Fraud Detection & Prediction

Column Name	Description
Transaction ID	A unique identifier for each transaction
Customer ID	A unique identifier for each customer
Transaction Amount	The amount of the transaction in the local currency
Transaction Date	The date when the transaction took place
Payment Method	The method used for payment (e.g., credit card, PayPal, etc.)
Product Category	The category of the purchased product
Quantity	The quantity of the purchased product
Customer Age	The age of the customer at the time of the transaction
Customer Location	The location (city, state, country) of the customer
Device Used	The device used for the transaction (e.g., smartphone, tablet)
IP Address	The IP address from which the transaction was made
Shipping Address	The address to which the purchased product was shipped
Billing Address	The billing address associated with the payment method
Is Fraudulent	A binary indicator (0 or 1) indicating whether the transaction is fraudulent
Account Age Days	The number of days since the customer's account was created
Transaction Hour	The hour of the day when the transaction took place (24-hour format)

In [1]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
import time
import warnings
warnings.filterwarnings('ignore')
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler, OneHotEncoder
```

```python
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import BernoulliNB
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import GradientBoostingClassifier, RandomForestClassifier
from xgboost import XGBClassifier
from sklearn.model_selection import GridSearchCV
import optuna
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
```

In [2]:

```python
train_df = pd.read_csv('Fraudulent_E-Commerce_Transaction_Data.csv')

train_df.shape
```

Out[2]: (1472952, 16)

In [3]:

```python
test_df = pd.read_csv('Fraudulent_E-Commerce_Transaction_Data_2.csv')

test_df.shape
```

Out[3]: (23634, 16)

In [4]:

```python
train_df.head()
```

Out[4]:

	Transaction ID	Customer ID	Transaction Amount	Transaction Date	Payment Method	Product Category	Quantity	Customer Age	Customer Location	Device Used	IP Address	
0	15d2e414-8735-46fc-9e02-80b472b2580f	d1b87f62-51b2-493b-ad6a-77e0fe13e785	58.09	2024-02-20 05:58:41	bank transfer	electronics	1	17	Amandaborough	tablet	212.195.49.198	
1	0bfee1a0-6d5e-40da-a446-d04e73b1b177	37de64d5-e901-4a56-9ea0-af0c24c069cf	389.96	2024-02-25 08:09:45	debit card	electronics	2	40	East Timothy	desktop	208.106.249.121	
2	e588eef4-b754-468e-	1bac88d6-4b22-409a-	134.19	2024-03-18 03:42:55	PayPal	home & garden	2	22	Davismouth	tablet	76.63.88.212	1

	Transaction ID	Customer ID	Transaction Amount	Transaction Date	Payment Method	Product Category	Quantity	Customer Age	Customer Location	Device Used	IP Address
	9d90-d0e0abfc1af0	a06b-425119c57225									7
3	4de46e52-60c3-49d9-be39-636681009789	2357c76e-9253-4ceb-b44e-ef4b71cb7d4d	226.17	2024-03-16 20:41:31	bank transfer	clothing	5	31	Lynnberg	desktop	207.208.171.73
4	074a76de-fe2d-443e-a00c-f044cdb68e21	45071bc5-9588-43ea-8093-023caec8ea1c	121.53	2024-01-15 05:08:17	bank transfer	clothing	2	51	South Nicole	tablet	190.172.14.169

In [5]:

```python
train_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1472952 entries, 0 to 1472951
Data columns (total 16 columns):
 #   Column              Non-Null Count    Dtype  
---  ------              --------------    -----  
 0   Transaction ID      1472952 non-null  object 
 1   Customer ID         1472952 non-null  object 
 2   Transaction Amount  1472952 non-null  float64
 3   Transaction Date    1472952 non-null  object 
 4   Payment Method      1472952 non-null  object 
 5   Product Category    1472952 non-null  object 
 6   Quantity            1472952 non-null  int64  
 7   Customer Age        1472952 non-null  int64  
 8   Customer Location   1472952 non-null  object 
 9   Device Used         1472952 non-null  object 
 10  IP Address          1472952 non-null  object 
 11  Shipping Address    1472952 non-null  object 
 12  Billing Address     1472952 non-null  object 
 13  Is Fraudulent       1472952 non-null  int64  
 14  Account Age Days    1472952 non-null  int64  
 15  Transaction Hour    1472952 non-null  int64  
dtypes: float64(1), int64(5), object(10)
memory usage: 179.8+ MB
```

Check for missing values & data consistency

In [6]:

```
# Check for missing values
missing_values = train_df.isna().sum()
missing_values
```

Out[6]:

```
Transaction ID        0
Customer ID           0
Transaction Amount    0
Transaction Date      0
Payment Method        0
Product Category      0
Quantity              0
Customer Age          0
Customer Location     0
Device Used           0
IP Address            0
Shipping Address      0
Billing Address       0
Is Fraudulent         0
Account Age Days      0
Transaction Hour      0
dtype: int64
```

In [7]:

```
# Check for non-negative transaction_amount
non_negative_transaction_amount = (train_df['Transaction Amount'] >= 0).all()
non_negative_transaction_amount # True indicates that all transactions are non-negative
```

Out[7]:

```
True
```

In [8]:

```
train_df["Transaction Date"] = pd.to_datetime(train_df["Transaction Date"])
test_df["Transaction Date"] = pd.to_datetime(test_df["Transaction Date"])

## Extract Day, Day of Week, and Month from the Transaction Date
train_df['Transaction Day'] = train_df["Transaction Date"].dt.day
train_df["Transaction DOW"] = train_df["Transaction Date"].dt.day_of_week
train_df["Transaction Month"] = train_df["Transaction Date"].dt.month

test_df['Transaction Day'] = test_df["Transaction Date"].dt.day
test_df["Transaction DOW"] = test_df["Transaction Date"].dt.day_of_week
test_df["Transaction Month"] = test_df["Transaction Date"].dt.month

train_df.head()
```

Out[8]:

	Transaction ID	Customer ID	Transaction Amount	Transaction Date	Payment Method	Product Category	Quantity	Customer Age	Customer Location	Device Used	IP Address
0	15d2e414-8735-46fc-9e02-80b472b2580f	d1b87f62-51b2-493b-ad6a-77e0fe13e785	58.09	2024-02-20 05:58:41	bank transfer	electronics	1	17	Amandaborough	tablet	212.195.49.198
1	0bfee1a0-6d5e-40da-a446-d04e73b1b177	37de64d5-e901-4a56-9ea0-af0c24c069cf	389.96	2024-02-25 08:09:45	debit card	electronics	2	40	East Timothy	desktop	208.106.249.121
2	e588eef4-b754-468e-9d90-d0e0abfc1af0	1bac88d6-4b22-409a-a06b-425119c57225	134.19	2024-03-18 03:42:55	PayPal	home & garden	2	22	Davismouth	tablet	76.63.88.212
3	4de46e52-60c3-49d9-be39-636681009789	2357c76e-9253-4ceb-b44e-ef4b71cb7d4d	226.17	2024-03-16 20:41:31	bank transfer	clothing	5	31	Lynnberg	desktop	207.208.171.73
4	074a76de-fe2d-443e-a00c-f044cdb68e21	45071bc5-9588-43ea-8093-023caec8ea1c	121.53	2024-01-15 05:08:17	bank transfer	clothing	2	51	South Nicole	tablet	190.172.14.169

In [9]:

```
## If the Shipping Address is the same as the Billing Address, the value is set to 1, otherwise, it is set to 0.
train_df["Is Address Match"] = (train_df["Shipping Address"] == train_df["Billing Address"]).astype(int)
test_df["Is Address Match"] = (test_df["Shipping Address"] == test_df["Billing Address"]).astype(int)

## Remove irrelevant features and downcast the datatype to reduce dataset size
train_df.drop(columns=["Transaction ID", "Customer ID", "Customer Location",
                      "IP Address", "Transaction Date","Shipping Address","Billing Address"], inplace=True)
test_df.drop(columns=["Transaction ID", "Customer ID", "Customer Location",
                     "IP Address", "Transaction Date","Shipping Address","Billing Address"], inplace=True)
```

In [10]:

```
# Calculate the mean of valid ages (ages greater than or equal to 9)
valid_age_mean = train_df[train_df['Customer Age'] >= 9]['Customer Age'].mean()
valid_age_mean = test_df[test_df['Customer Age'] >= 9]['Customer Age'].mean()
```

```
# Replace values less than -9 with their absolute values
train_df.loc[train_df['Customer Age'] < -9, 'Customer Age'] = train_df.loc[train_df['Customer Age'] < -9, 'Customer Age']
test_df.loc[test_df['Customer Age'] < -9, 'Customer Age'] = test_df.loc[test_df['Customer Age'] < -9, 'Customer Age'].abs

# Replace values between -9 and 8 with the mean of valid ages
train_df.loc[(train_df['Customer Age'] >= -9) & (train_df['Customer Age'] <= 8), 'Customer Age'] = valid_age_mean
test_df.loc[(test_df['Customer Age'] >= -9) & (test_df['Customer Age'] <= 8), 'Customer Age'] = valid_age_mean
```

In [11]:

```
# Converts integer and float columns to numeric data types, downcasting them to reduce memory usage while preserving the
int_col = train_df.select_dtypes(include="int").columns
int_col = test_df.select_dtypes(include="int").columns
float_col = train_df.select_dtypes(include="float").columns
float_col = test_df.select_dtypes(include="float").columns

train_df[int_col] = train_df[int_col].apply(pd.to_numeric, downcast='integer')
test_df[int_col] = test_df[int_col].apply(pd.to_numeric, downcast='integer')
train_df[float_col] = train_df[float_col].apply(pd.to_numeric, downcast='float')
test_df[float_col] = test_df[float_col].apply(pd.to_numeric, downcast='float')
```

In [12]:

```
train_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1472952 entries, 0 to 1472951
Data columns (total 13 columns):
 #   Column              Non-Null Count    Dtype  
---  ------              --------------    -----  
 0   Transaction Amount  1472952 non-null  float32
 1   Payment Method      1472952 non-null  object 
 2   Product Category    1472952 non-null  object 
 3   Quantity            1472952 non-null  int8   
 4   Customer Age        1472952 non-null  float32
 5   Device Used         1472952 non-null  object 
 6   Is Fraudulent       1472952 non-null  int8   
 7   Account Age Days    1472952 non-null  int16  
 8   Transaction Hour    1472952 non-null  int8   
 9   Transaction Day     1472952 non-null  int8   
 10  Transaction DOW     1472952 non-null  int8   
 11  Transaction Month   1472952 non-null  int8   
 12  Is Address Match    1472952 non-null  int8   
dtypes: float32(2), int16(1), int8(7), object(3)
memory usage: 57.6+ MB
```

The dataset has been cleaned and compressed, reducing its sie from 179.8 MB to 70.2 MB.

Exploratory Data Analysis

In [13]:

```
train_df.describe()
```

Out[13]:

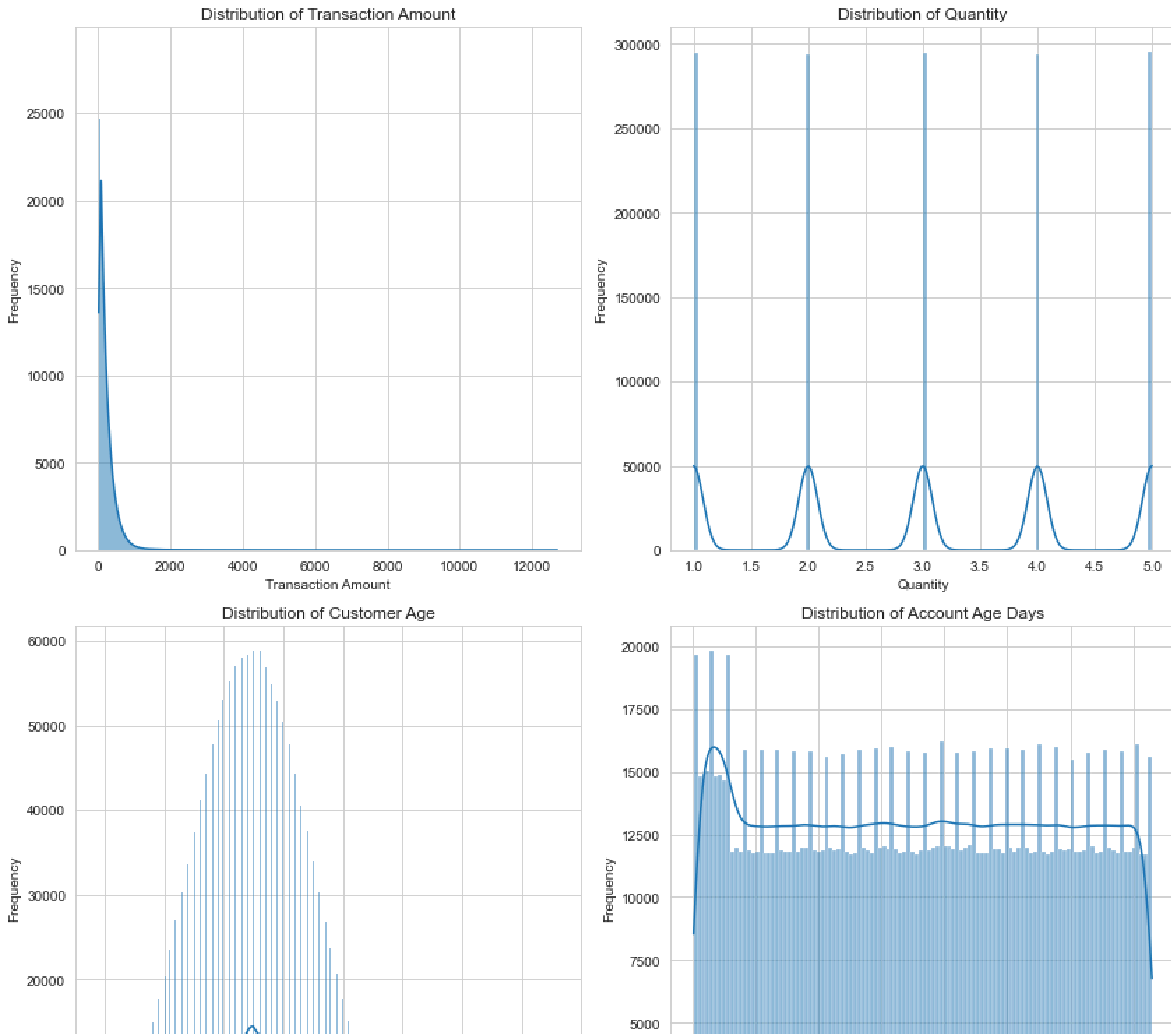
	Transaction Amount	Quantity	Customer Age	Is Fraudulent	Account Age Days	Transaction Hour	Transaction Day	Transaction DOW	Transaction Month	Is A
count	1.472952e+06	1.472952e+06	1.472952e+06	1.472952e+06	1.472952e+06	1.472952e+06	1.472952e+06	1.472952e+06	1.472952e+06	1.47295
mean	2.267885e+02	3.000230e+00	3.465031e+01	5.012926e-02	1.796464e+02	1.128696e+01	1.533193e+01	2.946404e+00	2.050155e+00	8.9984
std	2.702478e+02	1.414736e+00	9.798222e+00	2.182117e-01	1.068642e+02	6.975995e+00	8.938388e+00	2.009459e+00	8.727387e-01	3.0020
min	1.000000e+01	1.000000e+00	9.000000e+00	0.000000e+00	1.000000e+00	0.000000e+00	1.000000e+00	0.000000e+00	1.000000e+00	0.00000
25%	6.861000e+01	2.000000e+00	2.800000e+01	0.000000e+00	8.600000e+01	5.000000e+00	7.000000e+00	1.000000e+00	1.000000e+00	1.00000
