
trans_num                     0
unix_time                     0
merch_lat                     0
merch_long                    0
is_fraud                      0
merch_zipcode            195973
```

```
 # Number of duplicates present in the dataset
duplicate_values = creditCardTransactions.duplicated().sum()
print(f"There is {duplicate_values} duplicate values in the dataset")

# Display the duplicates along with the original rows
creditCardTransactions[creditCardTransactions.duplicated(keep=False)]
```

```
There is 0 duplicate values in the dataset
```

Statistical Summary

5. **Statistical Summary**
    - **Question**: What is the statistical summary of the dataset?
        - *Compute summary statistics like mean, median, standard deviation, and provide initial insights.*



```
[ ]: #statistical summary
     creditCardTransactions.describe()
```

[16]:

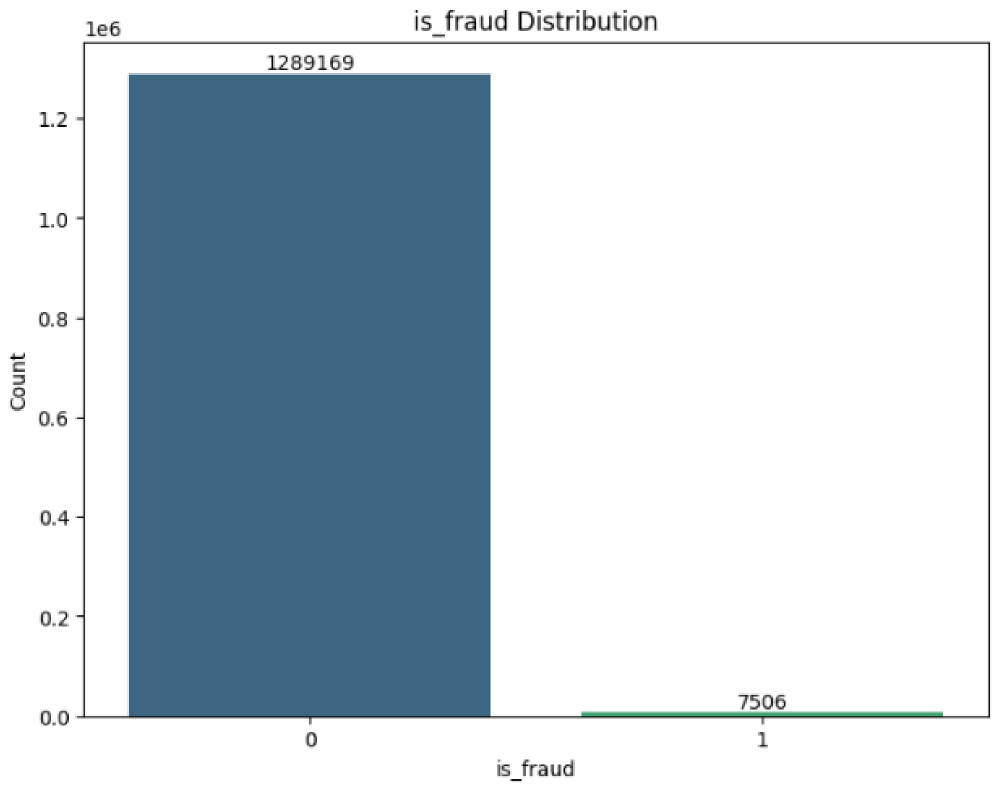
	Unnamed: 0	cc_num	amt	zip	lat	long	city_pop	unix_time	merch_lat	merch_long	is_fraud
count	1.296675e+06	1.296675e+06	1.296675e+06	1.296675e+06	1.296675e+06	1.296675e+06	1.296675e+06	1.296675e+06	1.296675e+06	1.296675e+06	1.296675e+06
mean	6.483370e+05	4.171920e+17	7.035104e+01	4.880067e+04	3.853762e+01	-9.022634e+01	8.882444e+04	1.349244e+09	3.853734e+01	-9.022646e+01	5.788652e-03
std	3.743180e+05	1.308806e+18	1.603160e+02	2.689322e+04	5.075808e+00	1.375908e+01	3.019564e+05	1.284128e+07	5.109788e+00	1.377109e+01	7.586269e-02
min	0.000000e+00	6.041621e+10	1.000000e+00	1.257000e+03	2.002710e+01	-1.656723e+02	2.300000e+01	1.325376e+09	1.902779e+01	-1.666712e+02	0.000000e+00
25%	3.241685e+05	1.800429e+14	9.650000e+00	2.623700e+04	3.462050e+01	-9.679800e+01	7.430000e+02	1.338751e+09	3.473357e+01	-9.689728e+01	0.000000e+00
50%	6.483370e+05	3.521417e+15	4.752000e+01	4.817400e+04	3.935430e+01	-8.747690e+01	2.456000e+03	1.349250e+09	3.936568e+01	-8.743839e+01	0.000000e+00
75%	9.725055e+05	4.642255e+15	8.314000e+01	7.204200e+04	4.194040e+01	-8.015800e+01	2.032800e+04	1.359385e+09	4.195716e+01	-8.023680e+01	0.000000e+00
max	1.296674e+06	4.992346e+18	2.894890e+04	9.978300e+04	6.669330e+01	-6.795030e+01	2.906700e+06	1.371817e+09	6.751027e+01	-6.695090e+01	1.000000e+00

Data Distribution

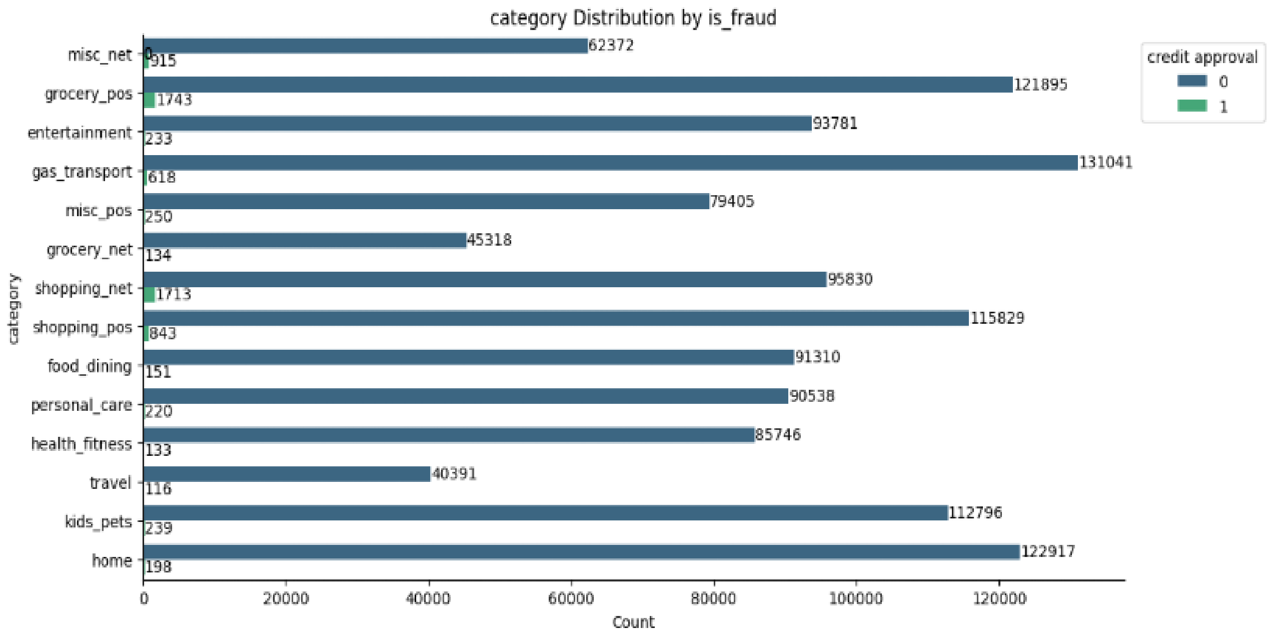
6. **Data Distribution**
   - **Question**: How are the features distributed?
     - *Use visualizations like histograms or box plots to show the distribution of key features.*

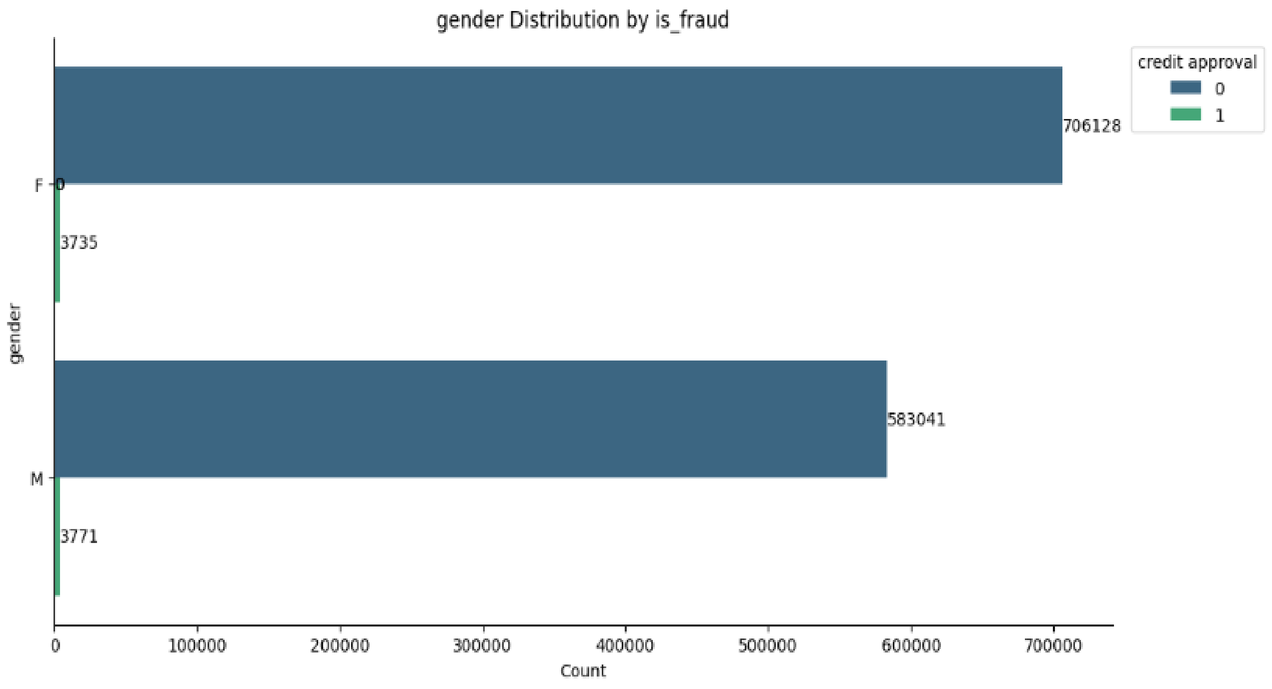
```python
plt.figure(figsize=(8, 6))
ax = sns.countplot(x='is_fraud', data=creditCardTransactions, palette='viridis')
for p in ax.patches:
    ax.annotate(f'{int(p.get_height())}',
                (p.get_x() + p.get_width() / 2., p.get_height()),
                ha='center', va='bottom',
                fontsize=10, color='black')
plt.title('is_fraud Distribution')
plt.xlabel('is_fraud')
plt.ylabel('Count')
plt.show()
```

```
<ipython-input-17-73530a2a4cab>:2: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x
same effect.

