# Credit Card Fraud Detection

**Import the Data Set**

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

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 archive (1).zip to archive (1).zip
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
# Import libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# Load the dataset
data = pd.read_csv('/content/creditcard.csv')  # Update the path as necessary
```

**Structure of the Dataset**

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 107046 entries, 0 to 107045
Data columns (total 31 columns):
 #   Column  Non-Null Count   Dtype
---  ------  --------------   -----
 0   Time    107046 non-null  int64
 1   V1      107046 non-null  float64
 2   V2      107046 non-null  float64
 3   V3      107046 non-null  float64
 4   V4      107046 non-null  float64
 5   V5      107046 non-null  float64
 6   V6      107046 non-null  float64
 7   V7      107046 non-null  float64
 8   V8      107046 non-null  float64
 9   V9      107046 non-null  float64
 10  V10     107046 non-null  float64
 11  V11     107046 non-null  float64
 12  V12     107046 non-null  float64
 13  V13     107046 non-null  float64
 14  V14     107046 non-null  float64
 15  V15     107046 non-null  float64
 16  V16     107046 non-null  float64
 17  V17     107046 non-null  float64
 18  V18     107046 non-null  float64
 19  V19     107046 non-null  float64
 20  V20     107046 non-null  float64
 21  V21     107046 non-null  float64
 22  V22     107046 non-null  float64
 23  V23     107046 non-null  float64
 24  V24     107046 non-null  float64
 25  V25     107046 non-null  float64
 26  V26     107046 non-null  float64
 27  V27     107046 non-null  float64
 28  V28     107045 non-null  float64
 29  Amount  107045 non-null  float64
 30  Class   107045 non-null  float64
dtypes: float64(30), int64(1)
memory usage: 25.3 MB
```

```
# Summary statistics
data_description = data.describe()
print(data_description)
```

```
                Time             V1             V2             V3  \
count  107046.000000  107046.000000  107046.000000  107046.000000   
mean    44163.193393      -0.257831      -0.026560       0.682250   
std     17718.024713       1.853150       1.647586       1.317292   
min         0.000000     -56.407510     -72.715728     -33.680984   
25%     34238.000000      -1.023275      -0.595414       0.176950   
50%     46023.000000      -0.260775       0.078063       0.757881   
75%     58413.750000       1.154981       0.738193       1.383324   
max     70256.000000       1.960497      18.902453       4.226108   

                  V4             V5             V6             V7  \
