CREDIT CARD FRAUD DETECTION

Objective:

The primary objective of this case study is to build a machine learning model that can identify fraudulent credit card transactions. By analyzing the dataset, the model will distinguish between legitimate and fraudulent transactions. We will also explore various visualization techniques to understand the patterns of fraudulent activity, such as the distribution of transaction amounts and fraud occurrence by time

Dataset Description:

- **Time**: Time of transaction in seconds since the first transaction.
- **V1-V28**: Anonymized features derived from PCA, representing transaction patterns.
- **Amount**: Monetary value of the transaction.
- **Class**: Target variable indicating transaction type:
- 0: Legitimate transaction
- 1: Fraudulent transaction

Key Steps:

- **Load Data**: Import and explore dataset for shape, missing values, and basic stats.
- **Class Distribution**: Analyze and visualize the count of fraudulent vs. legitimate transactions.
- **Amount Statistics**: Calculate transaction amount statistics (min, max, mean, median).
- **Visualizations**: Plot transaction counts, correlations, and time distributions.
- **Transaction Hour Analysis:** Analyze and visualize fraudulent transactions by hour.
- **Hourly Stats**: Calculate transaction amounts by hour (sum, mean, etc.).
- **Amount Distribution**: Compare amounts for fraudulent vs. non-fraudulent transactions.
- **Scatter Plot**: Visualize fraudulent transactions' time vs. amount.

Packages:

- **pandas**: Data manipulation and reading files.
- **numpy**: Numerical operations on arrays.
- **seaborn**: Statistical data visualization.
- **matplotlib.pyplot**: Plotting graphs and charts.
- **sklearn.model_selection**: Splitting data into training and test sets.
- **sklearn.preprocessing**: Scaling data with StandardScaler.
- **sklearn.ensemble**: Random forest model for classification.

- **sklearn.metrics**: Evaluating models with classification reports and confusion matrix.
- **sklearn.linear_model**: Logistic regression for binary classification.

Step 1: Importing Libraries

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

Step 2: Load the dataset

```
df = pd.read_csv('creditcard.csv')
print("Dataset loaded successfully:",df)

# Check for missing values and basic dataset info
print(f"Dataset shape: {df.shape}")
print("\nStatistical description of the dataset:\n",df.describe())
print(df.dtypes)
print(f"Missing values:\n{df.isnull().sum()}")
```

Output:

Dataset loaded successfully:

```
Dataset loaded successfully:             Time         V1         V2        V3        V4        V5        V6  ...       V24       V25       V26       V27
V28  Amount  Class
0            0.0  -1.359807  -0.072781  2.536347  1.378155 -0.338321  0.462388  ...  0.066928  0.128539 -0.189115  0.133558 -0.021053  149.62      0
1            0.0   1.191857   0.266151  0.166480  0.448154  0.060018 -0.082361  ... -0.339846  0.167170  0.125895 -0.008983  0.014724    2.69      0
2            1.0  -1.358354  -1.340163  1.773209  0.379780 -0.503198  1.800499  ... -0.689281 -0.327642 -0.139097 -0.055353 -0.059752  378.66      0
3            1.0  -0.966272  -0.185226  1.792993 -0.863291 -0.010309  1.247203  ... -1.175575  0.647376 -0.221929  0.062723  0.061458  123.50      0
4            2.0  -1.158233   0.877737  1.548718  0.403034 -0.407193  0.095921  ...  0.141267 -0.206010  0.502292  0.219422  0.215153   69.99      0
...          ...        ...        ...       ...       ...       ...       ...  ...       ...       ...       ...       ...       ...     ...    ...
284802  172786.0 -11.881118  10.071785 -9.834783 -2.066656 -5.364473 -2.606837  ... -0.509348  1.436807  0.250034  0.943651  0.823731    0.77      0
284803  172787.0  -0.732789  -0.055080  2.035030 -0.738589  0.868229  1.058415  ... -1.016226 -0.606624 -0.395255  0.068472 -0.053527   24.79      0
284804  172788.0   1.919565  -0.301254 -3.249640 -0.557828  2.630515  3.031260  ...  0.640134  0.265745 -0.087371  0.004455 -0.026561   67.88      0
284805  172788.0  -0.240440   0.530483  0.702510  0.689799 -0.377961  0.623708  ...  0.123205 -0.569159  0.546668  0.108821  0.104533   10.00      0
284806  172792.0  -0.533413  -0.189733  0.703337 -0.506271 -0.012546 -0.649617  ...  0.008797 -0.473649 -0.818267 -0.002415  0.013649  217.00      0
```

Dataset shape: (284807, 31)

