入力 [83]:

```python
# Import the modules
import numpy as np
import pandas as pd
from pathlib import Path
from sklearn.metrics import balanced_accuracy_score, confusion_matrix, classification_report
from datetime import datetime
from math import radians, cos, sin, asin, sqrt
from datetime import datetime
import matplotlib.pyplot as plt
import seaborn as sns
from scipy.stats import ttest_ind

```

入力 [84]:

```python
df_cc_data = pd.read_csv("fraudTest.csv")
```

入力 [85]:

```python
df_cc_data.head(5)
```

出力[85]:

	Unnamed: 0	trans_date_trans_time	cc_num	merchant	category	amt	first	last
0	0	2020-06-21 12:14:25	2291163933867244	fraud_Kirlin and Sons	personal_care	2.86	Jeff	Elliott
1	1	2020-06-21 12:14:33	3573030041201292	fraud_Sporer-Keebler	personal_care	29.84	Joanne	Williams
2	2	2020-06-21 12:14:53	3598215285024754	fraud_Swaniawski, Nitzsche and Welch	health_fitness	41.28	Ashley	Lopez
3	3	2020-06-21 12:15:15	3591919803438423	fraud_Haley Group	misc_pos	60.05	Brian	Williams
4	4	2020-06-21 12:15:17	3526826139003047	fraud_Johnston-Casper	travel	3.19	Nathan	Massey

5 rows × 23 columns

入力 [86]:

```python
# Convert the transaction date and date of birth to datetime format for easier manipulation
df_cc_data["trans_date_trans_time"] = pd.to_datetime(df_cc_data["trans_date_trans_time"])
df_cc_data["dob"] = pd.to_datetime(df_cc_data["dob"])
```

入力 [87]:

```python
# Print the datatypes of each column to ensure proper formats
print(df_cc_data.dtypes)
```

```
Unnamed: 0                        int64
trans_date_trans_time    datetime64[ns]
cc_num                            int64
merchant                         object
category                         object
amt                             float64
first                            object
last                             object
gender                           object
street                           object
city                             object
state                            object
zip                               int64
lat                             float64
long                            float64
city_pop                          int64
job                              object
dob                      datetime64[ns]
trans_num                        object
unix_time                         int64
merch_lat                       float64
merch_long                      float64
is_fraud                          int64
dtype: object
```

入力 [88]:

```python
df_cc_data.isnull().sum()
```

出力[88]:

```
Unnamed: 0               0
trans_date_trans_time    0
cc_num                   0
merchant                 0
category                 0
amt                      0
first                    0
last                     0
gender                   0
street                   0
city                     0
state                    0
zip                      0
lat                      0
long                     0
city_pop                 0
job                      0
dob                      0
trans_num                0
unix_time                0
merch_lat                0
merch_long               0
is_fraud                 0
dtype: int64
```

Feature Engineering

入力 [89]:

```
# Extract time-based features for further analysis
df_cc_data['hour'] = df_cc_data['trans_date_trans_time'].dt.hour
df_cc_data['day'] = df_cc_data['trans_date_trans_time'].dt.dayofweek
df_cc_data['week'] = df_cc_data['trans_date_trans_time'].dt.isocalendar().week
```

入力 [91]:

```
# Calculate the time gap between transactions for each customer (using cc_num)
df_cc_data['time_gap'] = df_cc_data.groupby('cc_num')['trans_date_trans_time'].diff()
df_cc_data['time_gap'] = df_cc_data['time_gap'].fillna(pd.Timedelta(seconds=0))
```

Time-based features such as hour, day, and week are extracted to analyze transaction patterns over time. 'time_gap' represents the time difference between consecutive transactions for the same credit card number, which could be a useful feature in fraud detection.

入力 [92]:

```
# Function to calculate the great circle distance between customer and merchant based on latitude

def haversine(lon1, lat1, lon2, lat2):
    """
    Calculate the great circle distance in kilometers between two points
    on the earth (specified in decimal degrees)
    """

    lon1, lat1, lon2, lat2 = map(radians, [lon1, lat1, lon2, lat2])


    dlon = lon2 - lon1
    dlat = lat2 - lat1
    a = sin(dlat/2)**2 + cos(lat1) * cos(lat2) * sin(dlon/2)**2
    c = 2 * asin(sqrt(a))
    r = 6371
    return c * r