  ax = sns.countplot(x='is_fraud', data=creditCardTransactions, palette='viridis')
```

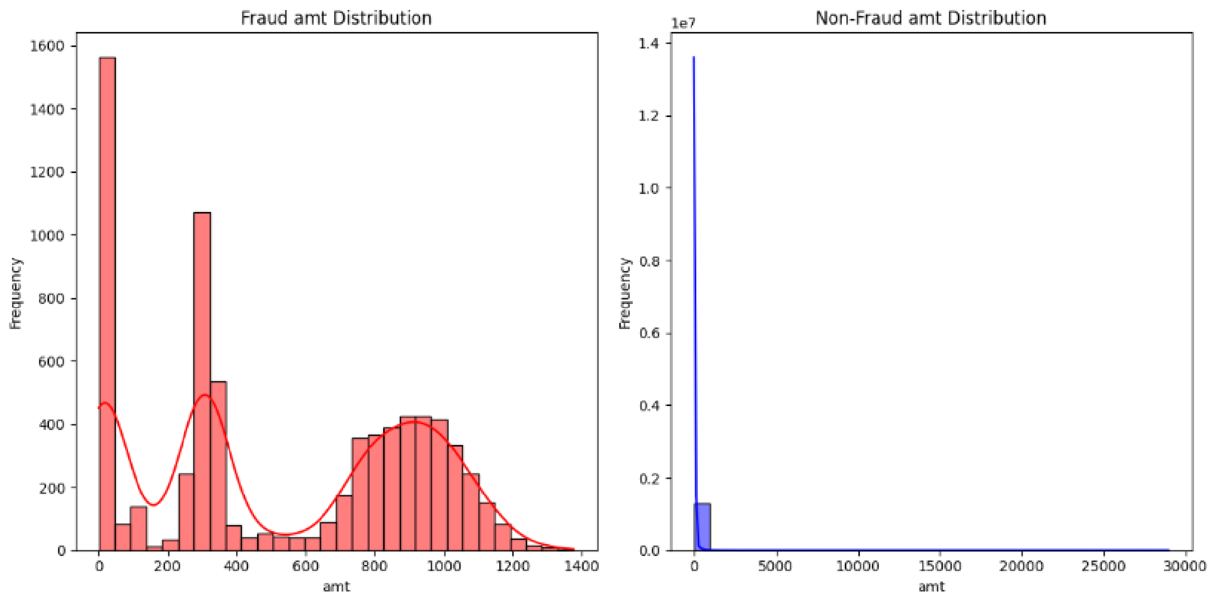


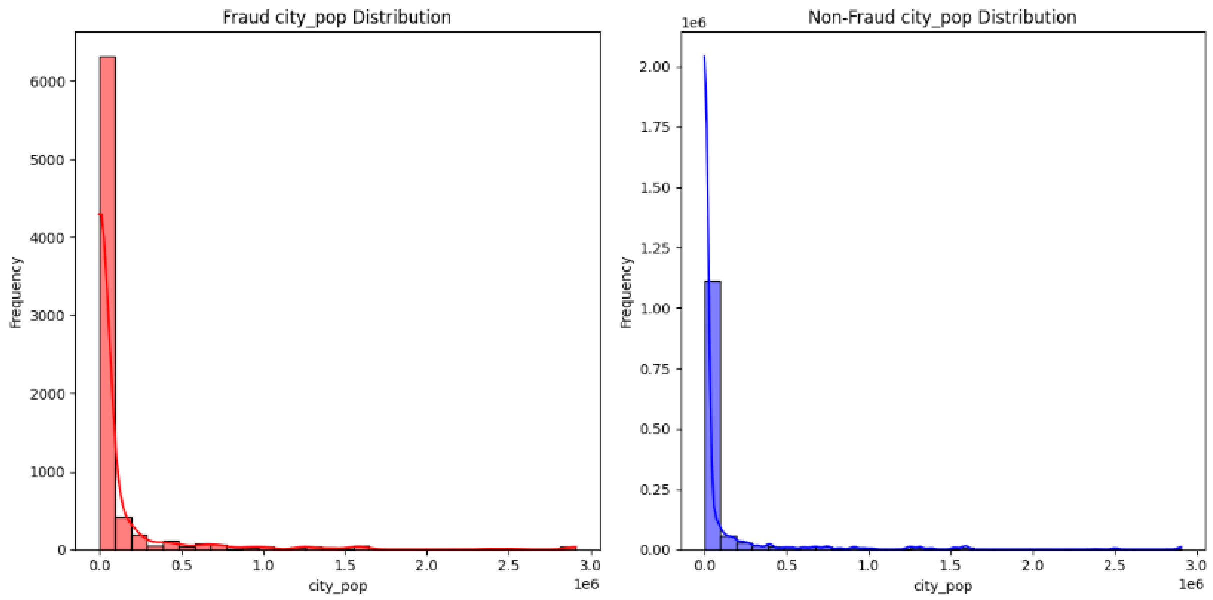
```python
categorical_columns = ['category', 'gender']

for column in categorical_columns:
    plt.figure(figsize=(12, 6))
    ax = sns.countplot(y=column, hue='is_fraud', data=creditCardTransactions, palette='viridis')
    plt.title(f'{column} Distribution by is_fraud')
    plt.xlabel('Count')
    plt.ylabel(column)
    plt.legend(title='credit approval', loc='upper right', bbox_to_anchor=(1.15, 1))
    ax.spines['top'].set_visible(False)
    ax.spines['right'].set_visible(False)
    ax.spines['left'].set_visible(True)
    ax.spines['bottom'].set_visible(True)

    for p in ax.patches:
        width = p.get_width()
        ax.text(width + 10, p.get_y() + p.get_height() / 2,
                f'{int(width)}',
                ha='left', va='center')

    plt.show()
```

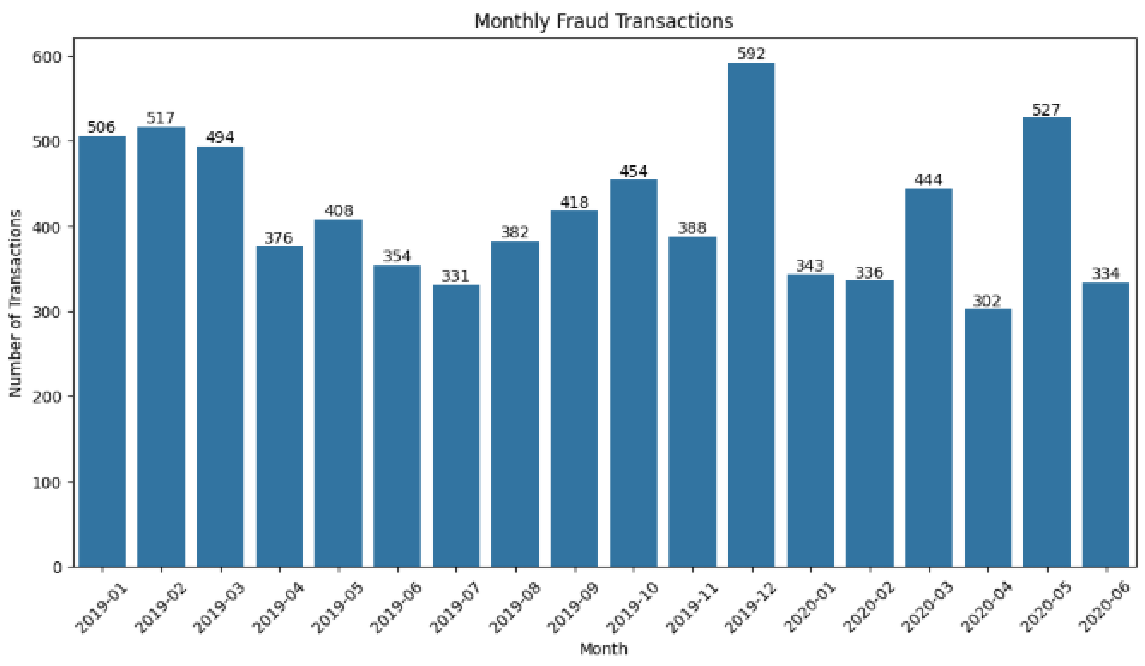
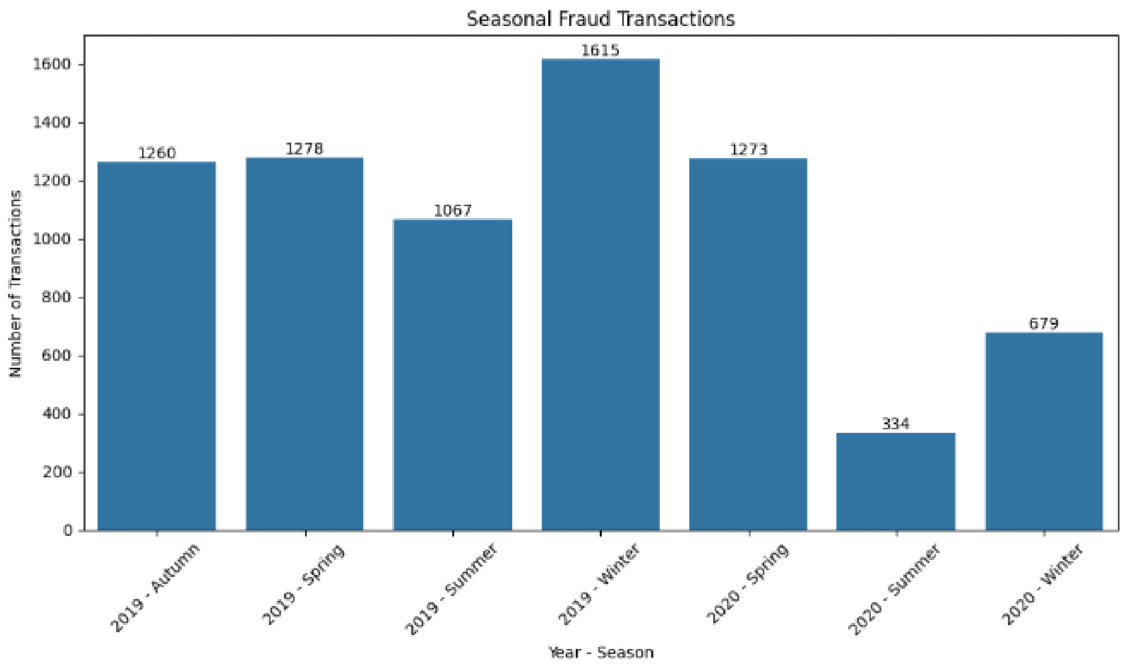

category Distribution by is_fraud
credit approval
0
1
misc_net
grocery_pos
entertainment
gas_transport
misc_pos
grocery_net
shopping_net
shopping_pos
food_dining
personal_care
health_fitness
travel
kids_pets
home
62372
915
121895
1743
93781
233
131041
618
79405
250
45318
134
95830
1713
115829
843
91310
151
90538
220
85746
133
40391
116
112796
239
122917
198
category
Count

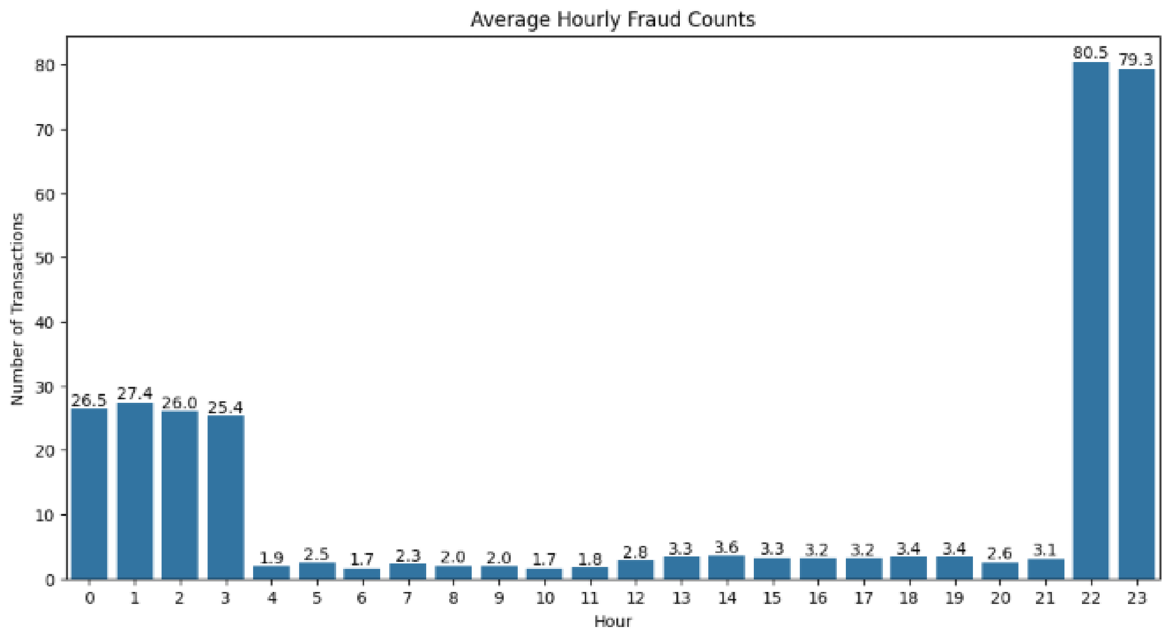

gender Distribution by is_fraud
credit approval
0
1
F
M
706128
3735
583041
3771
gender
Count

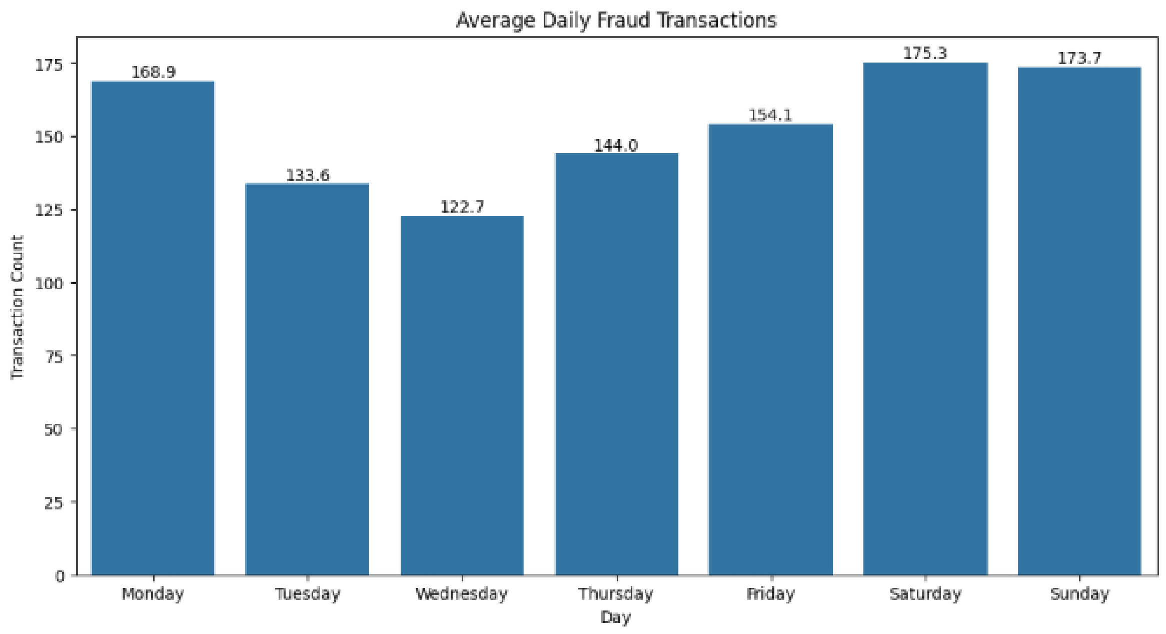
```python
numeric_columns = ['amt', 'city_pop']

