```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns
```

```
data = pd.read_csv("/content/credit_card_fraud_dataset.csv")
```

```
data.head()
```

	Location	IsFraud	TransactionHour	TransactionDay	TransactionMonth
0	7	0	14	3	4
1	1	0	13	19	3
2	4	0	10	8	1
3	5	0	23	13	4
4	6	0	18	12	7

Next steps: Generate code with data View recommended plots New interactive sheet

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100000 entries, 0 to 99999
Data columns (total 7 columns):
 #   Column           Non-Null Count   Dtype  
---  ------           --------------   -----  
 0   TransactionID    100000 non-null  int64  
 1   TransactionDate  100000 non-null  object 
 2   Amount           100000 non-null  float64
 3   MerchantID       100000 non-null  int64  
 4   TransactionType  100000 non-null  object 
 5   Location         100000 non-null  object 
 6   IsFraud          100000 non-null  int64  
dtypes: float64(1), int64(3), object(3)
memory usage: 5.3+ MB
```

```
data.describe()
```

	Location	IsFraud	TransactionHour	TransactionDay	TransactionMonth
count	100000.000000	100000.000000	100000.00000	100000.000000	100000.000000
mean	4.485300	0.010000	11.50084	15.782220	6.516240
std	2.876283	0.099499	6.91768	8.813795	3.448248
min	0.000000	0.000000	0.00000	1.000000	1.000000
25%	2.000000	0.000000	6.00000	8.000000	4.000000
50%	4.000000	0.000000	12.00000	16.000000	7.000000
75%	7.000000	0.000000	17.00000	23.000000	10.000000
max	9.000000	1.000000	23.00000	31.000000	12.000000

```
data.isnull().sum()
```

	0
TransactionID	0
TransactionDate	0
Amount	0
MerchantID	0
TransactionType	0
Location	0
IsFraud	0

dtype: int64

label encoding and onehot encoding

data_encoded = pd.get_dummies(data, columns=['TransactionID', 'TransactionDate', 'Amount'])

```
data.duplicated().sum()
```

```
np.int64(0)
```

```
data = data.drop_duplicates()
```

```
label_encoder_type = LabelEncoder()
label_encoder_location = LabelEncoder()

data['TransactionType'] = label_encoder_type.fit_transform(data['TransactionType'])
data['Location'] = label_encoder_location.fit_transform(data['Location'])

transaction_type_mapping = dict(zip(label_encoder_type.classes_, range(len(label_encoder_type.classes_))))
location_mapping = dict(zip(label_encoder_location.classes_, range(len(label_encoder_location.classes_))))

transaction_type_inverse_mapping = {v: k for k, v in transaction_type_mapping.items()}
location_inverse_mapping = {v: k for k, v in location_mapping.items()}
```

```
data['TransactionDate'] = pd.to_datetime(data['TransactionDate'])

data['TransactionHour'] = data['TransactionDate'].dt.hour
data['TransactionDay'] = data['TransactionDate'].dt.day
data['TransactionMonth'] = data['TransactionDate'].dt.month

data = data.drop(columns=['TransactionDate'])
```

```
data.head()
```

	Location	IsFraud	TransactionHour	TransactionDay	TransactionMonth
0	7	0	14	3	4
1	1	0	13	19	3
2	4	0	10	8	1
3	5	0	23	13	4
4	6	0	18	12	7

Next steps: Generate code with data View recommended plots New interactive sheet

```
data.describe()
```

	TransactionID	Amount	MerchantID	TransactionType	Location	IsFraud	TransactionHour	TransactionDay	T
count	100000.000000	100000.000000	100000.000000	100000.000000	100000.000000	100000.000000	100000.00000	100000.000000	
mean	50000.500000	2497.092666	501.676070	0.501310	4.485300	0.010000	11.50084	15.782220	
std	28867.657797	1442.415999	288.715868	0.500001	2.876283	0.099499	6.91768	8.813795	
min	1.000000	1.050000	1.000000	0.000000	0.000000	0.000000	0.00000	1.000000	
25%	25000.750000	1247.955000	252.000000	0.000000	2.000000	0.000000	6.00000	8.000000	
50%	50000.500000	2496.500000	503.000000	1.000000	4.000000	0.000000	12.00000	16.000000	
75%	75000.250000	3743.592500	753.000000	1.000000	7.000000	0.000000	17.00000	23.000000	
max	100000.000000	4999.770000	1000.000000	1.000000	9.000000	1.000000	23.00000	31.000000	

```
#label encoding and onehot encoding
df_encoded = pd.get_dummies(df, columns=['Geography', 'Gender', 'Card Type'])
```