Statistical description of the dataset:

```
Statistical description of the dataset:
                Time            V1            V2            V3            V4  ...           V26           V27           V28         Amount          Class
count  284807.000000  2.848070e+05  2.848070e+05  2.848070e+05  2.848070e+05  ...  2.848070e+05  2.848070e+05  2.848070e+05  284807.000000  284807.000000
mean    94813.859575  1.168375e-15  3.416908e-16 -1.379537e-15  2.074095e-15  ...  1.683437e-15 -3.660091e-16 -1.227390e-16      88.349619       0.001727
std     47488.145955  1.958696e+00  1.651309e+00  1.516255e+00  1.415869e+00  ...  4.822270e-01  4.036325e-01  3.300833e-01     250.120109       0.041527
min         0.000000 -5.640751e+01 -7.271573e+01 -4.832559e+01 -5.683171e+00  ... -2.604551e+00 -2.256568e+01 -1.543008e+01       0.000000       0.000000
25%     54201.500000 -9.203734e-01 -5.985499e-01 -8.903648e-01 -8.486401e-01  ... -3.269839e-01 -7.083953e-02 -5.295979e-02       5.600000       0.000000
50%     84692.000000  1.810880e-02  6.548556e-02  1.798463e-01 -1.984653e-02  ... -5.213911e-02  1.342146e-03  1.124383e-02      22.000000       0.000000
75%    139320.500000  1.315642e+00  8.037239e-01  1.027196e+00  7.433413e-01  ...  2.409522e-01  9.104512e-02  7.827995e-02      77.165000       0.000000
max    172792.000000  2.454930e+00  2.205773e+01  9.382558e+00  1.687534e+01  ...  3.517346e+00  3.161220e+01  3.384781e+01   25691.160000       1.000000

[8 rows x 31 columns]
```

datatypes: Time float64

V1 float64
V2 float64
V3 float64
V4 float64
V5 float64
V6 float64
V7 float64
V8 float64
V9 float64
V10 float64
V11 float64
V12 float64
V13 float64
V14 float64
V15 float64
V16 float64
V17 float64
V18 float64
V19 float64
V20 float64
V21 float64
V22 float64
V23 float64
V24 float64
V25 float64
V26 float64
V27 float64
V28 float64
Amount float64
Class int64

Missing values:
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

Step 3: Analyze transaction types (fraud vs. non-fraud)

```
# Count the number of fraudulent and legitimate transactions
transaction_counts = df['Class'].value_counts()
print("Number of legitimate transactions:", transaction_counts[0])
print("Number of fraudulent transactions:", transaction_counts[1])

# Calculate the percentage of fraudulent transactions
total_transactions = len(df)
transaction_counts = df['Class'].value_counts()
fraudulent_transactions = transaction_counts[1]
```

```
non_fraudulent_transactions = transaction_counts[0]
percentage_fraudulent = (fraudulent_transactions / total_transactions) * 100
print(f"Percentage of fraudulent transactions: {percentage_fraudulent:.2f}%")
```

Output:

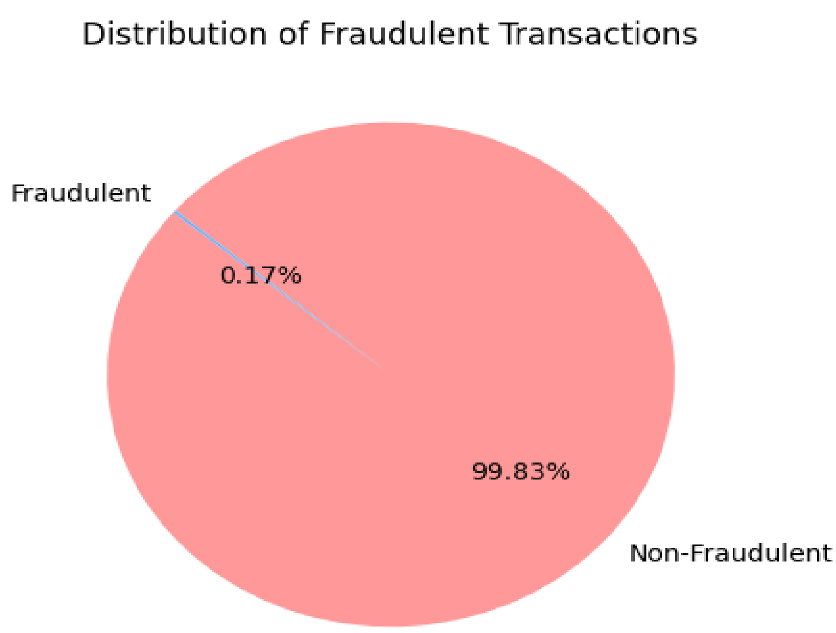
Number of legitimate transactions: 284315

Number of fraudulent transactions: 492

Percentage of fraudulent transactions: 0.17%

Step 4 : Visualize distribution of fraudulent vs non-fraudulent transactions

```
Data for the pie chart
labels = ['Non-Fraudulent', 'Fraudulent']
sizes = [non_fraudulent_transactions, fraudulent_transactions]
plt.pie(sizes, labels=labels, autopct='%2.2f%%', startangle=140,
colors=['#ff9999','#66b3ff'])
plt.title('Distribution of Fraudulent Transactions')
plt.show()
```