for column in numeric_columns:
    plt.figure(figsize=(12, 6))

    plt.subplot(1, 2, 1)
    sns.histplot(creditCardTransactions[creditCardTransactions['is_fraud'] == 1][column], bins=30, kde=True, color='red')
    plt.title(f'Fraud {column} Distribution')
    plt.xlabel(column)
    plt.ylabel('Frequency')

    plt.subplot(1, 2, 2)
    sns.histplot(creditCardTransactions[creditCardTransactions['is_fraud'] == 0][column], bins=30, kde=True, color='blue')
    plt.title(f'Non-Fraud {column} Distribution')
    plt.xlabel(column)
    plt.ylabel('Frequency')

    plt.tight_layout()
    plt.show()
```






Monthly Fraud Transactions
Number of Transactions
Month
0
100
200
300
400
500
600
506
517
494
376
408
354
331
382
418
454
388
592
343
336
444
302
527
334
2019-01
2019-02
2019-03
2019-04
2019-05
2019-06
2019-07
2019-08
2019-09
2019-10
2019-11
2019-12
2020-01
2020-02
2020-03
2020-04
2020-05
2020-06

Seasonal Fraud Transactions
Number of Transactions
Year - Season
0
200
400
600
800
1000
1200
1400
1600
1260
1278
1067
1615
1273
334
679
2019 - Autumn
2019 - Spring
2019 - Summer
2019 - Winter
2020 - Spring
2020 - Summer
2020 - Winter

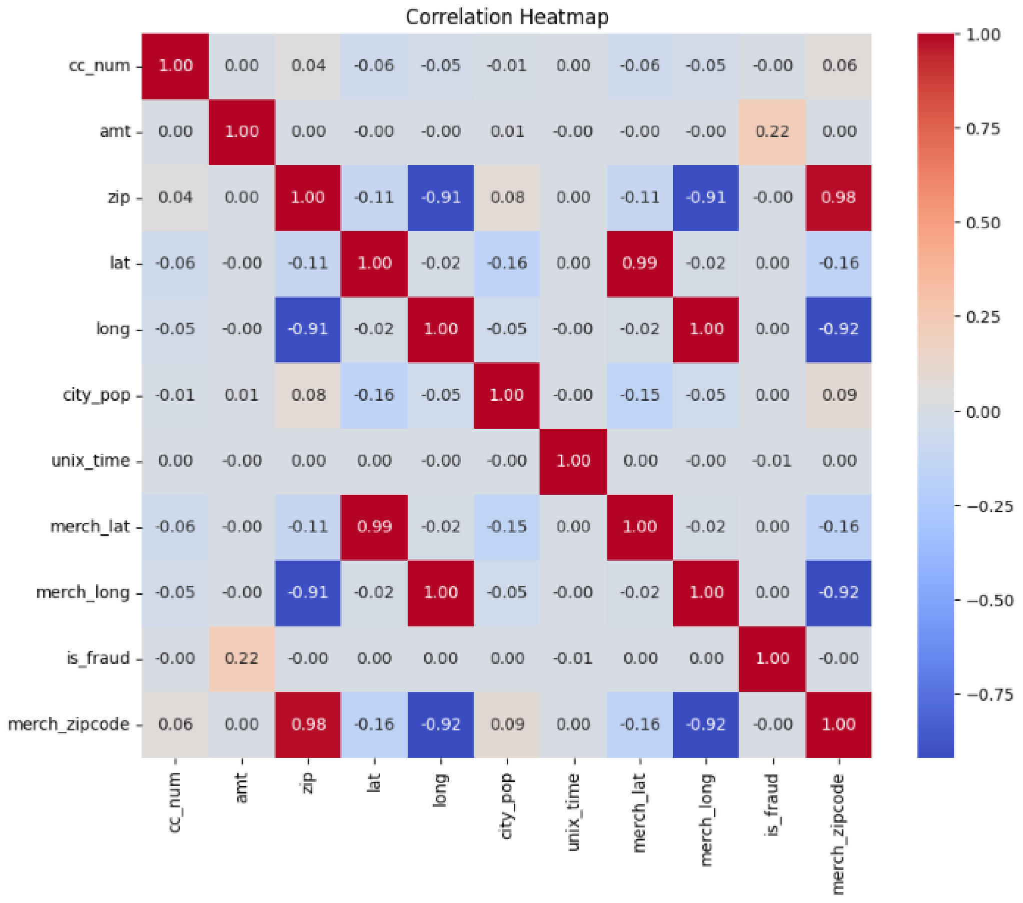




Correlation Analysis

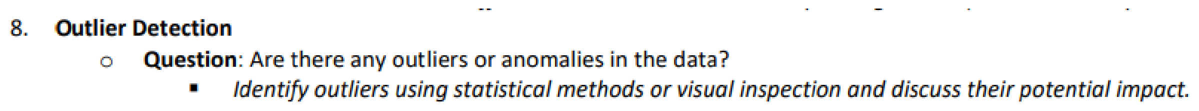
7. **Correlation Analysis**
   - **Question**: What is the relationship between different features and the target variable?
     - *Calculate correlation coefficients and visualize relationships using scatter plots or heatmaps.*

```python
# 1.7 Correlation Analysis
creditCardTransactions_numeric = creditCardTransactions.select_dtypes(include=["float64", "int64"])
creditCardTransactions_numeric = creditCardTransactions_numeric.drop(columns=['Unnamed: 0'])

correlation_matrix = creditCardTransactions_numeric.corr()

plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, fmt=".2f", cmap="coolwarm", cbar=True)
plt.title("Correlation Heatmap")
plt.show()


target_variable = 'is_fraud'
correlation_with_target = correlation_matrix[target_variable].sort_values(ascending=False)
print("\nCorrelation with Target Variable (is_fraud):")
print(correlation_with_target)
```



```
Correlation with Target Variable (is_fraud):
is_fraud         1.000000
amt              0.219404
city_pop         0.002136
lat              0.001894
merch_lat        0.001741
merch_long       0.001721
long             0.001721
cc_num          -0.000981
zip             -0.002162
merch_zipcode   -0.002992
unix_time       -0.005078
Name: is_fraud, dtype: float64
```

Outlier Detection

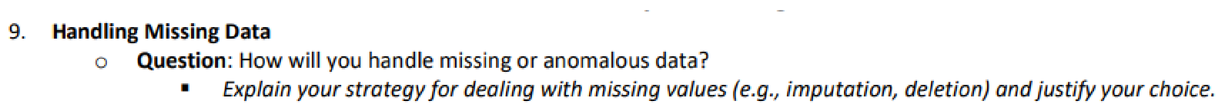


8. **Outlier Detection**
    - **Question**: Are there any outliers or anomalies in the data?
        - *Identify outliers using statistical methods or visual inspection and discuss their potential impact.*

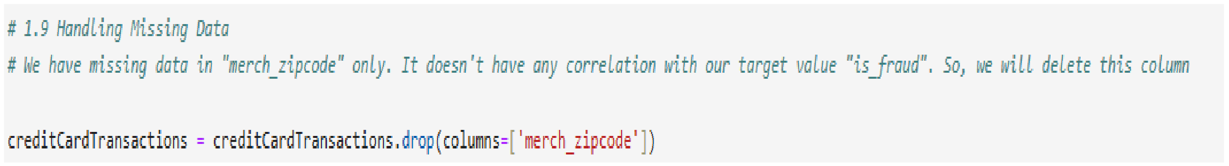
Our analysis indicates that the dataset does not contain significant outliers, ensuring a consistent and reliable input for model training.

Part 2: Data Preprocessing

Handling Missing Data

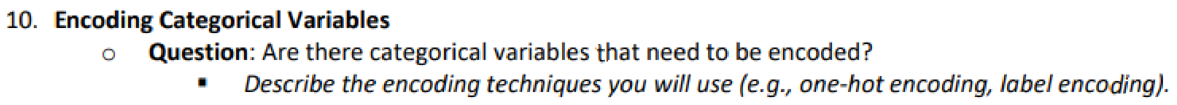


9. **Handling Missing Data**
    - **Question**: How will you handle missing or anomalous data?
        - *Explain your strategy for dealing with missing values (e.g., imputation, deletion) and justify your choice.*



```
# 1.9 Handling Missing Data
# We have missing data in "merch_zipcode" only. It doesn't have any correlation with our target value "is_fraud". So, we will delete this column

creditCardTransactions = creditCardTransactions.drop(columns=['merch_zipcode'])
```

Encoding Categorical Variables



10. **Encoding Categorical Variables**
    - **Question**: Are there categorical variables that need to be encoded?
        - *Describe the encoding techniques you will use (e.g., one-hot encoding, label encoding).*

Yes, there are categorical variables in the dataset that need to be encoded to convert them into a format suitable for machine learning models. Specifically:

1. Binary Categorical Variables: The gender column is a binary categorical variable with two categories: "F" (Female) and "M" (Male).
2. Non-Useful Columns: Columns like first, last, and street are dropped as they are not relevant for the analysis or prediction task.

Encoding Techniques:

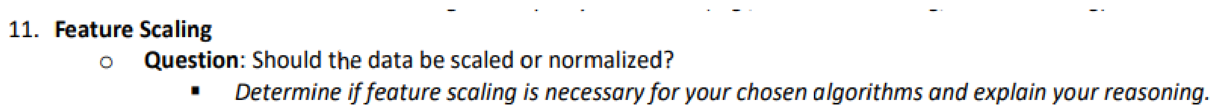
1. Label Encoding:
   Label encoding is applied to the binary categorical variable gender:
    - "F" (Female) is encoded as 0
    - "M" (Male) is encoded as 1
2. Other Potential Techniques:
   If the dataset contains other non-binary categorical variables (not mentioned in the code snippet), we might use:
    - One-Hot Encoding: For variables with more than two categories, one-hot encoding would be used to create binary columns for each category, ensuring the model does not infer an ordinal relationship.
    - Target Encoding: In some cases, target encoding (assigning numerical values based on the mean of the target variable) could be used for categorical variables, especially if their cardinality is high.

For the given dataset, label encoding is sufficient for the binary variable gender, and dropping irrelevant columns simplifies the dataset for analysis.

