Importing Libraries

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import sklearn
import pickle
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
```

Importing Dataset From keggle

```
import kagglehub
# Download latest version
path = kagglehub.dataset_download("kartik2112/fraud-detection")
```

Load train and test dataset using pandas

```
df=pd.read_csv("/kaggle/input/fraud-detection/fraudTest.csv")
df_t=pd.read_csv('/kaggle/input/fraud-detection/fraudTrain.csv')

#check uniqe value in fraud coloum
df['is_fraud'].value_counts()
```

```
is_fraud
0    553574
1      2145
Name: count, dtype: int64
```

```
# train dataset
df.head(3)
```

{"type":"dataframe","variable_name":"df"}

```
# test dataset
df_t.head(3)
```

{"type":"dataframe","variable_name":"df_t"}

Data Info

Observations on shape of data and data types of all attributes

```
# check the size of the both dataset
print("Train Data shape : ",df.shape)
print("Test Data shape : ",df.shape)
# create target variable
target_variable=df['is_fraud']
# check train datset info
df.info()
```

```
Train Data shape :  (555719, 23)
Test Data shape :  (555719, 23)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 555719 entries, 0 to 555718
Data columns (total 23 columns):
 #   Column                 Non-Null Count   Dtype
---  ------                 --------------   -----
 0   Unnamed: 0             555719 non-null  int64
 1   trans_date_trans_time  555719 non-null  object
 2   cc_num                 555719 non-null  int64
 3   merchant               555719 non-null  object
 4   category               555719 non-null  object
 5   amt                    555719 non-null  float64
 6   first                  555719 non-null  object
 7   last                   555719 non-null  object
 8   gender                 555719 non-null  object
 9   street                 555719 non-null  object
 10  city                   555719 non-null  object
 11  state                  555719 non-null  object
 12  zip                    555719 non-null  int64
 13  lat                    555719 non-null  float64
 14  long                   555719 non-null  float64
 15  city_pop               555719 non-null  int64
 16  job                    555719 non-null  object
 17  dob                    555719 non-null  object
 18  trans_num              555719 non-null  object
 19  unix_time              555719 non-null  int64
 20  merch_lat              555719 non-null  float64
 21  merch_long             555719 non-null  float64
 22  is_fraud               555719 non-null  int64
dtypes: float64(5), int64(6), object(12)
memory usage: 97.5+ MB
```

Display the statistical summary

```
# check train data summary
df.describe()
```

{"summary":"{\n \"name\": \"df\",\n \"rows\": 8,\n \"fields\": [\n
{\n   \"column\": \"Unnamed: 0\",\n   \"properties\": {\n
\"dtype\": \"number\",\n   \"std\": 200283.5973286697,\n
\"min\": 0.0,\n   \"max\": 555719.0,\n
\"num_unique_values\": 7,\n   \"samples\": [\n
555719.0,\n   277859.0,\n   416788.5\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   },\n   {\n   \"column\": \"cc_num\",\n   \"properties\":
{\n   \"dtype\": \"number\",\n   \"std\":
1.7384297719737928e+18,\n   \"min\": 555719.0,\n   \"max\":
4.992346398065154e+18,\n   \"num_unique_values\": 8,\n
\"samples\": [\n   4.178386955287641e+17,\n
3521417320836166.0,\n   555719.0\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   },\n   {\n   \"column\": \"amt\",\n   \"properties\": {\n
\"dtype\": \"number\",\n   \"std\": 195469.40292796041,\n
\"min\": 1.0,\n   \"max\": 555719.0,\n
\"num_unique_values\": 8,\n   \"samples\": [\n
69.39281023322938,\n   47.29,\n   555719.0\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   },\n   {\n   \"column\": \"zip\",\n   \"properties\": {\n
\"dtype\": \"number\",\n   \"std\": 182646.36451821792,\n
\"min\": 1257.0,\n   \"max\": 555719.0,\n
\"num_unique_values\": 8,\n   \"samples\": [\n
48842.62801523792,\n   48174.0,\n   555719.0\
n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"lat\",\n   \"properties\": {\n   \"dtype\": \"number\",\n
\"std\": 196463.95010223167,\n   \"min\": 5.061336211107229,\n
\"max\": 555719.0,\n   \"num_unique_values\": 8,\n
\"samples\": [\n   38.54325282129998,\n   39.3716,\n
555719.0\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"long\",\n   \"properties\": {\n   \"dtype\": \"number\",\n
\"std\": 196505.36351645383,\n   \"min\": -165.6723,\n
\"max\": 555719.0,\n   \"num_unique_values\": 8,\n
\"samples\": [\n   -90.23132507832197,\n   -87.4769,\n
555719.0\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"city_pop\",\n   \"properties\": {\n   \"dtype\":
\"number\",\n   \"std\": 998589.2440870891,\n   \"min\":
23.0,\n   \"max\": 2906700.0,\n   \"num_unique_values\": 8,\
n   \"samples\": [\n   88221.88791817447,\n
2408.0,\n   555719.0\n   ],\n   \"semantic_type\":
\"\",\n   \"description\": \"\"\n   }\n   },\n   {\n
\"column\": \"unix_time\",\n   \"properties\": {\n
\"dtype\": \"number\",\n   \"std\": 637790285.3074052,\n
\"min\": 555719.0,\n   \"max\": 1388534374.0,\n
\"num_unique_values\": 8,\n   \"samples\": [\n
1380678865.1667802,\n   1380761988.0,\n   555719.0\n

],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"merch_lat\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 196463.94132113908,\n   \"min\": 5.095829265180036,\n   \"max\": 555719.0,\n   \"num_unique_values\": 8,\n   \"samples\": [\n   38.54279777803892,\n   39.376593,\n   555719.0\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"merch_long\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 196505.37140014354,\n   \"min\": -166.671575,\n   \"max\": 555719.0,\n   \"num_unique_values\": 8,\n   \"samples\": [\n   -90.23138049244673,\n   -87.445204,\n   555719.0\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"is_fraud\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 196476.28283296054,\n   \"min\": 0.0,\n   \"max\": 555719.0,\n   \"num_unique_values\": 5,\n   \"samples\": [\n   0.0038598644278853163,\n   1.0,\n   0.062007844611745286\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   }\n]\n}","type":"dataframe"}

Check for missing value (if any)