```
#scalar standardization
scaler = StandardScaler()
df_scaled = scaler.fit_transform(df)
```

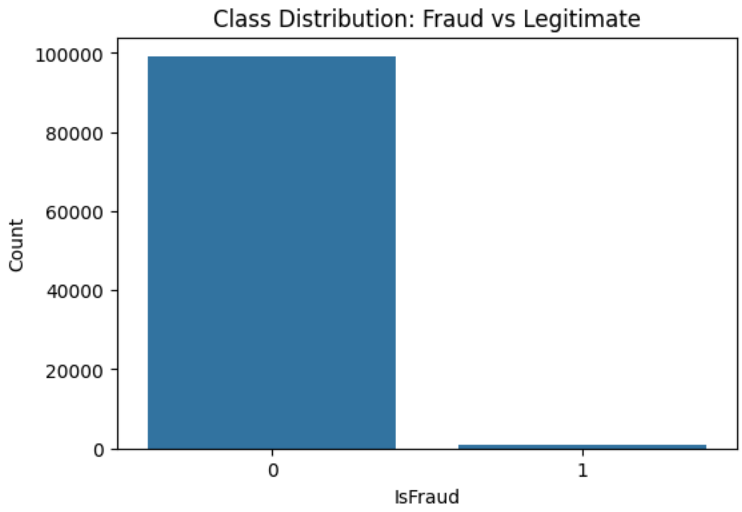
```
df
```

	MerchantID	TransactionType	Location	IsFraud
0	687	1	7	0
1	108	1	1	0
2	393	0	4	0
3	943	0	5	0
4	474	0	6	0
...	...	...	...	...
99995	288	1	7	0
99996	744	1	7	0
99997	689	0	7	0
99998	643	0	5	0
99999	674	1	2	0

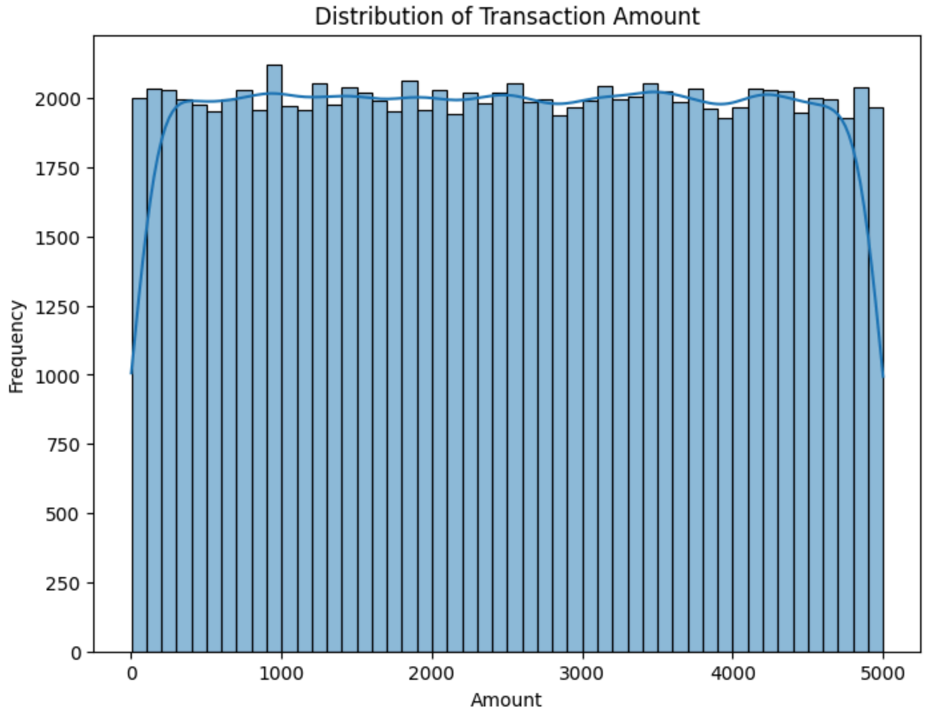
100000 rows × 4 columns

Next steps: Generate code with df View recommended plots New interactive sheet

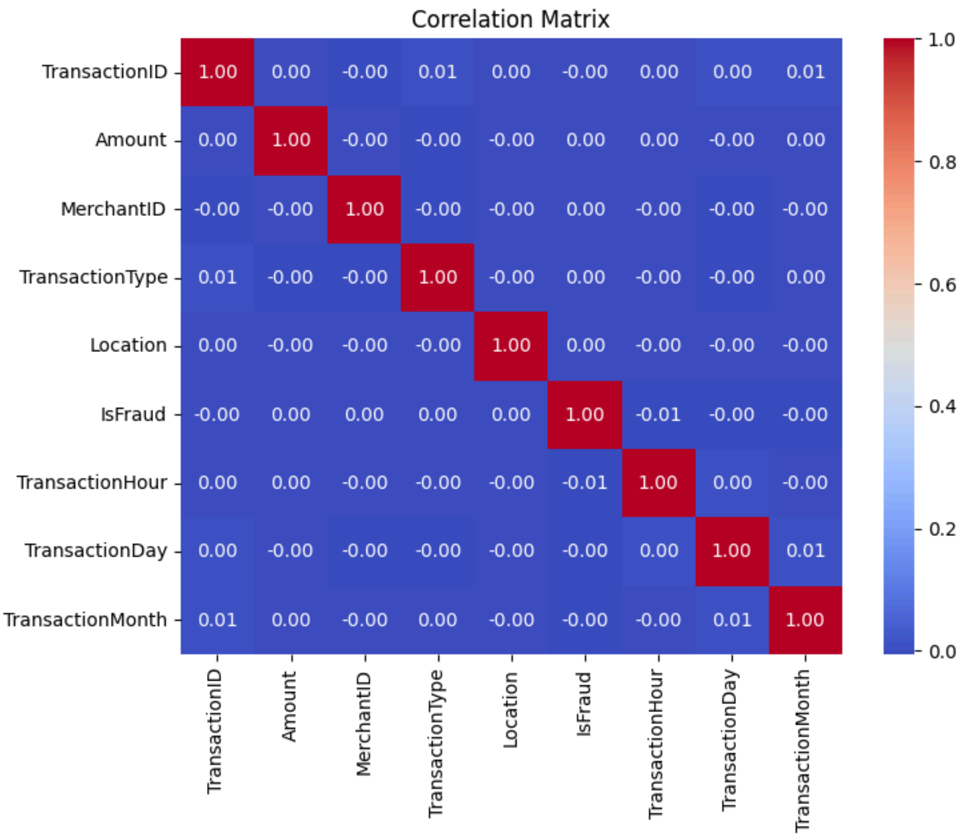
```
plt.figure(figsize=(6, 4))
sns.countplot(x='IsFraud', data=data)
plt.title('Class Distribution: Fraud vs Legitimate')
plt.xlabel('IsFraud')
plt.ylabel('Count')
plt.show()
```




```
plt.figure(figsize=(8, 6))
sns.histplot(data['Amount'], bins=50, kde=True)
plt.title('Distribution of Transaction Amount')
plt.xlabel('Amount')
plt.ylabel('Frequency')
plt.show()
```

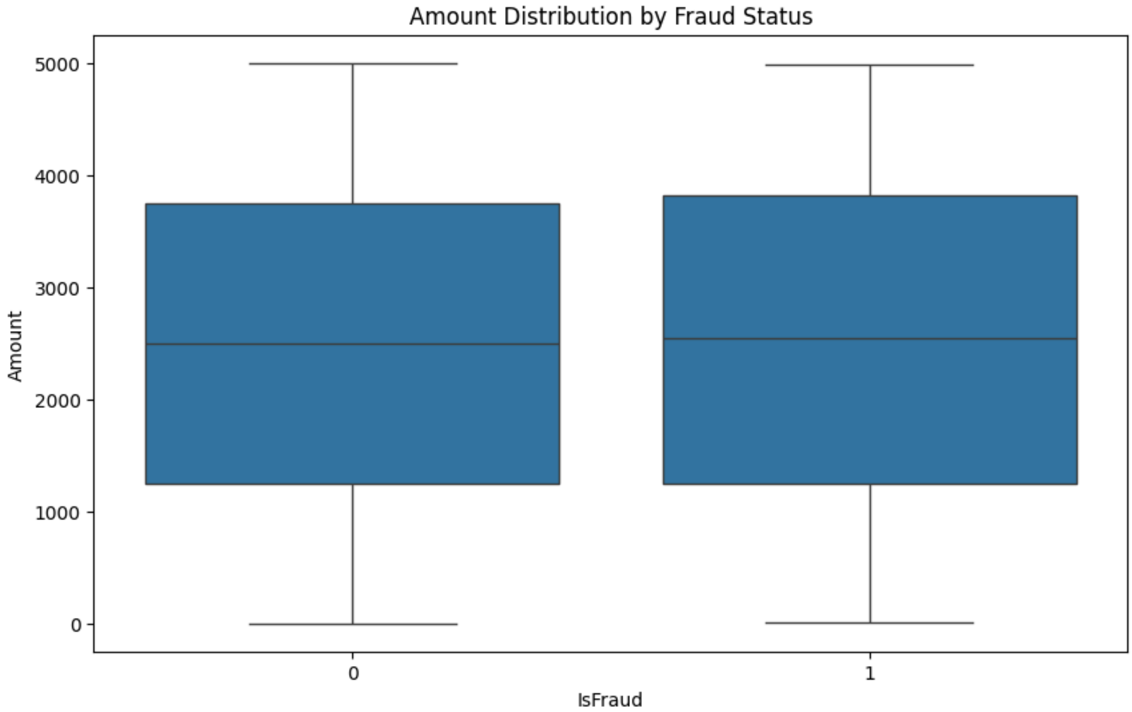