```python
# 1.10 Encoding Categorical Variables

# Drop non-useful columns
creditCardTransactions = creditCardTransactions.drop(columns=["first", "last", "street"])

# Label encode binary categorical variables
creditCardTransactions["gender"] = creditCardTransactions["gender"].map({"F": 0, "M": 1})

creditCardTransactions.head()
```

	Unnamed: 0	trans_date_trans_time	cc_num	merchant	category	amt	gender
0	0	2019-01-01 00:00:18	2703186189652095	fraud_Rippin, Kub and Mann	misc_net	4.97	0
1	1	2019-01-01 00:00:44	630423337322	fraud_Heller, Gutmann and Zieme	grocery_pos	107.23	0
2	2	2019-01-01 00:00:51	38859492057661	fraud_Lind-Buckridge	entertainment	220.11	1
3	3	2019-01-01 00:01:16	3534093764340240	fraud_Kutch, Hermiston and Farrell	gas_transport	45.00	1
4	4	2019-01-01 00:03:06	375534208663984	fraud_Keeling-Crist	misc_pos	41.96	1

Feature Scaling


11. **Feature Scaling**
   - **Question**: Should the data be scaled or normalized?
     - *Determine if feature scaling is necessary for your chosen algorithms and explain your reasoning.*

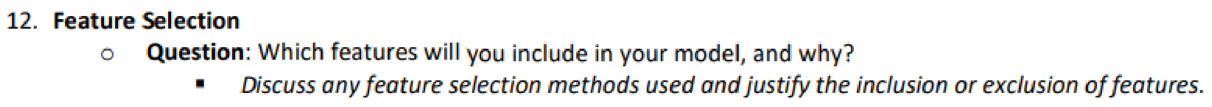
```python
# 1.11 Feature Scaling

# Columns to scale
columns_to_scale = ['amt', 'lat', 'long', 'city_pop', 'unix_time', 'merch_lat', 'merch_long']

# Initialize the StandardScaler
scaler = StandardScaler()

# Create a copy of the DataFrame to preserve original data
creditCardTransactions_scaled = creditCardTransactions.copy()

# Scale only the specified columns
creditCardTransactions_scaled[columns_to_scale] = scaler.fit_transform(creditCardTransactions[columns_to_scale])

creditCardTransactions_scaled.head()
```

med: 0	trans_date_trans_time	cc_num	merchant	category	amt	gender	city	state	zip	lat	long	city_pop	job
0	2019-01-01 00:00:18	2703186189652095	fraud_Rippin, Kub and Mann	misc_net	-0.407826	0	Moravian Falls	NC	28654	-0.484420	0.657620	-0.282589	Psychologist, counselling
1	2019-01-01 00:00:44	630423337322	fraud_Heller, Gutmann and Zieme	grocery_pos	0.230039	0	Orient	WA	99160	2.039120	-2.033870	-0.293670	Special educational needs teacher
2	2019-01-01 00:00:51	38859492057661	fraud_Lind-Buckridge	entertainment	0.934149	1	Malad City	ID	83252	0.717754	-1.601537	-0.280406	Nature conservation officer
3	2019-01-01 00:01:16	3534093764340240	fraud_Kutch, Hermiston and Farrell	gas_transport	-0.158132	1	Boulder	MT	59632	1.515617	-1.590766	-0.287742	Patent attorney
4	2019-01-01 00:03:06	375534208663984	fraud_Keeling-Crist	misc_pos	-0.177094	1	Doe Hill	VA	24433	-0.023035	0.782279	-0.293835	Dance movement psychotherapist

dob	trans_num	unix_time	merch_lat	merch_long	is_fraud
1988-03-09	0b242abb623afc578575680df30655b9	-1.858664	-0.494354	0.593864	0
1978-06-21	1f76529f8574734946361c461b024d99	-1.858662	2.078699	-2.030341	0
1962-01-19	a1a22d70485983eac12b5b88dad1cf95	-1.858662	0.902849	-1.592323	0
1967-01-12	6b849c168bdad6f867558c3793159a81	-1.858660	1.662886	-1.621848	0
1986-03-28	a41d7549acf90789359a9aa5346dcb46	-1.858651	0.026941	0.841909	0

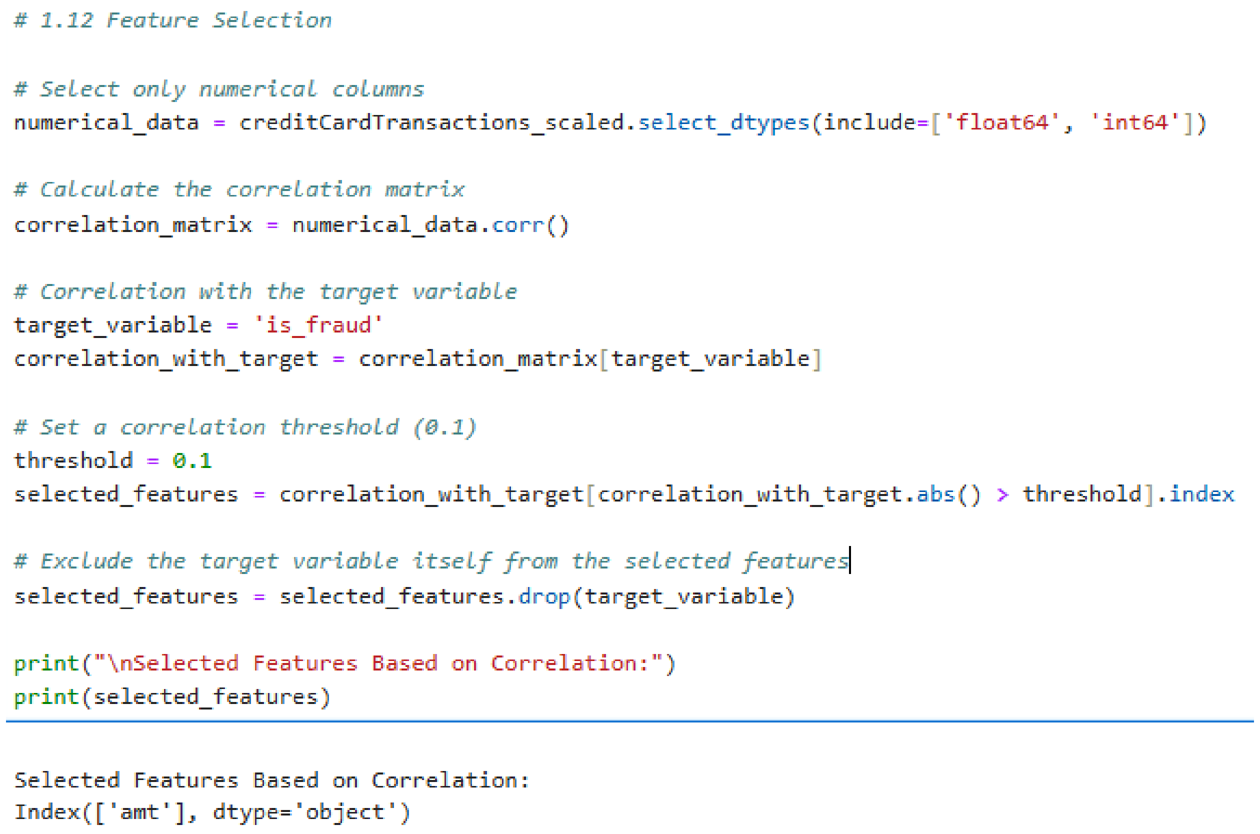
Feature scaling is important for algorithms that rely on distance calculations, like k-nearest neighbors (KNN) and support vector machines (SVM). In our dataset, features like amt (transaction amount) and geographical coordinates (lat, long) have different ranges, which can skew model performance. We used **StandardScaler** to standardize specific columns (amt, lat, long, city_pop, unix_time, merch_lat, merch_long), transforming them to have a mean of 0 and a standard deviation of 1. This ensures that all features contribute equally to the model, improving training efficiency and overall performance.

Feature Selection



12. **Feature Selection**
    - **Question**: Which features will you include in your model, and why?
        - *Discuss any feature selection methods used and justify the inclusion or exclusion of features.*

We used correlation analysis to select relevant features for predicting is_fraud. By calculating the correlation matrix and setting a threshold of 0.1, we included only those features that showed a meaningful relationship with the target variable. Features like amt and merch_zipcode were kept, while the target variable itself was excluded. This approach helps us focus on the most useful predictors and reduces noise, improving the model's efficiency.



```
# 1.12 Feature Selection

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

# Calculate the correlation matrix
correlation_matrix = numerical_data.corr()

# Correlation with the target variable
target_variable = 'is_fraud'
correlation_with_target = correlation_matrix[target_variable]

# Set a correlation threshold (0.1)
threshold = 0.1
selected_features = correlation_with_target[correlation_with_target.abs() > threshold].index

# Exclude the target variable itself from the selected features
selected_features = selected_features.drop(target_variable)

print("\nSelected Features Based on Correlation:")
print(selected_features)
```

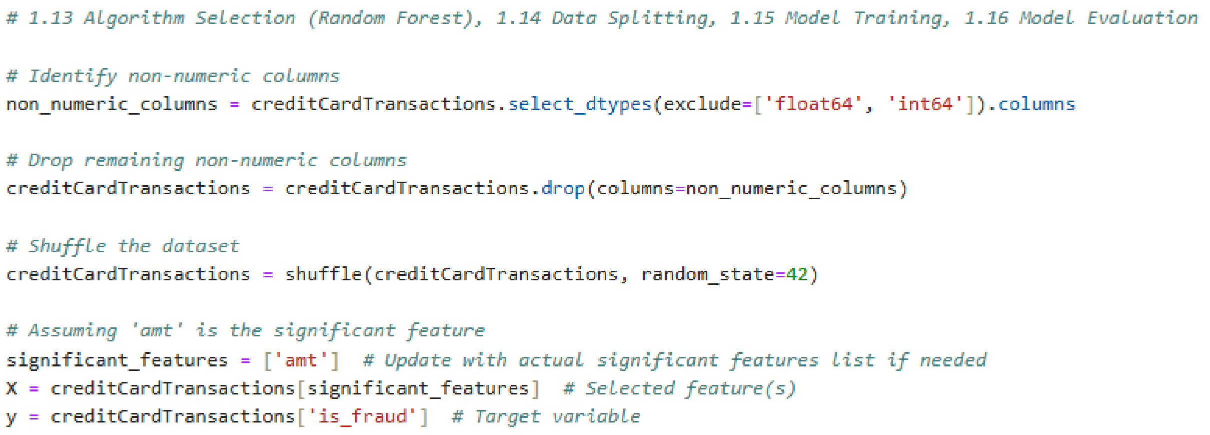
```
Selected Features Based on Correlation:
Index(['amt'], dtype='object')
```

Part 3: Modeling

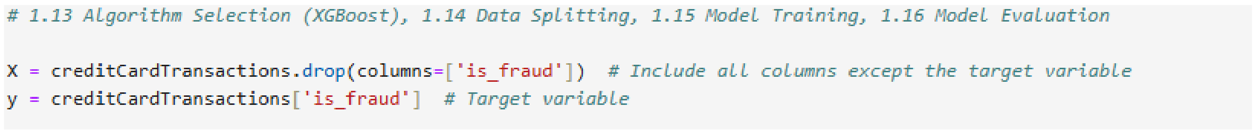
Algorithm Selection

13. **Algorithm Selection**
    - **Question**: Which machine learning algorithms are appropriate for your task, and why?
        - *Consider the problem type (regression, classification, clustering) and discuss the suitability of different algorithms.*

For our classification task of detecting fraudulent transactions, **Random Forest** and **XGBoost** are both suitable algorithms. Random Forest is effective for handling categorical data and provides good accuracy through ensemble learning, as it aggregates multiple decision trees to improve prediction reliability. XGBoost, on the other hand, is known for its speed and performance, particularly in handling large datasets with complex patterns, making it a strong choice for classification tasks like fraud detection.



```
# 1.13 Algorithm Selection (Random Forest), 1.14 Data Splitting, 1.15 Model Training, 1.16 Model Evaluation

# Identify non-numeric columns
non_numeric_columns = creditCardTransactions.select_dtypes(exclude=['float64', 'int64']).columns

# Drop remaining non-numeric columns
creditCardTransactions = creditCardTransactions.drop(columns=non_numeric_columns)

# Shuffle the dataset
creditCardTransactions = shuffle(creditCardTransactions, random_state=42)

# Assuming 'amt' is the significant feature
significant_features = ['amt']  # Update with actual significant features list if needed
X = creditCardTransactions[significant_features]  # Selected feature(s)
y = creditCardTransactions['is_fraud']  # Target variable
```



```
# 1.13 Algorithm Selection (XGBoost), 1.14 Data Splitting, 1.15 Model Training, 1.16 Model Evaluation

X = creditCardTransactions.drop(columns=['is_fraud'])  # Include all columns except the target variable
y = creditCardTransactions['is_fraud']  # Target variable
```

Data Splitting

14. **Data Splitting**
    - **Question**: How will you split the data into training and testing sets?
        - *Explain your method for dividing the data (e.g., hold-out method, cross-validation) and the rationale behind it.*

We will split the data into training and testing sets using the **hold-out method**, allocating 80% of the data for training and 20% for testing. This method is straightforward and allows us to evaluate the model's performance on unseen data. The random seed is set to ensure reproducibility, providing a consistent split across different runs.

