```python
# For data manipulation and analysis
import pandas as pd              # For data manipulation and analysis
import numpy as np               # For numerical computations
import matplotlib.pyplot as plt  # For data visualization (basic plots)
import seaborn as sns            # For advanced statistical visualizations

%matplotlib inline
```

```python
# Load the fraud detection dataset
data = pd.read_csv('Fraud_Detection.csv')

# Display the first five rows of the dataset
print(data.head())
```

```
   step      type    amount     nameOrig  oldbalanceOrg  newbalanceOrig  \
0     1   PAYMENT   9839.64  C1231006815       170136.0       160296.36
1     1   PAYMENT   1864.28  C1666544295        21249.0        19384.72
2     1  TRANSFER    181.00  C1305486145          181.0            0.00
3     1  CASH_OUT    181.00   C840083671          181.0            0.00
4     1   PAYMENT  11668.14  C2048537720        41554.0        29885.86

      nameDest  oldbalanceDest  newbalanceDest  isFraud  isFlaggedFraud
0  M1979787155             0.0             0.0        0               0
1  M2044282225             0.0             0.0        0               0
2   C553264065             0.0             0.0        1               0
3    C38997010         21182.0             0.0        1               0
4  M1230701703             0.0             0.0        0               0
```

```python
# Convert 'step' into actual time features (day, hour, etc.)
data['hour'] = data['step'] % 24
```

```python
#Diff of Orig and Dest
data['balance_diff_orig'] = data['oldbalanceOrg'] - data['newbalanceOrig']
data['balance_diff_dest'] = data['oldbalanceDest'] - data['newbalanceDest']
```

```python
# Check for missing values in the dataset
data.isnull().sum()
```

	0
step	0
type	0
amount	0
nameOrig	0
oldbalanceOrg	0
newbalanceOrig	0
nameDest	0
oldbalanceDest	0
newbalanceDest	0
isFraud	0
isFlaggedFraud	0
hour	0
balance_diff_orig	0
balance_diff_dest	0

```python
# Remove rows with NaN values
data = data.dropna()
```

```python
# Map the 'type' column's categorical values to numerical representations
data["type"] = data["type"].map({"CASH_OUT": 1, "PAYMENT": 2,
                                 "CASH_IN": 3, "TRANSFER": 4,
                                 "DEBIT": 5})
```

```
!pip install imbalanced-learn
```