Step 5: Basic statistics on the 'Amount' column

```
# Calculate descriptive statistics for the 'Amount' column
min_amount = df['Amount'].min()
max_amount = df['Amount'].max()
mean_amount = df['Amount'].mean()
median_amount = df['Amount'].median()

# Print the statistics
print(f"Minimum amount: {min_amount:.2f}")
print(f"Maximum amount: {max_amount:.2f}")
print(f"Mean amount: {mean_amount:.2f}")
print(f"Median amount: {median_amount:.2f}")
```

Output

Minimum amount: 0.00

Maximum amount: 25691.16

Mean amount: 88.35

Median amount: 22.00

Step 6: Analyze the maximum transaction amount

```
# Find the maximum transaction amount
max_amount = df['Amount'].max()
max_amount_row = df[df['Amount'] == max_amount]
is_fraudulent = max_amount_row['Class'].values[0]
print(f"The maximum transaction amount is {max_amount}, and it is {'fraudulent'
if is_fraudulent else 'legitimate'}.")
```

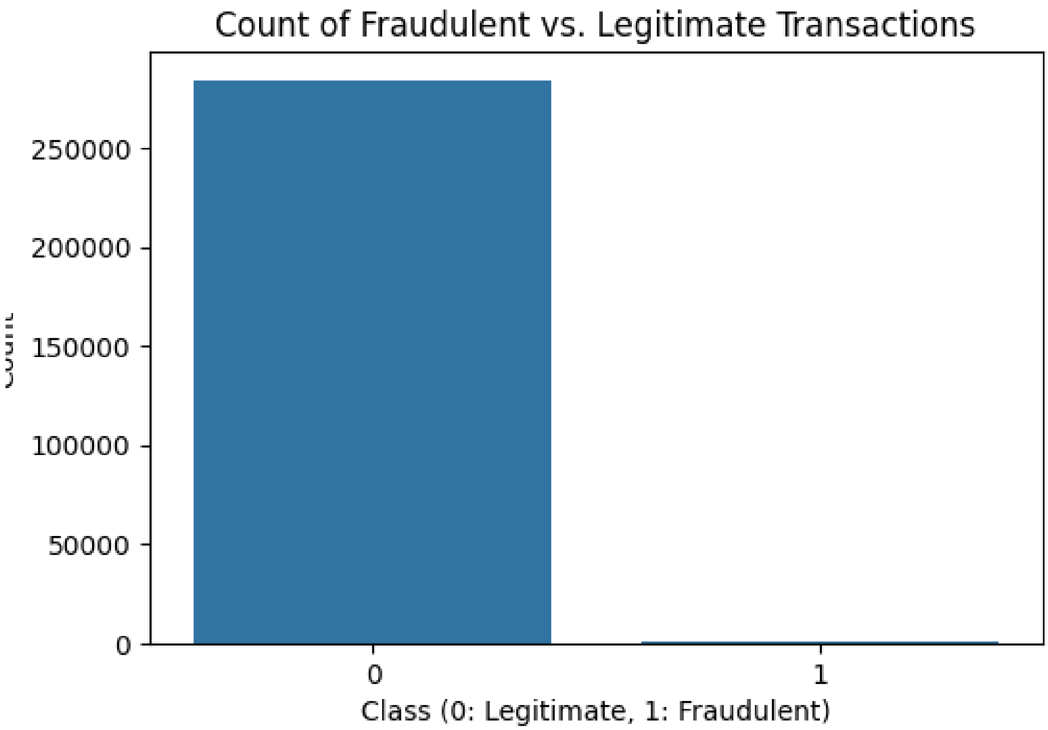
Output:

The maximum transaction amount is 25691.16, and it is legitimate.

