+ Code + Text

```
from google.colab import files
uploaded=files.upload()
print(uploaded)
```

Choose Files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving Online Payment Fraud Detection.csv to Online Payment Fraud Detection.csv
{'Online Payment Fraud Detection.csv': b'step,type,amount,nameOrig,oldbalanceOrg,newbalanceOrig,nameDest,oldbalanceDest,newbalanceDest,is
```

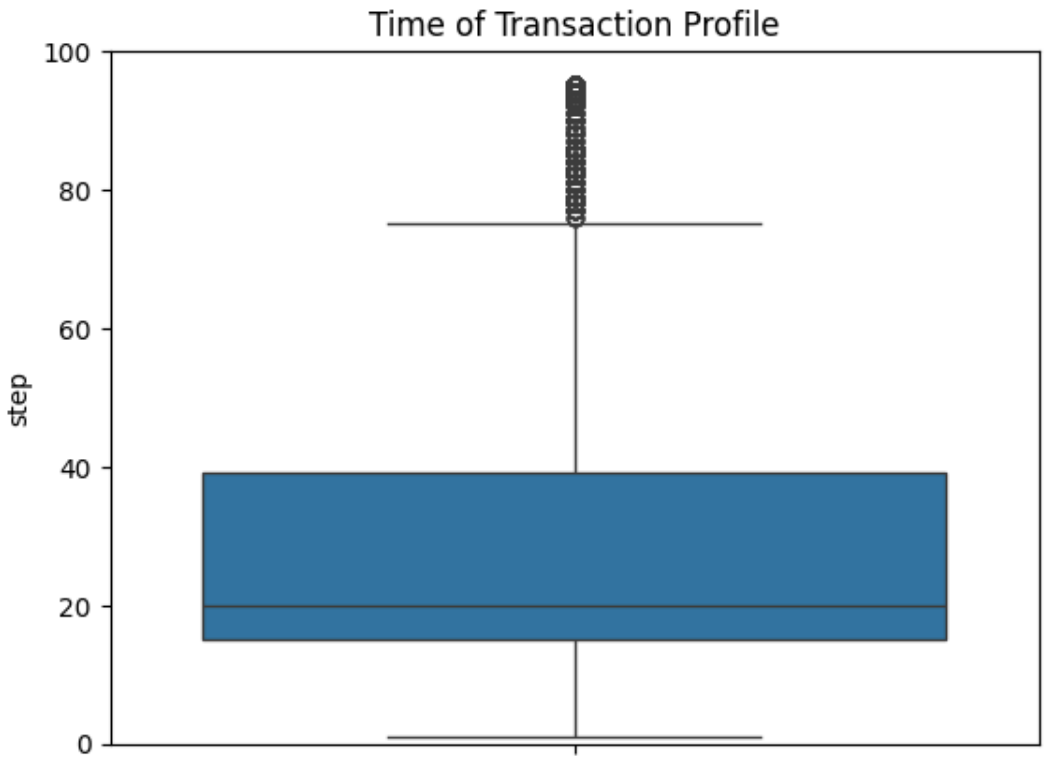
Start coding or generate with AI.

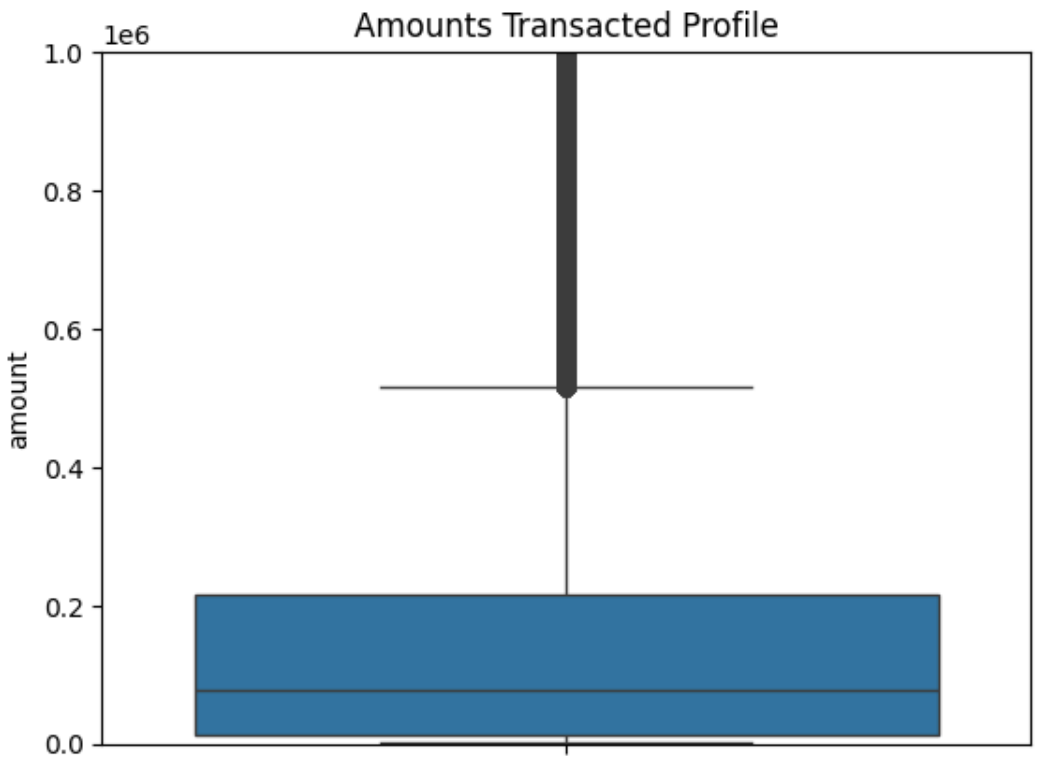
```
#import the libaries to work with for EDA
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
#loading the data set
data = pd.read_csv('Online Payment Fraud Detection.csv',encoding='unicode-escape')
# Checking the size of the dataset (Rows,Columns)
data.shape
data.head()
data.tail()
data.describe()
data.columns
data.info()
data.dtypes
data.isna().sum()
#Checking which recipients stand out

data.nameDest.unique()
#Investigating to check unique customers
data.nameOrig.unique()
#investigating to see how many times a customer started a transaction
data.nameOrig.value_counts()
#How many times a recipient got a transaction
data.nameDest.value_counts()
data.amount.max()


#Investigating how many times a particular type of transaction was carried out.

data.type.value_counts()

#Investigating the top customers and the type of transactions they initiated
top_ten = data.groupby('nameOrig').type.sum().sort_values(ascending=False)[:10]
top_ten
#Checking the average amounttransacted
data['amount'].mean()
sns.boxplot(y=data.step)
plt.title('Time of Transaction Profile')
plt.ylim(0,100)
plt.show()
sns.boxplot(y=data.amount)
plt.title('Amounts Transacted Profile')
plt.ylim(0,1000000)
plt.show()
sns.boxplot(y=data.isFraud)
plt.title('Fraud Profile')
plt.ylim(-1,1)
plt.show()
#Visualising the spread of fraud variables across the dataset