```
# check missing value
if df.isna().sum().any():
    df.isna().sum()
else:
    print("No missing value in train data")
```

```
No missing value in train data
```

Preprocessing

create this function to apply preprocessing on both dataset

```
def preprocess_data(df):
    # Drop unnecessary columns
    df = df.drop(['Unnamed: 0','cc_num', 'trans_date_trans_time', 
'first', 'last', 'dob', 'street', 'trans_num', 'unix_time', 
'merchant'], axis=1)

    # Handle outliers for 'amt' and 'city_pop'
    numerical_columns = df.select_dtypes(include=['float64', 
'int64']).columns
    for col in numerical_columns:
        if col in ['amt', 'city_pop']:
            Q1 = df[col].quantile(0.25)
```

```
            Q3 = df[col].quantile(0.75)
            IQR = Q3 - Q1
            lower_bound = Q1 - 1.5 * IQR
            upper_bound = Q3 + 1.5 * IQR
            df[col] = df[col].clip(lower=lower_bound,
upper=upper_bound)

     # Encode categorical variables
    category_le = LabelEncoder()
    df['category'] = category_le.fit_transform(df["category"]) + 1
    pickle.dump(category_le, open('category_le.pkl', 'wb'))

    gender_le = LabelEncoder()
    df['gender'] = gender_le.fit_transform(df["gender"]) + 1
    pickle.dump(gender_le, open('gender_le.pkl', 'wb'))

    city_le = LabelEncoder()
    df['city'] = city_le.fit_transform(df["city"]) + 1
    pickle.dump(city_le, open('city_le.pkl', 'wb'))

    state_le = LabelEncoder()
    df['state'] = state_le.fit_transform(df["state"]) + 1
    pickle.dump(state_le, open('state_le.pkl', 'wb'))

    job_le = LabelEncoder()
    df['job'] = job_le.fit_transform(df["job"]) + 1
    pickle.dump(job_le, open('job_le.pkl', 'wb'))

    gender_le = LabelEncoder()
    df['gender'] = gender_le.fit_transform(df["gender"]) + 1
    pickle.dump(gender_le, open('gender_le.pkl', 'wb'))

    # # Convert gender to binary
    # df['gender'] = df['gender'].apply(lambda x: 1 if x == 'M' else
0)

    # Normalize numerical features
    scaler = MinMaxScaler()
    df_scaled = pd.DataFrame(scaler.fit_transform(df),
columns=df.columns)
    pickle.dump(scaler, open('scaler_le.pkl', 'wb'))

    return df_scaled

df_preprocessed = preprocess_data(df)
df_t_preprocessed = preprocess_data(df_t)

df_preprocessed.head()
```

{"type":"dataframe","variable_name":"df_preprocessed"}

```
df_t_preprocessed.head()
```

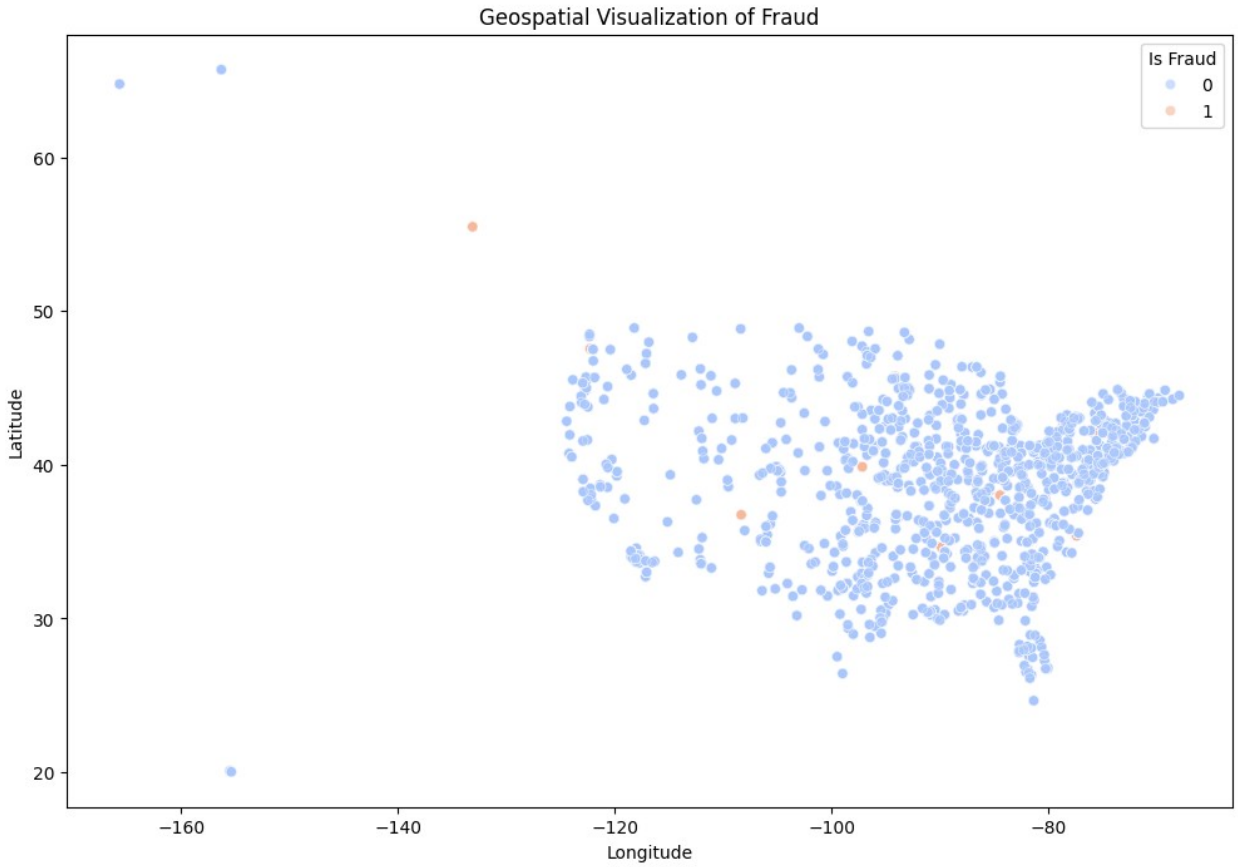
{"type":"dataframe","variable_name":"df_t_preprocessed"}

```
X = df_preprocessed.drop('is_fraud', axis=1)        # features
y = df_preprocessed['is_fraud']