```
correlation = data.corr()

plt.figure(figsize=(8, 6))
sns.heatmap(correlation, annot=True, cmap='coolwarm', fmt='.2f')
plt.title('Correlation Matrix')
plt.show()
```

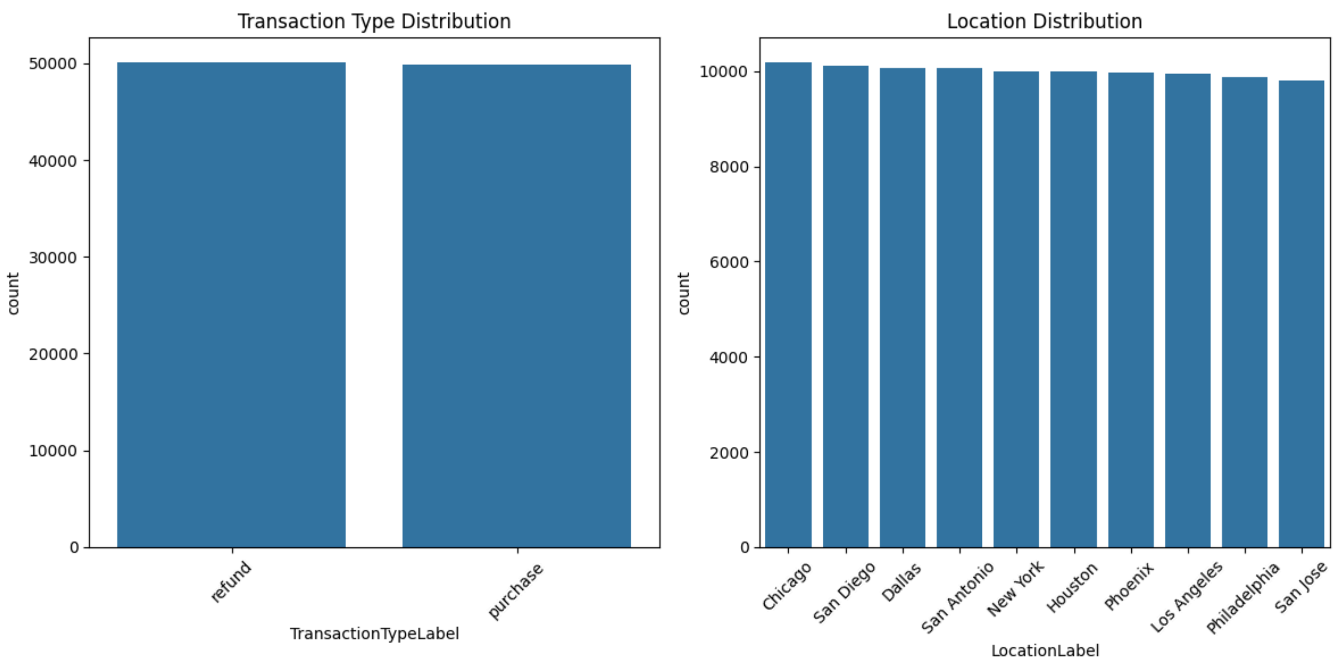


	TransactionID	Amount	MerchantID	TransactionType	Location	IsFraud	TransactionHour	TransactionDay	TransactionMonth
TransactionID	1.00	0.00	-0.00	0.01	0.00	-0.00	0.00	0.00	0.01
Amount	0.00	1.00	-0.00	-0.00	-0.00	0.00	0.00	-0.00	0.00
MerchantID	-0.00	-0.00	1.00	-0.00	-0.00	0.00	-0.00	-0.00	-0.00
TransactionType	0.01	-0.00	-0.00	1.00	-0.00	0.00	-0.00	-0.00	0.00
Location	0.00	-0.00	-0.00	-0.00	1.00	0.00	-0.00	-0.00	-0.00
IsFraud	-0.00	0.00	0.00	0.00	0.00	1.00	-0.01	-0.00	-0.00
TransactionHour	0.00	0.00	-0.00	-0.00	-0.00	-0.01	1.00	0.00	-0.00
TransactionDay	0.00	-0.00	-0.00	-0.00	-0.00	-0.00	0.00	1.00	0.01
TransactionMonth	0.01	0.00	-0.00	0.00	-0.00	-0.00	-0.00	0.01	1.00

```
plt.figure(figsize=(10, 6))
sns.boxplot(x='IsFraud', y='Amount', data=data)
plt.title('Amount Distribution by Fraud Status')
plt.xlabel('IsFraud')
plt.ylabel('Amount')
plt.show()
```



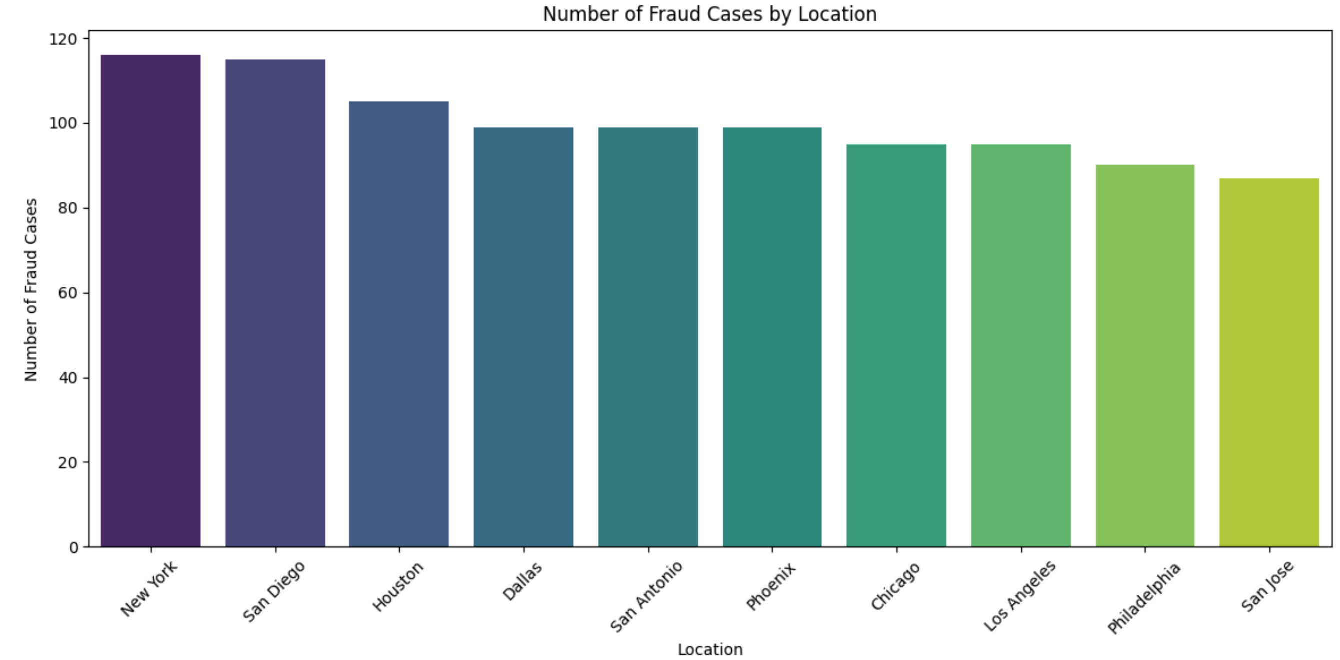