Online_Payment_layout = sns.PairGrid(data, vars = ['step', 'oldbalanceOrg', 'newbalanceOrig', 'oldbalanceDest', 'newbalanceDest'], hue = 'isFra

Online_Payment_layout.map_diag(plt.hist, alpha = 0.6)
Online_Payment_layout.map_offdiag(plt.scatter, alpha = 0.5)
Online_Payment_layout.add_legend()
sns.barplot(x='amount', y='type', hue= 'isFraud', data=data)
plt.show()
sns.catplot(data=data,kind='box')

plt.ylim(0,2000000)

# Sepearating Fraudulent transaction from non fraudulent

Fraudulent_Transaction = data[data.isFraud ==1]
Not_Fraudulent_Transaction = data[data.isFraud ==0]
print('Fraudulent Transaction: {}'.format(len(Fraudulent_Transaction)))
print('Not Fraudulent Transaction: {}'.format(len(Not_Fraudulent_Transaction)))
   #Understanding The statistical nature of Non Fraudulent Transactions.

Not_Fraudulent_Transaction.amount.describe()
```

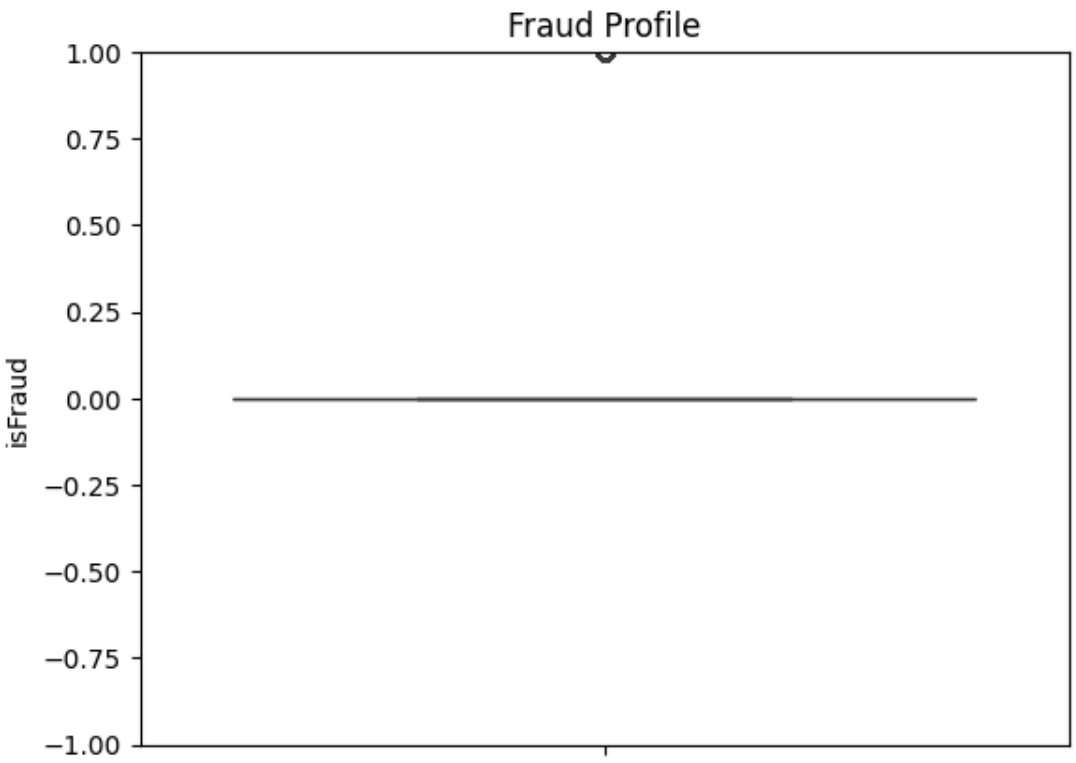
```
#understanding the statistical nature of fraudulent transactions.

Fraudulent_Transaction.amount.describe()

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline

# Define features (X) and target (Y)
X = data.drop('isFraud', axis=1)
Y = data['isFraud']

# Identify categorical and numerical features
categorical_features = ['type']
numerical_features = ['step', 'amount', 'oldbalanceOrg', 'newbalanceOrig', 'oldbalanceDest', 'newbalanceDest']

# Create a column transformer for one-hot encoding
preprocessor = ColumnTransformer(
    transformers=[
        ('cat', OneHotEncoder(handle_unknown='ignore'), categorical_features),
        ('num', 'passthrough', numerical_features)
    ],
    remainder='drop' # Drop other columns
)

# Create a pipeline with the preprocessor and Logistic Regression model
model = Pipeline(steps=[('preprocessor', preprocessor),
                        ('classifier', LogisticRegression())])


#create X_train, X_test, Y_train, Y_test
# using test_size of 20%

X_train, X_test, Y_train, Y_test = train_test_split(X,Y,test_size=0.2, stratify=Y, random_state=2)
print('\n',X_train.head(2))

print('\n',X_test.head(2))

print('\n',Y_train.head(2))

print('\n',Y_test.head(2))

#Training model with Training data
model.fit(X_train, Y_train)
# LogisticRegression() # This line is not needed

model_pred = model.predict(X_test)
# Obtain model probabilities
probs = model.predict_proba(X_test)
#importing the methods
from sklearn import metrics
from sklearn.metrics import confusion_matrix, classification_report, f1_score, accuracy_score, precision_recall_curve, average_precision_score,
print('\nClassification Report:')
print(classification_report(Y_test, model_pred))
# check True Negatives/Positives, False Negatives/Positives
pd.DataFrame(confusion_matrix(Y_test, model_pred),
             columns=['Predicted Negative(0) ', 'Predicted Positive(1)'],
             index=['Actually Negative(0)', 'Actually Positive(1)'])

# Print confusion matrix using predictions in context
pd.DataFrame(confusion_matrix(Y_test, model_pred),
             columns=['Predicted Not Fraud(0) ', 'Predicted Fraud(1)'],
             index=['Actually Not Fraud(0)', 'Actually Fraud(1)'])

# ACCURACY SCORE
print('Accuracy:',accuracy_score(Y_test, model_pred))
# Calculate average precision and the P-R curve
average_precision = average_precision_score(Y_test, model_pred)
average_precision
#define metrics
y_pred_proba = model.predict_proba(X_test)[::,1]
fpr, tpr, _ = metrics.roc_curve(Y_test,  y_pred_proba)
auc = metrics.roc_auc_score(Y_test, y_pred_proba)