X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2,
stratify=y, random_state=42)

X_test = df_t_preprocessed.drop('is_fraud', axis=1)
y_test = df_t_preprocessed['is_fraud']
```

Exploratory Data Analysis (EDA)

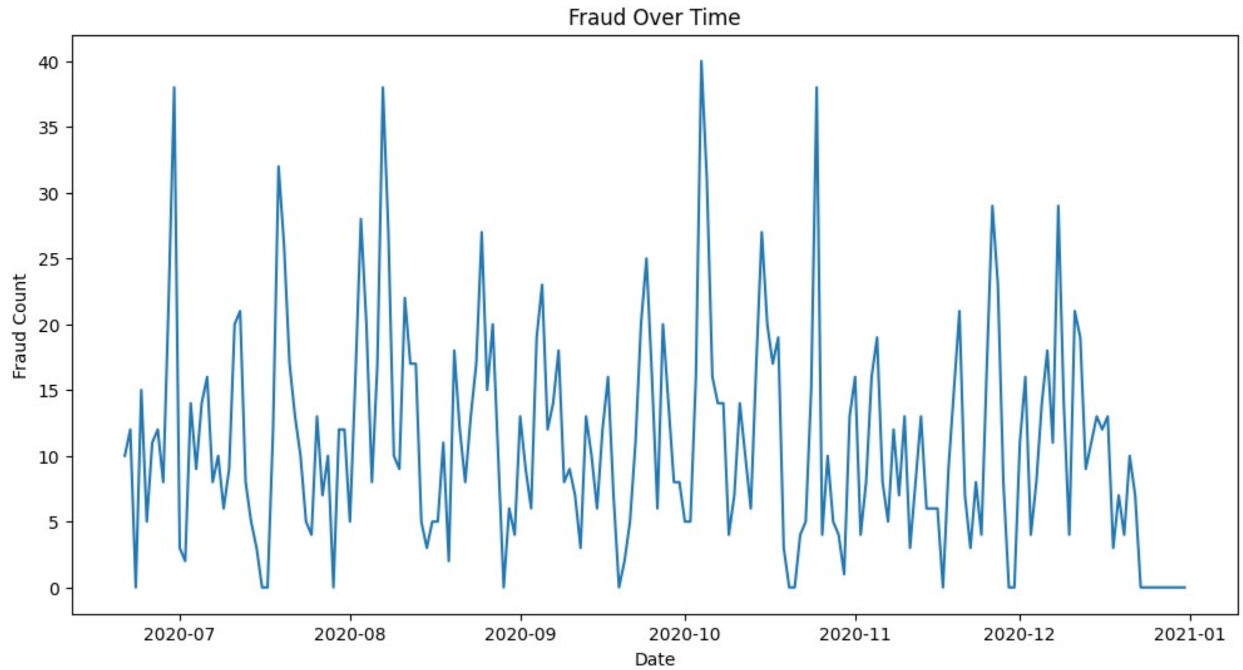
Geospatial Analysis

```
# Plot fraud incidents on a map
plt.figure(figsize=(12, 8))
sns.scatterplot(x='long', y='lat', hue='is_fraud', data=df,
palette='coolwarm', alpha=0.6)
plt.title('Geospatial Visualization of Fraud')
plt.xlabel('Longitude')
plt.ylabel('Latitude')
plt.legend(title='Is Fraud', loc='upper right')
plt.show()
```



Temporal Analysis

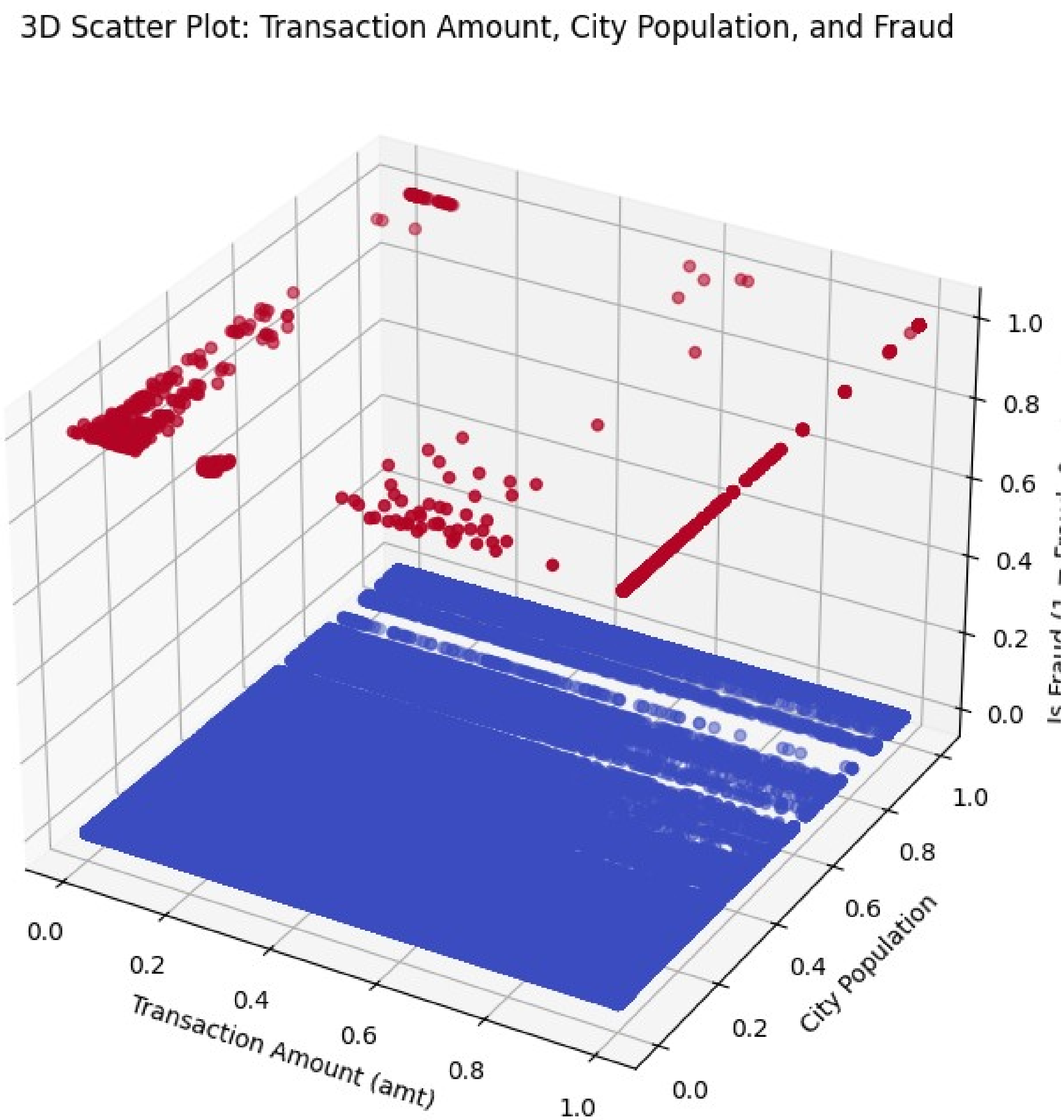
```
# Convert transaction time to datetime
df['trans_date_trans_time'] =
pd.to_datetime(df['trans_date_trans_time'])