count  107046.000000  107046.000000  107046.000000  107046.000000   
mean        0.157009      -0.283548       0.098077      -0.116249   
std         1.344114       1.348799       1.299593       1.208427   
min        -5.172595     -42.147898     -26.160506     -31.764946   
25%        -0.712661      -0.906588      -0.645561      -0.605549   
50%         0.184854      -0.318181      -0.154784      -0.071994   
75%         1.024500       0.244910       0.493216       0.409485   
max        16.715537      34.801666      22.529298      36.677268   

                  V8             V9  ...            V21            V22  \
count  107046.000000  107046.000000  ...  107046.000000  107046.000000   
mean        0.058826      -0.054332  ...      -0.030490      -0.106799   
std         1.232005       1.110405  ...       0.741472       0.639652   
min       -73.216718      -9.283925  ...     -34.830382     -10.933144   
25%        -0.134727      -0.697239  ...      -0.224126      -0.533261   
50%         0.077532      -0.121239  ...      -0.056729      -0.082983   
75%         0.369074       0.542787  ...       0.120478       0.314211   
max        20.007208      10.392889  ...      27.202839      10.503090   

                 V23            V24            V25            V26  \
count  107046.000000  107046.000000  107046.000000  107046.000000   
mean       -0.037448       0.010167       0.133350       0.025764   
std         0.623527       0.595680       0.440034       0.491594   
min       -44.807735      -2.836627     -10.295397      -2.534330   
25%        -0.176768      -0.323162      -0.130802      -0.323569   
50%        -0.049385       0.066011       0.171578      -0.069111   
75%         0.080706       0.407201       0.421048       0.294199   
max        19.002942       4.016342       5.541598       3.517346   

                 V27            V28         Amount          Class  
count  107046.000000  107045.000000  107045.000000  107045.000000  
mean        0.001604       0.001640      96.201778       0.002195  
std         0.391995       0.320114     261.083669       0.046803  
min        -9.390980      -9.617915       0.000000       0.000000  
25%        -0.061142      -0.005091       7.050000       0.000000  
50%         0.010696       0.023365      25.150000       0.000000  
75%         0.084614       0.076725      87.000000       0.000000  
max        12.152401      33.847808   19656.530000       1.000000  

[8 rows x 31 columns]
```

```
# Correlation matrix
data_correlation = data.corr()
print(data_correlation)
```

```
V6      -0.000891  0.048999  0.003346 -0.008288 -0.012057  0.209054 -0.059887
V7       0.001811 -0.095439 -0.032614 -0.093095 -0.046591  0.385821 -0.245636