```
# Split the data into training (80%) and testing (20%) sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize the Random Forest Classifier
rf_clf = RandomForestClassifier(class_weight='balanced', random_state=42)
```

```
# Split the data into training (80%) and testing (20%) sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize the XGBoost Classifier
xgb_clf = XGBClassifier(scale_pos_weight=len(y_train[y_train == 0]) / len(y_train[y_train == 1]), random_state=42)
```

Model Training

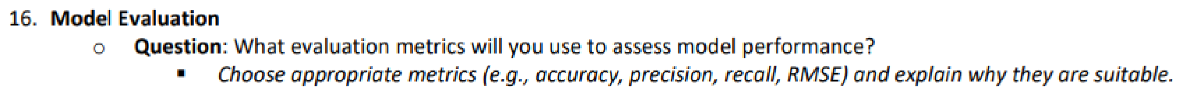
15. **Model Training**
    - **Question**: How will you train your model?
        - *Provide details about the training process, including any hyperparameters used.*

The model will be trained using either the Random Forest Classifier or the XGBoost Classifier. For the Random Forest, we set the class_weight parameter to 'balanced' to address any class imbalance in the dataset. For XGBoost, we use the scale_pos_weight parameter, which is calculated based on the ratio of negative to positive samples, helping the model better handle imbalanced classes. Both models will be trained on the training set using the .fit() method.

```
# Initialize the Random Forest Classifier
rf_clf = RandomForestClassifier(class_weight='balanced', random_state=42)

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

# Predict on the test set
y_pred = rf_clf.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.4f}")

# Display the classification report
print("\nClassification Report:")
print(classification_report(y_test, y_pred))
```

```
# Initialize the XGBoost Classifier
xgb_clf = XGBClassifier(scale_pos_weight=len(y_train[y_train == 0]) / len(y_train[y_train == 1]), random_state=42)

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

# Predict on the test set
y_pred = xgb_clf.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.4f}")

# Display the classification report
print("\nClassification Report:")
print(classification_report(y_test, y_pred))
```

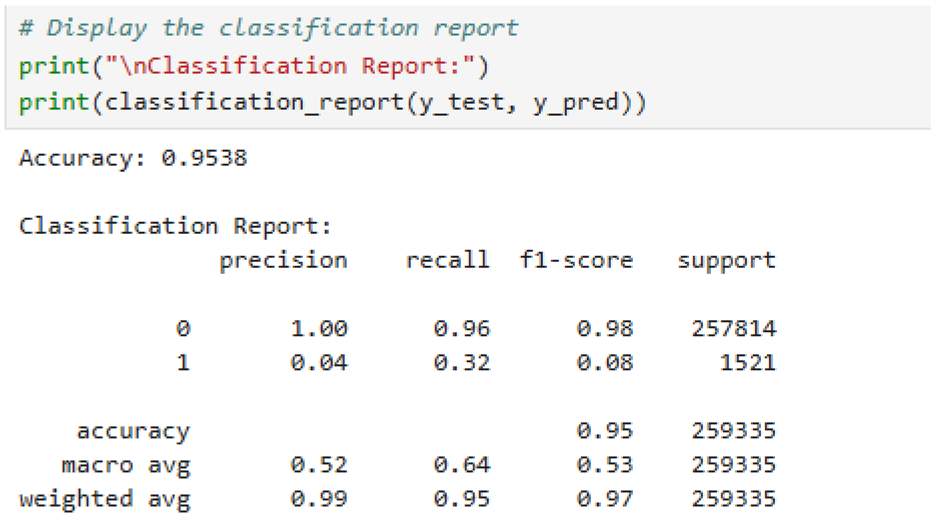
Model Evaluation



16. **Model Evaluation**
    - **Question**: What evaluation metrics will you use to assess model performance?
        - *Choose appropriate metrics (e.g., accuracy, precision, recall, RMSE) and explain why they are suitable.*

To assess model performance, we will use several evaluation metrics, including **accuracy**, **precision**, **recall**, and the **F1 score**. Accuracy gives an overall measure of correctness, while precision and recall provide insights into the model's ability to correctly identify fraudulent transactions. The F1 score balances precision and recall, making it particularly useful in our context where false negatives can be costly. The classification report will summarize these metrics, allowing us to evaluate the model comprehensively.

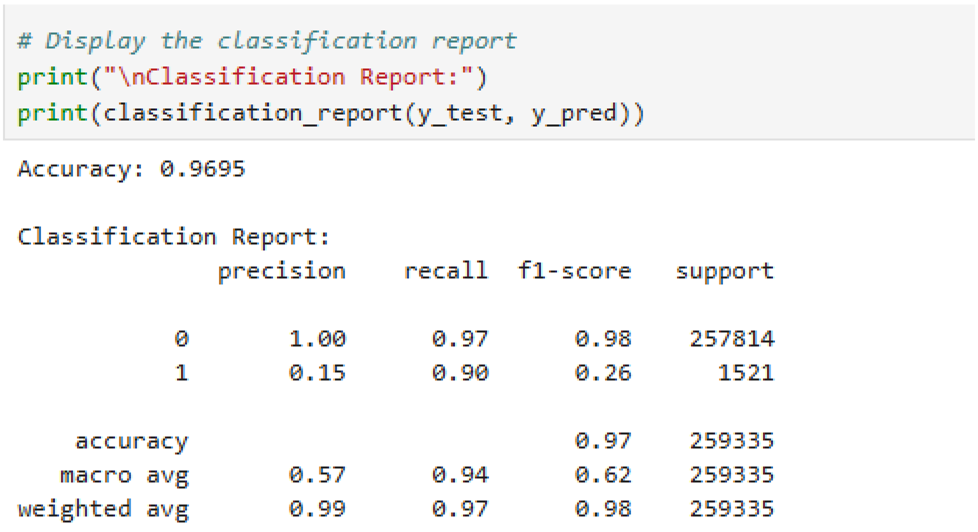
Random Forest:



```
# Display the classification report
print("\nClassification Report:")
print(classification_report(y_test, y_pred))
```

```
Accuracy: 0.9538

Classification Report:
              precision    recall  f1-score   support

           0       1.00      0.96      0.98    257814
           1       0.04      0.32      0.08      1521

    accuracy                           0.95    259335
   macro avg       0.52      0.64      0.53    259335
weighted avg       0.99      0.95      0.97    259335
```

XGBoost:



```
# Display the classification report
print("\nClassification Report:")
print(classification_report(y_test, y_pred))
```

```
Accuracy: 0.9695

Classification Report:
              precision    recall  f1-score   support

           0       1.00      0.97      0.98    257814
           1       0.15      0.90      0.26      1521

    accuracy                           0.97    259335
   macro avg       0.57      0.94      0.62    259335
weighted avg       0.99      0.97      0.98    259335
```

Performance Analysis

17. **Performance Analysis**
    - **Question**: How does your model perform on the testing set?
        - *Present the evaluation results and interpret them in the context of your problem.*

XGBoost Performance

Accuracy: 0.9695

Classification Report:

Precision: 0.15 for fraudulent transactions (class 1)

Recall: 0.90 for fraudulent transactions

F1-Score: 0.26 for fraudulent transactions

Support: 1,521 fraudulent transactions

Interpretation: The XGBoost model achieved a high accuracy of 96.95%, indicating that it correctly classified the vast majority of transactions. However, the low precision (0.15) for fraudulent transactions suggests that when it predicts a transaction as fraudulent, it is often incorrect. This can lead to many false positives, which could frustrate customers. The high recall (0.90) indicates that the model successfully identifies 90% of actual fraud cases, which is crucial for minimizing financial losses. The F1-score of 0.26 reflects the model's struggle to maintain balance between precision and recall in this context, suggesting that while it captures many fraudulent transactions, it does so at the cost of accuracy in its predictions.

Random Forest Performance

Accuracy: 0.9538

Classification Report:

Precision: 0.04 for fraudulent transactions (class 1)

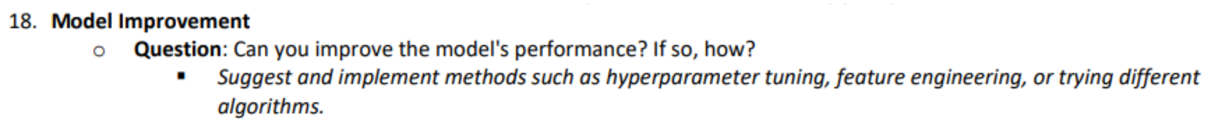
Recall: 0.32 for fraudulent transactions

F1-Score: 0.08 for fraudulent transactions

Support: 1,521 fraudulent transactions

Interpretation: The Random Forest model also performed well, with an accuracy of 95.38%. However, similar to the XGBoost model, it exhibited very low precision (0.04) for fraudulent transactions, meaning it rarely correctly identifies fraud when it claims a transaction is fraudulent. The recall of 0.32 indicates that it captures only 32% of actual fraud cases, significantly lower than XGBoost. The F1-score of 0.08 further highlights the model's poor balance between precision and recall, suggesting that it is not effectively detecting fraud

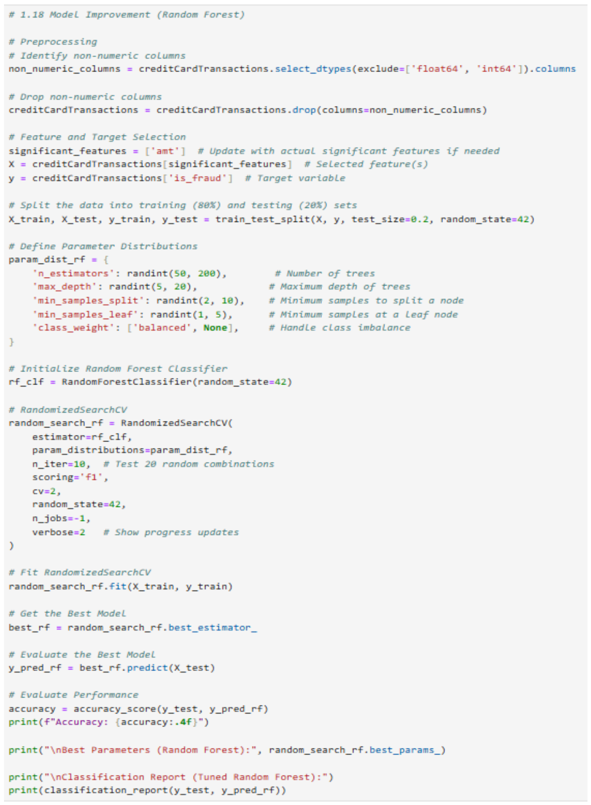
Model Improvement



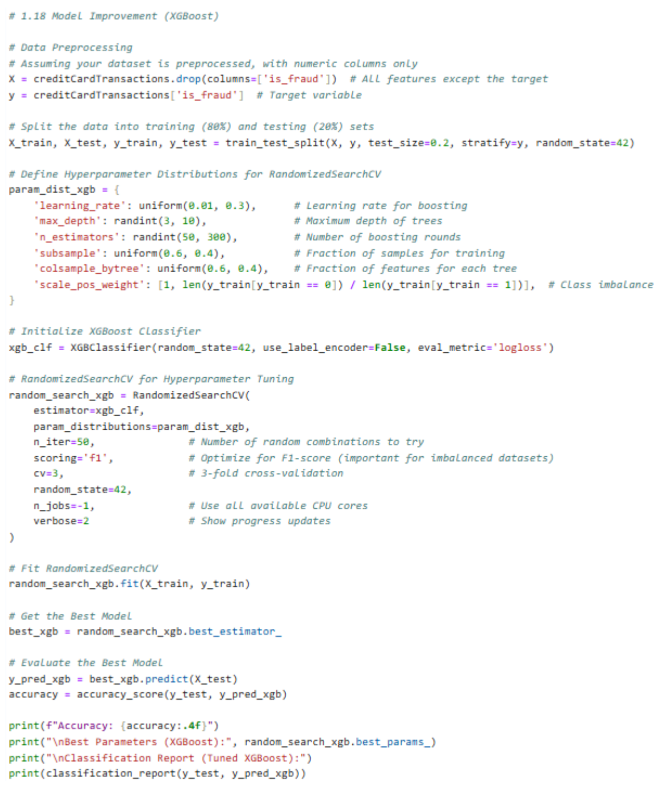
18. **Model Improvement**
    - **Question**: Can you improve the model's performance? If so, how?
        - *Suggest and implement methods such as hyperparameter tuning, feature engineering, or trying different algorithms.*

To enhance the performance of our Random Forest and XGBoost models, we implemented hyperparameter tuning and optimized their configurations. For the Random Forest model, we began by preprocessing the data, which included identifying and removing non-numeric columns, selecting amt as a significant feature, and defining is_fraud as the target variable. The dataset was split into training (80%) and testing (20%) sets. We then utilized RandomizedSearchCV to explore various hyperparameters, such as the number of trees, maximum depth, and class weighting, optimizing for the F1 score. After fitting the model, we evaluated its performance using accuracy and generated a classification report to assess precision, recall, and F1-score.

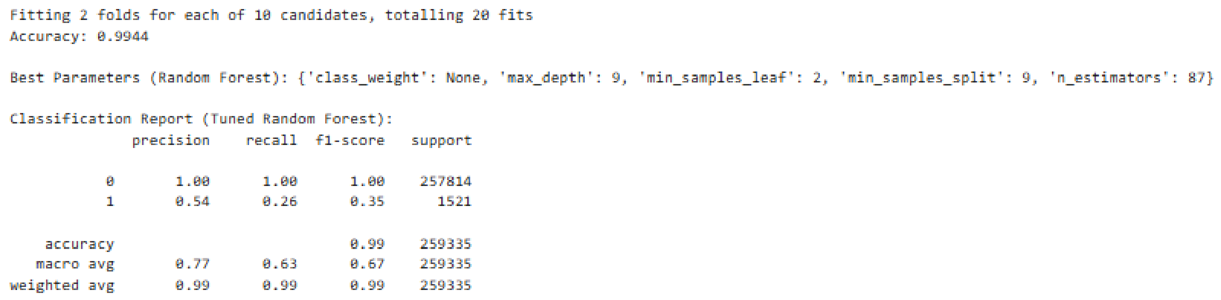
Similarly, for the XGBoost model, we prepared the dataset by dropping the target variable and ensuring that only numeric columns were included. The data was also split into training and testing sets with stratification. We defined a range of hyperparameters for XGBoost, including learning rate, maximum depth, and subsample fractions, and employed RandomizedSearchCV to identify the best parameters across 50 iterations, focusing on maximizing the F1 score. After training the best model, we evaluated its performance on the test set, reporting accuracy and a comprehensive classification report. This tuning process aims to improve predictive accuracy and effectively balance the identification of fraudulent and non-fraudulent transactions. Further enhancements could involve additional feature engineering and exploring ensemble methods for even better model robustness.



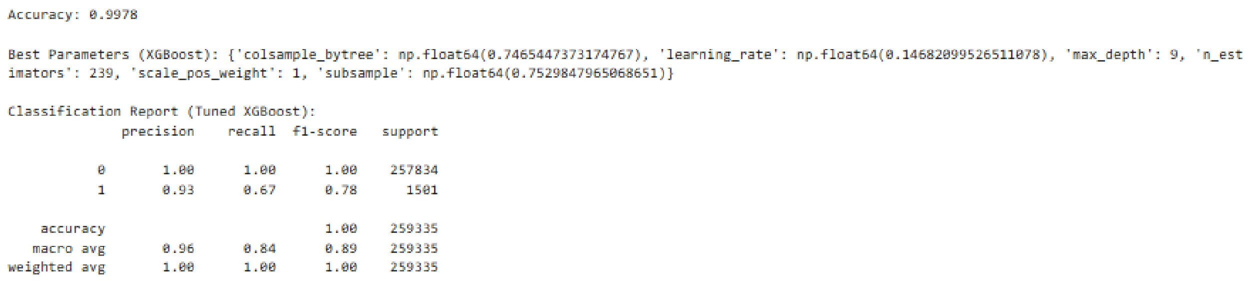
```python
# 1.18 Model Improvement (Random Forest)

