Untitled

September 30, 2023

[1]:
```
import pyforest
import psycopg2
from sqlalchemy import create_engine
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import classification_report, precision_recall_curve, auc
from imblearn.over_sampling import SMOTE
```

[2]:
```
# Creating SQLAlchemy engine using the psycopg2 connection
engine = create_engine('postgresql+psycopg2://postgres:Dishit@127.0.0.1:5432/
 ↪Project_1')

# Loading sample of the data
query = "SELECT * FROM your_table;"
df = pd.read_sql_query(query, engine)

# Displaying basic statistics
print(df.describe())

# Close the database connection
engine.dispose()
```

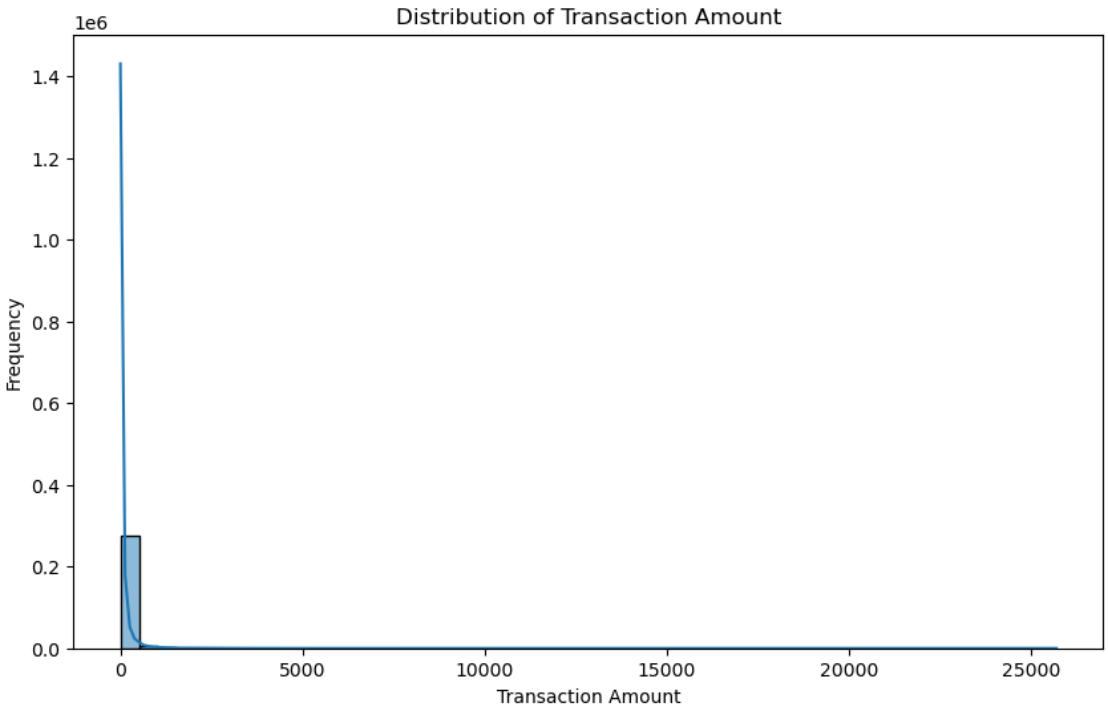
```
<IPython.core.display.Javascript object>
```