#create ROC curve
plt.plot(fpr,tpr,label="AUC="+str(auc))
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.legend(loc=4)
plt.show()

print('AUC Score:')
print(roc_auc_score(Y_test, probs[:,1]))
```

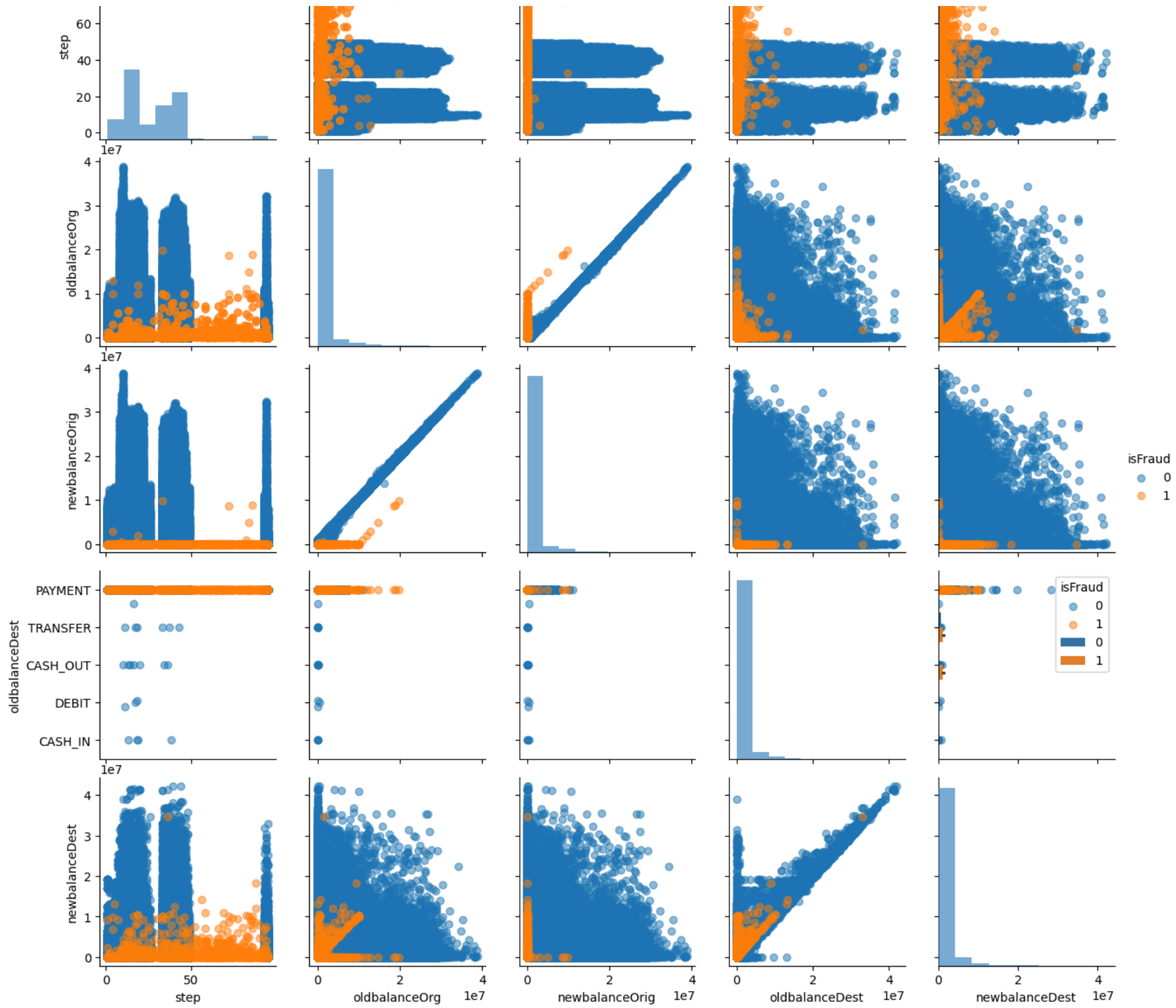
```
# Create a pipeline with the preprocessor and Random Forest model
# model = Pipeline(steps=[('preprocessor', preprocessor),   ('classifier', RandomForestClassifier(random_state=5, n_estimators=20))])


model.fit(X_train,Y_train)
# RandomForestClassifier(n_estimators=20, random_state=5) # This line is not needed

model_pred = model.predict(X_test)
# Obtain model probabilities
probs = model.predict_proba(X_test)
#importing the methods
from sklearn.metrics import confusion_matrix, classification_report, f1_score, accuracy_score, precision_recall_curve, average_precision_score,
# Print classification report using predictions
print('Classification_Report:\n',classification_report(Y_test, model_pred))

# Print confusion matrix using predictions
pd.DataFrame(confusion_matrix(Y_test, model_pred),
             columns=['Predicted Negative(0) ', 'Predicted Positive(1)'],
             index=['Actually Negative(0)', 'Actually Positive(1)'])

# Print confusion matrix using predictions in Context
pd.DataFrame(confusion_matrix(Y_test, model_pred),
             columns=['Predicted Not Fraud(0) ', 'Predicted Fraud(1)'],
             index=['Actually Not Fraud(0)', 'Actually Fraud(1)'])

# ACCURACY SCORE
print('Accuracy:',accuracy_score(Y_test, model_pred))
# Calculate average precision and the P-R curve
average_precision = average_precision_score(Y_test, model_pred)
average_precision

#define metrics
y_pred_proba = model.predict_proba(X_test)[::,1]
fpr, tpr, _ = metrics.roc_curve(Y_test,  y_pred_proba)
auc = metrics.roc_auc_score(Y_test, y_pred_proba)


#create ROC curve
plt.plot(fpr,tpr,label="AUC="+str(auc))
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.legend(loc=4)
plt.show()
# Print ROC_AUC score using probabilities
print('AUC Score:')
print(roc_auc_score(Y_test, probs[:, 1]))



labels = data['isFraud'].value_counts().plot(kind='bar')
plt.xlabel('Fraudulent (1) / Non-Fraudulent (0)')
plt.ylabel('Count')
plt.title('Fraudulent vs Non-Fraudulent Transactions')
plt.show()

# Sample data for online payment fraud detection
categories = ['Fraudulent', 'Non-Fraudulent']
fraud_counts = [150, 850]

# Create a bar chart
plt.figure(figsize=(8, 6))
plt.bar(categories, fraud_counts, color=['red', 'green'], edgecolor='black')

# Customize the chart
plt.title('Online Payment Fraud Detection', fontsize=14, pad=15)
plt.xlabel('Transaction Type', fontsize=12)
plt.ylabel('Number of Transactions', fontsize=12)
plt.grid(axis='y', linestyle='--', alpha=0.7)

# Add value labels on top of each bar
for i, count in enumerate(fraud_counts):
    plt.text(i, count + 10, str(count), ha='center', fontsize=12)