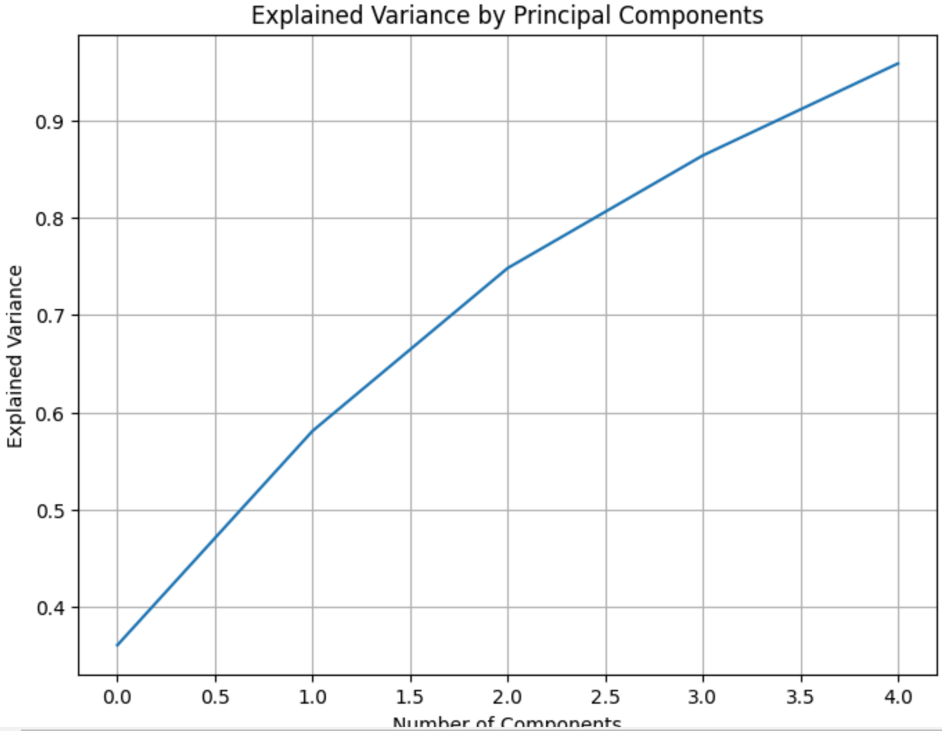
```
Requirement already satisfied: imbalanced-learn in /usr/local/lib/python3.10/dist-packages (0.12.4)
Requirement already satisfied: numpy>=1.17.3 in /usr/local/lib/python3.10/dist-packages (from imbalanced-learn) (1.26.4)
Requirement already satisfied: scipy>=1.5.0 in /usr/local/lib/python3.10/dist-packages (from imbalanced-learn) (1.13.1)
Requirement already satisfied: scikit-learn>=1.0.2 in /usr/local/lib/python3.10/dist-packages (from imbalanced-learn) (1.5.2)
Requirement already satisfied: joblib>=1.1.1 in /usr/local/lib/python3.10/dist-packages (from imbalanced-learn) (1.4.2)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from imbalanced-learn) (3.5.0)
```

```
import pandas as pd
from imblearn.over_sampling import SMOTE
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import classification_report, accuracy_score

# Assume 'data' is your dataset
# Drop irrelevant columns: 'nameOrig', 'nameDest' (unique to each transaction), and 'isFlaggedFraud'
X = data.drop(['isFraud', 'nameDest', 'nameOrig', 'isFlaggedFraud','step'], axis=1)
y = data['isFraud']  # Target variable: Fraud flag

# Handling Class Imbalance
# The dataset is imbalanced since fraudulent transactions make up only a small portion using SMOTE

# Apply SMOTE to balance the dataset
smote = SMOTE(random_state=42)
X_resampled, y_resampled = smote.fit_resample(X, y)

# Split the resampled data into training and testing sets (using 10% for testing)
X_train, X_test, y_train, y_test = train_test_split(X_resampled, y_resampled, test_size=0.1, random_state=42)

# Scale the features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)  # Fit and transform on the training data
X_test = scaler.transform(X_test)  # Only transform the test data

# Output the shape of the original and resampled datasets to check the balancing
print(f"Original dataset shape: {X.shape}")
print(f"Resampled dataset shape: {X_resampled.shape}")

print(f"Before SMOTE, counts of label '1' (fraud): {sum(y == 1)}")
print(f"After SMOTE, counts of label '1' (fraud): {sum(y_resampled == 1)}")
```

```
Original dataset shape: (6362620, 9)
Resampled dataset shape: (12708814, 9)
Before SMOTE, counts of label '1' (fraud): 8213
After SMOTE, counts of label '1' (fraud): 6354407
```

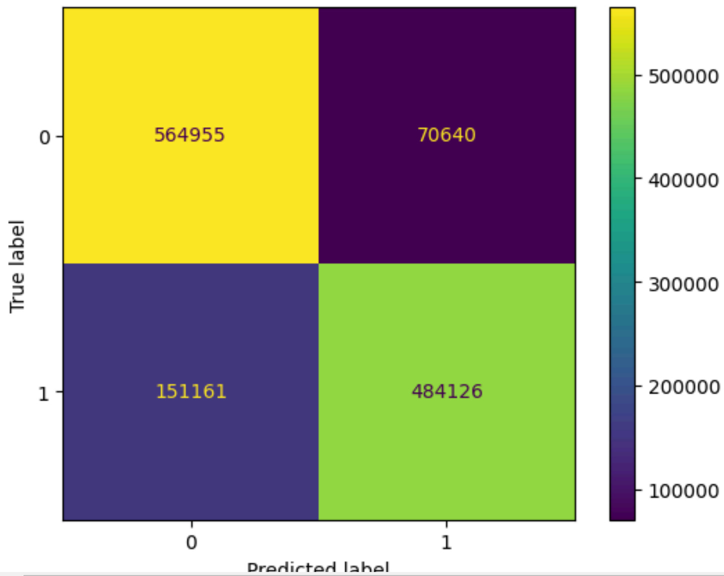
```
from sklearn.decomposition import PCA

# Scale the features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X_train)



# Apply PCA to reduce dimensionality
pca = PCA(n_components=0.95)  # Keep 95% of the variance
X_pca = pca.fit_transform(X_scaled)

# Print the number of components after PCA
print(f"Number of components after PCA: {pca.n_components_}")

# Plot the explained variance ratio of each principal component
plt.figure(figsize=(8, 6))
plt.plot(np.cumsum(pca.explained_variance_ratio_))
plt.xlabel('Number of Components')
plt.ylabel('Explained Variance')
plt.title('Explained Variance by Principal Components')
plt.grid(True)
plt.show()
```

```
Number of components after PCA: 5
```