```
data['TransactionTypeLabel'] = data['TransactionType'].map(transaction_type_inverse_mapping)
data['LocationLabel'] = data['Location'].map(location_inverse_mapping)

plt.figure(figsize=(12, 6))

plt.subplot(1, 2, 1)
sns.countplot(x='TransactionTypeLabel', data=data, order=data['TransactionTypeLabel'].value_counts().index)
plt.title('Transaction Type Distribution')
plt.xticks(rotation=45)

plt.subplot(1, 2, 2)
sns.countplot(x='LocationLabel', data=data, order=data['LocationLabel'].value_counts().index)
plt.title('Location Distribution')
plt.xticks(rotation=45)

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

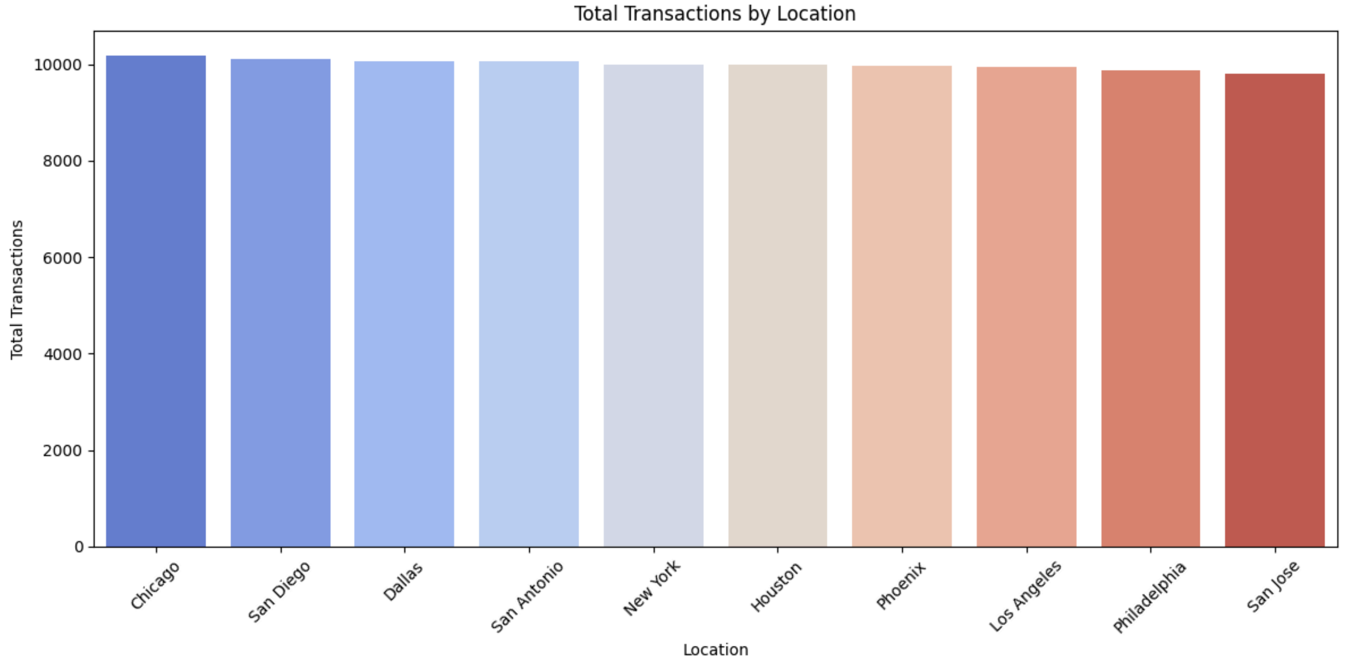



```
plt.figure(figsize=(12, 12))
```

```
plt.subplot(2, 1, 1)

fraud_by_location = data[data['IsFraud'] == 1].groupby('LocationLabel').size().reset_index(name='FraudCount')

fraud_by_location = fraud_by_location.sort_values(by='FraudCount', ascending=False)

# Create the barplot for fraud cases
sns.barplot(x='LocationLabel', y='FraudCount', data=fraud_by_location, palette='viridis')
plt.title('Number of Fraud Cases by Location')
plt.xlabel('Location')
plt.ylabel('Number of Fraud Cases')
plt.xticks(rotation=45)

plt.subplot(2, 1, 2)

total_transactions_by_location = data.groupby('LocationLabel').size().reset_index(name='TotalTransactions')

total_transactions_by_location = total_transactions_by_location.sort_values(by='TotalTransactions', ascending=False)

sns.barplot(x='LocationLabel', y='TotalTransactions', data=total_transactions_by_location, palette='coolwarm')
plt.title('Total Transactions by Location')
plt.xlabel('Location')
plt.ylabel('Total Transactions')
plt.xticks(rotation=45)

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

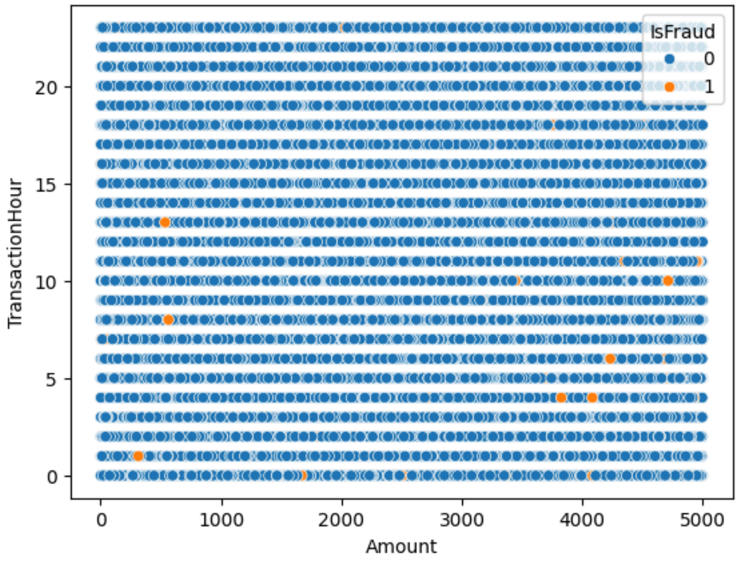
```
<ipython-input-41-8f0b452c3b54>:10: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `le