# Display the chart (optional)
plt.tight_layout()
plt.show()

plt.figure(figsize=(12,6))
sns.heatmap(data.apply(lambda x: x.factorize()[0]).corr(), annot=True, cmap='Blues')

plt.show()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1048575 entries, 0 to 1048574
Data columns (total 10 columns):
 #   Column          Non-Null Count    Dtype
---  ------          --------------    -----
 0   step            1048575 non-null  int64
 1   type            1048575 non-null  object
 2   amount          1048575 non-null  float64
 3   nameOrig        1048575 non-null  object
 4   oldbalanceOrg   1048575 non-null  float64
 5   newbalanceOrig  1048575 non-null  float64
 6   nameDest        1048575 non-null  object
 7   oldbalanceDest  1048575 non-null  float64
 8   newbalanceDest  1048575 non-null  float64
 9   isFraud         1048575 non-null  int64
dtypes: float64(5), int64(2), object(3)
memory usage: 80.0+ MB
```

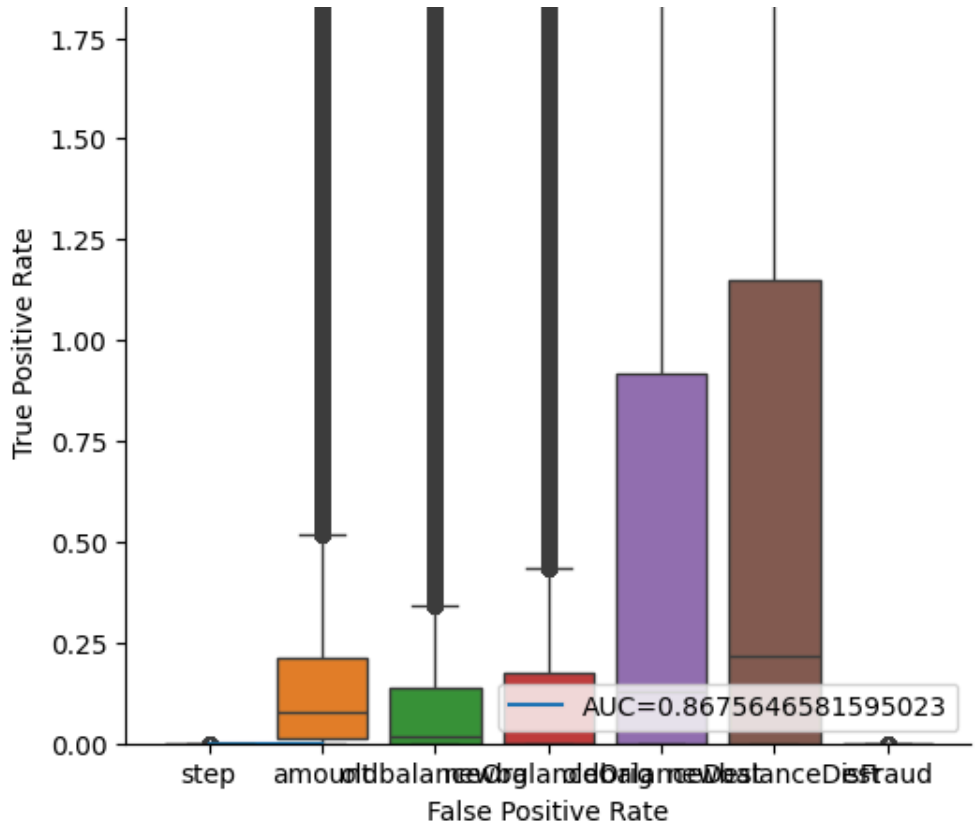




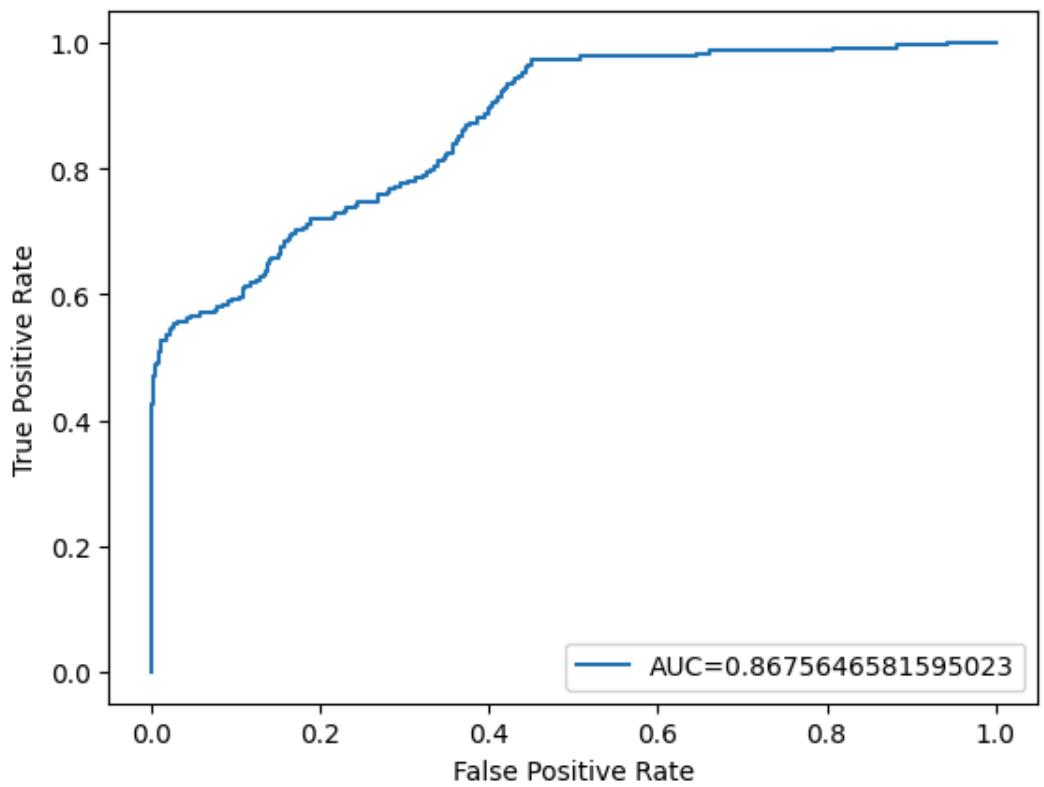




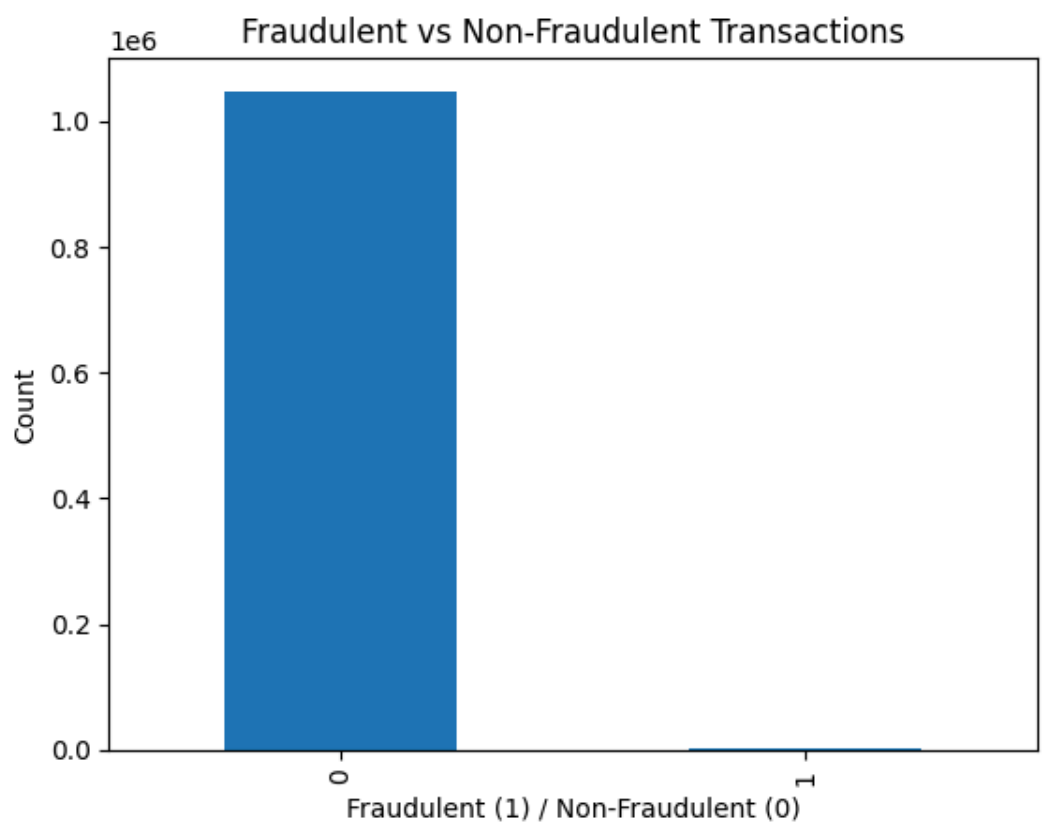