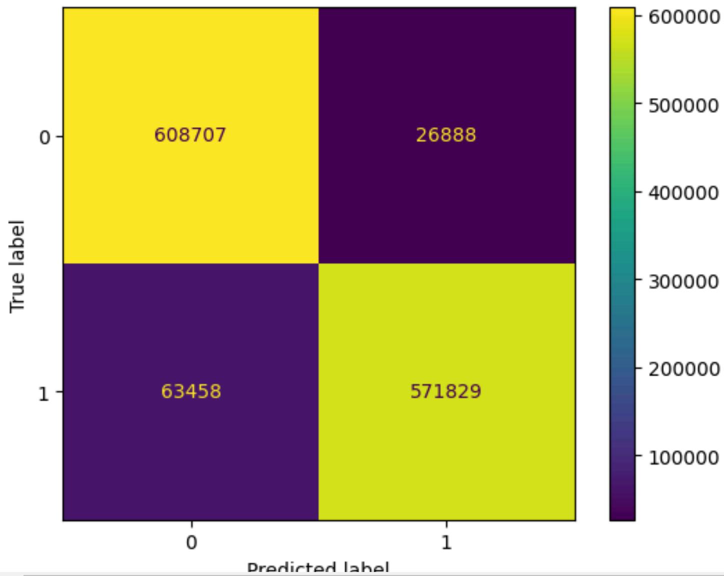

```
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, accuracy_score, confusion_matrix, ConfusionMatrixDisplay
import matplotlib.pyplot as plt

# Scale the features
scaler = StandardScaler()

# Fit the scaler on the training data and transform
X_train_scaled = scaler.fit_transform(X_train)

# Apply PCA to the training set
pca = PCA(n_components=0.95)  # Keep 95% of the variance
X_train_pca = pca.fit_transform(X_train_scaled)

# Apply the same transformations to the test set
X_test_scaled = scaler.transform(X_test)
X_test_pca = pca.transform(X_test_scaled)

# Train Logistic Regression
lr = LogisticRegression(random_state=42)
lr.fit(X_train_pca, y_train)

# Make predictions with Logistic Regression
y_pred_lr = lr.predict(X_test_pca)

# Evaluate Logistic Regression
print("Logistic Regression - Classification Report:")
print(classification_report(y_test, y_pred_lr))
print(f"Logistic Regression Accuracy: {accuracy_score(y_test, y_pred_lr)}")

# Confusion Matrix
cm_lr = confusion_matrix(y_test, y_pred_lr)
ConfusionMatrixDisplay(confusion_matrix=cm_lr, display_labels=lr.classes_).plot()

# Show the confusion matrix plot
plt.show()
```

```
Logistic Regression - Classification Report:
              precision    recall  f1-score   support

           0       0.79      0.89      0.84    635595
           1       0.87      0.76      0.81    635287

    accuracy                           0.83   1270882
   macro avg       0.83      0.83      0.82   1270882
weighted avg       0.83      0.83      0.82   1270882

Logistic Regression Accuracy: 0.8254747490325617
```