  sns.barplot(x='LocationLabel', y='FraudCount', data=fraud_by_location, palette='viridis')
<ipython-input-41-8f0b452c3b54>:22: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `le

  sns.barplot(x='LocationLabel', y='TotalTransactions', data=total_transactions_by_location, palette='coolwarm')
```



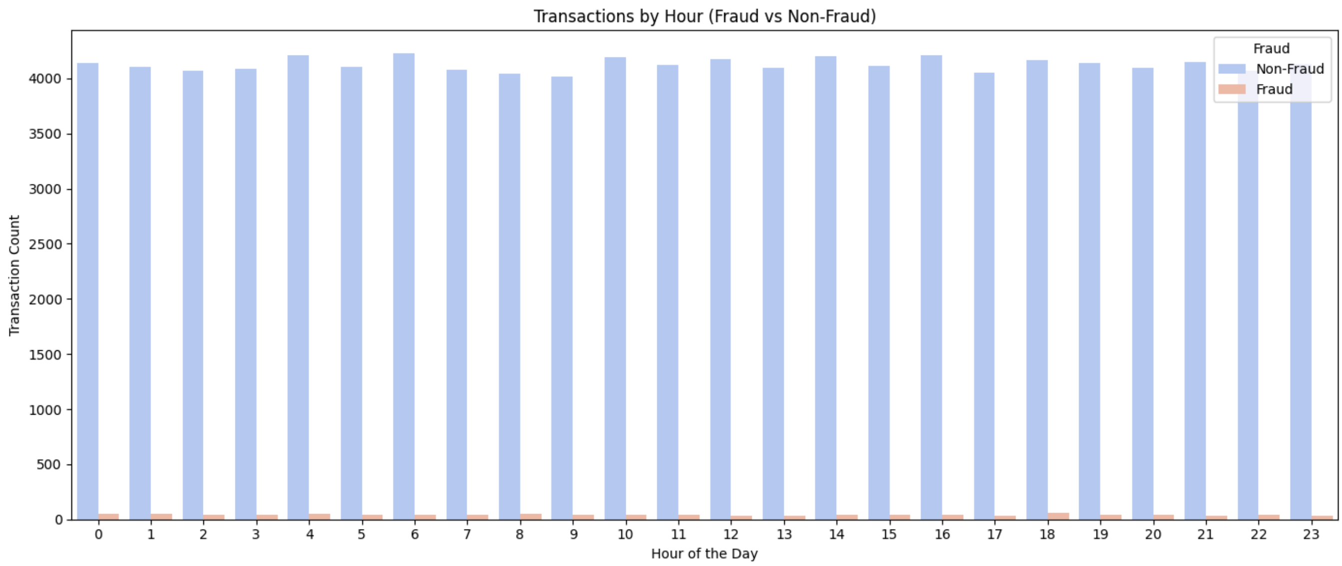



```
sns.scatterplot(x='Amount', y='TransactionHour', hue='IsFraud', data=data)
```

```
<Axes: xlabel='Amount', ylabel='TransactionHour'>
```

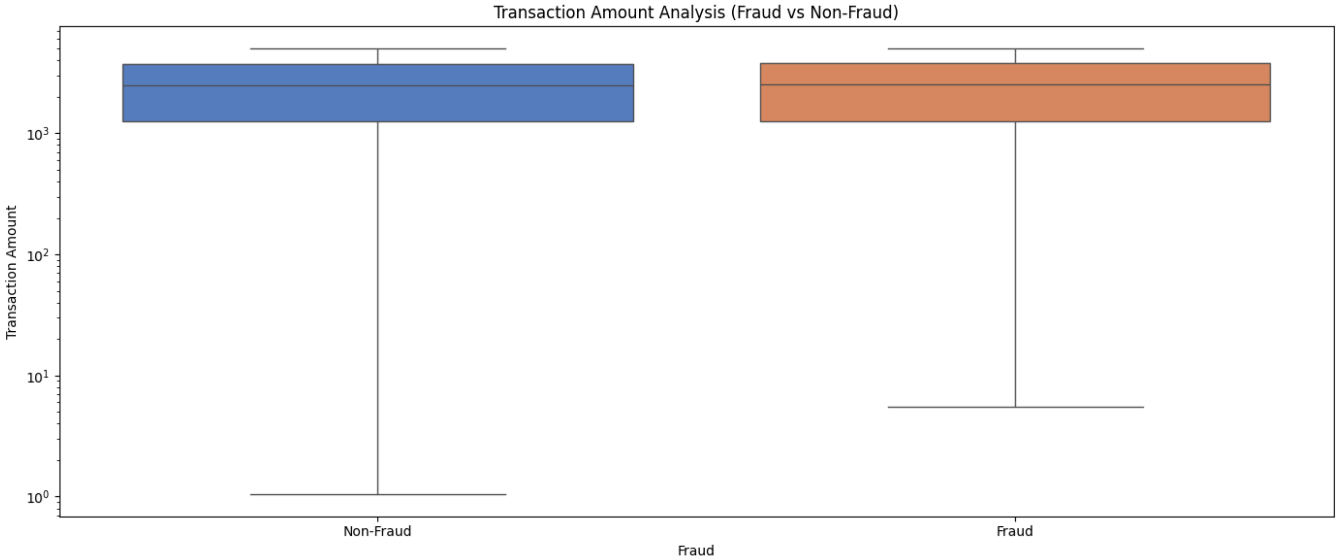