```

入力 [93]:

```
# Apply the Haversine formula to calculate the distance between the customer and merchant
df_cc_data['distance_km'] = df_cc_data.apply(lambda row: haversine(row['long'], row['lat'], row[
df_cc_data['distance_km'] = df_cc_data['distance_km'].round(2)
```

入力 [94]:

```
df_cc_data['amt'].describe()
```

出力[94]:

```
count    555719.000000
mean         69.392810
std         156.745941
min           1.000000
25%           9.630000
50%          47.290000
75%          83.010000
max       22768.110000
Name: amt, dtype: float64
```

入力 [95]:

```
bin_edges = [0, 9.63, 47.29, 83.01, 1000, df_cc_data['amt'].max()]
bin_labels = ['Very Low', 'Low', 'Medium', 'High', 'Very High']
df_cc_data['transaction_bins'] = pd.cut(df_cc_data['amt'], bins=bin_edges, labels=bin_labels, in
print(df_cc_data[['amt', 'transaction_bins']])
```

```
          amt transaction_bins
0        2.86         Very Low
1       29.84              Low
2       41.28              Low
3       60.05           Medium
4        3.19         Very Low
...       ...              ...
555714  43.77              Low
555715  111.84            High
555716  86.88             High
555717   7.99         Very Low
555718  38.13              Low

[555719 rows x 2 columns]
```

Note: "Transaction amounts are categorized into bins (Very Low to Very High) to simplify the analysis and make it easier to observe fraud patterns across different transaction ranges."

入力 [96]:

```
df_cc_data['dob']
```

出力[96]:

```
0        1968-03-19
1        1990-01-17
2        1970-10-21
3        1987-07-25
4        1955-07-06
            ...
555714   1966-02-13
555715   1999-12-27
555716   1981-11-29
555717   1965-12-15
555718   1993-05-10
Name: dob, Length: 555719, dtype: datetime64[ns]
```

入力 [97]:

```
today = datetime.today()

df_cc_data['age'] = today.year - df_cc_data['dob'].dt.year - (
    (today.month < df_cc_data['dob'].dt.month) |
    ((today.month == df_cc_data['dob'].dt.month) & (today.day < df_cc_data['dob'].dt.day))
)

```

Note: "The age of each customer is calculated based on their date of birth. Age can be a key factor when analyzing patterns in fraudulent transactions."

入力 [98]:

```
df_cc_data['age']
```

出力[98]:

```
0         56
1         34
2         53
3         37
4         69
          ..
555714    58
555715    24
555716    42
555717    58
555718    31
Name: age, Length: 555719, dtype: int64
```

*** Exploratory Data Analysis***

入力 [99]:

```
df_cc_data.describe()
```

出力[99]:

	Unnamed: 0	cc_num	amt	zip	lat	long	city_pop
count	555719.000000	5.557190e+05	555719.000000	555719.000000	555719.000000	555719.000000	5.557190e+05
mean	277859.000000	4.178387e+17	69.392810	48842.628015	38.543253	-90.231325	8.822189e+04
std	160422.401459	1.309837e+18	156.745941	26855.283328	5.061336	13.721780	3.003909e+05
min	0.000000	6.041621e+10	1.000000	1257.000000	20.027100	-165.672300	2.300000e+01
25%	138929.500000	1.800429e+14	9.630000	26292.000000	34.668900	-96.798000	7.410000e+02
50%	277859.000000	3.521417e+15	47.290000	48174.000000	39.371600	-87.476900	2.408000e+03
75%	416788.500000	4.635331e+15	83.010000	72011.000000	41.894800	-80.175200	1.968500e+04
max	555718.000000	4.992346e+18	22768.110000	99921.000000	65.689900	-67.950300	2.906700e+06