```
Fraudulent Transaction: 1142
Not Fraudulent Transaction: 1047433

         step      type   amount     nameOrig  oldbalanceOrg  newbalanceOrig  \
628981     34   PAYMENT  2118.79  C1066817532           96.0            0.00
413872     18  TRANSFER   834.48   C764330759       141762.0       140927.52

          nameDest  oldbalanceDest  newbalanceDest
628981  M435429538             0.0            0.00
413872  C509958751        100187.0        54578.32

         step      type     amount    nameOrig  oldbalanceOrg  newbalanceOrig  \
66544       9  CASH_OUT  104630.56  C441630332            0.0            0.00
945978     44   PAYMENT   18627.21  C641483617       206710.0       188082.79

          nameDest  oldbalanceDest  newbalanceDest
66544   C597676461      1223188.32      1995386.37
945978  M169655611            0.00            0.00

 628981    0
413872    0
Name: isFraud, dtype: int64

 66544     0
945978    0
Name: isFraud, dtype: int64

Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    209487
           1       0.57      0.36      0.44       228

    accuracy                           1.00    209715
   macro avg       0.78      0.68      0.72    209715
weighted avg       1.00      1.00      1.00    209715

Accuracy: 0.9990081777650621
```



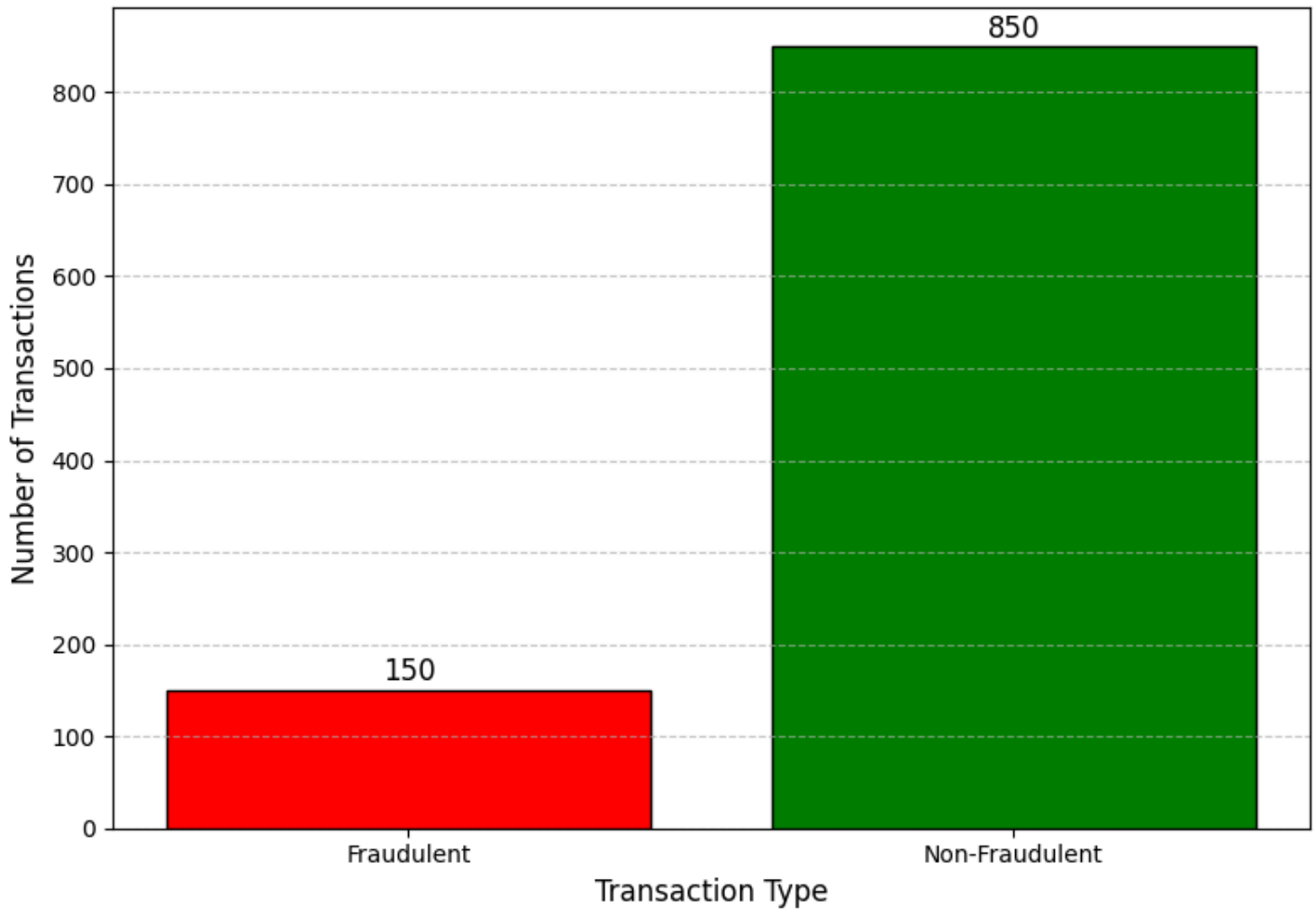

```
AUC Score:
0.8675646581595023
Classification_Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    209487
           1       0.57      0.36      0.44       228

    accuracy                           1.00    209715
   macro avg       0.78      0.68      0.72    209715
weighted avg       1.00      1.00      1.00    209715