```
plt.figure(figsize=(14, 6))

sns.countplot(x='TransactionHour', hue='IsFraud', data=data, palette='coolwarm')
plt.title('Transactions by Hour (Fraud vs Non-Fraud)')
plt.xlabel('Hour of the Day')
plt.ylabel('Transaction Count')
plt.legend(title='Fraud', loc='upper right', labels=['Non-Fraud', 'Fraud'])
plt.tight_layout()
plt.show()
```

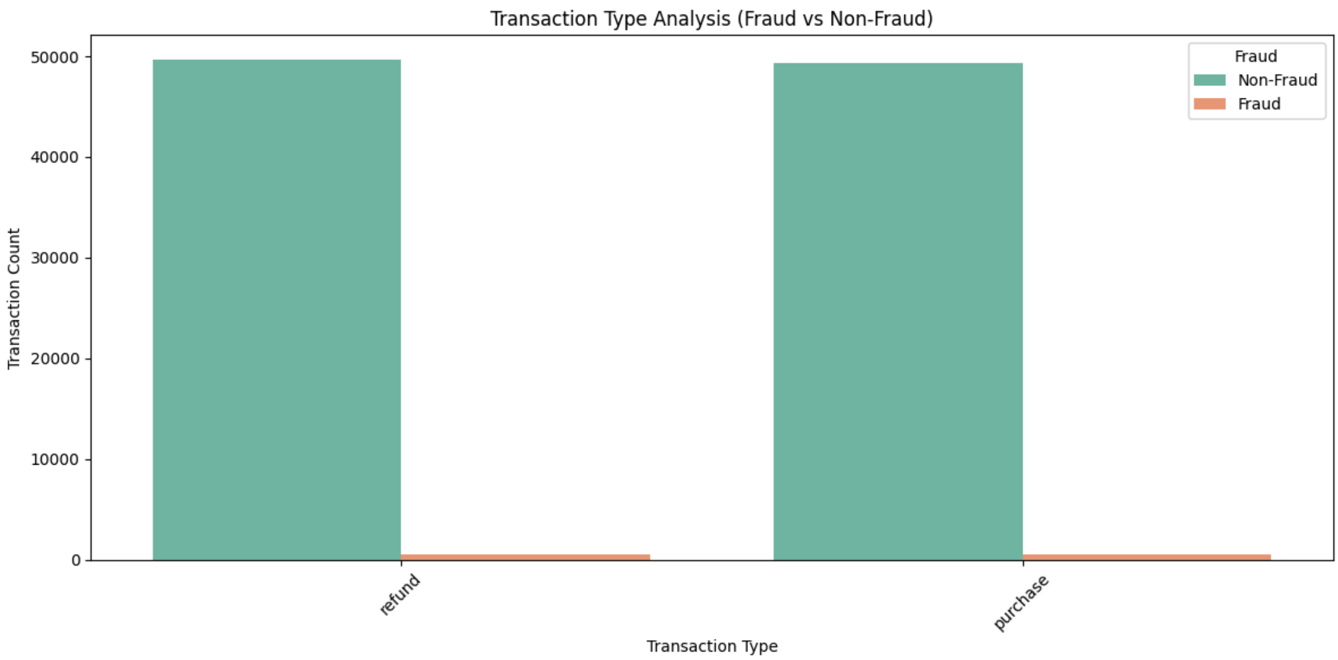



```
plt.figure(figsize=(14, 6))

sns.boxplot(x='IsFraud', y='Amount', data=data, palette='muted')
plt.title('Transaction Amount Analysis (Fraud vs Non-Fraud)')
plt.xlabel('Fraud')
plt.ylabel('Transaction Amount')
plt.xticks(ticks=[0, 1], labels=['Non-Fraud', 'Fraud'])
plt.yscale('log')
plt.tight_layout()
plt.show()
```

```
<ipython-input-44-4f6b52f4f5ce>:3: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `le

  sns.boxplot(x='IsFraud', y='Amount', data=data, palette='muted')
```



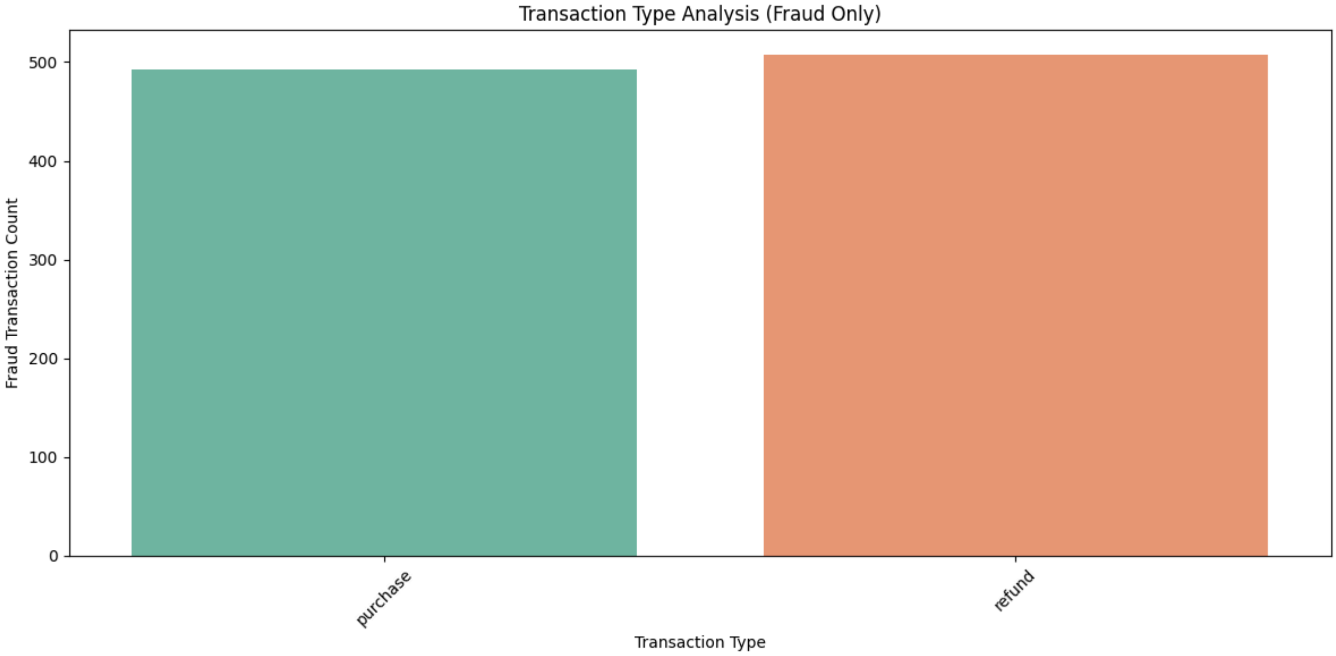