Step 7: Count plot for fraudulent vs legitimate transactions

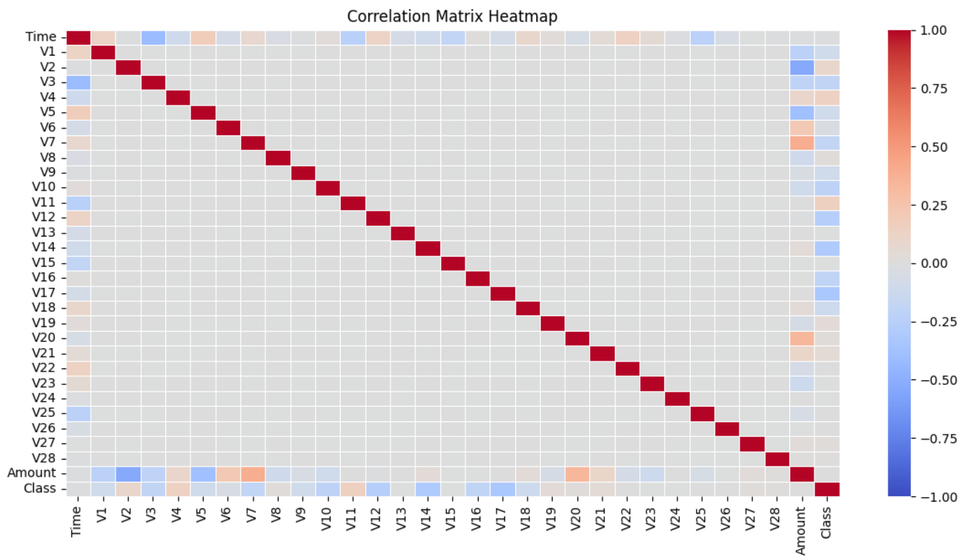
```
#Bar Chart Showing the Count of Fraudulent vs. Legitimate Transactions¶
plt.figure(figsize=(6, 4))
sns.countplot(x='Class', data=df)
plt.title('Count of Fraudulent vs. Legitimate Transactions')
```

```python
plt.xlabel('Class (0: Legitimate, 1: Fraudulent)')
plt.ylabel('Count')
plt.show()
```



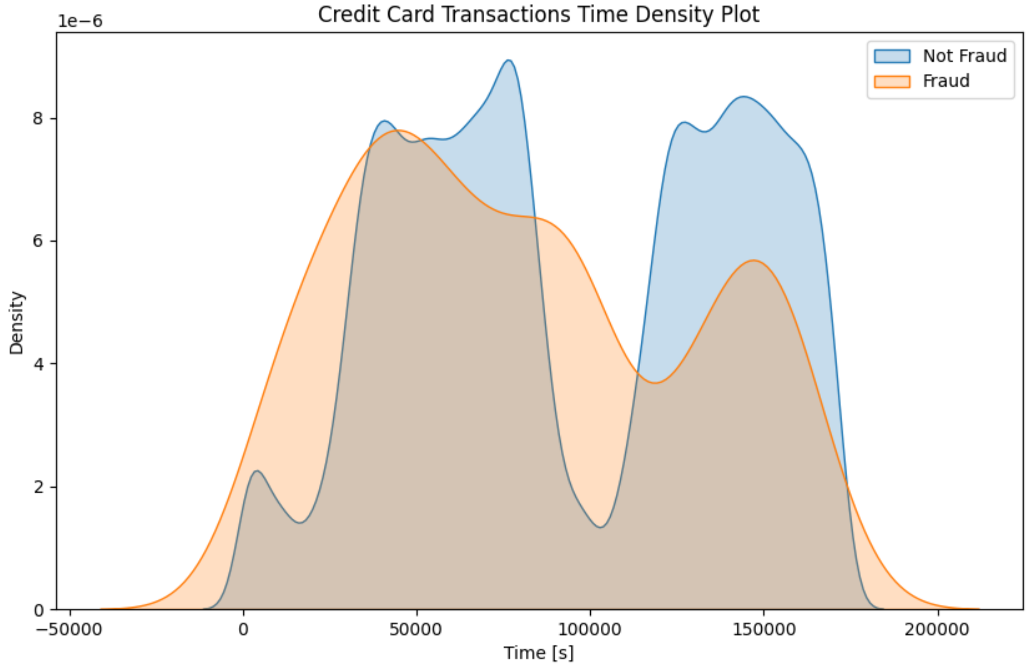
Step 8: Correlation matrix heatmap

```python
#Heatmap to Visualize the Correlation Between Numerical Features
correlation_matrix=df.corr()
plt.figure(figsize=(10,6))
sns.heatmap(correlation_matrix,cmap='coolwarm',vmin=-
1,vmax=1,annot=False,linewidths=0.5)
plt.title('Correlation Matrix Heatmap')
plt.show()
```



Step 9: Time-related analysis: Plot fraudulent vs non-fraudulent transaction times