50%	1.517600e+02	3.000000e+00	3.500000e+01	0.000000e+00	1.790000e+02	1.100000e+01	1.500000e+01	3.000000e+00	2.000000e+00	1.00000
75%	2.960500e+02	4.000000e+00	4.100000e+01	0.000000e+00	2.720000e+02	1.700000e+01	2.300000e+01	5.000000e+00	3.000000e+00	1.00000
max	1.270175e+04	5.000000e+00	8.600000e+01	1.000000e+00	3.650000e+02	2.300000e+01	3.100000e+01	6.000000e+00	4.000000e+00	1.00000

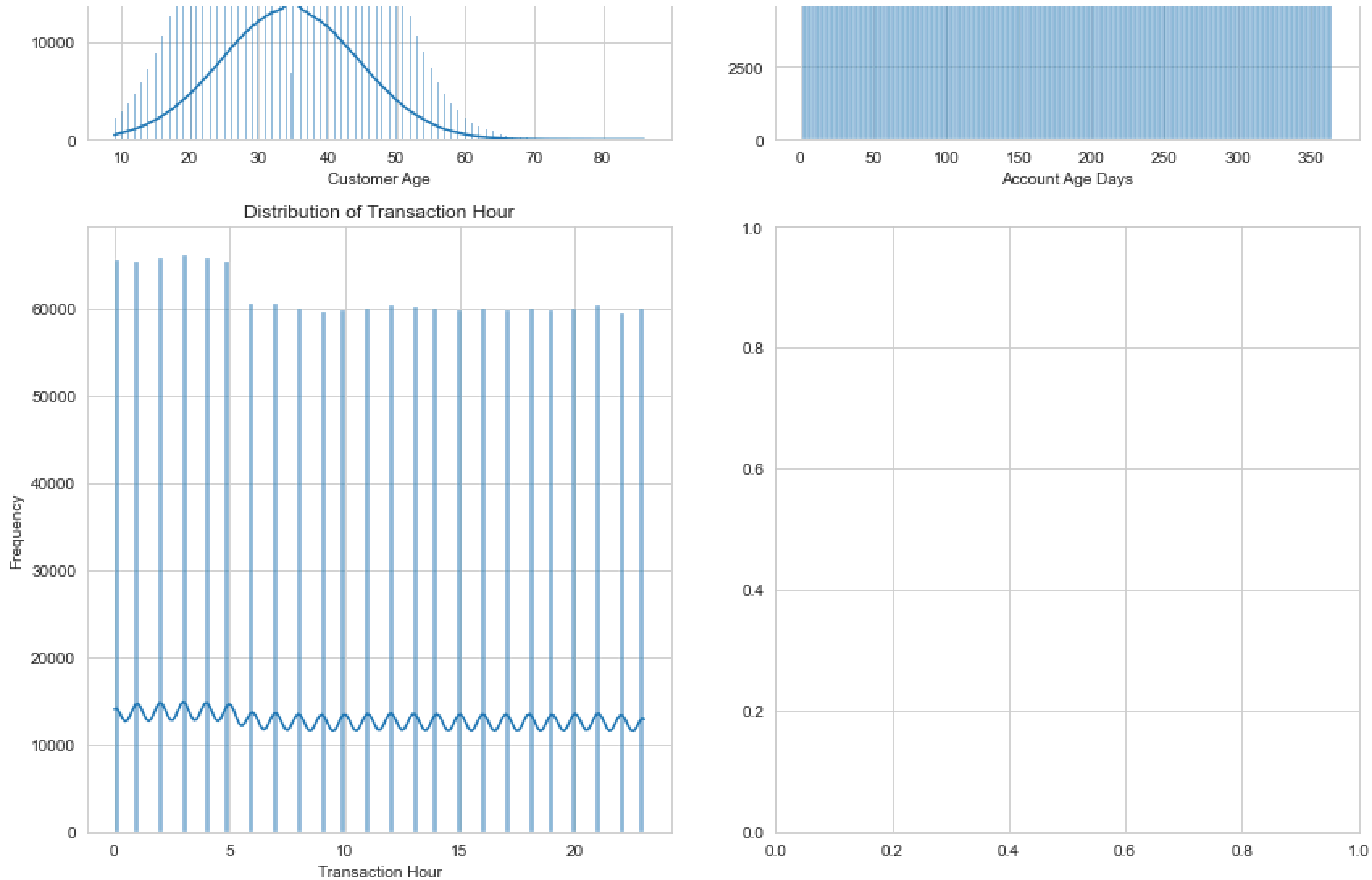
- Count: Indicates the number of non-null values in each column. All columns have the same count, which suggests there are no missing values.
- std: The dispersion or spread of the data around the mean. The std of Transaction Amount is approximately 270.25 which indcating significant variability.
- min: The minimum value observed in each column. The minimum transaction amount is 10.
- max: The maximum transaction amount is 12701.75

In [14]:

```
import matplotlib.pyplot as plt
import seaborn as sns

# Set the style of seaborn
sns.set_style("whitegrid")

# Define numerical columns
```

```python
numerical_cols = ['Transaction Amount', 'Quantity', 'Customer Age', 'Account Age Days', 'Transaction Hour']

# Calculate the number of rows needed for subplots
num_rows = (len(numerical_cols) + 1) // 2

# Create a 1 by 2 grid layout for the subplots
fig, axes = plt.subplots(num_rows, 2, figsize=(12, 6*num_rows))

# Iterate over each numerical column and create a histogram
for i, col in enumerate(numerical_cols):
    row = i // 2  # Calculate the row index
    c = i % 2   # Calculate the column index
    sns.histplot(train_df[col], kde=True, ax=axes[row, c])
    axes[row, c].set_title(f'Distribution of {col}')
    axes[row, c].set_xlabel(col)
    axes[row, c].set_ylabel('Frequency')

# Adjust layout to prevent overlap
plt.tight_layout()
plt.show()
```

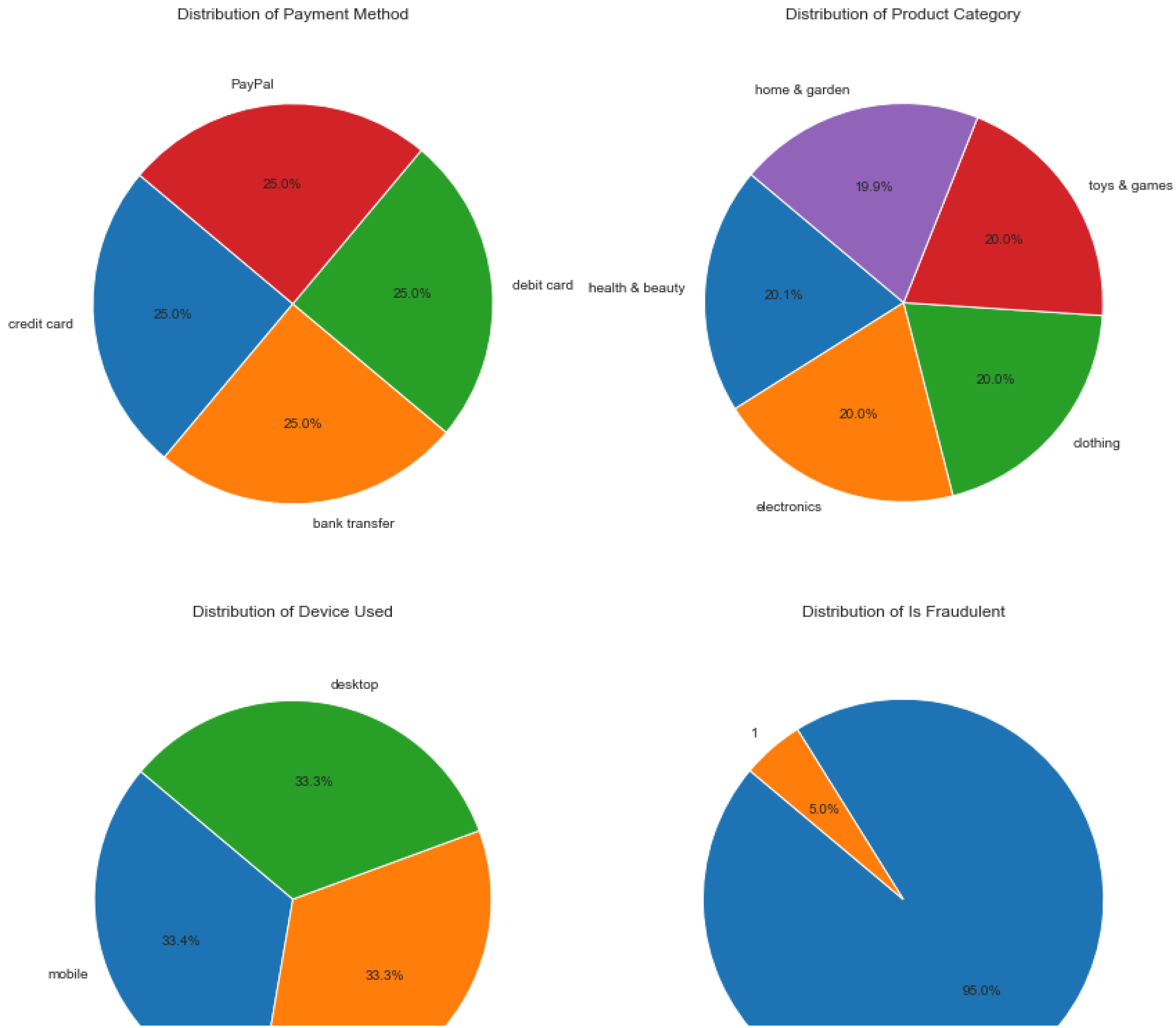

Distribution of Transaction Amount
Frequency
Transaction Amount
0
5000
10000
15000
20000
25000
0
2000
4000
6000
8000
10000
12000
Distribution of Quantity
Frequency
Quantity
0
50000
100000
150000
200000
250000
300000
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0
Distribution of Customer Age
Frequency
20000
30000
40000
50000
60000
Distribution of Account Age Days
Frequency
5000
7500
10000
12500
15000
17500
20000



In [15]:

```
# Explore categorical variables
categorical_cols = ['Payment Method', 'Product Category', 'Device Used', 'Is Fraudulent']

# Calculate the number of rows needed for subplots
num_rows = (len(categorical_cols) + 1) // 2

# Create a 1 by 2 grid layout for the subplots
fig, axes = plt.subplots(num_rows, 2, figsize=(12, 6*num_rows))
```

```python
# Iterate over each categorical column and create a pie chart
for i, col in enumerate(categorical_cols):
    row = i // 2  # Calculate the row index
    c = i % 2   # Calculate the column index
    counts = train_df[col].value_counts()
    axes[row, c].pie(counts, labels=counts.index, autopct='%1.1f%%', startangle=140)
    axes[row, c].set_title(f'Distribution of {col}')
    axes[row, c].axis('equal')  # Equal aspect ratio ensures that pie is drawn as a circle

# Adjust layout to prevent overlap
plt.tight_layout()
plt.show()
```


Distribution of Payment Method
PayPal
25.0%
debit card
25.0%
credit card
25.0%
25.0%
bank transfer
Distribution of Product Category
home & garden
19.9%
toys & games
20.0%
health & beauty
20.1%
20.0%
20.0%
clothing
electronics
Distribution of Device Used
desktop
33.3%
33.4%
mobile
33.3%
Distribution of Is Fraudulent
1
5.0%
95.0%

In [16]:

```
# Calculate the correlation matrix
correlation_matrix = train_df.corr()

# Visualize the correlation matrix using a heatmap
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f", vmin=-1, vmax=1)
plt.title('Correlation Matrix')
plt.show()
```

The correlation coefficient of 0.27 suggests a moderate postive relationship between 'Is fraudulent' and 'Transaction Amount'. This means that as the transaction amount increases, there is a tendency for the likelihood of the transaction being fraudulent to also increase. However, the strength of this relationship is not very strong.

Why did I choose Logistic regression?

- Naive Bayes classifiers are useful when there are many variables, when the input variables have many categorical values, and when used for text classification.

- Decision trees are useful when input variables interact with the output variable in an 'if-then' structure. They are also useful when input variables have 'AND' relationships with each other, when input variabls overlap or have high correlations.
- Support Vector Machines (SVM) are useful when there are many input variables or when input variables interact with each other or with the output variable in a complex (non-linear) manner.
- **Logistic regression is well-suited for binary classification tasks, where the target variable is binary data. It models the probability of one of the classes.**
- Random Forest classifier is well-suited for binary classifier and effective at handling complex datasets and identifying patterns and anomalies in

Logistic regression model (full model)

In [17]:

```
import statsmodels.api as sm

# Separate features and target variable
X = train_df.drop(columns=['Is Fraudulent'])
y = train_df['Is Fraudulent']

# Perform one-hot encoding for categorical variables
X = pd.get_dummies(X, drop_first=True)

# Add constant term to the features (intercept)
X = sm.add_constant(X)

# Fit logistic regression model
model = sm.GLM(y, X, family=sm.families.Binomial())
result = model.fit()

# Print summary of the model
print(result.summary())
```

```
                 Generalized Linear Model Regression Results
==============================================================================
Dep. Variable:          Is Fraudulent   No. Observations:              1472952
Model:                            GLM   Df Residuals:                  1472933
Model Family:                Binomial   Df Model:                           18
Link Function:                  logit   Scale:                          1.0000
Method:                          IRLS   Log-Likelihood:            -2.4313e+05
Date:                Sun, 02 Jun 2024   Deviance:                   4.8626e+05
Time:                        16:04:33   Pearson chi2:                 1.70e+06
No. Iterations:                     7
Covariance Type:            nonrobust
```

```
==============================================================================================
                                    coef    std err          z      P>|z|      [0.025      0.975]
----------------------------------------------------------------------------------------------
const                              -1.9679      0.029    -68.074      0.000      -2.025      -1.911
Transaction Amount                  0.0024   1.15e-05    206.286      0.000       0.002       0.002
Quantity                           -0.0032      0.003     -1.114      0.265      -0.009       0.002
Customer Age                       -0.0006      0.000     -1.440      0.150      -0.001       0.000
Account Age Days                   -0.0062   4.18e-05   -148.175      0.000      -0.006      -0.006
Transaction Hour                   -0.0776      0.001   -123.935      0.000      -0.079      -0.076
Transaction Day                 -1.519e-05      0.000     -0.033      0.973      -0.001       0.001
Transaction DOW                    -0.0001      0.002     -0.072      0.943      -0.004       0.004
Transaction Month                  -0.0044      0.005     -0.947      0.343      -0.014       0.005
Is Address Match                    0.0051      0.014      0.380      0.704      -0.021       0.032
Payment Method_bank transfer        0.0085      0.011      0.744      0.457      -0.014       0.031
Payment Method_credit card         -0.0099      0.011     -0.865      0.387      -0.032       0.013
Payment Method_debit card           0.0020      0.011      0.173      0.863      -0.020       0.024
Product Category_electronics       -0.0196      0.013     -1.527      0.127      -0.045       0.006
Product Category_health & beauty   -0.0060      0.013     -0.467      0.640      -0.031       0.019
Product Category_home & garden     -0.0074      0.013     -0.577      0.564      -0.033       0.018
Product Category_toys & games       0.0026      0.013      0.204      0.838      -0.022       0.028
Device Used_mobile                  0.0138      0.010      1.386      0.166      -0.006       0.033
Device Used_tablet                  0.0052      0.010      0.518      0.604      -0.014       0.025
==============================================================================================
```

From summary, we know that the variable 'Transaction Amount, 'Account Age Days' and 'Transaction Hour' are statistically significant at the significance level of 0.05.

In [18]:

```
X_test = test_df.drop(columns=['Is Fraudulent'])
y_test = test_df['Is Fraudulent']