```
plt.figure(figsize=(12, 6))

sns.countplot(x='TransactionTypeLabel', hue='IsFraud', data=data, palette='Set2')
plt.title('Transaction Type Analysis (Fraud vs Non-Fraud)')
plt.xlabel('Transaction Type')
plt.ylabel('Transaction Count')
plt.legend(title='Fraud', loc='upper right', labels=['Non-Fraud', 'Fraud'])
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```




```
plt.figure(figsize=(12, 6))

fraud_data = data[data['IsFraud'] == 1]

sns.countplot(x='TransactionTypeLabel', data=fraud_data, palette='Set2')
plt.title('Transaction Type Analysis (Fraud Only)')
```

```
plt.xlabel('Transaction Type')
plt.ylabel('Fraud Transaction Count')
plt.xticks(rotation=45)  # Rotate labels for better readability
plt.tight_layout()
plt.show()
```

```
<ipython-input-46-94a04abfd6b2>:5: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `le

  sns.countplot(x='TransactionTypeLabel', data=fraud_data, palette='Set2')
```




```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100000 entries, 0 to 99999
Data columns (total 11 columns):
 #   Column                Non-Null Count   Dtype  
---  ------                --------------   -----  
 0   TransactionID         100000 non-null  int64  
 1   Amount                100000 non-null  float64
 2   MerchantID            100000 non-null  int64  
 3   TransactionType       100000 non-null  int64  
 4   Location              100000 non-null  int64  
 5   IsFraud               100000 non-null  int64  
 6   TransactionHour       100000 non-null  int32  
 7   TransactionDay        100000 non-null  int32  
 8   TransactionMonth      100000 non-null  int32  
 9   TransactionTypeLabel  100000 non-null  object 
 10  LocationLabel         100000 non-null  object 
dtypes: float64(1), int32(3), int64(5), object(2)
memory usage: 7.2+ MB
```

```
data_model = data.drop(columns=['TransactionID', 'TransactionTypeLabel', 'LocationLabel', 'MerchantID'])

X = data_model.drop(columns=['IsFraud'])
y = data_model['IsFraud']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42, stratify=y)
```

```
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

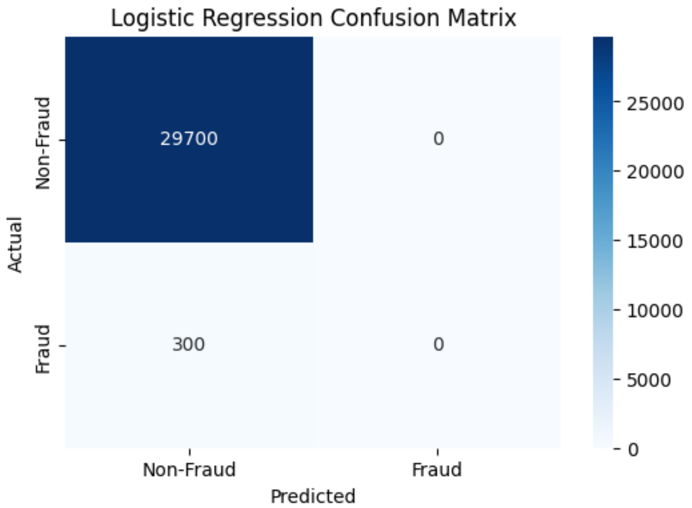
```
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier

models = {
```

```
    "Logistic Regression": LogisticRegression(random_state=42),
    "Decision Tree": DecisionTreeClassifier(random_state=42),
    "Random Forest": RandomForestClassifier(random_state=42),
    "XGBoost": XGBClassifier(random_state=42, use_label_encoder=False, eval_metric='logloss'),
}

for model_name, model in models.items():
    model.fit(X_train, y_train)
    models[model_name] = model
```

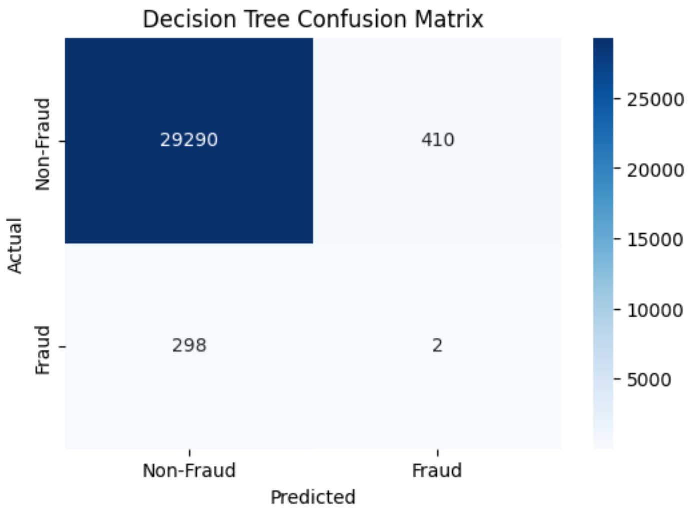
```
/usr/local/lib/python3.11/dist-packages/xgboost/core.py:158: UserWarning: [07:08:16] WARNING: /workspace/src/learner.cc:740:
Parameters: { "use_label_encoder" } are not used.

  warnings.warn(smsg, UserWarning)
```

```
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

# Evaluation dictionary to store results
evaluation_results = {}

# Evaluasi setiap model
for model_name, model in models.items():
    y_pred = model.predict(X_test)

    # Hitung accuracy
    accuracy = accuracy_score(y_test, y_pred)

    # Classification report
    clf_report = classification_report(y_test, y_pred)

     # Confusion matrix
    conf_matrix = confusion_matrix(y_test, y_pred)

    # Store results in dictionary
    evaluation_results[model_name] = {
        "accuracy": accuracy,
        "classification_report": clf_report,
        "confusion_matrix": conf_matrix
    }

    # Print evaluation for each model
    print(f"\n{model_name} Accuracy: {accuracy}")
    print(f"\n{model_name} Classification Report:\n", clf_report)

    # Plot confusion matrix
    plt.figure(figsize=(6, 4))
    sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=['Non-Fraud', 'Fraud'], yticklabels=['Non-Fraud', 'Fraud'])
    plt.title(f'{model_name} Confusion Matrix')
    plt.xlabel('Predicted')
    plt.ylabel('Actual')
    plt.show()
```

```
Logistic Regression Accuracy: 0.99

Logistic Regression Classification Report:
              precision    recall  f1-score   support

           0       0.99      1.00      0.99     29700
           1       0.00      0.00      0.00       300

    accuracy                           0.99     30000
   macro avg       0.49      0.50      0.50     30000
weighted avg       0.98      0.99      0.99     30000