V8       0.003615  0.013440  0.006673  0.003476  0.020038 -0.094854  0.059624
V9       0.012819  0.120192  0.109231 -0.032188 -0.026413 -0.019445 -0.114625
V10      0.004346  0.016112 -0.022666 -0.075493 -0.020344 -0.134181 -0.264604
V11      0.036859 -0.103081 -0.004226  0.007644  0.004513 -0.010298  0.181361
V12      0.011362  0.011422  0.024001 -0.022224 -0.005584  0.021705 -0.291451
V13     -0.015258  0.040377  0.000849 -0.001750 -0.004817 -0.005640 -0.002931
V14      0.025787 -0.070869  0.020707 -0.024153  0.013479  0.014753 -0.368841
V15     -0.001174 -0.093143 -0.014780  0.008115 -0.006544 -0.034282  0.001143
V16     -0.005177  0.106066  0.020579 -0.005525 -0.021017 -0.013158 -0.249897
V17     -0.007457 -0.078437 -0.070094 -0.011900 -0.012963  0.017496 -0.404573
V18     -0.019184  0.018809  0.018182 -0.014404 -0.000888  0.048608 -0.154981
V19     -0.011815 -0.008452 -0.002260  0.003989 -0.010012 -0.057533  0.044073
V20     -0.006610 -0.030547  0.002778 -0.027855  0.108452  0.427260  0.012235
V21     -0.000216 -0.012366 -0.026530 -0.015192  0.054282  0.131681  0.088744
V22      0.000614 -0.009072 -0.089506 -0.010555 -0.034442 -0.078011 -0.018004
V23      0.004199  0.106387  0.036862 -0.016658  0.027209 -0.138558 -0.005879
V24      1.000000 -0.039003 -0.008761 -0.004799 -0.008779  0.019995 -0.009851
V25     -0.039003  1.000000 -0.112737  0.048402  0.056925 -0.065504  0.008701
V26     -0.008761 -0.112737  1.000000  0.000181  0.001912  0.002163  0.006176
V27     -0.004799  0.048402  0.000181  1.000000 -0.005604  0.012196  0.063883
V28     -0.008779  0.056925  0.001912 -0.005604  1.000000  0.010710  0.009220
Amount   0.019995 -0.065504  0.002163  0.012196  0.010710  1.000000  0.003383
Class   -0.009851  0.008701  0.006176  0.063883  0.009220  0.003383  1.000000

[31 rows x 31 columns]
```

```
# Filter the dataset to only include fraudulent transactions
fraudulent_count = data[data['Class'] == 1]
```

```
# Filter the dataset for fraudulent transactions
fraudulent_data = data[data['Class'] == 1]

#  summary statistics for fraudulent transactions
fraudulent_stats = fraudulent_data['Amount'].describe()
print("Summary Statistics for Fraudulent Transactions:")
print(fraudulent_stats)
```

```
Summary Statistics for Fraudulent Transactions:
count     235.000000
mean      115.033362
std       252.411933
min         0.000000
25%         1.000000
50%         7.580000
75%        99.990000
max      1809.680000
Name: Amount, dtype: float64
```

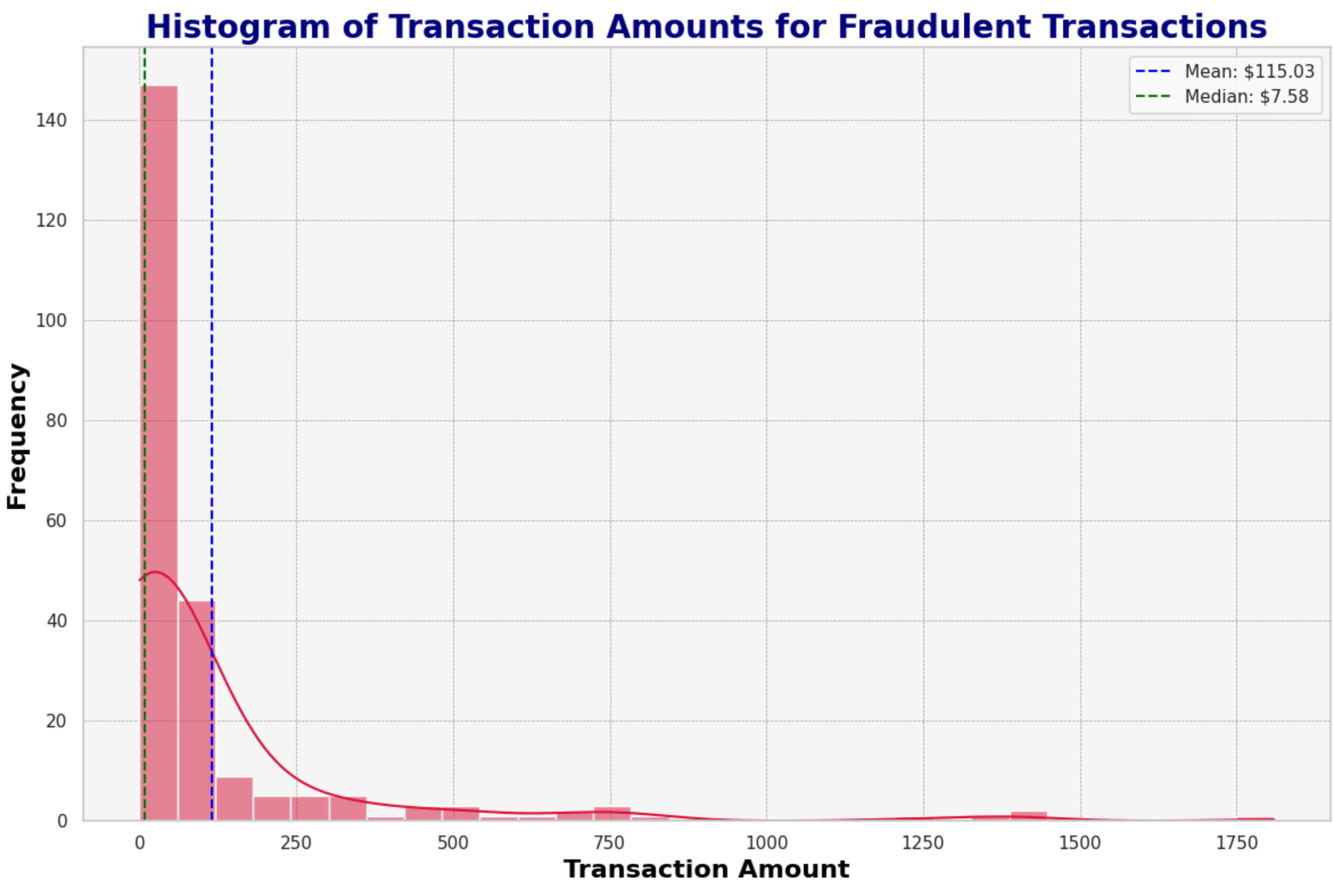
```
# Checking how many fraudulent transactions there are
fraudulent_count = data['Class'].value_counts()