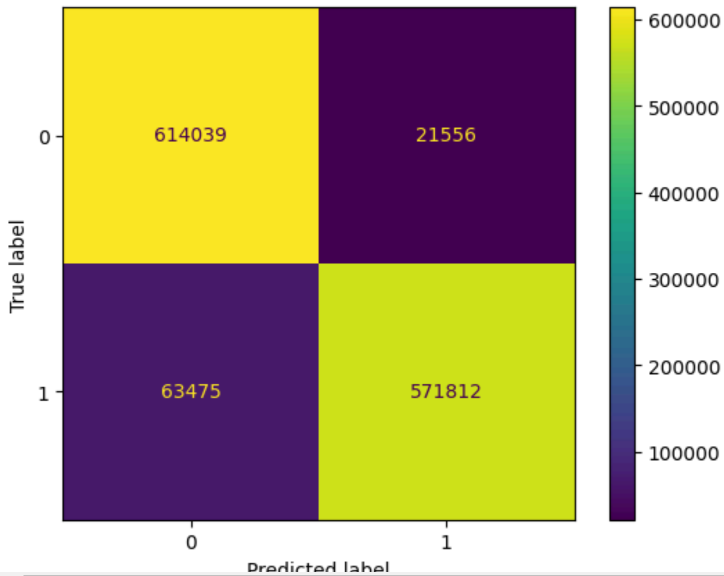

```
from sklearn.metrics import classification_report, accuracy_score, confusion_matrix, ConfusionMatrixDisplay

# Logistic Regression
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, accuracy_score


# Train a Logistic Regression model
lr = LogisticRegression()
lr.fit(X_train, y_train)

# Make predictions and evaluate
y_pred = lr.predict(X_test)
print(classification_report(y_test, y_pred))
print(f"Accuracy: {accuracy_score(y_test, y_pred)}")

# Display confusion matrix
print('Logistic Regression: Confusion Matrix')


# Confusion Matrix
cm = confusion_matrix(y_test, y_pred)
ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=lr.classes_).plot()

# Show the confusion matrix plot
plt.show()
```

```
              precision    recall  f1-score   support

           0       0.91      0.96      0.93    635595
           1       0.96      0.90      0.93    635287

    accuracy                           0.93   1270882
   macro avg       0.93      0.93      0.93   1270882
weighted avg       0.93      0.93      0.93   1270882

Accuracy: 0.9289107879409733
Logistic Regression: Confusion Matrix
```



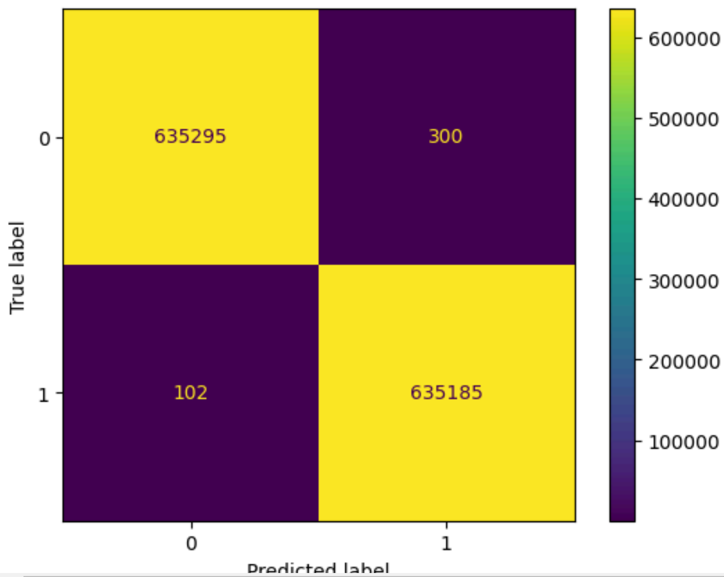

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import classification_report, accuracy_score, confusion_matrix, ConfusionMatrixDisplay
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA

# Scale the training data
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)

# Apply PCA
pca = PCA(n_components=0.95)  # Retain 95% of variance
X_train_pca = pca.fit_transform(X_train_scaled)

X_test_scaled = scaler.transform(X_test)
X_test_pca = pca.transform(X_test_scaled)


# Train Decision Tree Classifier
dt = DecisionTreeClassifier(max_depth=10, random_state=42)  # You can adjust `max_depth` as needed
dt.fit(X_train_pca, y_train)

# Make predictions with Decision Tree
y_pred_dt = dt.predict(X_test_pca)

# Evaluate Decision Tree Classifier
print("Decision Tree - Classification Report:")
print(classification_report(y_test, y_pred_dt))
print(f"Decision Tree Accuracy: {accuracy_score(y_test, y_pred_dt)}")