入力 [100]:

```
df_cc_data['gender'].value_counts()
```

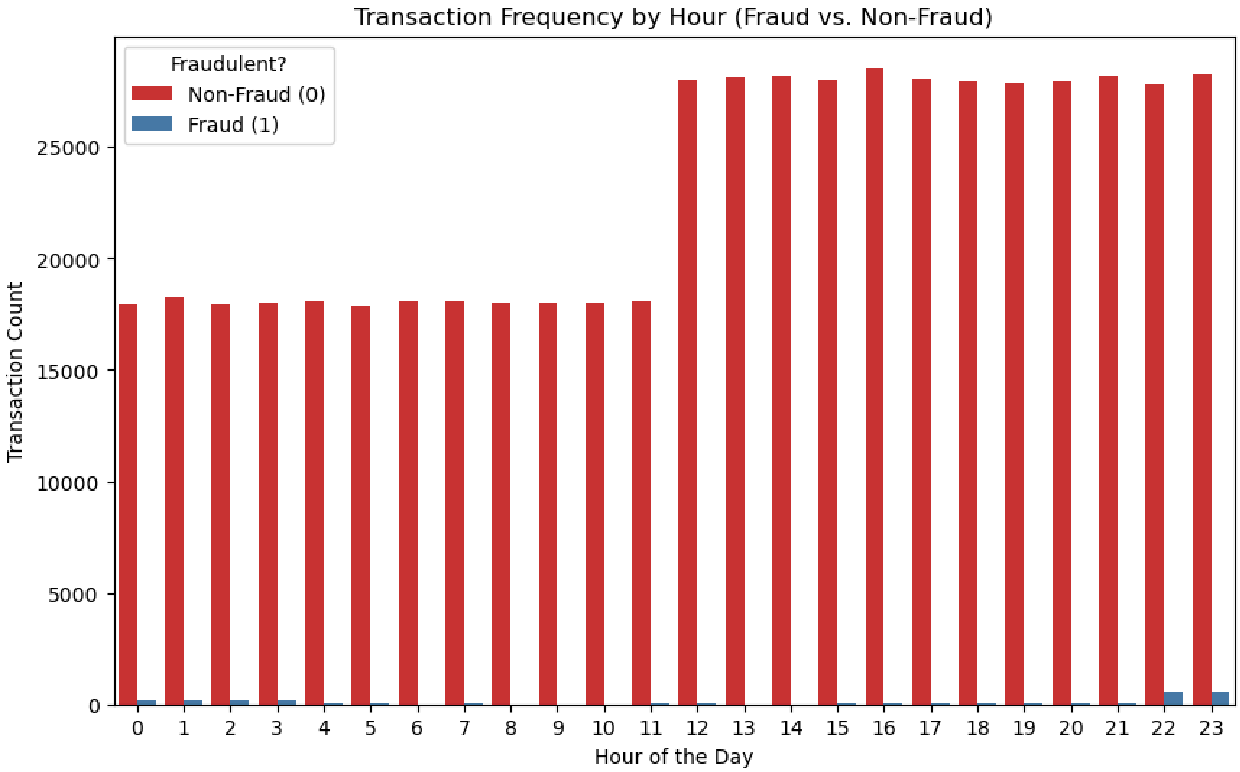
出力[100]:

```
F    304886
M    250833
Name: gender, dtype: int64
```