# Group by date and calculate fraud count
fraud_over_time = df.groupby(df['trans_date_trans_time'].dt.date)
['is_fraud'].sum()

# Plot time series
plt.figure(figsize=(12, 6))
fraud_over_time.plot()
plt.title('Fraud Over Time')
plt.xlabel('Date')
plt.ylabel('Fraud Count')
plt.show()
```



Features

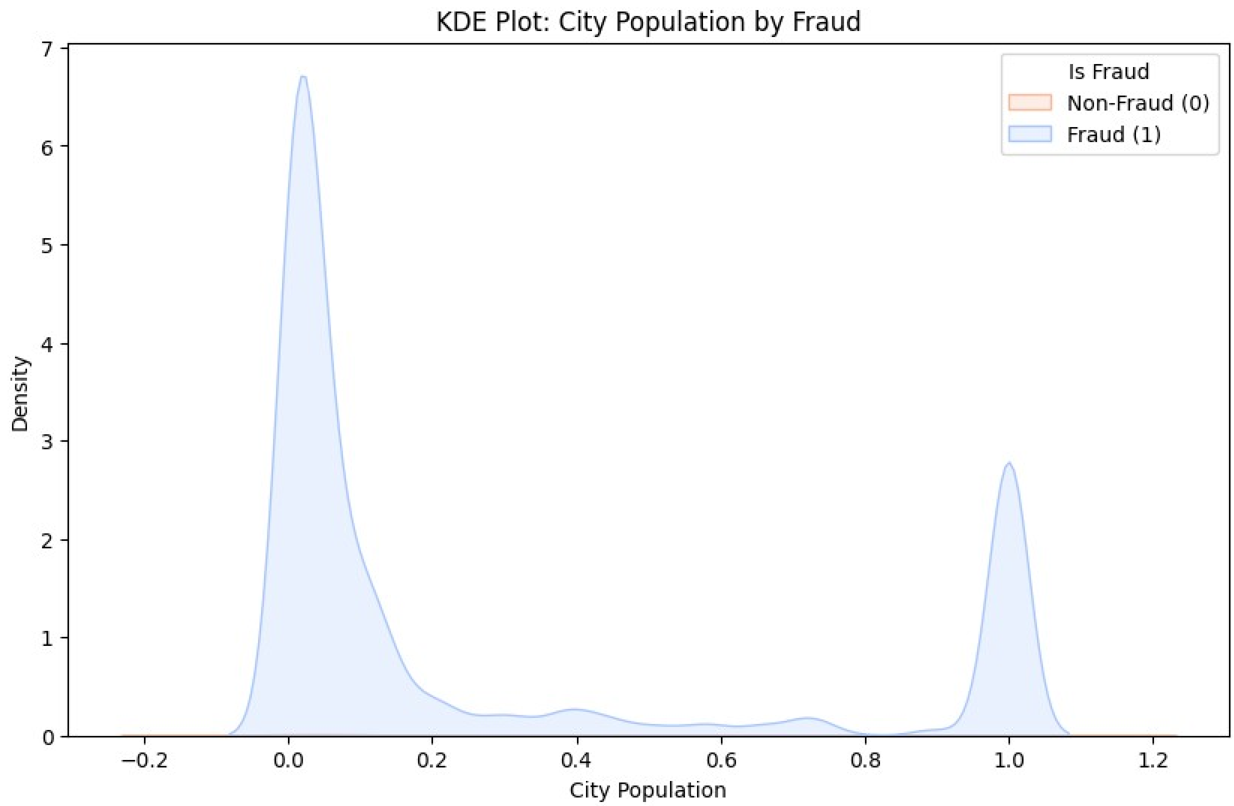
Transaction Amount, City Population, and Fraud

```
# 3D Scatter Plot: Transaction Amount, City Population, and Fraud
fig = plt.figure(figsize=(10, 8))
ax = fig.add_subplot(111, projection='3d')

# Scatter plot
scatter = ax.scatter(
    df_preprocessed['amt'],
    df_preprocessed['city_pop'],
    df_preprocessed['is_fraud'],
    c=df_preprocessed['is_fraud'],
    cmap='coolwarm',
    s=20
)