print(f'Number of Legitimate Transactions: {fraudulent_count[0]}')
print(f'Number of Fraudulent Transactions: {fraudulent_count[1]}')
```

```
Number of Legitimate Transactions: 106810
Number of Fraudulent Transactions: 235
```

```
# Convert 'Time' column to 24-hour format as a string in a new column
fraudulent_data = fraudulent_data.copy()  # Create a copy
fraudulent_data['Time_24hr'] = pd.to_datetime(fraudulent_data['Time'], unit='s').dt.strftime('%H:%M:%S')

# Convert 'Time' to minutes
fraudulent_data['Time_in_minutes'] = fraudulent_data['Time'] / 60  # Retain minutes for plotting
```

```
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
sns.set(style='whitegrid')

# Creating the histogram
plt.figure(figsize=(12, 8))
sns.histplot(fraudulent_data['Amount'], bins=30, color='crimson', kde=True)
mean_amount = fraudulent_data['Amount'].mean()
median_amount = fraudulent_data['Amount'].median()
plt.axvline(mean_amount, color='blue', linestyle='--', label=f'Mean: ${mean_amount:.2f}')
plt.axvline(median_amount, color='green', linestyle='--', label=f'Median: ${median_amount:.2f}')
plt.title('Histogram of Transaction Amounts for Fraudulent Transactions', fontsize=20, fontweight='bold', color='navy')
```

```
plt.xlabel('Transaction Amount', fontsize=16, fontweight='bold', color='black')
plt.ylabel('Frequency', fontsize=16, fontweight='bold', color='black')
plt.legend()
plt.grid(color='gray', linestyle='--', linewidth=0.5, alpha=0.7)
plt.gca().set_facecolor('whitesmoke')
plt.tight_layout()
plt.show()
```



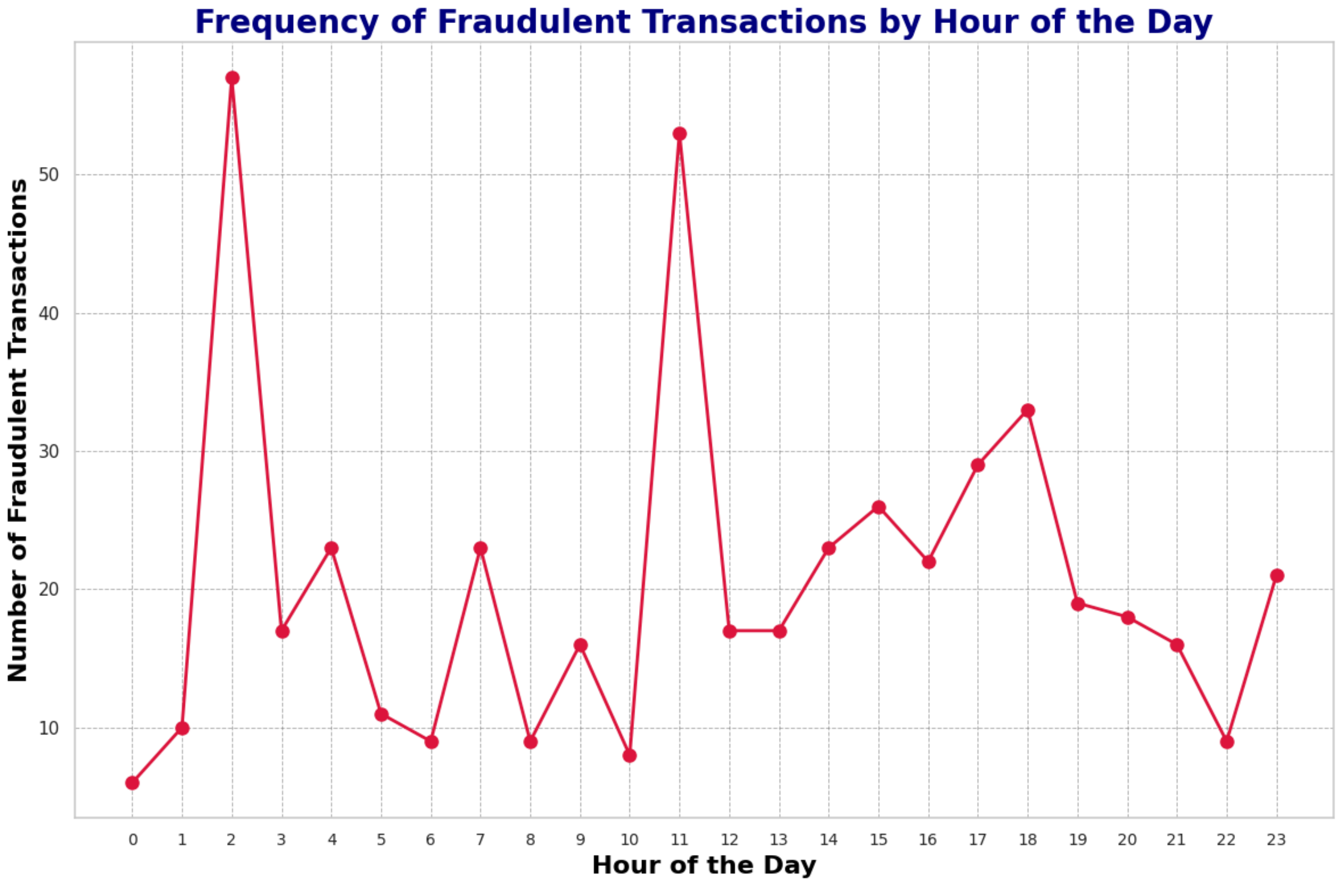

*Observation: Most fraudulent transactions are relatively small, with 75% under 105.89. This pattern suggests that fraudsters may opt for smaller amounts to avoid detection. However, there is a broad range in amounts, from $0 up to 2,125.87, showing that while larger fraud attempts are less frequent, they do occur.*

```
# Convert Time from seconds to 24-hour clock format (HH:MM:SS)
fraudulent_data['Time_24hr'] = pd.to_datetime(fraudulent_data['Time'], unit='s').dt.strftime('%H:%M:%S')


# Creating hour Column
# Converting tie to hours
data['Hour'] = (data['Time'] // 3600) % 24  # Get the hour in a 24-hour format

# Filter for fraudulent transactions
fraudulent_data = data[data['Class'] == 1]

#  Count fraudulent transactions by hour
hourly_counts = fraudulent_data['Hour'].value_counts().sort_index()

#  hour with the highest and lowest number of transactions
highest_hour = hourly_counts.idxmax()
highest_count = hourly_counts.max()
lowest_hour = hourly_counts.idxmin()
lowest_count = hourly_counts.min()

print(f"The hour with the most fraudulent transactions is {highest_hour}:00 with {highest_count} transactions.")