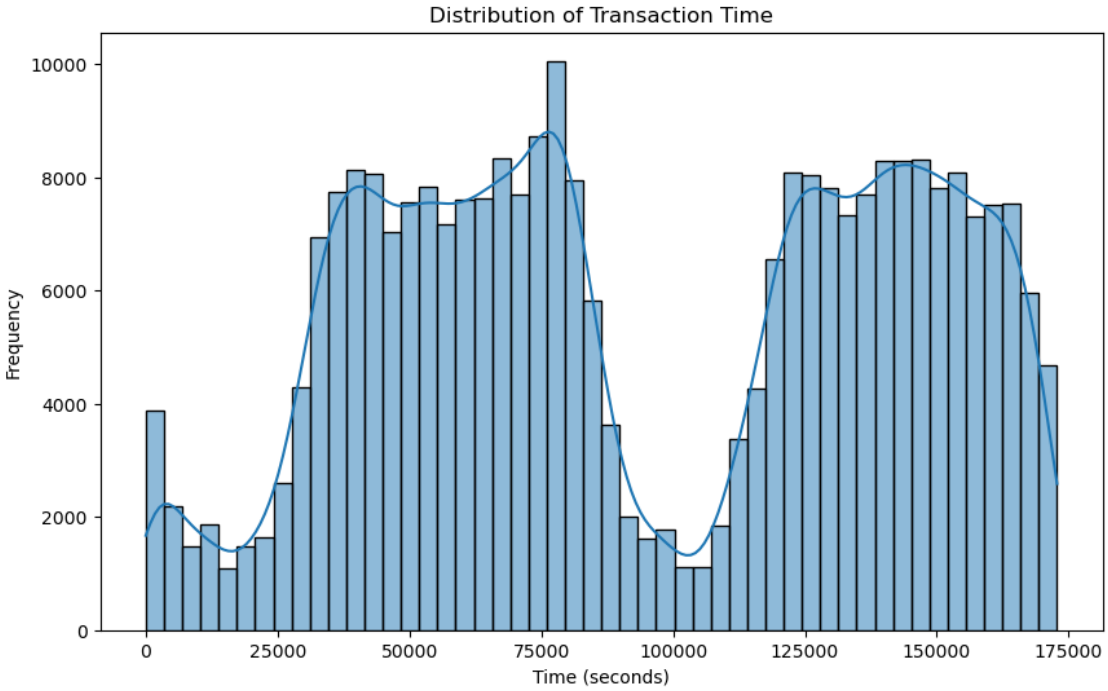
```
                time             v1             v2             v3             v4  \
count  284807.000000  2.848070e+05  2.848070e+05  2.848070e+05  2.848070e+05
mean    94813.859575  1.187535e-15  3.384974e-16 -1.430631e-15  2.074095e-15
std     47488.145955  1.958696e+00  1.651309e+00  1.516255e+00  1.415869e+00
min         0.000000 -5.640751e+01 -7.271573e+01 -4.832559e+01 -5.683171e+00
25%     54201.500000 -9.203734e-01 -5.985499e-01 -8.903648e-01 -8.486401e-01
50%     84692.000000  1.810880e-02  6.548556e-02  1.798463e-01 -1.984653e-02
75%    139320.500000  1.315642e+00  8.037239e-01  1.027196e+00  7.433413e-01
max    172792.000000  2.454930e+00  2.205773e+01  9.382558e+00  1.687534e+01

                 v5             v6             v7             v8             v9  \
```

```
count  2.848070e+05  2.848070e+05  2.848070e+05  2.848070e+05  2.848070e+05
mean   1.011501e-15  1.487313e-15 -5.620335e-16  1.217473e-16 -2.409574e-15
std    1.380247e+00  1.332271e+00  1.237094e+00  1.194353e+00  1.098632e+00
min   -1.137433e+02 -2.616051e+01 -4.355724e+01 -7.321672e+01 -1.343407e+01
25%   -6.915971e-01 -7.682956e-01 -5.540759e-01 -2.086297e-01 -6.430976e-01
50%   -5.433583e-02 -2.741871e-01  4.010308e-02  2.235804e-02 -5.142873e-02
75%    6.119264e-01  3.985649e-01  5.704361e-01  3.273459e-01  5.971390e-01
max    3.480167e+01  7.330163e+01  1.205895e+02  2.000721e+01  1.559499e+01

       …           v21           v22           v23           v24  \
count  …  2.848070e+05  2.848070e+05  2.848070e+05  2.848070e+05
mean   …  1.623131e-16 -3.552626e-16  2.610582e-16  4.472268e-15
std    …  7.345240e-01  7.257016e-01  6.244603e-01  6.056471e-01
min    … -3.483038e+01 -1.093314e+01 -4.480774e+01 -2.836627e+00
25%    … -2.283949e-01 -5.423504e-01 -1.618463e-01 -3.545861e-01
50%    … -2.945017e-02  6.781943e-03 -1.119293e-02  4.097606e-02
75%    …  1.863772e-01  5.285536e-01  1.476421e-01  4.395266e-01
max    …  2.720284e+01  1.050309e+01  2.252841e+01  4.584549e+00

                v25           v26           v27           v28         amount  \
count  2.848070e+05  2.848070e+05  2.848070e+05  2.848070e+05  284807.000000
mean   5.277047e-16  1.687030e-15 -3.662087e-16 -1.227328e-16      88.349619
std    5.212781e-01  4.822270e-01  4.036325e-01  3.300833e-01     250.120109
min   -1.029540e+01 -2.604551e+00 -2.256568e+01 -1.543008e+01       0.000000
25%   -3.171451e-01 -3.269839e-01 -7.083953e-02 -5.295979e-02       5.600000
50%    1.659350e-02 -5.213911e-02  1.342146e-03  1.124383e-02      22.000000
75%    3.507156e-01  2.409522e-01  9.104512e-02  7.827995e-02      77.165000
max    7.519589e+00  3.517346e+00  3.161220e+01  3.384781e+01   25691.160000

              class
count  284807.000000
mean        0.001727
std         0.041527
min         0.000000
25%         0.000000
50%         0.000000
75%         0.000000
max         1.000000

[8 rows x 31 columns]
```