/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-definec
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-definec
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-definec
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
```

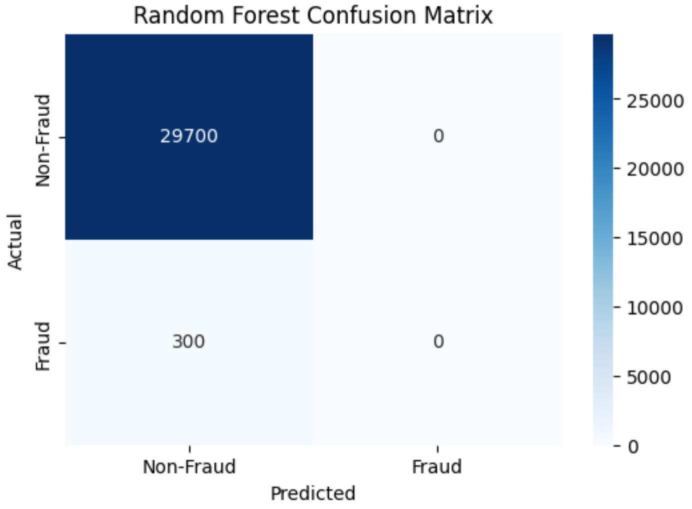



```
Decision Tree Accuracy: 0.9764

Decision Tree Classification Report:
              precision    recall  f1-score   support

           0       0.99      0.99      0.99     29700
           1       0.00      0.01      0.01       300

    accuracy                           0.98     30000
   macro avg       0.50      0.50      0.50     30000
weighted avg       0.98      0.98      0.98     30000
```

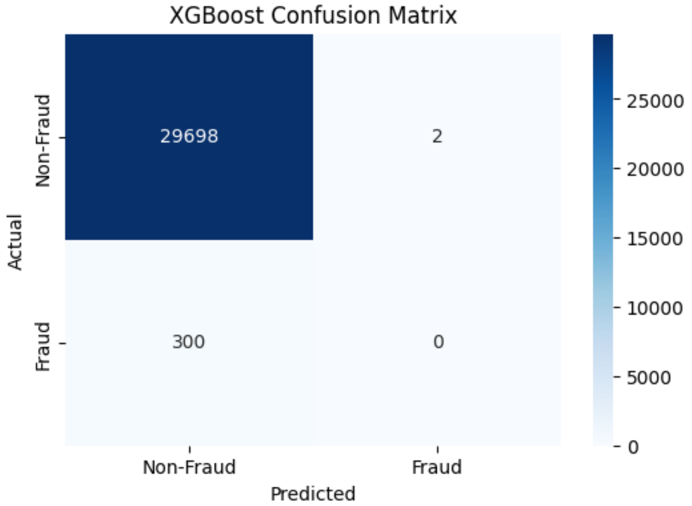



```
Random Forest Accuracy: 0.99

Random Forest Classification Report:
              precision    recall  f1-score   support

           0       0.99      1.00      0.99     29700
           1       0.00      0.00      0.00       300

    accuracy                           0.99     30000
   macro avg       0.49      0.50      0.50     30000
weighted avg       0.98      0.99      0.99     30000
```

```
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-definec
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-definec
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-definec
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
```




```
XGBoost Accuracy: 0.9899333333333333

XGBoost Classification Report:
              precision    recall  f1-score   support

           0       0.99      1.00      0.99     29700
           1       0.00      0.00      0.00       300

    accuracy                           0.99     30000
   macro avg       0.49      0.50      0.50     30000
weighted avg       0.98      0.99      0.98     30000
```



```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100000 entries, 0 to 99999
Data columns (total 11 columns):
 #   Column               Non-Null Count   Dtype  
---  ------               --------------   -----  
 0   TransactionID        100000 non-null  int64  
 1   Amount               100000 non-null  float64
 2   MerchantID           100000 non-null  int64  
 3   TransactionType      100000 non-null  int64  
 4   Location             100000 non-null  int64  
 5   IsFraud              100000 non-null  int64  
 6   TransactionHour      100000 non-null  int32  
 7   TransactionDay       100000 non-null  int32  
 8   TransactionMonth     100000 non-null  int32  
 9   TransactionTypeLabel 100000 non-null  object 
 10  LocationLabel        100000 non-null  object 
dtypes: float64(1), int32(3), int64(5), object(2)
memory usage: 7.2+ MB
```

```
data = data.drop(['TransactionID', 'TransactionTypeLabel', 'LocationLabel'], axis=1)
```

```
X = data.drop('IsFraud', axis=1)
y = data['IsFraud']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
from imblearn.over_sampling import SMOTE

smote = SMOTE(random_state=42)
X_train_res, y_train_res = smote.fit_resample(X_train, y_train)
```

```
# Print the count before SMOTE
print("Before SMOTE:")
print(f"Train Data - Positive class (Fraud): {y_train.sum()} | Negative class (Non-Fraud): {(y_train == 0).sum()}")
print(f"Test Data - Positive class (Fraud): {y_test.sum()} | Negative class (Non-Fraud): {(y_test == 0).sum()}")
```

```
Before SMOTE:
Train Data - Positive class (Fraud): 787 | Negative class (Non-Fraud): 79213
Test Data - Positive class (Fraud): 213 | Negative class (Non-Fraud): 19787
```

```
# Print the count after SMOTE
print("\nAfter SMOTE (Training Data):")
print(f"Train Data - Positive class (Fraud): {y_train_res.sum()} | Negative class (Non-Fraud): {(y_train_res == 0).sum()}")
print(f"Train Data Shape: {X_train_res.shape}")

print("\nTest Data (No Change):")
print(f"Test Data - Positive class (Fraud): {y_test.sum()} | Negative class (Non-Fraud): {(y_test == 0).sum()}")
print(f"Test Data Shape: {X_test.shape}")
```

```

After SMOTE (Training Data):
Train Data - Positive class (Fraud): 79213 | Negative class (Non-Fraud): 79213
Train Data Shape: (158426, 7)

Test Data (No Change):
Test Data - Positive class (Fraud): 213 | Negative class (Non-Fraud): 19787
Test Data Shape: (20000, 7)
```

for model_name, model in models.items(): model.fit(X_train_res, y_train_res) models[model_name] = model

```
# Evaluation dictionary to store results
evaluation_results = {}

# Evaluasi setiap model
for model_name, model in models.items():
    y_pred = model.predict(X_test)

    # Hitung accuracy
    accuracy = accuracy_score(y_test, y_pred)

    # Classification report
    clf_report = classification_report(y_test, y_pred)

    # Confusion matrix
    conf_matrix = confusion_matrix(y_test, y_pred)

    # Store results in dictionary
```

```
        evaluation_results[model_name] = {
            "accuracy": accuracy,
            "classification_report": clf_report,
            "confusion_matrix": conf_matrix
        }

        # Print evaluation for each model
        print(f"\n{model_name} Accuracy: {accuracy}")
        print(f"\n{model_name} Classification Report:\n", clf_report)

        # Plot confusion matrix
        plt.figure(figsize=(6, 4))
        sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=['Non-Fraud', 'Fraud'], yticklabels=['Non-Fraud', 'Fraud'])
        plt.title(f'{model_name} Confusion Matrix')
        plt.xlabel('Predicted')
        plt.ylabel('Actual')
        plt.show()
```