# Labels
ax.set_xlabel('Transaction Amount (amt)')
ax.set_ylabel('City Population')
ax.set_zlabel('Is Fraud (1 = Fraud, 0 = Non-Fraud)')
plt.title('3D Scatter Plot: Transaction Amount, City Population, and
Fraud')
plt.show()
```

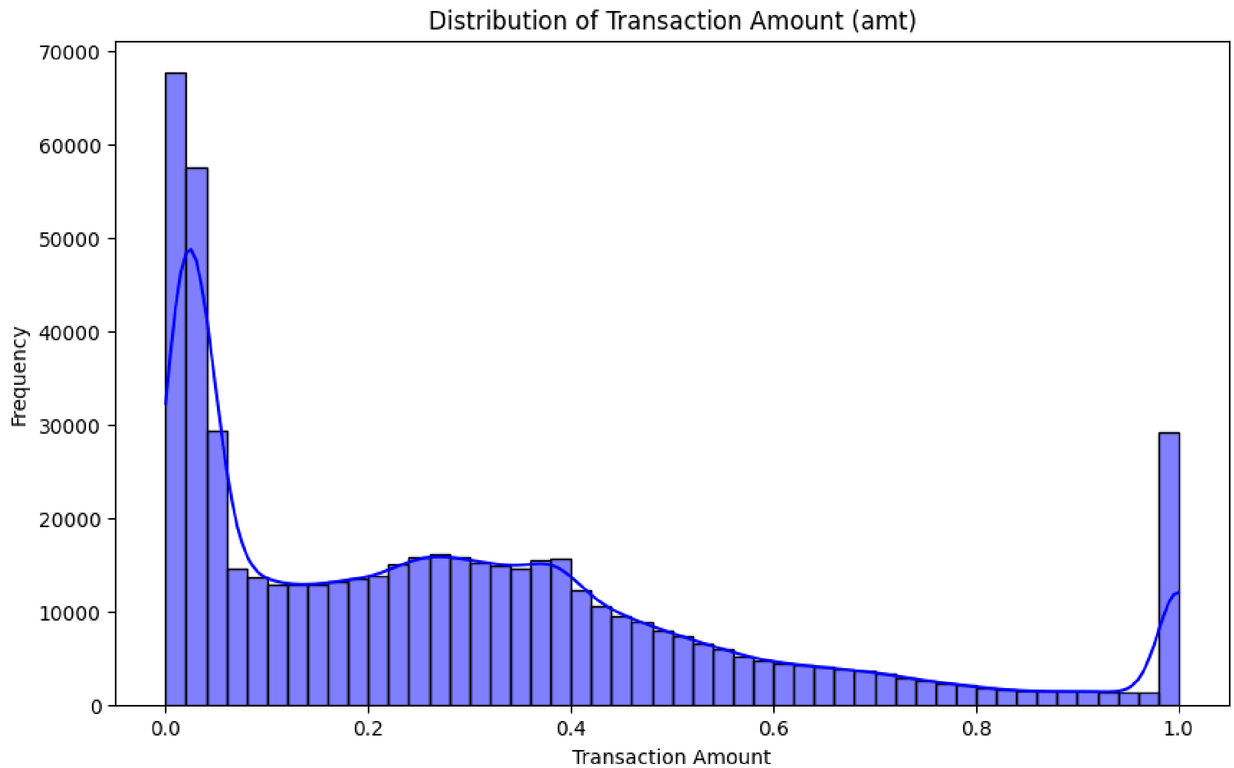


City Population

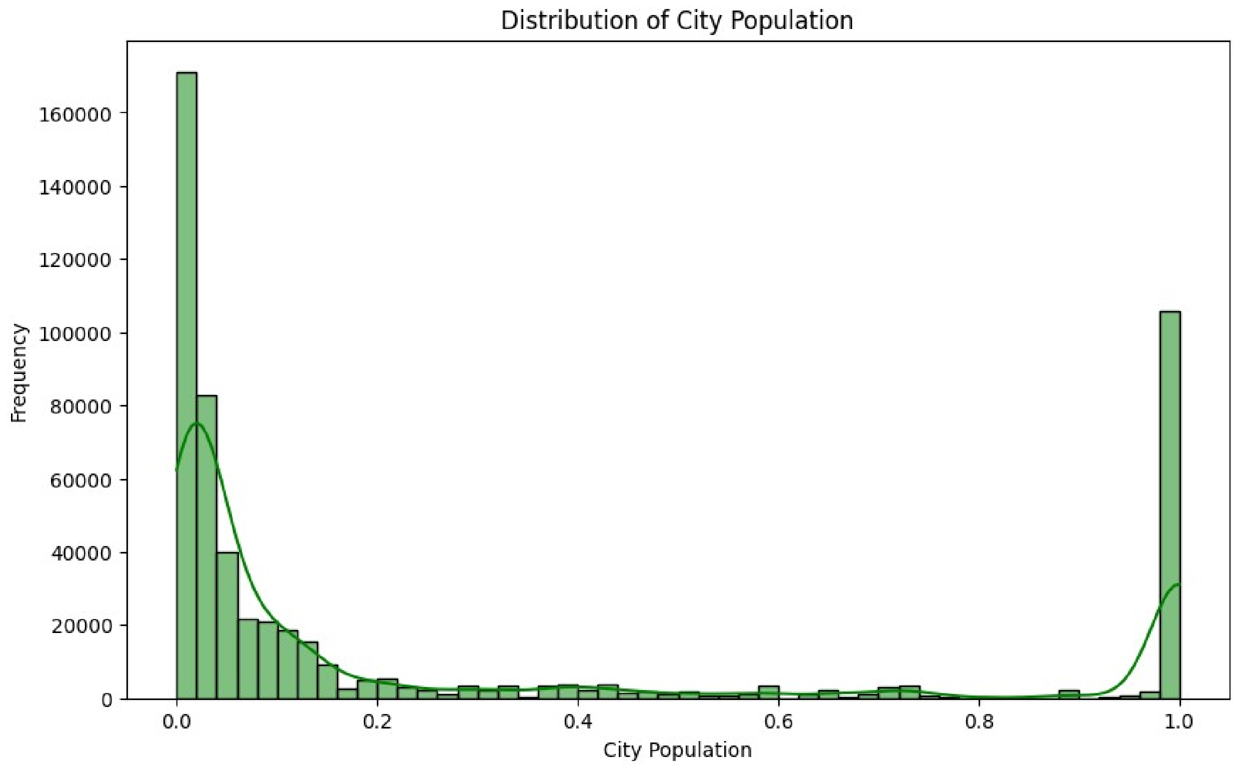
```
plt.figure(figsize=(10, 6))
sns.kdeplot(data=df_preprocessed, x='city_pop', hue='is_fraud',
palette='coolwarm', fill=True)
plt.title('KDE Plot: City Population by Fraud')
plt.xlabel('City Population')
plt.ylabel('Density')
plt.legend(title='Is Fraud', labels=['Non-Fraud (0)', 'Fraud (1)'])
plt.show()
```



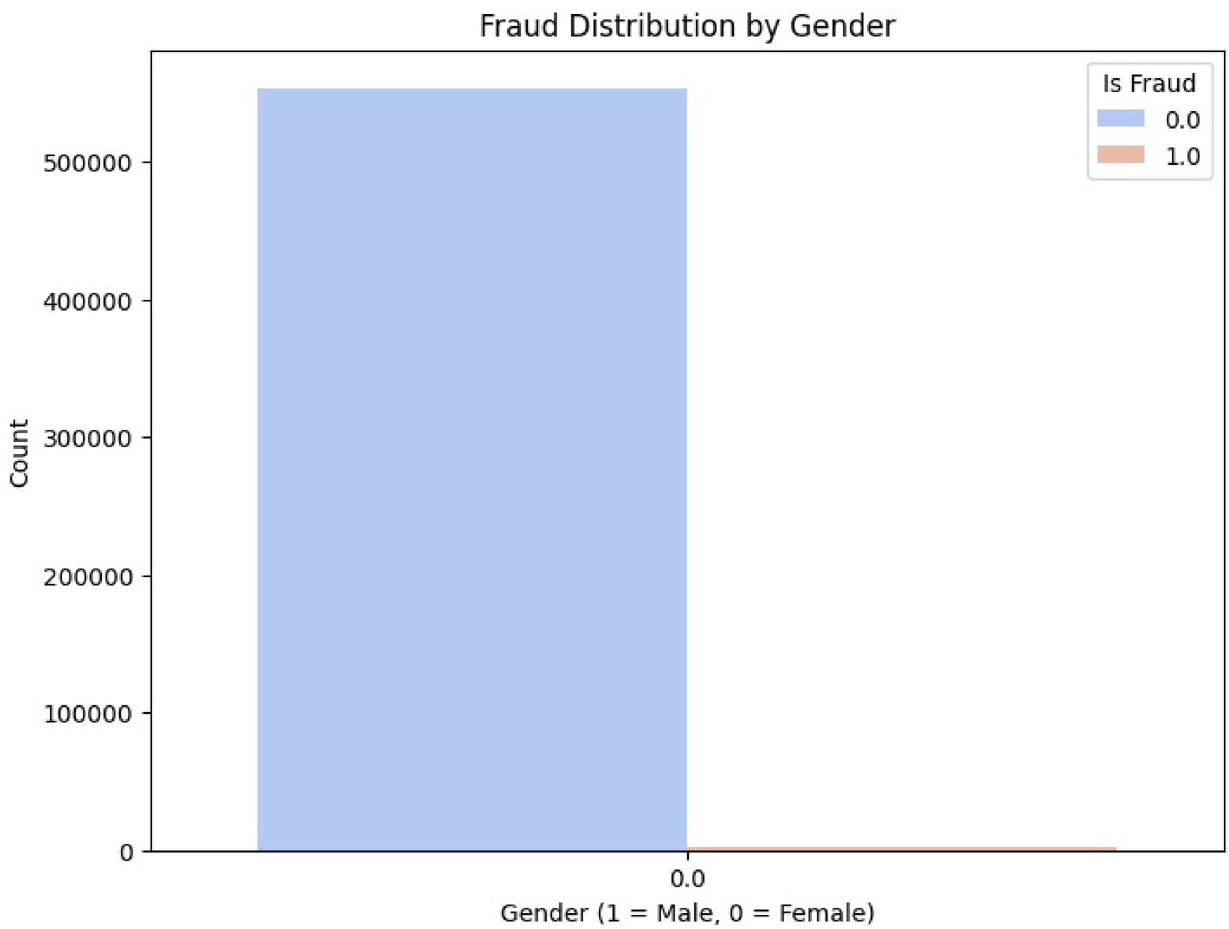