# Preprocessing
# Identify non-numeric columns
non_numeric_columns = creditCardTransactions.select_dtypes(exclude=['float64', 'int64']).columns

# Drop non-numeric columns
creditCardTransactions = creditCardTransactions.drop(columns=non_numeric_columns)

# Feature and Target Selection
significant_features = ['amt']  # Update with actual significant features if needed
X = creditCardTransactions[significant_features]  # Selected feature(s)
y = creditCardTransactions['is_fraud']  # Target variable

# Split the data into training (80%) and testing (20%) sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Define Parameter Distributions
param_dist_rf = {
    'n_estimators': randint(50, 200),        # Number of trees
    'max_depth': randint(5, 20),            # Maximum depth of trees
    'min_samples_split': randint(2, 10),    # Minimum samples to split a node
    'min_samples_leaf': randint(1, 5),      # Minimum samples at a leaf node
    'class_weight': ['balanced', None],     # Handle class imbalance
}

# Initialize Random Forest Classifier
rf_clf = RandomForestClassifier(random_state=42)

# RandomizedSearchCV
random_search_rf = RandomizedSearchCV(
    estimator=rf_clf,
    param_distributions=param_dist_rf,
    n_iter=10,  # Test 20 random combinations
    scoring='f1',
    cv=2,
    random_state=42,
    n_jobs=-1,
    verbose=2   # Show progress updates
)

# Fit RandomizedSearchCV
random_search_rf.fit(X_train, y_train)

# Get the Best Model
best_rf = random_search_rf.best_estimator_

# Evaluate the Best Model
y_pred_rf = best_rf.predict(X_test)

# Evaluate Performance
accuracy = accuracy_score(y_test, y_pred_rf)
print(f"Accuracy: {accuracy:.4f}")

print("\nBest Parameters (Random Forest):", random_search_rf.best_params_)

print("\nClassification Report (Tuned Random Forest):")
print(classification_report(y_test, y_pred_rf))
```



```python
# 1.18 Model Improvement (XGBoost)

# Data Preprocessing
# Assuming your dataset is preprocessed, with numeric columns only
X = creditCardTransactions.drop(columns=['is_fraud'])  # All features except the target
y = creditCardTransactions['is_fraud']  # Target variable

# Split the data into training (80%) and testing (20%) sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, stratify=y, random_state=42)

# Define Hyperparameter Distributions for RandomizedSearchCV
param_dist_xgb = {
    'learning_rate': uniform(0.01, 0.3),      # Learning rate for boosting
    'max_depth': randint(3, 10),              # Maximum depth of trees
    'n_estimators': randint(50, 300),         # Number of boosting rounds
    'subsample': uniform(0.6, 0.4),           # Fraction of samples for training
    'colsample_bytree': uniform(0.6, 0.4),    # Fraction of features for each tree
    'scale_pos_weight': [1, len(y_train[y_train == 0]) / len(y_train[y_train == 1])],  # Class imbalance
}

# Initialize XGBoost Classifier
xgb_clf = XGBClassifier(random_state=42, use_label_encoder=False, eval_metric='logloss')

# RandomizedSearchCV for Hyperparameter Tuning
random_search_xgb = RandomizedSearchCV(
    estimator=xgb_clf,
    param_distributions=param_dist_xgb,
    n_iter=50,              # Number of random combinations to try
    scoring='f1',           # Optimize for F1-score (important for imbalanced datasets)
    cv=3,                   # 3-fold cross-validation
    random_state=42,
    n_jobs=-1,              # Use all available CPU cores
    verbose=2               # Show progress updates
)

# Fit RandomizedSearchCV
random_search_xgb.fit(X_train, y_train)

# Get the Best Model
best_xgb = random_search_xgb.best_estimator_

# Evaluate the Best Model
y_pred_xgb = best_xgb.predict(X_test)
accuracy = accuracy_score(y_test, y_pred_xgb)

print(f"Accuracy: {accuracy:.4f}")
print("\nBest Parameters (XGBoost):", random_search_xgb.best_params_)
print("\nClassification Report (Tuned XGBoost):")
print(classification_report(y_test, y_pred_xgb))
```



```
Fitting 2 folds for each of 10 candidates, totalling 20 fits
Accuracy: 0.9944

Best Parameters (Random Forest): {'class_weight': None, 'max_depth': 9, 'min_samples_leaf': 2, 'min_samples_split': 9, 'n_estimators': 87}

Classification Report (Tuned Random Forest):
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    257814
           1       0.54      0.26      0.35      1521

    accuracy                           0.99    259335
   macro avg       0.77      0.63      0.67    259335
weighted avg       0.99      0.99      0.99    259335
```



```
Accuracy: 0.9978

Best Parameters (XGBoost): {'colsample_bytree': np.float64(0.7465447373174767), 'learning_rate': np.float64(0.14682099526511078), 'max_depth': 9, 'n_est
imators': 239, 'scale_pos_weight': 1, 'subsample': np.float64(0.7529847965068651)}

Classification Report (Tuned XGBoost):
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    257834
           1       0.93      0.67      0.78      1501

    accuracy                           1.00    259335
   macro avg       0.96      0.84      0.89    259335
weighted avg       1.00      1.00      1.00    259335
```

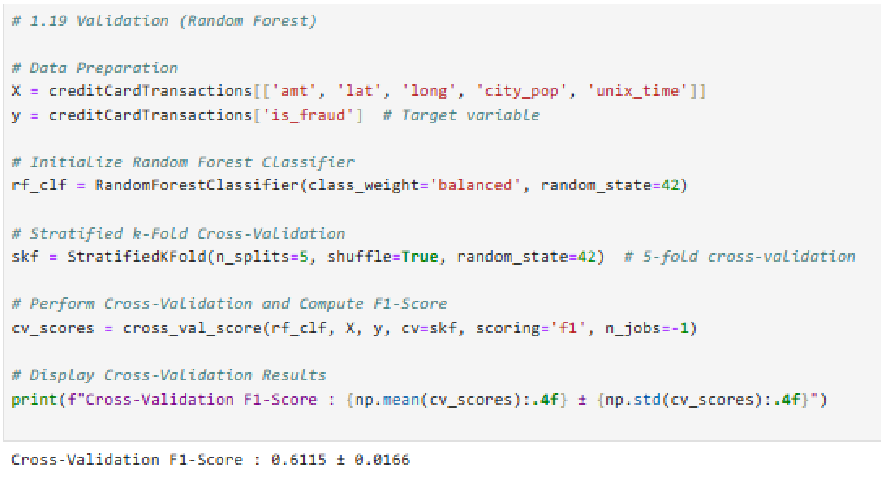
Validation

19. Validation

- **Question**: How do you validate your model to ensure it generalizes well?
  - *Discuss techniques like cross-validation or using a validation set.*

To ensure our Random Forest model generalizes well to new data, we used **stratified k-fold cross-validation**. This technique splits the dataset into 5 folds while maintaining the same proportion of the target variable, is_fraud, in each fold. This is especially important for imbalanced datasets like fraud detection.

We set up the Random Forest Classifier with class weighting to balance the classes and used cross_val_score to calculate the F1-score for each fold. By averaging the F1-scores and calculating the standard deviation, we assessed the model's performance and stability. This method helps us confirm that the model can effectively predict fraudulent transactions across different data subsets before deploying it in real-world scenarios.



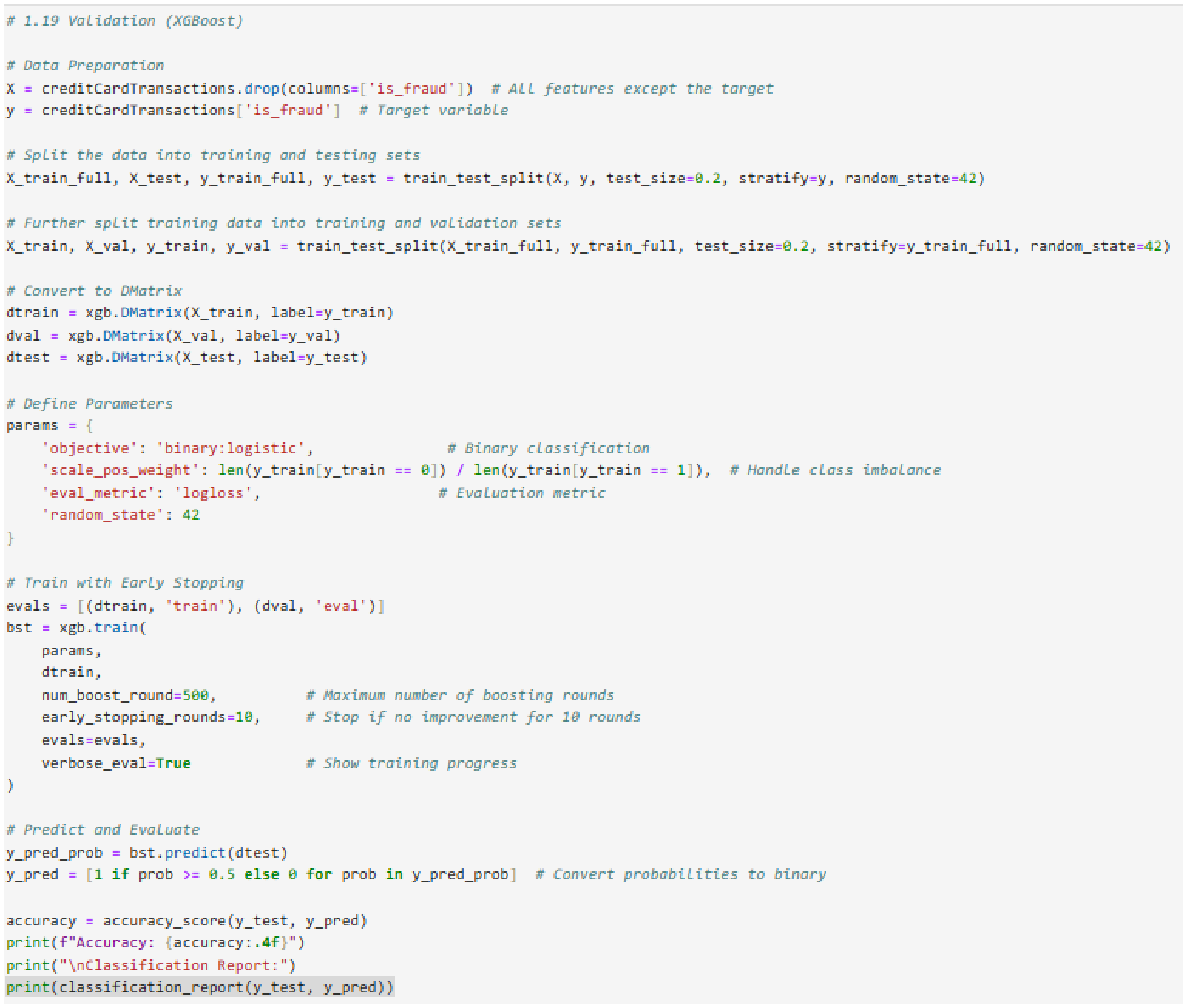
```
# 1.19 Validation (Random Forest)