Accuracy: 0.9990081777650621
```

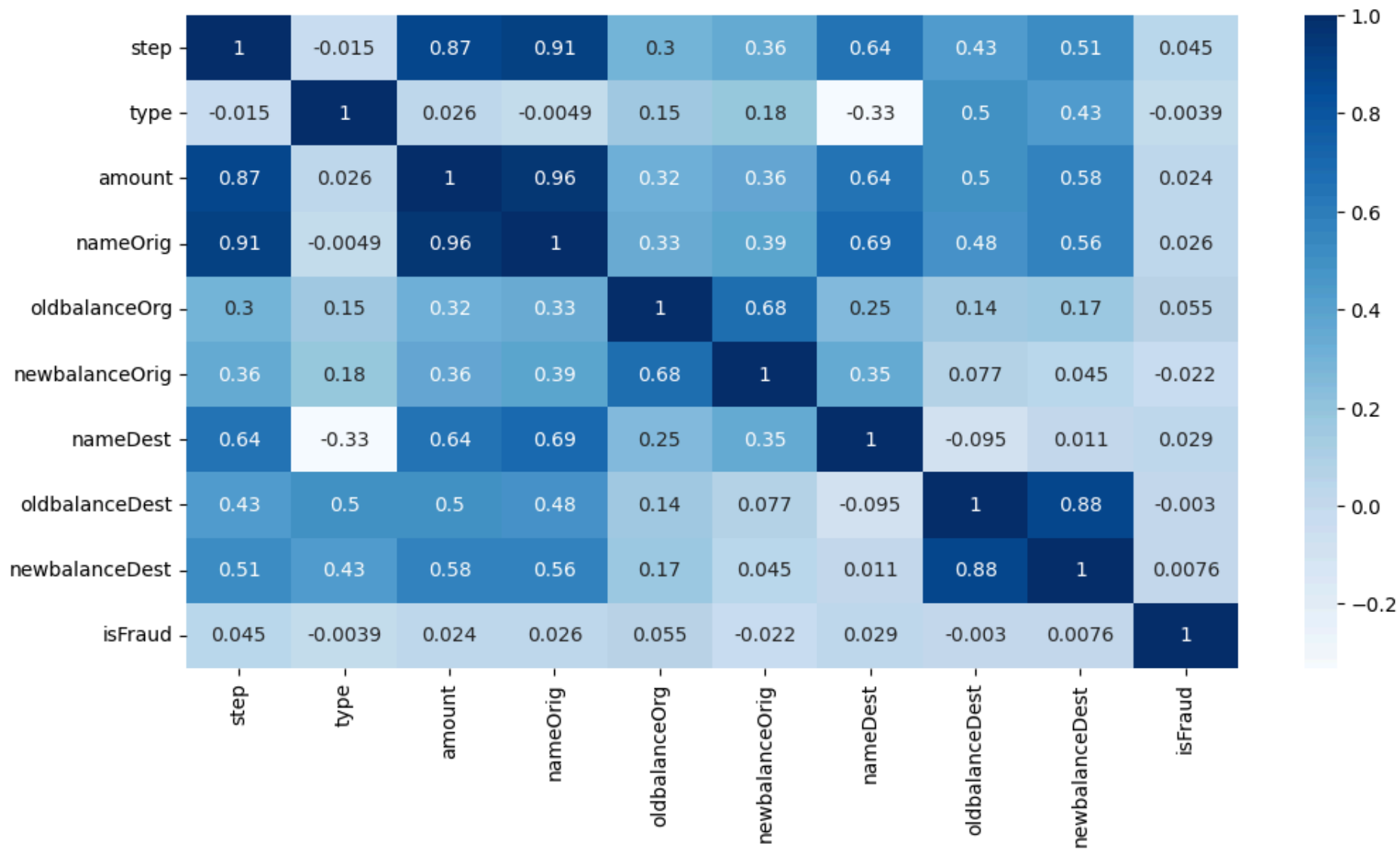



```
AUC Score:
0.8675646581595023
```



Online Payment Fraud Detection


850
150
Number of Transactions
0
100
200
300
400
500
600
700
800
Fraudulent
Non-Fraudulent
Transaction Type


step 1 -0.015 0.87 0.91 0.3 0.36 0.64 0.43 0.51 0.045
type -0.015 1 0.026 -0.0049 0.15 0.18 -0.33 0.5 0.43 -0.0039
amount 0.87 0.026 1 0.96 0.32 0.36 0.64 0.5 0.58 0.024
nameOrig 0.91 -0.0049 0.96 1 0.33 0.39 0.69 0.48 0.56 0.026
oldbalanceOrg 0.3 0.15 0.32 0.33 1 0.68 0.25 0.14 0.17 0.055
newbalanceOrig 0.36 0.18 0.36 0.39 0.68 1 0.35 0.077 0.045 -0.022
nameDest 0.64 -0.33 0.64 0.69 0.25 0.35 1 -0.095 0.011 0.029
oldbalanceDest 0.43 0.5 0.5 0.48 0.14 0.077 -0.095 1 0.88 -0.003
newbalanceDest 0.51 0.43 0.58 0.56 0.17 0.045 0.011 0.88 1 0.0076
isFraud 0.045 -0.0039 0.024 0.026 0.055 -0.022 0.029 -0.003 0.0076 1
step
type
amount
nameOrig
oldbalanceOrg
newbalanceOrig
nameDest
oldbalanceDest
newbalanceDest
isFraud
1.0
0.8
0.6
0.4
0.2
0.0
−0.2

```
# Line 1: Import necessary libraries.
import uuid # For generating unique user IDs
import hashlib # For password hashing (conceptual, use bcrypt in production)
import json # For simulating JWT payload encoding/decoding
import base64 # For base64 encoding in JWT simulation
import time # For simulating JWT expiration