```
# Distribution of Transaction Amount (amt)
plt.figure(figsize=(10, 6))
sns.histplot(df_preprocessed['amt'], bins=50, kde=True, color='blue')
plt.title('Distribution of Transaction Amount (amt)')
plt.xlabel('Transaction Amount')
plt.ylabel('Frequency')
plt.show()
```



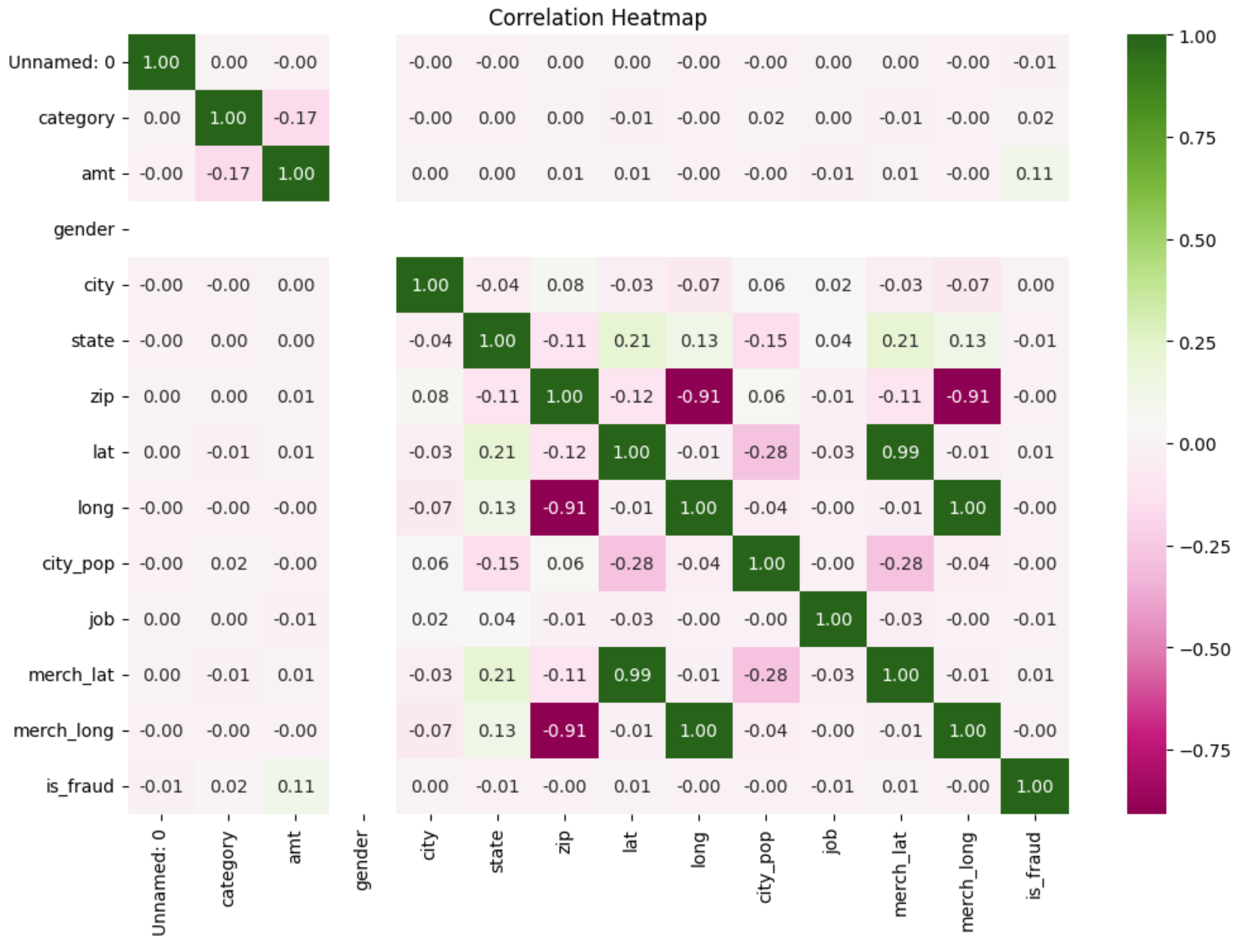