```

```
print(f"The hour with the least fraudulent transactions is {lowest_hour}:00 with {lowest_count} transactions.")
```

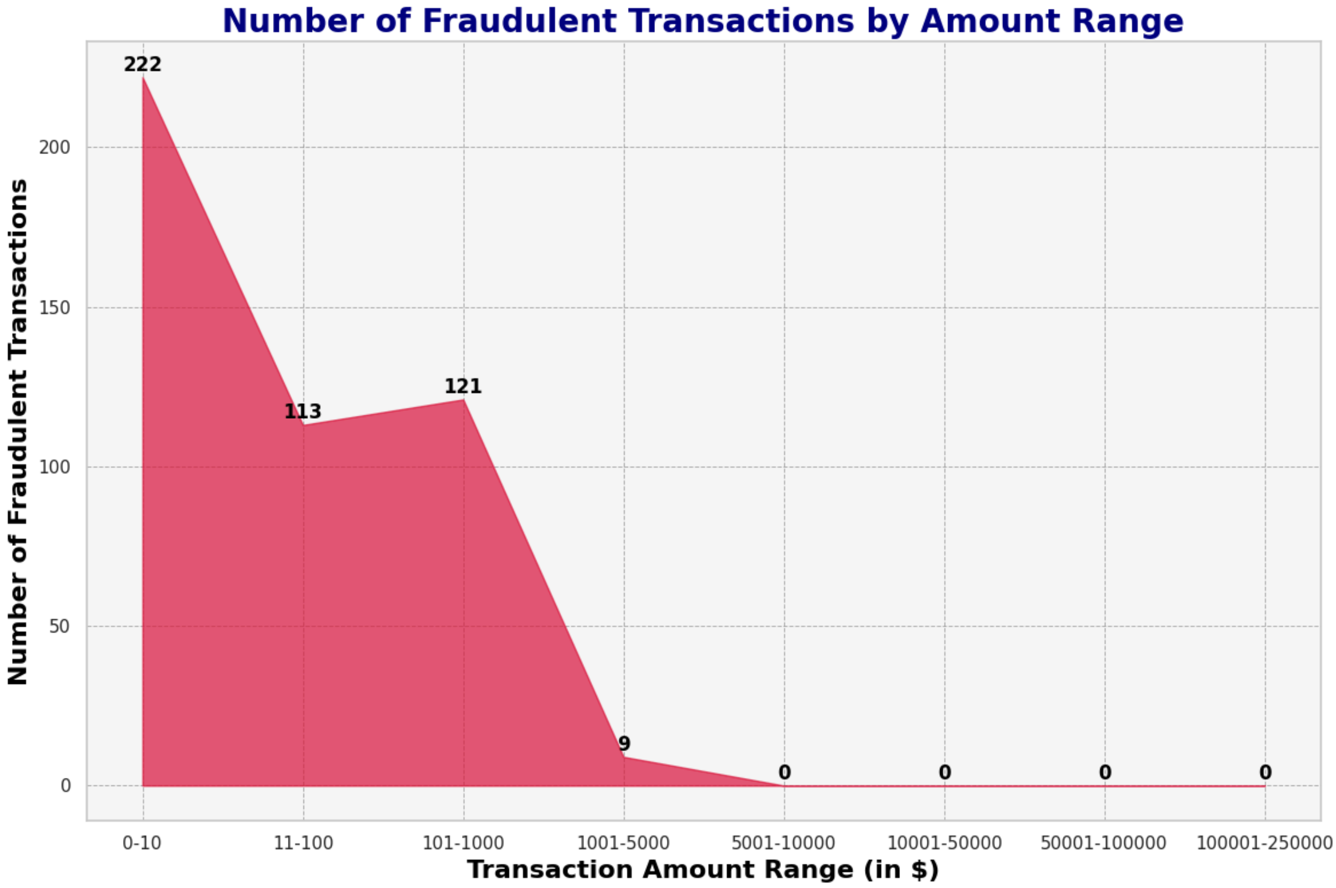
```
The hour with the most fraudulent transactions is 11:00 with 43 transactions.