# Confusion Matrix
cm_dt = confusion_matrix(y_test, y_pred_dt)
ConfusionMatrixDisplay(confusion_matrix=cm_dt, display_labels=dt.classes_).plot()

# Show the confusion matrix plot
plt.show()
```

```
Decision Tree - Classification Report:
              precision    recall  f1-score   support

           0       0.91      0.97      0.94    635595
           1       0.96      0.90      0.93    635287

    accuracy                           0.93   1270882
   macro avg       0.93      0.93      0.93   1270882
weighted avg       0.93      0.93      0.93   1270882

Decision Tree Accuracy: 0.9330929228677407
```



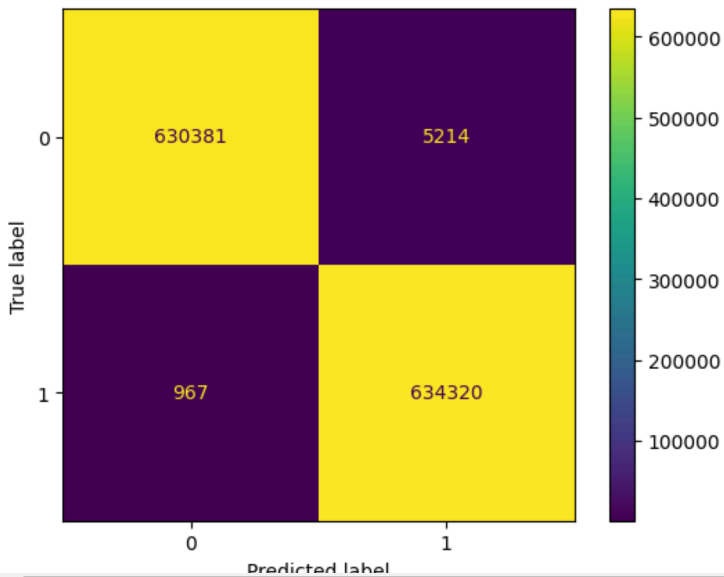

```
#Decision Tree Classifier
from sklearn.tree import DecisionTreeClassifier

# Train a Decision Tree model
dt = DecisionTreeClassifier(random_state=42)
dt.fit(X_train, y_train)

# Make predictions and evaluate
y_pred_dt = dt.predict(X_test)
print(classification_report(y_test, y_pred_dt))
print(f"Accuracy: {accuracy_score(y_test, y_pred_dt)}")

# Display confusion matrix
print('Decision Tree Classifier: Confusion Matrix')


# Confusion Matrix
cm_dt= confusion_matrix(y_test, y_pred_dt)
ConfusionMatrixDisplay(confusion_matrix=cm_dt , display_labels=lr.classes_).plot()

# Show the confusion matrix plot
plt.show()
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    635595
           1       1.00      1.00      1.00    635287

    accuracy                           1.00   1270882
   macro avg       1.00      1.00      1.00   1270882
weighted avg       1.00      1.00      1.00   1270882

Accuracy: 0.9996836842444853
Decision Tree Classifier: Confusion Matrix
```




```python
# Random Forest Classifier

from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, accuracy_score


# Train a Random Forest model
rf = RandomForestClassifier(n_estimators=50, max_depth=10, random_state=42, n_jobs=-1)
rf.fit(X_train, y_train)

# Make predictions and evaluate
y_pred_rf = rf.predict(X_test)

# Print classification report and accuracy
print(classification_report(y_test, y_pred_rf))
print(f"Accuracy: {accuracy_score(y_test, y_pred_rf)}")


# Display confusion matrix
print('Random Forest Classifier: Confusion Matrix')


# Confusion Matrix
cm_rf= confusion_matrix(y_test, y_pred_rf)
ConfusionMatrixDisplay(confusion_matrix=cm_rf , display_labels=lr.classes_).plot()