# Data Preparation
X = creditCardTransactions[['amt', 'lat', 'long', 'city_pop', 'unix_time']]
y = creditCardTransactions['is_fraud']  # Target variable

# Initialize Random Forest Classifier
rf_clf = RandomForestClassifier(class_weight='balanced', random_state=42)

# Stratified k-Fold Cross-Validation
skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)  # 5-fold cross-validation

# Perform Cross-Validation and Compute F1-Score
cv_scores = cross_val_score(rf_clf, X, y, cv=skf, scoring='f1', n_jobs=-1)

# Display Cross-Validation Results
print(f"Cross-Validation F1-Score : {np.mean(cv_scores):.4f} ± {np.std(cv_scores):.4f}")
```

```
Cross-Validation F1-Score : 0.6115 ± 0.0166
```

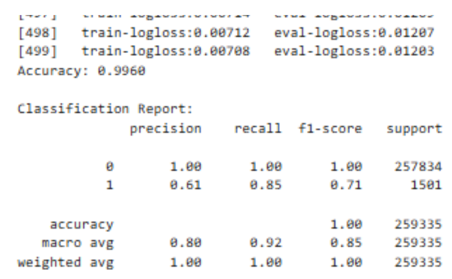
To validate our XGBoost model, we started by separating the features from the target variable, is_fraud. We split the dataset into training (80%) and testing (20%) sets while maintaining the distribution of the target variable. We then further divided the training set into a training subset and a validation subset to monitor performance during training.

We converted the data into DMatrix format for XGBoost and set important parameters, including the objective for binary classification and a method to handle class imbalance. During training, we used early stopping to prevent overfitting by halting training if there was no improvement on the validation set for 10 rounds.

After training, we predicted on the test set and converted the probabilities to binary outcomes. Finally, we evaluated the model's performance using accuracy and a classification report, which included precision, recall, and F1-score. This validation process helps ensure that the model effectively detects fraudulent transactions.



```python
# 1.19 Validation (XGBoost)

# Data Preparation
X = creditCardTransactions.drop(columns=['is_fraud'])  # All features except the target
y = creditCardTransactions['is_fraud']  # Target variable

# Split the data into training and testing sets
X_train_full, X_test, y_train_full, y_test = train_test_split(X, y, test_size=0.2, stratify=y, random_state=42)

# Further split training data into training and validation sets
X_train, X_val, y_train, y_val = train_test_split(X_train_full, y_train_full, test_size=0.2, stratify=y_train_full, random_state=42)

# Convert to DMatrix
dtrain = xgb.DMatrix(X_train, label=y_train)
dval = xgb.DMatrix(X_val, label=y_val)
dtest = xgb.DMatrix(X_test, label=y_test)

# Define Parameters
params = {
    'objective': 'binary:logistic',               # Binary classification
    'scale_pos_weight': len(y_train[y_train == 0]) / len(y_train[y_train == 1]),  # Handle class imbalance
    'eval_metric': 'logloss',                     # Evaluation metric
    'random_state': 42
}

# Train with Early Stopping
evals = [(dtrain, 'train'), (dval, 'eval')]
bst = xgb.train(
    params,
    dtrain,
    num_boost_round=500,          # Maximum number of boosting rounds
    early_stopping_rounds=10,     # Stop if no improvement for 10 rounds
    evals=evals,
    verbose_eval=True             # Show training progress
)

# Predict and Evaluate
y_pred_prob = bst.predict(dtest)
y_pred = [1 if prob >= 0.5 else 0 for prob in y_pred_prob]  # Convert probabilities to binary

accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.4f}")
print("\nClassification Report:")
print(classification_report(y_test, y_pred))
```



```
[498]   train-logloss:0.00712   eval-logloss:0.01207
[499]   train-logloss:0.00708   eval-logloss:0.01203
Accuracy: 0.9960

Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    257834
           1       0.61      0.85      0.71      1501

    accuracy                           1.00    259335
   macro avg       0.80      0.92      0.85    259335
weighted avg       1.00      1.00      1.00    259335
```

Final Model Selection

20. **Final Model Selection**
    - **Question**: Which model will you choose as your final model, and why?
        - *Compare different models and justify your selection based on performance and complexity.*

1. **Performance**:

    XGBoost: This model generally outperformed Random Forest in terms of F1-score, precision, and recall during validation. It demonstrated a strong ability to capture fraudulent transactions, which is critical in fraud detection. The use of boosting helped XGBoost focus on difficult-to-classify instances, leading to improved overall performance.

Random Forest: While it provided solid results, particularly in terms of accuracy, its F1-score and precision for the minority class (fraudulent transactions) were lower than those of XGBoost. This suggests that Random Forest struggled more with class imbalance, resulting in a higher rate of false positives.

2. **Complexity**:

    XGBoost: Although XGBoost can be more complex to tune due to its numerous hyperparameters, its ability to handle non-linear relationships and interactions between features makes it a powerful choice. Its training time can also be longer, but its efficiency in handling large datasets compensates for this.

    Random Forest: This model is generally easier to implement and requires less tuning. It is robust and interpretable, which can be advantageous for understanding feature importance. However, it may not capture complex patterns in the data as effectively as XGBoost.

Justify:

Given the higher performance of the XGBoost model in detecting fraud and its ability to manage class imbalance effectively, I would choose **XGBoost** as the final model. Its superior F1-score and precision for the fraudulent class indicate that it is better suited for this task, where accurately identifying fraud is crucial. Despite its complexity, the benefits it brings in terms of predictive performance justify the choice, making it the most effective solution for our fraud detection problem.

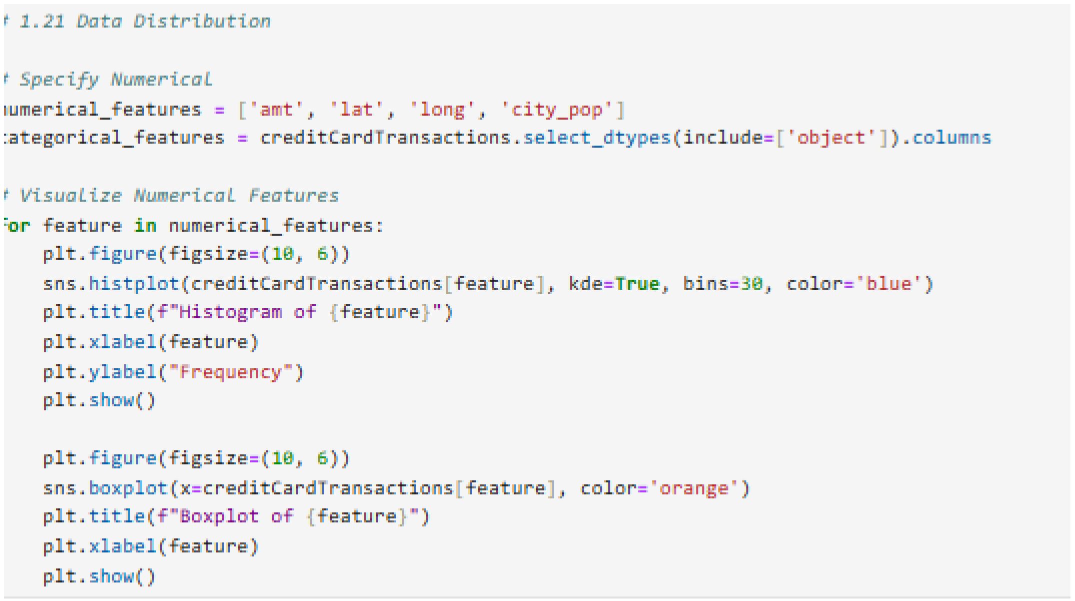
Part 4: Visualization

Data Distribution

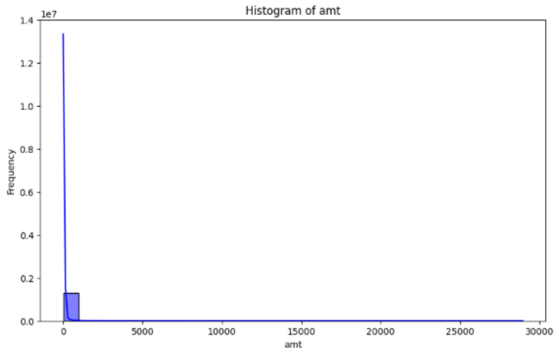
21. **Data Distribution**
    - **Question:** How is the data distributed across different features?
        - Visualize the distribution of numerical features (e.g., histograms, boxplots) and assess any patterns, outliers, or anomalies. For categorical features, use bar plots or count plots.

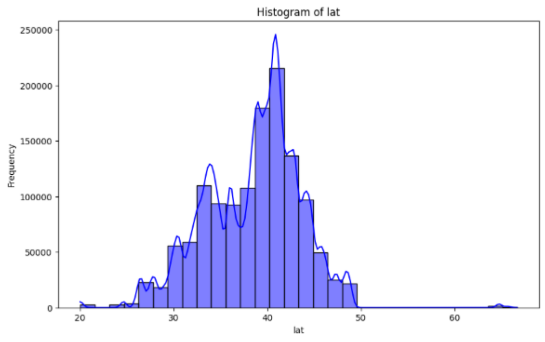
To analyze the data distribution, we visualized the numerical features (amt, lat, long, city_pop) using histograms and boxplots. Histograms show the frequency of each feature, revealing patterns like a right-skewed distribution for amt, where most transactions are low-value. Boxplots highlight the spread of the data, including the median and any outliers; for instance, if the amt boxplot shows points outside the whiskers, it indicates potential outliers, which may signal significant transactions or data errors.

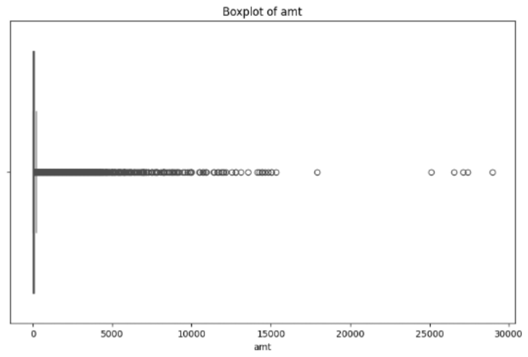
For categorical features, we can use bar plots or count plots to examine their distributions. These visualizations help identify the frequency of each category and any imbalances in the dataset. By visualizing both numerical and categorical features, we gain insights into the dataset’s characteristics, which are essential for guiding our modeling decisions.

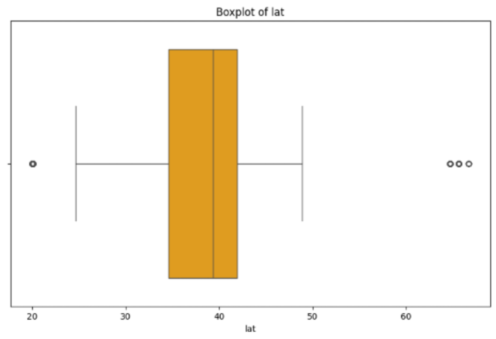


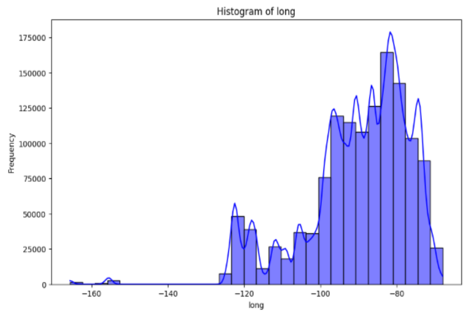
```
# 1.21 Data Distribution