# --- 1. User Class (Profile Management) ---
# Line 2: Defines the structure for a user, including their profile data, preferences, and roles.
class User:
    def __init__(self, username, password_hash, user_id=None, preferences=None, interests=None, roles=None):
        self.user_id = user_id if user_id else str(uuid.uuid4()) # Unique ID for the user
        self.username = username
        self.password_hash = password_hash # Hashed password
        self.preferences = preferences if preferences is not None else {} # User settings/preferences
        self.interests = interests if interests is not None else [] # User interests for recommendations
        self.roles = roles if roles is not None else ["user"] # Roles for access control (e.g., "user", "admin", "analyst")

    # Line 3: Converts user object to a dictionary, useful for saving to a database or JSON.
    def to_dict(self):
        return {
            "user_id": self.user_id,
            "username": self.username,
            "password_hash": self.password_hash,
            "preferences": self.preferences,
            "interests": self.interests,
            "roles": self.roles
        }

    # Line 4: Creates a User object from a dictionary.
    @classmethod
    def from_dict(cls, data):
        return cls(
            username=data['username'],
            password_hash=data['password_hash'],
            user_id=data['user_id'],
            preferences=data.get('preferences', {}),
            interests=data.get('interests', []),
            roles=data.get('roles', ["user"])
        )

# --- 2. AuthService (JWT-based Authentication & Authorization Simulation) ---
# Line 5: Simulates JWT (JSON Web Token) generation and validation.
# In a real application, you'd use a dedicated JWT library (e.g., `PyJWT`).
class AuthService:
    # Line 6: A secret key for signing JWTs. Keep this very secure in production!
    SECRET_KEY = "your_super_secret_jwt_key_for_signing" # Replace with a strong, random key
    EXPIRATION_TIME_SECONDS = 3600 # Token expires in 1 hour

    # Line 7: Simulates creating a JWT.
    # A real JWT has three parts: Header.Payload.Signature
    # Here, we simplify by just encoding Header and Payload.
    def create_token(self, user_id, username, roles):
        header = {"alg": "HS256", "typ": "JWT"}
        payload = {
            "user_id": user_id,
            "username": username,
            "roles": roles,
            "exp": time.time() + self.EXPIRATION_TIME_SECONDS, # Expiration timestamp
            "iat": time.time() # Issued at timestamp
        }
        # Line 8: Encode header and payload to base64.
        encoded_header = base64.urlsafe_b64encode(json.dumps(header).encode()).decode().rstrip("=")
        encoded_payload = base64.urlsafe_b64encode(json.dumps(payload).encode()).decode().rstrip("=")

        # Line 9: Simulate signature (in real JWT, this is a cryptographic hash).
        # For this conceptual example, we'll just concatenate with a dummy secret.
        signature_input = f"{encoded_header}.{encoded_payload}.{self.SECRET_KEY}"
        # In a real JWT, this would be:
        # signature = hmac.new(self.SECRET_KEY.encode(), signature_input.encode(), hashlib.sha256).digest()
        # encoded_signature = base64.urlsafe_b64encode(signature).decode().rstrip("=")
        # token = f"{encoded_header}.{encoded_payload}.{encoded_signature}"
        token = f"{encoded_header}.{encoded_payload}.[SIMULATED_SIGNATURE]" # Placeholder for actual signature

        return token

    # Line 10: Simulates validating a JWT and extracting payload.
    def validate_token(self, token):
        try:
            parts = token.split('.')
            if len(parts) != 3:
                return None # Invalid token format

            encoded_payload = parts[1]
            # Line 11: Pad base64 string if necessary
```

```
        # Line 11: Pad base64 string if necessary
        missing_padding = len(encoded_payload) % 4
        if missing_padding:
            encoded_payload += '=' * (4 - missing_padding)

        payload_bytes = base64.urlsafe_b64decode(encoded_payload.encode())
        payload = json.loads(payload_bytes.decode())

        # Line 12: Check expiration time
        if payload.get("exp") and time.time() > payload["exp"]:
            print("Token expired.")
            return None

        # Line 13: In a real system, you'd verify the signature here.
        # If signature verification fails, return None.

        return payload
    except Exception as e:
        print(f"Token validation error: {e}")
        return None

# Line 14: Role-based access control check.
def has_role(self, token, required_roles):
    payload = self.validate_token(token)
    if not payload:
        return False
    user_roles = set(payload.get("roles", []))
    return any(role in user_roles for role in required_roles)

# --- 3. UserManager (Handles User Operations) ---
# Line 15: Manages user registration, login, and profile updates.
class UserManager:
    def __init__(self):
        # Line 16: In a real application, this would be a database connection.
        # For this example, we use a dictionary to simulate user storage.
        self.users_db = {} # Stores User objects keyed by username
        self.auth_service = AuthService()

    # Line 17: Hashes a password. Use a strong, dedicated hashing library like `bcrypt` in production.
    def _hash_password(self, password):
        # Conceptual hashing using SHA256. NOT SECURE FOR PRODUCTION.
        return hashlib.sha256(password.encode()).hexdigest()

    # Line 18: Registers a new user.
    def register_user(self, username, password, preferences=None, interests=None, roles=None):
        if username in self.users_db:
            return {"success": False, "message": "Username already exists."}

        password_hash = self._hash_password(password)
        new_user = User(username, password_hash, preferences=preferences, interests=interests, roles=roles)
        self.users_db[username] = new_user
        print(f"User '{username}' registered with ID: {new_user.user_id}")
        return {"success": True, "message": "User registered successfully.", "user_id": new_user.user_id}

    # Line 19: Authenticates a user and generates a JWT.
    def login_user(self, username, password):
        user = self.users_db.get(username)
        if not user:
            return {"success": False, "message": "Invalid username or password."}

        if user.password_hash == self._hash_password(password):
            token = self.auth_service.create_token(user.user_id, user.username, user.roles)
            return {"success": True, "message": "Login successful.", "token": token}
        else:
            return {"success": False, "message": "Invalid username or password."}

    # Line 20: Retrieves user profile information.
    def get_user_profile(self, token):
        payload = self.auth_service.validate_token(token)
        if not payload:
            return {"success": False, "message": "Invalid or expired token."}

        username = payload.get("username")
        user = self.users_db.get(username)
        if user:
            # Return a copy to prevent direct modification of stored object
            profile = user.to_dict()
            profile.pop('password_hash', None) # Do not expose password hash
            return {"success": True, "profile": profile}
        return {"success": False, "message": "User not found."}

    # Line 21: Updates user profile information (preferences, interests).
    def update_user_profile(self, token, new_preferences=None, new_interests=None):
        payload = self.auth_service.validate_token(token)
        if not payload:
            return {"success": False, "message": "Invalid or expired token."}
```

```
        username = payload.get("username")
        user = self.users_db.get(username)
        if user:
            if new_preferences is not None:
                user.preferences.update(new_preferences)
            if new_interests is not None:
                user.interests = list(set(new_interests)) # Ensure unique interests
            print(f"User '{username}' profile updated.")
            return {"success": True, "message": "Profile updated successfully."}
        return {"success": False, "message": "User not found."}