# Show the confusion matrix plot
plt.show()
```

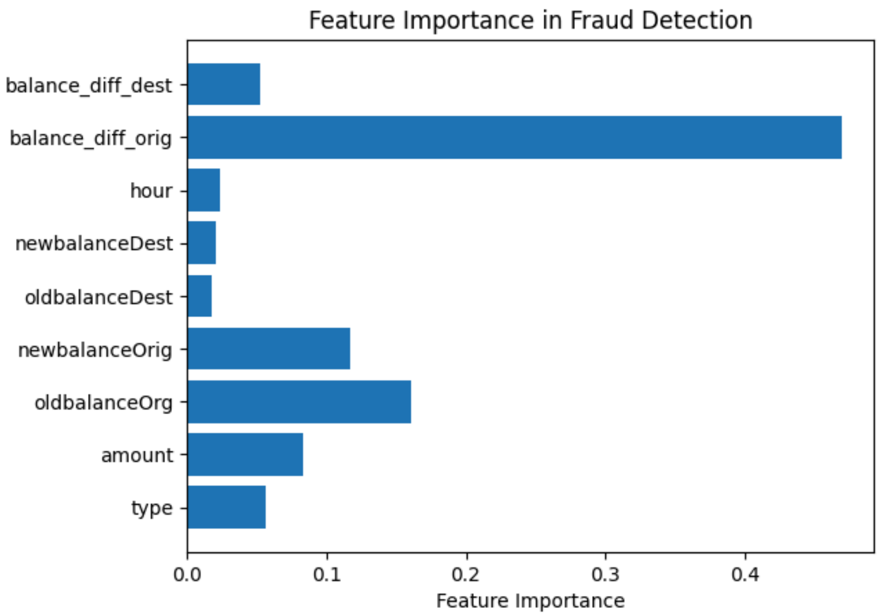
```
              precision    recall  f1-score   support

           0       1.00      0.99      1.00    635595
           1       0.99      1.00      1.00    635287

    accuracy                           1.00   1270882
   macro avg       1.00      1.00      1.00   1270882
weighted avg       1.00      1.00      1.00   1270882

Accuracy: 0.9951364485451836
Random Forest Classifier: Confusion Matrix
```



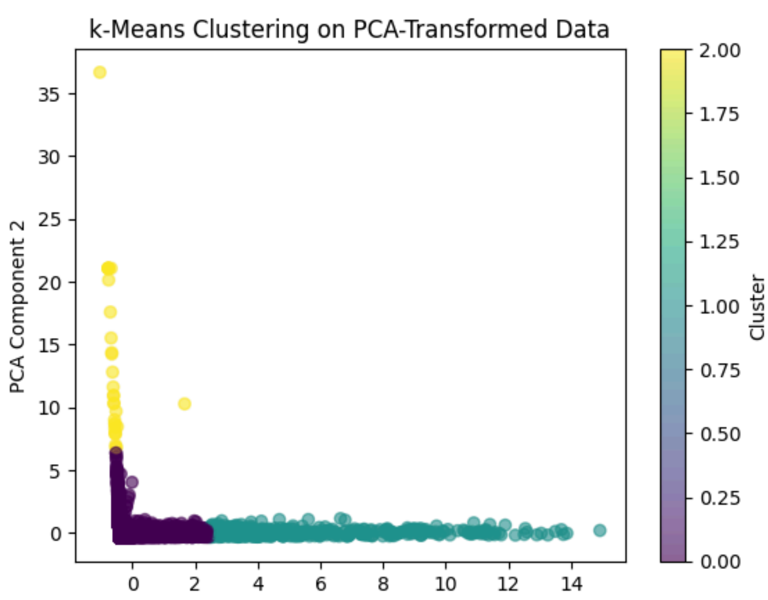

```
# Unsupervised Learning

from sklearn.ensemble import IsolationForest

# Train an Isolation Forest model for anomaly detection
iso_forest = IsolationForest(contamination=0.01)
y_pred_if = iso_forest.fit_predict(X)

# Convert anomaly labels (-1 for outliers) to binary classification (1 for fraud, 0 for normal)
y_pred_if = [1 if x == -1 else 0 for x in y_pred_if]

# Evaluate anomaly detection results
print(classification_report(y, y_pred_if))
print(f"Accuracy: {accuracy_score(y, y_pred_if)}")
```

```
              precision    recall  f1-score   support

           0       1.00      0.99      0.99   6354407
           1       0.02      0.15      0.03      8213

    accuracy                           0.99   6362620
   macro avg       0.51      0.57      0.51   6362620
weighted avg       1.00      0.99      0.99   6362620