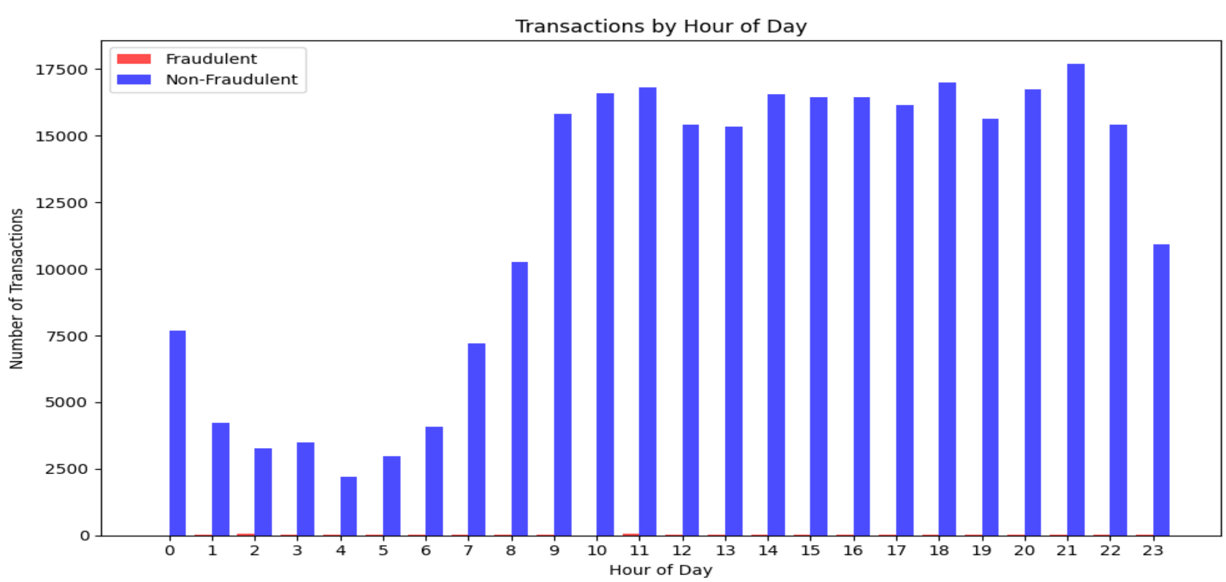
```
# Ensure there are no infinite values in the 'Time' column
df['Time'].replace([float('inf'), -float('inf')], float('nan'), inplace=True)

# Splitting the data
class_0 = df.loc[df['Class'] == 0]["Time"]
class_1 = df.loc[df['Class'] == 1]["Time"]

# Creating a Seaborn KDE plot
plt.figure(figsize=(10, 6))
sns.kdeplot(class_0, label='Not Fraud', fill=True, common_norm=False)
sns.kdeplot(class_1, label='Fraud', fill=True, common_norm=False)

# Adding titles and labels
plt.title('Credit Card Transactions Time Density Plot')
plt.xlabel('Time [s]')
plt.ylabel('Density')
plt.legend()
plt.show()
```