    # Line 22: Updates user roles (Admin function).
    def update_user_roles(self, admin_token, target_username, new_roles):
        if not self.auth_service.has_role(admin_token, ["admin"]):
            return {"success": False, "message": "Unauthorized: Admin access required."}

        target_user = self.users_db.get(target_username)
        if not target_user:
            return {"success": False, "message": "Target user not found."}

        target_user.roles = list(set(new_roles)) # Ensure unique roles
        print(f"Roles for user '{target_username}' updated to: {target_user.roles}")
        return {"success": True, "message": f"Roles for {target_username} updated."}

# --- Example Usage ---
# Line 23: Initialize the User Manager.
user_manager = UserManager()

print("--- User Registration ---")
# Line 24: Register a regular user.
reg_result_user = user_manager.register_user("alice", "password123",
                                              preferences={"theme": "dark"},
                                              interests=["electronics", "gaming"])
print(reg_result_user)

# Line 25: Register an admin user.
reg_result_admin = user_manager.register_user("admin_user", "adminpass",
                                               roles=["admin", "analyst"])
print(reg_result_admin)

# Line 26: Attempt to register an existing user.
reg_result_fail = user_manager.register_user("alice", "anotherpass")
print(reg_result_fail)

print("\n--- User Login ---")
# Line 27: Log in Alice.
login_result_alice = user_manager.login_user("alice", "password123")
print(login_result_alice)
alice_token = login_result_alice.get("token")

# Line 28: Log in Admin.
login_result_admin = user_manager.login_user("admin_user", "adminpass")
print(login_result_admin)
admin_token = login_result_admin.get("token")

# Line 29: Attempt failed login.
login_result_fail = user_manager.login_user("alice", "wrongpassword")
print(login_result_fail)

print("\n--- Profile Management ---")
# Line 30: Get Alice's profile.
profile_alice = user_manager.get_user_profile(alice_token)
print(f"Alice's Profile: {profile_alice}")

# Line 31: Update Alice's preferences and interests.
update_result_alice = user_manager.update_user_profile(alice_token,
                                                        new_preferences={"notifications": True, "language": "en"},
                                                        new_interests=["electronics", "audio", "smart devices"])
print(update_result_alice)
profile_alice_updated = user_manager.get_user_profile(alice_token)
print(f"Alice's Updated Profile: {profile_alice_updated}")

# Line 32: Attempt to update profile with invalid token.
update_result_fail = user_manager.update_user_profile("invalid_token", new_preferences={"theme": "light"})
print(update_result_fail)

print("\n--- Role-Based Access Control ---")
# Line 33: Check if Alice has 'admin' role (should be False).
has_admin_alice = user_manager.auth_service.has_role(alice_token, ["admin"])
print(f"Does Alice have 'admin' role? {has_admin_alice}")

# Line 34: Check if Admin has 'admin' role (should be True).
has_admin_admin = user_manager.auth_service.has_role(admin_token, ["admin"])
print(f"Does Admin have 'admin' role? {has_admin_admin}")
```

```
# Line 35: Admin updates Alice's roles.
update_roles_result = user_manager.update_user_roles(admin_token, "alice", ["user", "premium"])
print(update_roles_result)
profile_alice_roles_updated = user_manager.get_user_profile(alice_token)
print(f"Alice's Profile after role update: {profile_alice_roles_updated}")

# Line 36: Alice tries to update roles (should fail).
update_roles_fail = user_manager.update_user_roles(alice_token, "admin_user", ["user", "moderator"])
print(update_roles_fail)

# Line 37: Simulate token expiration (for demonstration, in real app, tokens expire automatically)
print("\n--- Token Expiration Simulation ---")
# Create a short-lived token for demonstration
auth_service_short_lived = AuthService()
auth_service_short_lived.EXPIRATION_TIME_SECONDS = 1 # 1 second expiration
short_lived_token = auth_service_short_lived.create_token("temp_user_id", "temp_user", ["user"])
print(f"Short-lived token created: {short_lived_token}")
time.sleep(2) # Wait for token to expire
validation_after_expire = auth_service_short_lived.validate_token(short_lived_token)
print(f"Validation after expiration: {validation_after_expire}")
```

```
--- User Registration ---
User 'alice' registered with ID: f9f11217-dac6-4069-ac99-446d464bef73
{'success': True, 'message': 'User registered successfully.', 'user_id': 'f9f11217-dac6-4069-ac99-446d464bef73'}
User 'admin_user' registered with ID: 00b62513-97d2-45a1-a31b-cbcc2f8b400a
{'success': True, 'message': 'User registered successfully.', 'user_id': '00b62513-97d2-45a1-a31b-cbcc2f8b400a'}
{'success': False, 'message': 'Username already exists.'}

--- User Login ---
{'success': True, 'message': 'Login successful.', 'token': 'eyJhbGciOiAiSFMyNTYiLCAidHlwIjogIkpXVCJ9.eyJ1c2VyX2lkIjogImY5ZjExMjE3LWRhYzYt
{'success': True, 'message': 'Login successful.', 'token': 'eyJhbGciOiAiSFMyNTYiLCAidHlwIjogIkpXVCJ9.eyJ1c2VyX2lkIjogIjAwYjYyNTEzLTk3ZDIt
{'success': False, 'message': 'Invalid username or password.'}

--- Profile Management ---
Alice's Profile: {'success': True, 'profile': {'user_id': 'f9f11217-dac6-4069-ac99-446d464bef73', 'username': 'alice', 'preferences': {'t
User 'alice' profile updated.
{'success': True, 'message': 'Profile updated successfully.'}
Alice's Updated Profile: {'success': True, 'profile': {'user_id': 'f9f11217-dac6-4069-ac99-446d464bef73', 'username': 'alice', 'preferenc
{'success': False, 'message': 'Invalid or expired token.'}

--- Role-Based Access Control ---
Does Alice have 'admin' role? False
Does Admin have 'admin' role? True
Roles for user 'alice' updated to: ['premium', 'user']
{'success': True, 'message': 'Roles for alice updated.'}
Alice's Profile after role update: {'success': True, 'profile': {'user_id': 'f9f11217-dac6-4069-ac99-446d464bef73', 'username': 'alice',
{'success': False, 'message': 'Unauthorized: Admin access required.'}

--- Token Expiration Simulation ---
Short-lived token created: eyJhbGciOiAiSFMyNTYiLCAidHlwIjogIkpXVCJ9.eyJ1c2VyX2lkIjogInRlbXBfdXNlcl9pZCIsICJ1c2VybmFtZSI6ICJ0ZW1wX3VzZXIiL
Token expired.
Validation after expiration: None
```