Accuracy: 0.9890947439891115
```

```
#Feature Importance
import matplotlib.pyplot as plt

# Plot feature importance for Random Forest model
importances = rf.feature_importances_
feature_names = X.columns

plt.barh(feature_names, importances)
plt.xlabel('Feature Importance')
plt.title('Feature Importance in Fraud Detection')
plt.show()
plt.savefig('Feature Importance in Fraud Detection.png')
```

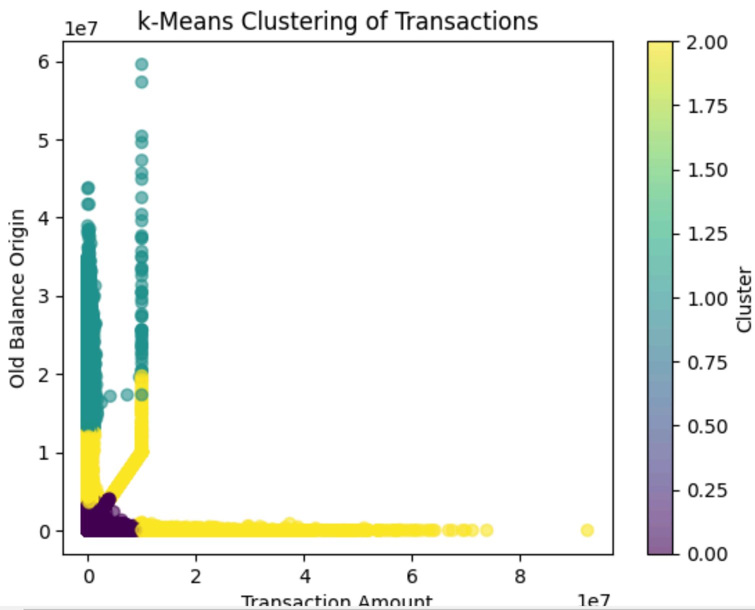



```
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt

# Take a random sample for clustering
data_sample = data.sample(n=10000, random_state=42)

# Scale the sample and apply PCA
scaled_features_sample = scaler.fit_transform(data_sample[['amount', 'oldbalanceOrg', 'newbalanceOrig']])
pca_sample = pca.fit_transform(scaled_features_sample)

# Apply K-Means on PCA-transformed data
kmeans = KMeans(n_clusters=3, random_state=42)
data_sample['cluster'] = kmeans.fit_predict(pca_sample)

# Visualize the clusters
plt.scatter(pca_sample[:, 0], pca_sample[:, 1], c=data_sample['cluster'], cmap='viridis', alpha=0.6)
plt.title('k-Means Clustering on PCA-Transformed Data')
plt.xlabel('PCA Component 1')
plt.ylabel('PCA Component 2')
plt.colorbar(label='Cluster')
plt.show()
```




```
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
```

```
# Select relevant features for clustering
features = data[['amount', 'oldbalanceOrg', 'newbalanceOrig']]

# Scale the features
scaler = StandardScaler()
scaled_features = scaler.fit_transform(features)

# Apply k-Means clustering with 3 clusters
kmeans = KMeans(n_clusters=3, random_state=42)
data['cluster'] = kmeans.fit_predict(scaled_features)

# Visualize the clusters
plt.scatter(data['amount'], data['oldbalanceOrg'], c=data['cluster'], cmap='viridis', alpha=0.6)
plt.title('k-Means Clustering of Transactions')
plt.xlabel('Transaction Amount')
plt.ylabel('Old Balance Origin')
plt.colorbar(label='Cluster')
plt.show()
```




```
from sklearn.ensemble import IsolationForest

# Apply Isolation Forest on PCA-transformed data
iso_forest = IsolationForest(n_estimators=100, random_state=42)
data_sample['anomaly_score'] = iso_forest.fit_predict(pca_sample)

# Visualize the anomalies
plt.scatter(pca_sample[:, 0], pca_sample[:, 1], c=data_sample['anomaly_score'], cmap='coolwarm', alpha=0.6)
plt.title('Anomaly Detection using Isolation Forest on PCA-Transformed Data')
plt.xlabel('PCA Component 1')
plt.ylabel('PCA Component 2')
plt.colorbar(label='Anomaly Score')
plt.show()
```