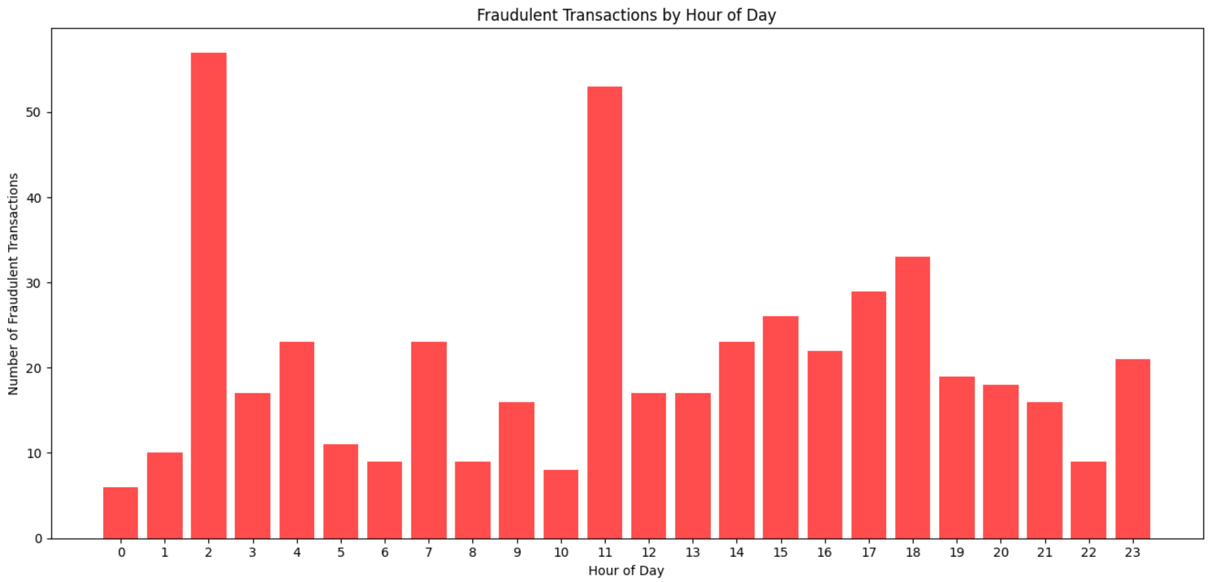


Step 10: Hourly analysis of fraudulent transactions

```
#Hour with the Most Frequent Fraudulent Transactions
# Convert the 'Time' column to hours
df['transaction_hour'] = (df['Time'] // 3600) % 24

# Assuming 'Class' column indicates fraudulent transactions (1 for fraud, 0 for
non-fraud)
df['is_fraudulent'] = df['Class']

# Separate the fraudulent and non-fraudulent transactions
fraudulent_df = df[df['is_fraudulent'] == 1]
non_fraudulent_df = df[df['is_fraudulent'] == 0]
# Aggregate by hour for fraudulent and non-fraudulent transactions
fraudulent_by_hour = fraudulent_df.groupby('transaction_hour').size()
non_fraudulent_by_hour = non_fraudulent_df.groupby('transaction_hour').size()

# Plotting
plt.figure(figsize=(10, 6))
plt.bar(fraudulent_by_hour.index - 0.2, fraudulent_by_hour.values, width=0.4,
color='red', alpha=0.7, label='Fraudulent')
plt.bar(non_fraudulent_by_hour.index + 0.2, non_fraudulent_by_hour.values,
width=0.4, color='blue', alpha=0.7, label='Non-Fraudulent')
plt.xlabel('Hour of Day')
```

```
plt.ylabel('Number of Transactions')
plt.title('Transactions by Hour of Day')
plt.xticks(range(0, 24))  # Set x-ticks for hours
plt.legend()
plt.tight_layout()
plt.show()

# Convert the 'Time' column to hours
df['transaction_hour'] = (df['Time'] // 3600) % 24

# Assuming 'Class' column indicates fraudulent transactions (1 for fraud, 0 for
non-fraud)
df['is_fraudulent'] = df['Class']

# Separate the fraudulent transactions
fraudulent_df = df[df['is_fraudulent'] == 1]

# Aggregate by hour for fraudulent transactions
fraudulent_by_hour = fraudulent_df.groupby('transaction_hour').size()
# Find the hour with the most frequent fraudulent transactions
most_frequent_hour = fraudulent_by_hour.idxmax()
most_frequent_count = fraudulent_by_hour.max()

# Print the result
print(f"The hour with the most frequent fraudulent transactions is:
{most_frequent_hour}:00")
print(f"Number of fraudulent transactions during this hour:
{most_frequent_count}")
```





The hour with the most frequent fraudulent transactions is: 2.0:00

Number of fraudulent transactions during this hour: 57

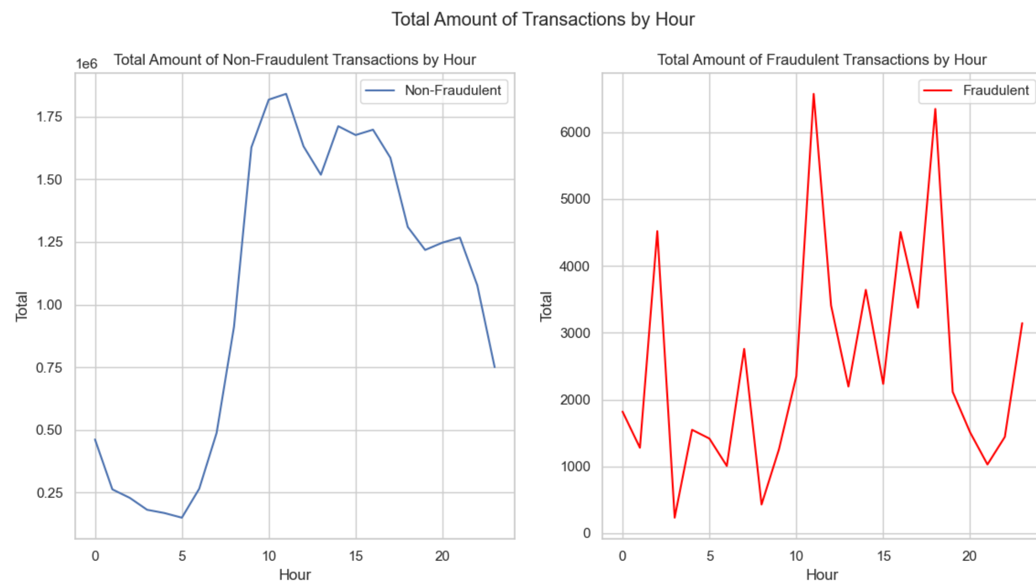
Step 11: 10. Compare transaction statistics (Total, Mean, Max, Median) by hour for fraudulent and non-fraudulent transactions