Data Visualization

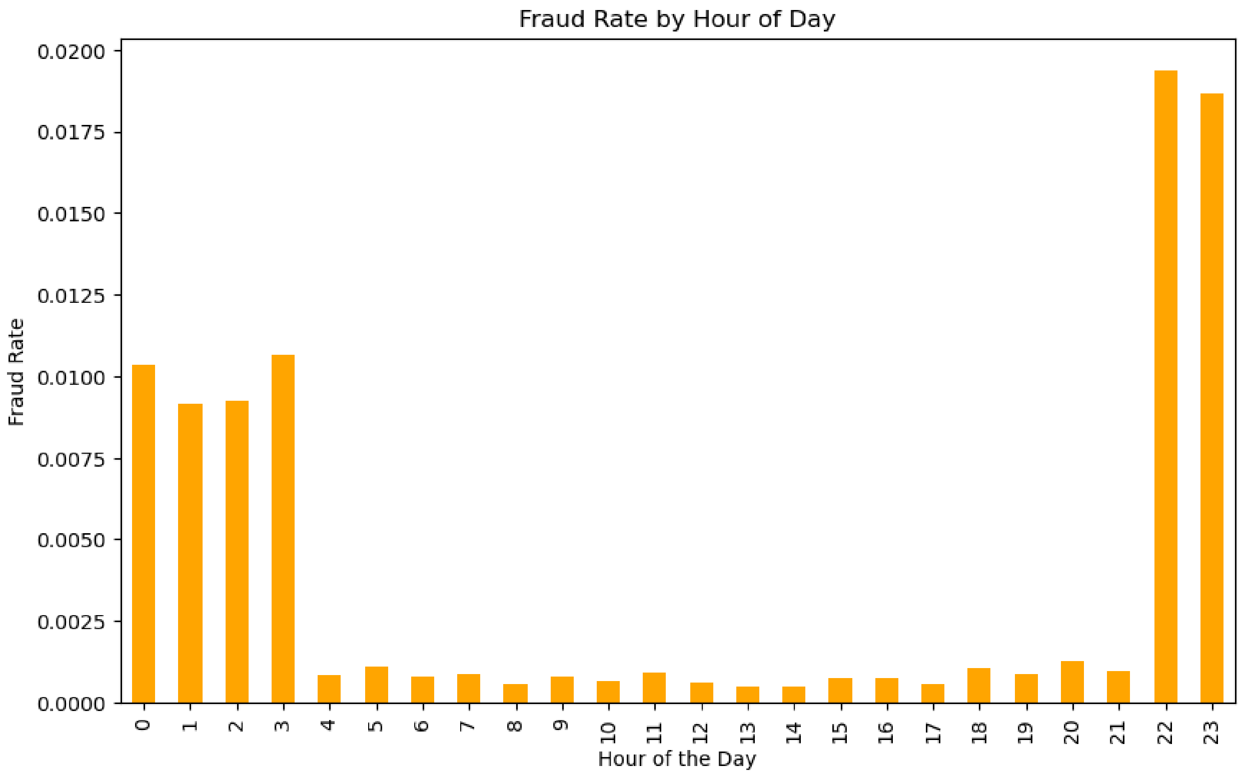
入力 [101]:

```
plt.figure(figsize=(10, 6))
sns.countplot(data=df_cc_data, x='hour', hue='is_fraud', palette='Set1')
plt.title('Transaction Frequency by Hour (Fraud vs. Non-Fraud)')
plt.xlabel('Hour of the Day')
plt.ylabel('Transaction Count')
plt.legend(title='Fraudulent?', labels=['Non-Fraud (0)', 'Fraud (1)'])
plt.show()
```



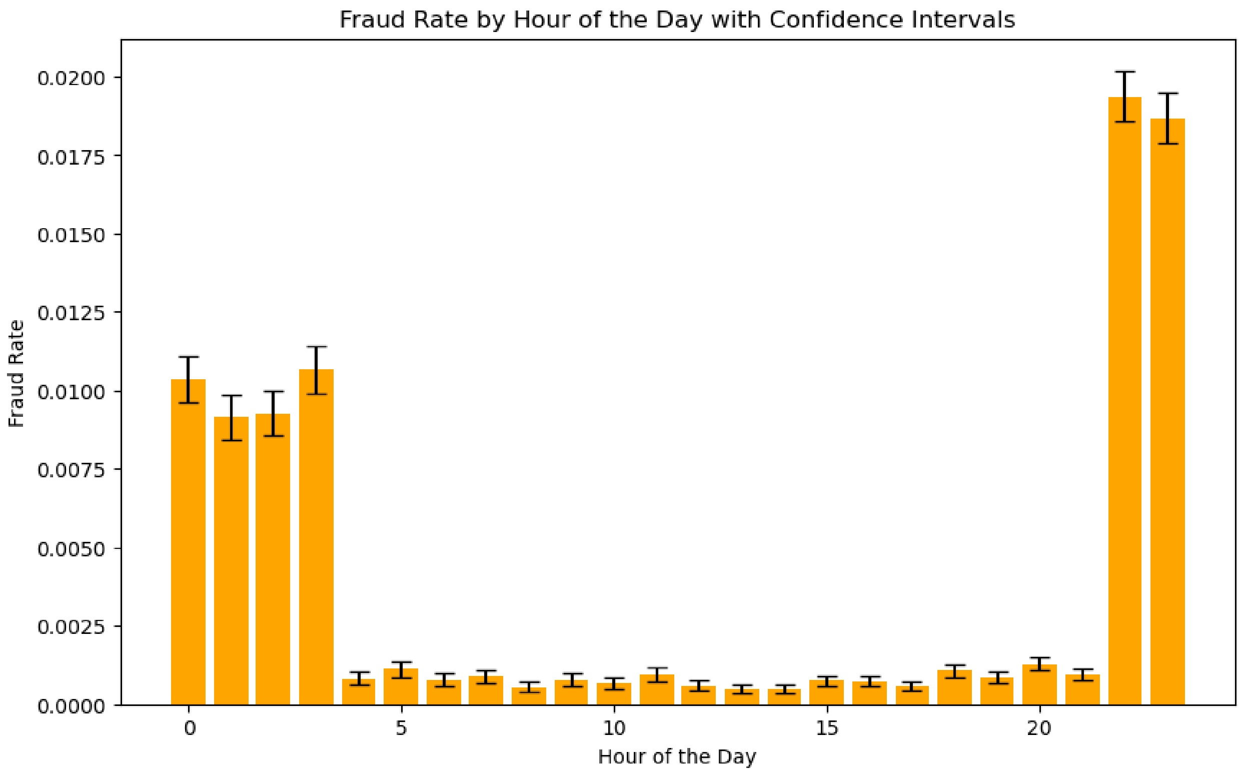
入力 [102]:

```python
fraud_rate_by_hour = df_cc_data.groupby('hour')['is_fraud'].mean()

plt.figure(figsize=(10, 6))
fraud_rate_by_hour.plot(kind='bar', color='orange')
plt.title('Fraud Rate by Hour of Day')
plt.xlabel('Hour of the Day')
plt.ylabel('Fraud Rate')
plt.show()