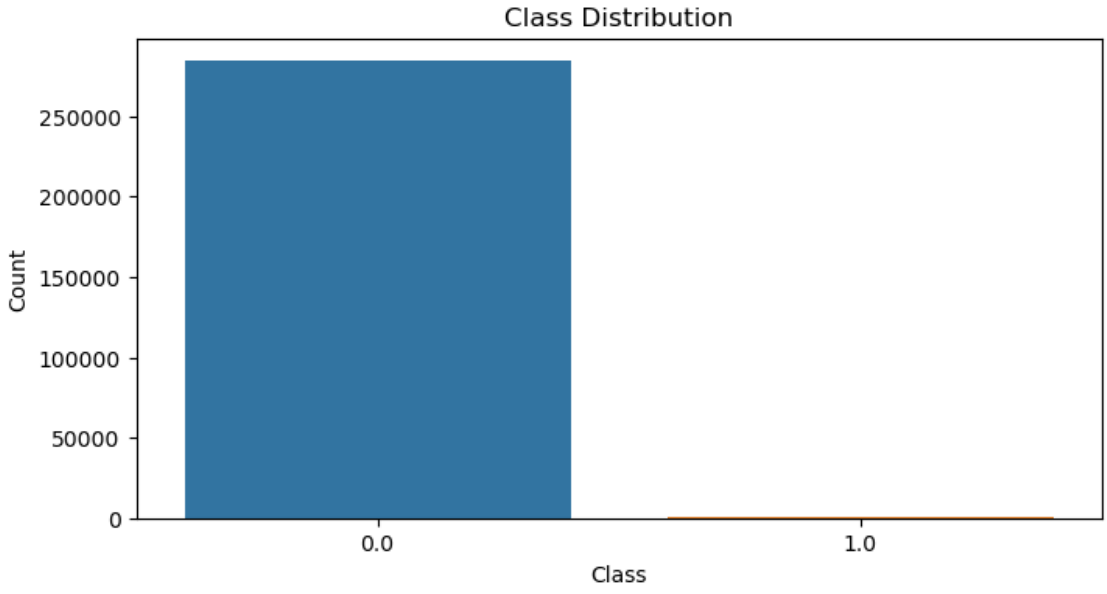
[3]:
```
# Distribution of Amount
plt.figure(figsize=(10, 6))
sns.histplot(df['amount'], bins=50, kde=True)
plt.xlabel('Transaction Amount')
plt.ylabel('Frequency')
plt.title('Distribution of Transaction Amount')
```

```
plt.show()

# Distribution of Time
plt.figure(figsize=(10, 6))
sns.histplot(df['time'], bins=50, kde=True)
plt.xlabel('Time (seconds)')
plt.ylabel('Frequency')
plt.title('Distribution of Transaction Time')
plt.show()
```





[4]:
```
# Class distribution
plt.figure(figsize=(8, 4))
sns.countplot(data=df, x='class')
plt.xlabel('Class')
plt.ylabel('Count')
plt.title('Class Distribution')
plt.show()
```



[6]:
```
# Separating features (X) and target (y)
X = df.drop(columns=['class'])
y = df['class']

# Splitting the data 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)
```

[7]:
```
# Applying SMOTE to oversample the minority class in the training data
smote = SMOTE(random_state=42)
X_train_resampled, y_train_resampled = smote.fit_resample(X_train, y_train)

# Checking the class distribution after resampling
print("Class distribution after SMOTE:")
print(y_train_resampled.value_counts())
```

```
Class distribution after SMOTE:
class
0.0    227459
1.0    227459
Name: count, dtype: int64
```

[8]:
```
# Creating a Random Forest classifier
clf = RandomForestClassifier(random_state=42)

# Training the model on the resampled training data
clf.fit(X_train_resampled, y_train_resampled)
```

[8]:
```
RandomForestClassifier(random_state=42)
```

[9]:
```
from sklearn.metrics import classification_report, precision_recall_curve, auc

# Predicting on the test data
y_pred = clf.predict(X_test)

# Evaluating the model
print(classification_report(y_test, y_pred))

# Calculating AUPRC
precision, recall, _ = precision_recall_curve(y_test, clf.
 ↪predict_proba(X_test)[:, 1])
auprc = auc(recall, precision)
print(f'AUPRC: {auprc:.2f}')

# Plot Precision-Recall curve
```

```
plt.figure(figsize=(8, 6))
plt.plot(recall, precision, color='blue', lw=2, label='Precision-Recall curve')
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Precision-Recall Curve')
plt.legend(loc='lower left')
plt.show()
```

```
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00     56856
         1.0       0.85      0.82      0.84       106

    accuracy                           1.00     56962
   macro avg       0.93      0.91      0.92     56962
weighted avg       1.00      1.00      1.00     56962

AUPRC: 0.85
```