# Perform one-hot encoding for categorical variables
X_test = pd.get_dummies(X_test, drop_first=True)

# Add constant term to the features (intercept)
X_test = sm.add_constant(X_test)

y_test_pred_proba = result.predict(X_test)

# Convert predicted probabilities to binary predictions (0 or 1) using a threshold of 0.5
y_test_pred_binary = np.where(y_test_pred_proba >= 0.5, 1, 0)

# Calculate accuracy
accuracy = np.mean(y_test == y_test_pred_binary)
```

```python
# Calculate confusion matrix
confusion_matrix = pd.crosstab(y_test, y_test_pred_binary, rownames=['Actual'], colnames=['Predicted'])

# Calculate precision, recall, and F1-score
true_positives = confusion_matrix.loc[1, 1]
false_positives = confusion_matrix.loc[0, 1]
false_negatives = confusion_matrix.loc[1, 0]

precision = true_positives / (true_positives + false_positives)
recall = true_positives / (true_positives + false_negatives)
f1_score = 2 * (precision * recall) / (precision + recall)

# Print the evaluation metrics
print("Accuracy:", accuracy)
print("Confusion Matrix:")
print(confusion_matrix)
print("Precision:", precision)
print("Recall:", recall)
print("F1-score:", f1_score)
```

```
Accuracy: 0.9531607006854531
Confusion Matrix:
Predicted      0    1
Actual
0          22386   26
1           1081  141
Precision: 0.844311377245509
Recall: 0.11538461538461539
F1-score: 0.20302375809935205
```

- Accuracy: The accuracy of the model is approximately 95.32%, indicating that around 95.32% of all transactions are correctly classified as fraudulent or non-fraudulent.
- Confusion Matrix:
    - True Negatives (TN): 22386 (non-fraudulent transactions correctly classified)
    - False Positives (FP): 26 (non-fraudulent transactions incorrectly classified as fraudulent)
    - False Negatives (FN): 1081 (fraudulent transactions incorrectly classified as non-fraudulent)
    - True Positives (TP): 141 (fraudulent transactions correctly classified)
- Precision: The precision of the model is approximately 84.43%, indicating that among all transactions predicted as fraudulent, around 84.43% are truly fraudulent.

- Recall: The recall (or sensitivity) of the model is approximately 11.54%, indicating that only around 11.54% of all fraudulent transactions are correctly identified by the model.
- F1-score: The F1-score of the model is approximately 20.30%, which is the harmonic mean of precision and recall. It provides a balance between precision and recall and is useful when there is an uneven class distribution.

Random Forest Classifier (Full model)

In [19]:

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

# Instantiate Random Forest classifier model
rf_model = RandomForestClassifier(n_estimators=100, random_state=42)

# Fit Random Forest model on the training data
rf_model.fit(X, y)
# Predict using Random Forest model
rf_y_pred = rf_model.predict(X_test)

# Evaluate performance of Random Forest model
rf_accuracy = accuracy_score(y_test, rf_y_pred)
rf_precision = precision_score(y_test, rf_y_pred)
rf_recall = recall_score(y_test, rf_y_pred)
rf_f1 = f1_score(y_test, rf_y_pred)

print("Performance of Random Forest model:")
print("Accuracy:", rf_accuracy)
print("Precision:", rf_precision)
print("Recall:", rf_recall)
print("F1-score:", rf_f1)
```

```
Performance of Random Forest model:
Accuracy: 0.9538800033849539
Precision: 0.7727272727272727
Recall: 0.1530278232405892
F1-score: 0.25546448087431695
```

In [21]:

```
# Get feature importances
feature_importances = rf_model.feature_importances_
feature_names = X.columns

# Create a DataFrame for feature importances
```

```python
importances_df = pd.DataFrame({'Feature': feature_names, 'Importance': feature_importances})

# Sort the DataFrame by importance
importances_df = importances_df.sort_values(by='Importance', ascending=False)

# Print the top 10 important features
print("Top 5 Important Features:")
print(importances_df.head(5))
```

```
Top 5 Important Features:
              Feature  Importance
1  Transaction Amount    0.276945
4    Account Age Days    0.208694
3        Customer Age    0.115055
5    Transaction Hour    0.096897
6     Transaction Day    0.078195
```

Both models shows that the 'Transaction Amount, 'Account Age Days' and 'Transaction Hour' are important features.

Based on these comparisons, if the goal is to prioritize precision (minimize false positives) while maintaining a reasonable level of recall, the logistic regression model may be preferred due to its higher precision. However, if maximizing recall (capturing as many fraudulent transactions as possible) is more important, the Random Forest model may be favored due to its higher recall and slightly higher F1-score.

In [ ]:

```python

```