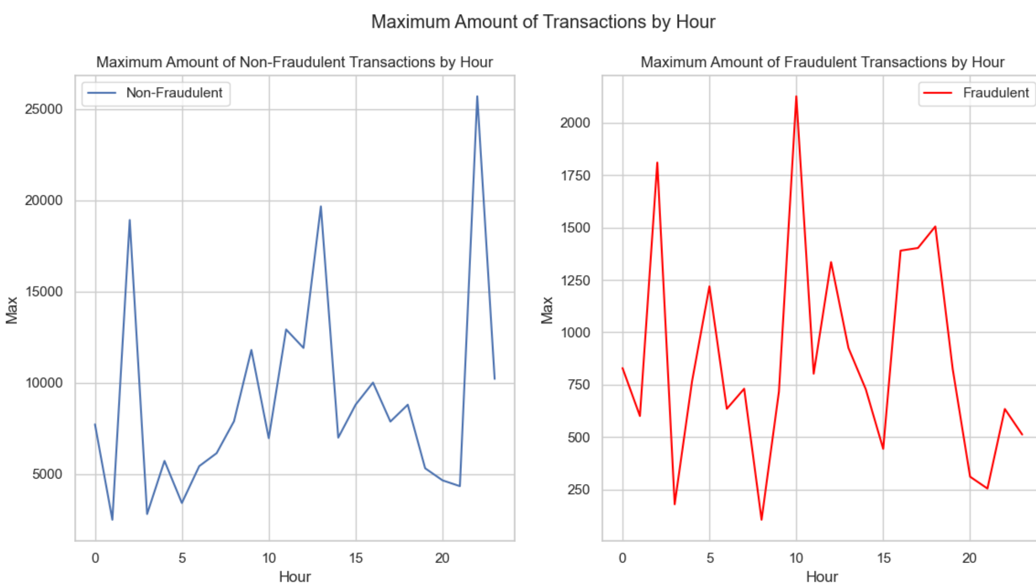
```
# Convert the 'Time' column to hours
df['transaction_hour'] = (df['Time'] // 3600) % 24

# Replace inf values with NaN
df.replace([np.inf, -np.inf], np.nan, inplace=True)

# Calculate required statistics for each hour
hourly_stats = df.groupby('transaction_hour').agg({
    'Amount': ['sum', 'mean', 'min', 'max', 'median']
}).reset_index()

# Rename columns for easier access
hourly_stats.columns = ['Hour', 'Total', 'Mean', 'Min', 'Max', 'Median']
# Separate fraudulent and non-fraudulent transactions
fraudulent_df = df[df['Class'] == 1]
non_fraudulent_df = df[df['Class'] == 0]

# Calculate required statistics for fraudulent transactions
fraudulent_stats = fraudulent_df.groupby('transaction_hour').agg({
    'Amount': ['sum', 'mean', 'min', 'max', 'median']
}).reset_index()
```

```
fraudulent_stats.columns = ['Hour', 'Total', 'Mean', 'Min', 'Max', 'Median']

# Calculate required statistics for non-fraudulent transactions
non_fraudulent_stats = non_fraudulent_df.groupby('transaction_hour').agg({
    'Amount': ['sum', 'mean', 'min', 'max', 'median']
}).reset_index()
non_fraudulent_stats.columns = ['Hour', 'Total', 'Mean', 'Min', 'Max', 'Median']
```

Step 12: Plotting total and maximum transaction amounts

```
# Set up the plotting environment
sns.set(style="whitegrid")

# Plot Total Amount
fig, (ax1, ax2) = plt.subplots(ncols=2, figsize=(18, 6))
sns.lineplot(ax=ax1, x='Hour', y='Total', data=non_fraudulent_stats, label='Non-
Fraudulent')
sns.lineplot(ax=ax2, x='Hour', y='Total', data=fraudulent_stats,
label='Fraudulent', color='red')
ax1.set_title('Total Amount of Non-Fraudulent Transactions by Hour')
ax2.set_title('Total Amount of Fraudulent Transactions by Hour')
plt.suptitle('Total Amount of Transactions by Hour')
plt.show()

# Plot Maximum Amount
fig, (ax1, ax2) = plt.subplots(ncols=2, figsize=(18, 6))
sns.lineplot(ax=ax1, x='Hour', y='Max', data=non_fraudulent_stats, label='Non-
Fraudulent')
sns.lineplot(ax=ax2, x='Hour', y='Max', data=fraudulent_stats,
label='Fraudulent', color='red')
ax1.set_title('Maximum Amount of Non-Fraudulent Transactions by Hour')
ax2.set_title('Maximum Amount of Fraudulent Transactions by Hour')
plt.suptitle('Maximum Amount of Transactions by Hour')
plt.show()
```






```
# Create a temporary DataFrame with 'Amount' and 'Class' columns
tmp = df[['Amount', 'Class']].copy()

# Separate the data into non-fraudulent and fraudulent transactions
class_0 = tmp.loc[tmp['Class'] == 0]['Amount']
class_1 = tmp.loc[tmp['Class'] == 1]['Amount']

# Display summary statistics
print("Non-Fraudulent Transactions:",class_0.describe())
print("\nFraudulent Transactions:",class_1.describe())
```

```
# Filter data for fraudulent transactions
fraud = df.loc[df['Class'] == 1]
print("data for fraudulent transactions:",fraud)

# Create the scatter plot using Seaborn
plt.figure(figsize=(10, 6))
sns.scatterplot(x=fraud['Time'],y=fraud['Amount'],color='red',alpha=0.5)
plt.title('Amount of Fraudulent Transactions')
plt.xlabel('Time [s]')
plt.ylabel('Amount')
plt.show()
```