```



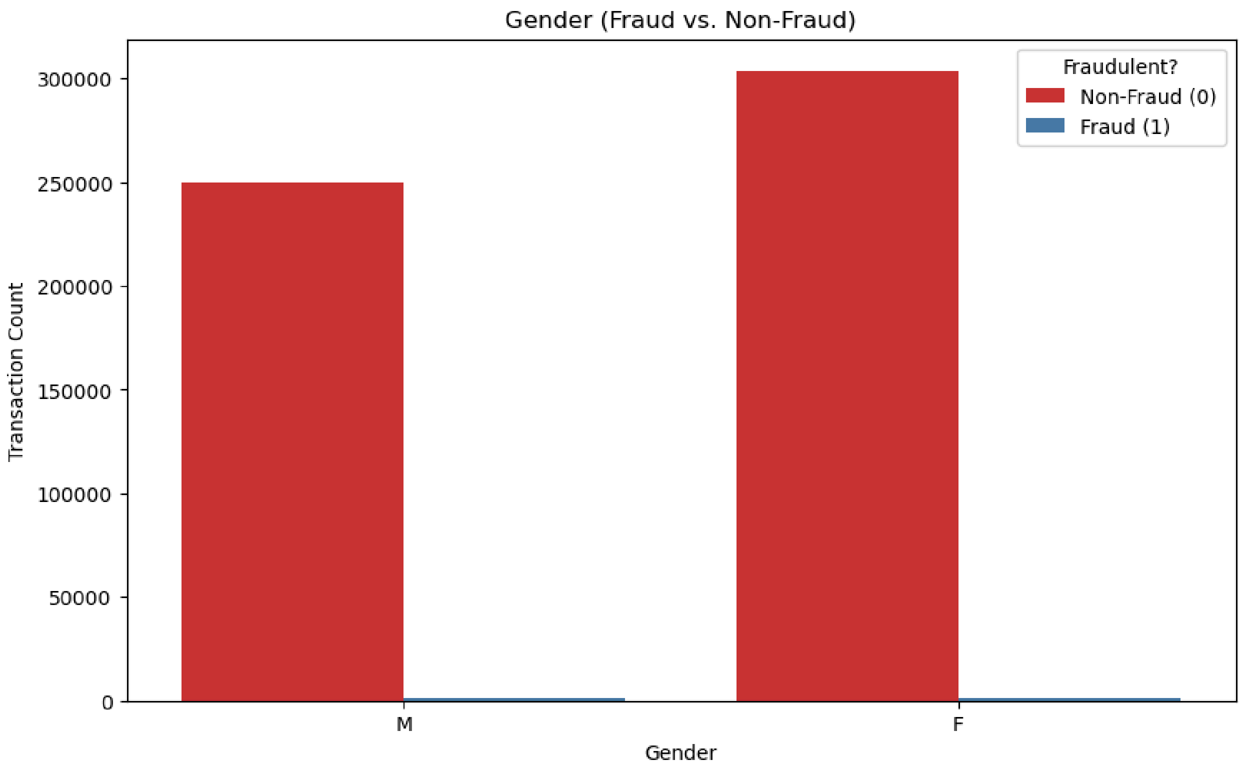
入力 [103]:

```
fraud_std_by_hour = df_cc_data.groupby('hour')['is_fraud'].std()
fraud_count_by_hour = df_cc_data.groupby('hour')['is_fraud'].count()

# Standard error
fraud_se_by_hour = fraud_std_by_hour / np.sqrt(fraud_count_by_hour)

plt.figure(figsize=(10, 6))
plt.bar(fraud_rate_by_hour.index, fraud_rate_by_hour, yerr=fraud_se_by_hour, color='orange', caps
plt.title('Fraud Rate by Hour of the Day with Confidence Intervals')
plt.xlabel('Hour of the Day')
plt.ylabel('Fraud Rate')
plt.show()