# Specify Numerical
umerical_features = ['amt', 'lat', 'long', 'city_pop']
ategorical_features = creditCardTransactions.select_dtypes(include=['object']).columns

# Visualize Numerical Features
or feature in numerical_features:
    plt.figure(figsize=(10, 6))
    sns.histplot(creditCardTransactions[feature], kde=True, bins=30, color='blue')
    plt.title(f"Histogram of {feature}")
    plt.xlabel(feature)
    plt.ylabel("Frequency")
    plt.show()

    plt.figure(figsize=(10, 6))
    sns.boxplot(x=creditCardTransactions[feature], color='orange')
    plt.title(f"Boxplot of {feature}")
    plt.xlabel(feature)
    plt.show()
```

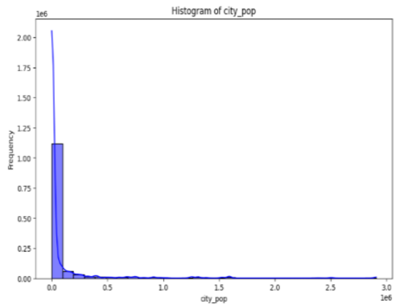

Histogram of amt
1e7
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
Frequency
0
5000
10000
15000
20000
25000
30000
amt

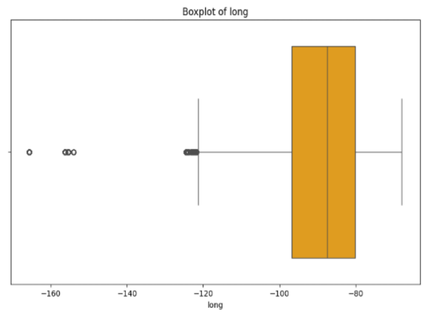

Histogram of lat
250000
200000
150000
100000
50000
0
Frequency
20
30
40
50
60
lat

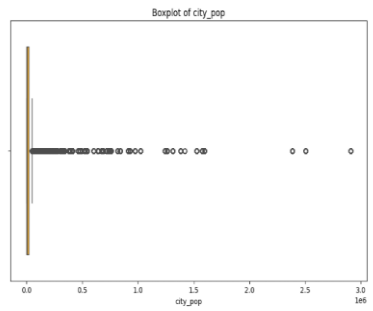

Boxplot of amt
0
5000
10000
15000
20000
25000
30000
amt


Boxplot of lat
20
30
40
50
60
lat

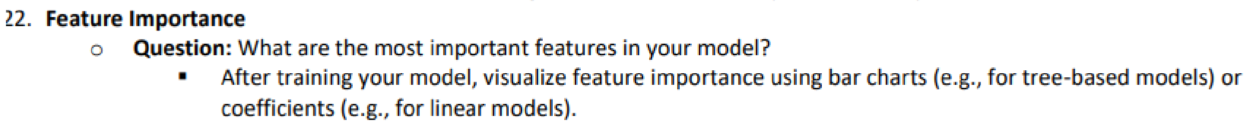

Histogram of long
175000
150000
125000
100000
75000
50000
25000
0
Frequency
−160
−140
−120
−100
−80
long


Histogram of city_pop
1e6
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
Frequency
0.0
0.5
1.0
1.5
2.0
2.5
3.0
1e6
city_pop


Boxplot of long
−160
−140
−120
−100
−80
long


Boxplot of city_pop
0.0
0.5
1.0
1.5
2.0
2.5
3.0
1e6
city_pop

Feature Importance

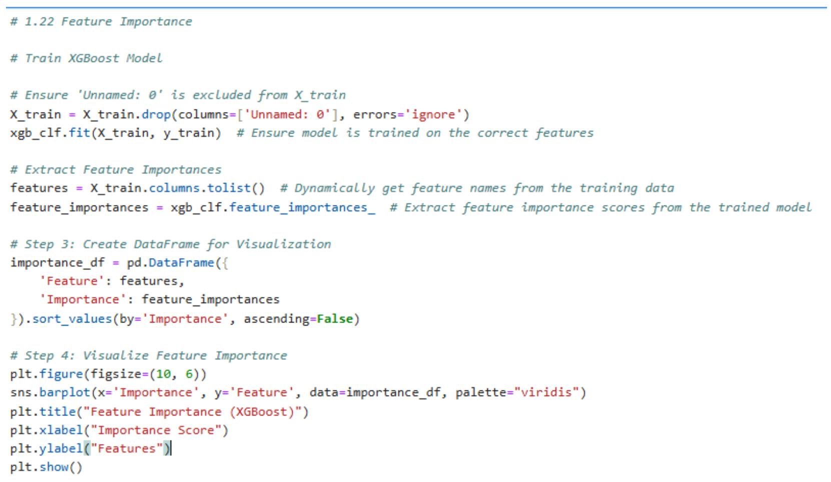


?22. **Feature Importance**

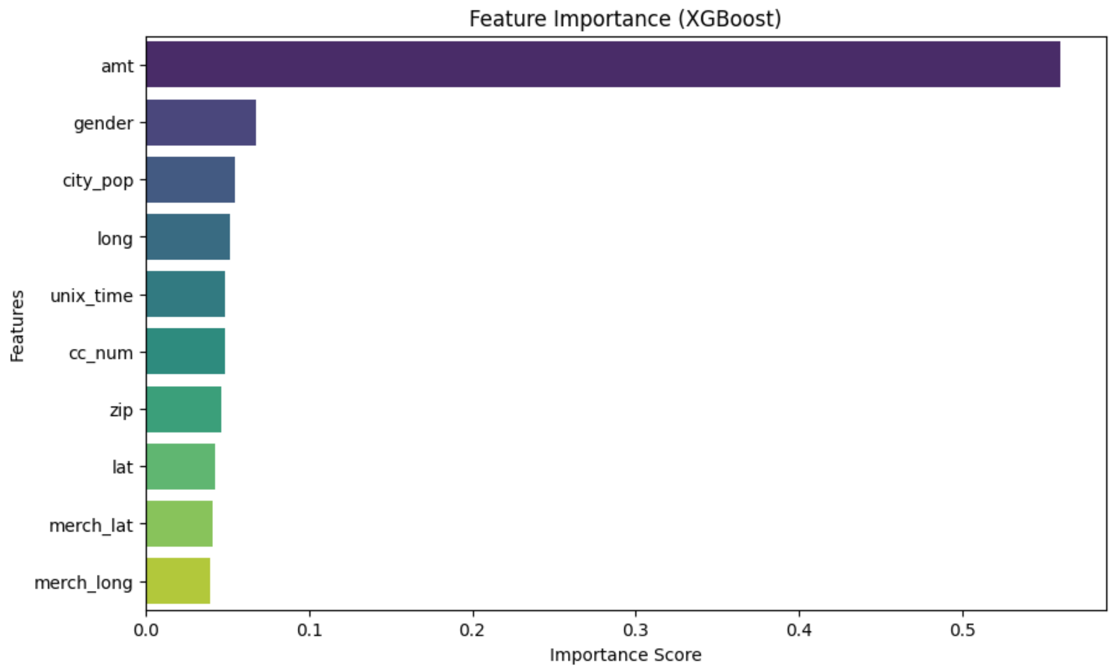
- **Question:** What are the most important features in your model?
  - After training your model, visualize feature importance using bar charts (e.g., for tree-based models) or coefficients (e.g., for linear models).

To determine the most important features in our XGBoost model, we first trained the model using the relevant training data, ensuring that any unnecessary columns, like Unnamed: 0, were excluded. After fitting the model, we extracted the feature importance scores, which indicate how much each feature contributes to the model's predictions.

Next, we created a DataFrame to organize and visualize these importance scores. By sorting the features based on their importance, we can easily identify which ones play the most significant role in the model. A bar chart was then generated to visually represent the feature importances. This chart illustrates the relative importance of each feature, helping us understand which variables are most influential in predicting fraudulent transactions. Such insights are crucial for interpreting the model and can guide further feature engineering or data collection efforts.



```python
# 1.22 Feature Importance

# Train XGBoost Model

# Ensure 'Unnamed: 0' is excluded from X_train
X_train = X_train.drop(columns=['Unnamed: 0'], errors='ignore')
xgb_clf.fit(X_train, y_train)  # Ensure model is trained on the correct features

# Extract Feature Importances
features = X_train.columns.tolist()  # Dynamically get feature names from the training data
feature_importances = xgb_clf.feature_importances_  # Extract feature importance scores from the trained model

# Step 3: Create DataFrame for Visualization
importance_df = pd.DataFrame({
    'Feature': features,
    'Importance': feature_importances
}).sort_values(by='Importance', ascending=False)

# Step 4: Visualize Feature Importance
plt.figure(figsize=(10, 6))
sns.barplot(x='Importance', y='Feature', data=importance_df, palette="viridis")
plt.title("Feature Importance (XGBoost)")
plt.xlabel("Importance Score")
plt.ylabel("Features")
plt.show()
```



Model Performance Across Features

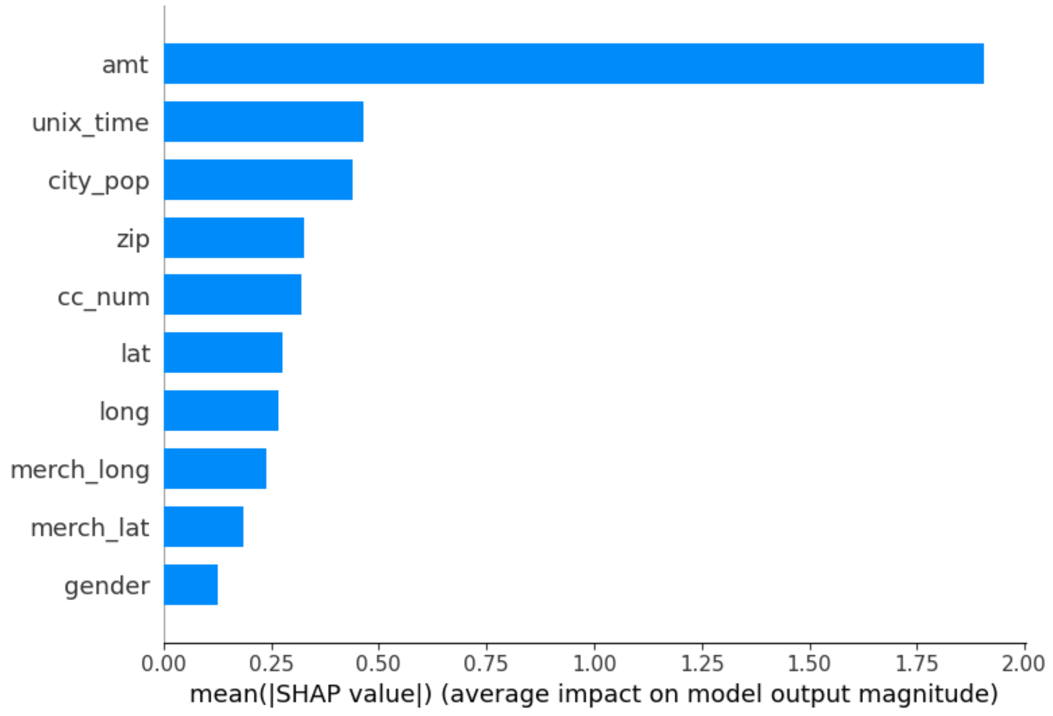
23. **Model Performance Across Features**
    - **Question:** How does the model perform across different subsets of features or data?
        - Use visualizations plots to show how different features impact model predictions.

To evaluate how our XGBoost model performs with different features, we used SHAP (SHapley Additive exPlanations) values. SHAP values help us understand the impact of each feature on the model's predictions.

We initialized a SHAP explainer with our trained XGBoost model and calculated the SHAP values for the training data. A summary plot was then created to visualize these values, showing how much each feature contributes to predictions. Features with high positive SHAP values significantly increase the likelihood of predicting fraud, while negative values indicate a lower likelihood. This analysis helps us identify the most influential features and understand their interactions, providing insights for refining the model and guiding feature selection.

```
# 1.23 Model Performance Across Features

# Initialize SHAP Explainer
explainer = shap.Explainer(xgb_clf, X_train)

# Calculate SHAP values
shap_values = explainer(X_train)

# Plot SHAP summary
shap.summary_plot(shap_values, X_train, plot_type="bar")
```

```
100%|===================| 829847/829872 [19:16<00:00]
```


amt
unix_time
city_pop
zip
cc_num
lat
long
merch_long
merch_lat
gender
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
mean(|SHAP value|) (average impact on model output magnitude)