Output:

Non-Fraudulent Transactions: count 284315.000000

mean 88.291022

std 250.105092

min 0.000000

25% 5.650000

50% 22.000000

75% 77.050000

max 25691.160000

Name: Amount, dtype: float64

Fraudulent Transactions: count 492.000000

mean 122.211321

std 256.683288

min 0.000000

25%   1.000000

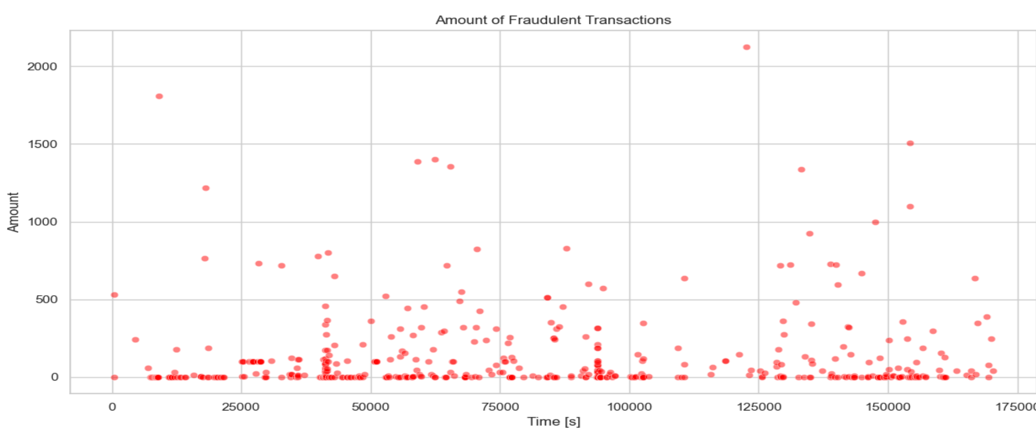
50%   9.250000

75%   105.890000

max   2125.870000

data for fraudulent transactions:

```
data for fraudulent transactions:               Time        V1        V2        V3        V4        V5  ...       V27       V28  Amount  Class  transaction
_hour  is_fraudulent
541        406.0 -2.312227  1.951992 -1.609851  3.997906 -0.522188  ...  0.261145 -0.143276    0.00      1         0.0          1
623        472.0 -3.043541 -3.157307  1.088463  2.288644  1.359805  ... -0.252773  0.035764  529.00      1         0.0          1
4920      4462.0 -2.303350  1.759247 -0.359745  2.330243 -0.821628  ...  0.039566 -0.153029  239.93      1         1.0          1
6108      6986.0 -4.397974  1.358367 -2.592844  2.679787 -1.128131  ... -0.827136  0.849573   59.00      1         1.0          1
6329      7519.0  1.234235  3.019740 -4.304597  4.732795  3.624201  ... -0.010016  0.146793    1.00      1         2.0          1
...          ...       ...       ...       ...       ...       ...  ...       ...       ...     ...    ...         ...        ...
279863  169142.0 -1.927883  1.125653 -4.518331  1.749293 -1.566487  ...  0.292680  0.147968  390.00      1        22.0          1
280143  169347.0  1.378559  1.289381 -5.004247  1.411850  0.442581  ...  0.389152  0.186637    0.76      1        23.0          1
280149  169351.0 -0.676143  1.126366 -2.213700  0.468308 -1.120541  ...  0.385107  0.194361   77.89      1        23.0          1
281144  169966.0 -3.113832  0.585864 -5.399730  1.817092 -0.840618  ...  0.884876 -0.253700  245.00      1        23.0          1
281674  170348.0  1.991976  0.158476 -2.583441  0.408670  1.151147  ...  0.002988 -0.015309   42.53      1        23.0          1
```



Conclusion:

The analysis of credit card transactions highlighted the rarity of fraudulent transactions, which account for only 0.17% of the total dataset. Despite their low occurrence, fraudulent transactions exhibit distinct patterns, such as a higher frequency during late hours (12 AM to 3 AM) and often involving higher transaction amounts. Visualizations revealed that these transactions tend to be more concentrated in specific hours, and high-value fraudulent transactions occur within shorter time intervals. To improve fraud detection, it is crucial to address the class imbalance through techniques like oversampling or under sampling. Additionally, developing a machine learning model that leverages time-based features and transaction amounts could enhance detection accuracy. Evaluating model performance with metrics like precision, recall, and F1-score will be key to ensuring its effectiveness in identifying fraud.