```
# Distribution of City Population
plt.figure(figsize=(10, 6))
sns.histplot(df_preprocessed['city_pop'], bins=50, kde=True,
color='green')
plt.title('Distribution of City Population')
plt.xlabel('City Population')
plt.ylabel('Frequency')
plt.show()
```



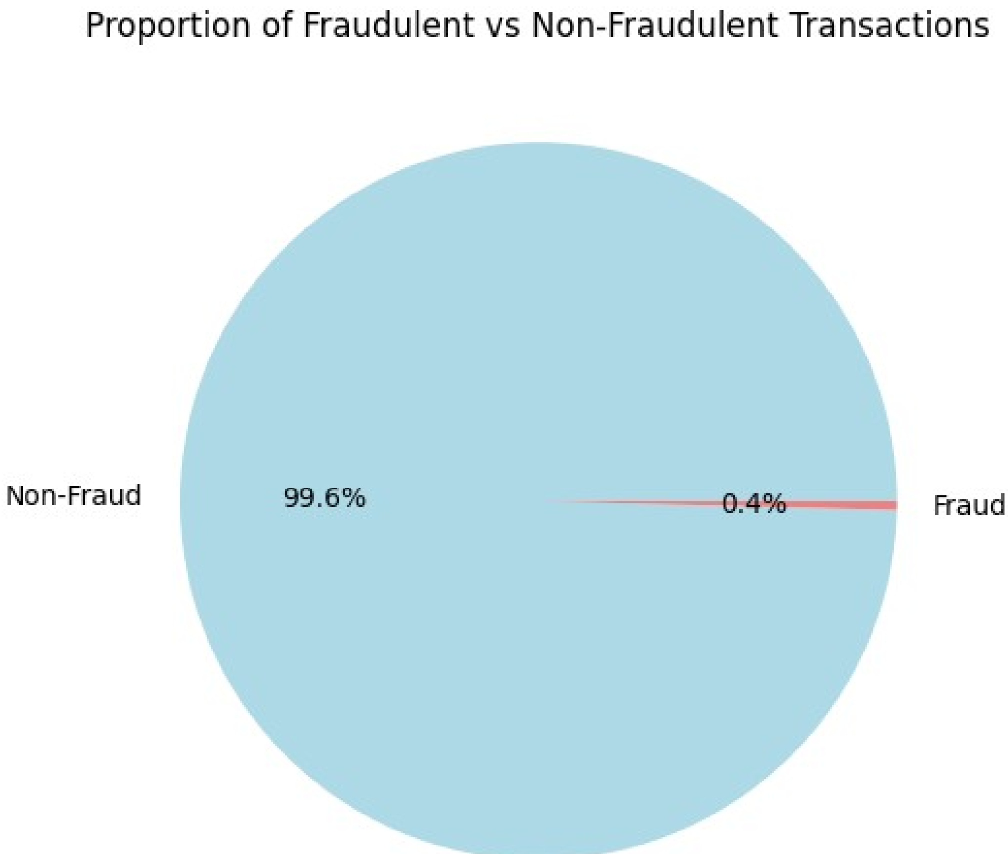

```
# Fraud Distribution by Gender
plt.figure(figsize=(8, 6))
sns.countplot(x='gender', hue='is_fraud', data=df_preprocessed,
palette='coolwarm')
plt.title('Fraud Distribution by Gender')
plt.xlabel('Gender (1 = Male, 0 = Female)')
plt.ylabel('Count')
plt.legend(title='Is Fraud', loc='upper right')
plt.show()
```




```
# Correlation Heatmap
plt.figure(figsize=(12, 8))
corr = df_preprocessed.corr()
sns.heatmap(corr, annot=True, cmap='PiYG', fmt='.2f')
plt.title('Correlation Heatmap')
plt.show()
```




```python
# Calculate fraud proportion
fraud_proportion = df['is_fraud'].value_counts(normalize=True)