The hour with the least fraudulent transactions is 0:00 with 2 transactions.
```

```
import pandas as pd
import matplotlib.pyplot as plt
hourly_fraud_counts = fraudulent_data[fraudulent_data['Class'] == 1]['Hour'].value_counts().sort_index()
plt.figure(figsize=(12, 8), facecolor='white')
plt.plot(hourly_fraud_counts.index, hourly_fraud_counts.values, marker='o', color='crimson', linewidth=2, markersize=8)
plt.title('Frequency of Fraudulent Transactions by Hour of the Day', fontsize=20, fontweight='bold', color='navy')
plt.xlabel('Hour of the Day', fontsize=16, fontweight='bold', color='black')
plt.ylabel('Number of Fraudulent Transactions', fontsize=16, fontweight='bold', color='black')
plt.grid(color='gray', linestyle='--', linewidth=0.7, alpha=0.6)
plt.xticks(hourly_fraud_counts.index, fontsize=10)
plt.tight_layout()
plt.show()
```




*Observations: Fraudulent activity peaks in the early morning hours, suggesting fraudsters may take advantage of lower oversight during this time. Midnight has the least activity, indicating a potential preference for times when fewer transactions occur.*

```
# Area Plot for fraudulent transactions by amount range
plt.figure(figsize=(12, 8), facecolor='white')

# Creating an area plot
plt.fill_between(amount_range_counts.index, amount_range_counts.values, color='crimson', alpha=0.7)
for i, value in enumerate(amount_range_counts.values):
    plt.text(i, value + 2, str(value), ha='center', fontsize=12, fontweight='bold', color='black')
plt.title('Number of Fraudulent Transactions by Amount Range', fontsize=20, fontweight='bold', color='navy')
plt.xlabel('Transaction Amount Range (in $)', fontsize=16, fontweight='bold', color='black')
plt.ylabel('Number of Fraudulent Transactions', fontsize=16, fontweight='bold', color='black')
plt.grid(color='gray', linestyle='--', linewidth=0.7, alpha=0.6)
plt.gca().set_facecolor('whitesmoke')
plt.tight_layout()
plt.show()
```




```
 Check for missing values in the dataset
print("Missing values before handling:")
print(data.isnull().sum())
drop rows with any NaN values)
data = data.dropna()
# Verify that there are no missing values left
print("Missing values after handling:")
print(data.isnull().sum())
# splitting and modeling
X = data.drop(columns=['Class'])  # Features
y = data['Class']                 # Target
# Train-test split and model training as before
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
model = RandomForestClassifier(random_state=42)
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
plot_confusion_matrix_heatmap(y_test, y_pred)
```

```
Missing values before handling:
Time      0
V1        0
V2        0
V3        0
V4        0
V5        0
V6        0
V7        0
V8        0
V9        0
V10       0
V11       0
V12       0
V13       0
V14       0
V15       0
V16       0
V17       0
V18       0
V19       0
V20       0
V21       0
V22       0
V23       0
V24       0
V25       0
V26       0
V27       0
V28       1
Amount    1
Class     1
Hour      0
dtype: int64
Missing values after handling:
Time      0
V1        0
V2        0
V3        0
V4        0
V5        0
V6        0
V7        0
V8        0
V9        0
V10       0
V11       0
V12       0
V13       0
V14       0
V15       0
V16       0
V17       0
V18       0
V19       0
V20       0
V21       0
V22       0
V23       0
V24       0
V25       0
V26       0
V27       0
V28       0
Amount    0
Class     0
Hour      0
dtype: int64
```