# Plot pie chart
plt.figure(figsize=(6, 6))
plt.pie(fraud_proportion, labels=['Non-Fraud', 'Fraud'],
autopct='%1.1f%%', colors=['lightblue', 'lightcoral'])
plt.title('Proportion of Fraudulent vs Non-Fraudulent Transactions')
plt.show()
```



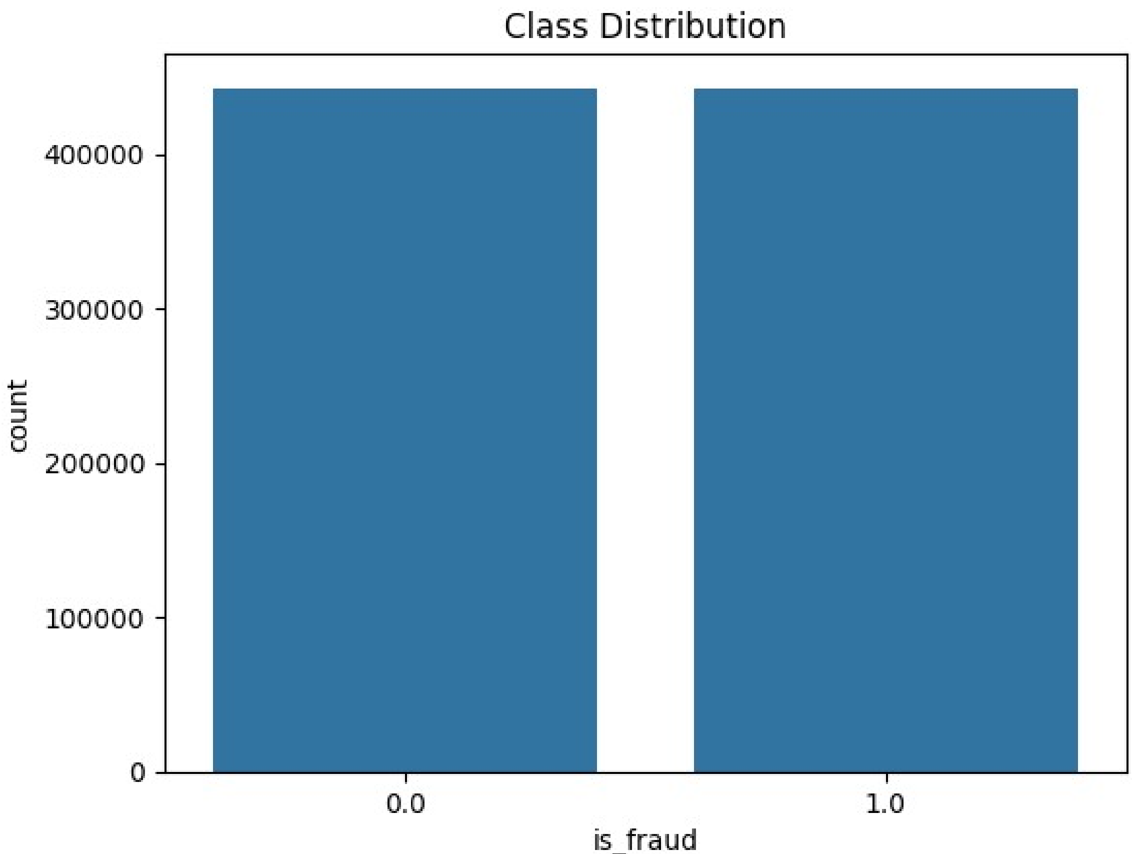
Apply SMOTE Oversampling

```
from imblearn.over_sampling import SMOTE
from collections import Counter

# Check class distribution before applying SMOTE
print("Class distribution before SMOTE:", Counter(y_train))

# Apply SMOTE with a sampling strategy that makes sense based on class
distribution
smote = SMOTE(sampling_strategy='auto', random_state=42)  # Balances
both classes equally
X_train_balanced, y_train_balanced = smote.fit_resample(X_train,
y_train)

# Check the new class distribution after applying SMOTE
print("Class distribution after SMOTE:", Counter(y_train_balanced))
```

```
Class distribution before SMOTE: Counter({0.0: 442859, 1.0: 1716})
Class distribution after SMOTE: Counter({0.0: 442859, 1.0: 442859})
```

```
print(y_train_balanced.value_counts())
sns.countplot(x=y_train_balanced)
plt.title("Class Distribution")
plt.show()
```

```
is_fraud
0.0    442859
1.0    442859
Name: count, dtype: int64
```



Build model

Classify All Model

```
model1 = LogisticRegression()
model2 = RandomForestClassifier()
model3 = DecisionTreeClassifier()
```

Create function to check accuracy

```
from sklearn.metrics import accuracy_score

# Define a function for each metric
def acc_score(test, pred):
    acc_ = accuracy_score(test, pred)
    return acc_

# Print the scores
def print_score(test, pred, model):

    print(f"Classifier: {model}")
    print(f"ACCURACY: {accuracy_score(test, pred)*100}")
```

LogisticRegression

```
model1.fit(X_train_balanced,y_train_balanced)
```

```
LogisticRegression()
```

```
y_pred = model1.predict(X_test)
lr=print_score(y_test, y_pred, "Logistic Regression")
lr
```

```
Classifier: Logistic Regression
ACCURACY: 83.90529623845605
```

```
model_list = []
acc_list = []
```

```
model_list.append(model1.__class__.__name__)
acc_list.append(round(acc_score(y_test, y_pred), 4)*100)
```

RandomForestClassifier

```
model2.fit(X_train_balanced,y_train_balanced)
```

```
RandomForestClassifier()
```

```
y_pred1 = model2.predict(X_test)
print_score(y_test,y_pred1,"Random Forest")
```

```
Classifier: Random Forest
ACCURACY: 99.33175236663003
```

```
model_list.append(model2.__class__.__name__)
acc_list.append(round(acc_score(y_test, y_pred1), 4)*100)
```

DecisionTreeClassifier

```
model3.fit(X_train_balanced,y_train_balanced)
```

```
DecisionTreeClassifier()
```

```
Y_Pred = model3.predict(X_test)
print_score(y_test,Y_Pred,"Decision Tree")
```

```
Classifier: Decision Tree
ACCURACY: 98.32548634006208
```

```
model_list.append(model3.__class__.__name__)
acc_list.append(round(acc_score(y_test, Y_Pred), 3)*100)
```

```
pickle.dump(model3, open('model.pkl', 'wb'))
```

```
import joblib
```

```
joblib.dump(model3,"Decision_Tree.joblib")
```

```
['Decision_Tree.joblib']
```

Campare All Model

```
model_results = pd.DataFrame({"Model": model_list,
                              "Accuracy_Score": acc_list,
                              })
```

```
model_results
```

{"summary":"{\n \"name\": \"model_results\",\n \"rows\": 2,\n \"fields\": [\n   {\n   \"column\": \"Model\",\n   \"properties\": {\n   \"dtype\": \"string\",\n   \"num_unique_values\": 2,\n   \"samples\": [\n   \"RandomForestClassifier\",\n   \"LogisticRegression\"\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"Accuracy_Score\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 10.903586565896564,\n   \"min\": 83.91,\n   \"max\": 99.33,\n   \"num_unique_values\": 2,\n   \"samples\": [\n   99.33,\n   83.91\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   }\n]\n}","type":"dataframe","variable_name":"model_results"}

```
plt.figure(figsize=(10, 6))
sns.barplot(x='Model', y='Accuracy_Score', data=model_results)
plt.title('Model Accuracy Comparison')
plt.xlabel('Model')
plt.ylabel('Accuracy Score')
```

```
Text(0, 0.5, 'Accuracy Score')
```