```



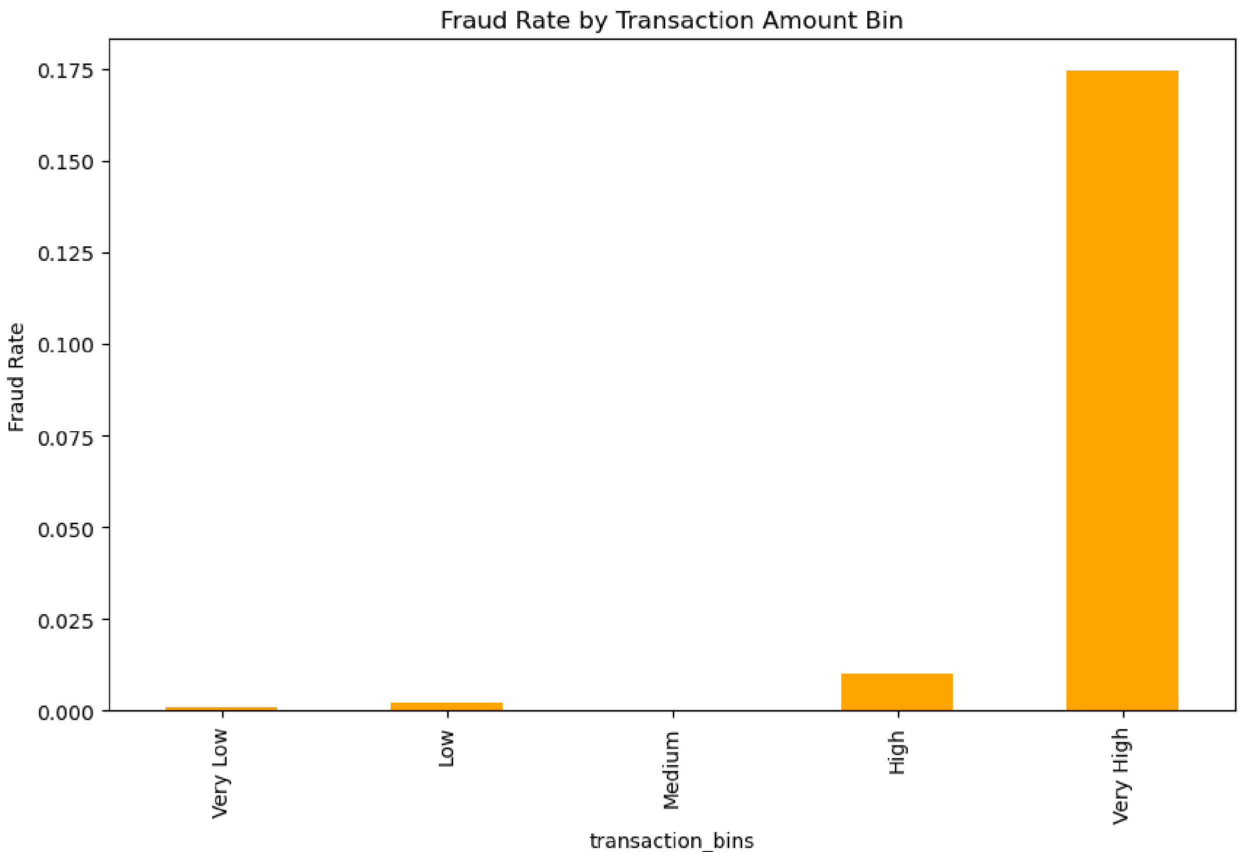
入力 [104]:

```
plt.figure(figsize=(10, 6))
sns.countplot(data=df_cc_data, x='gender', hue='is_fraud', palette='Set1')
plt.title('Gender (Fraud vs. Non-Fraud)')
plt.xlabel('Gender')
plt.ylabel('Transaction Count')
plt.legend(title='Fraudulent?', labels=['Non-Fraud (0)', 'Fraud (1)'])
plt.show()
```



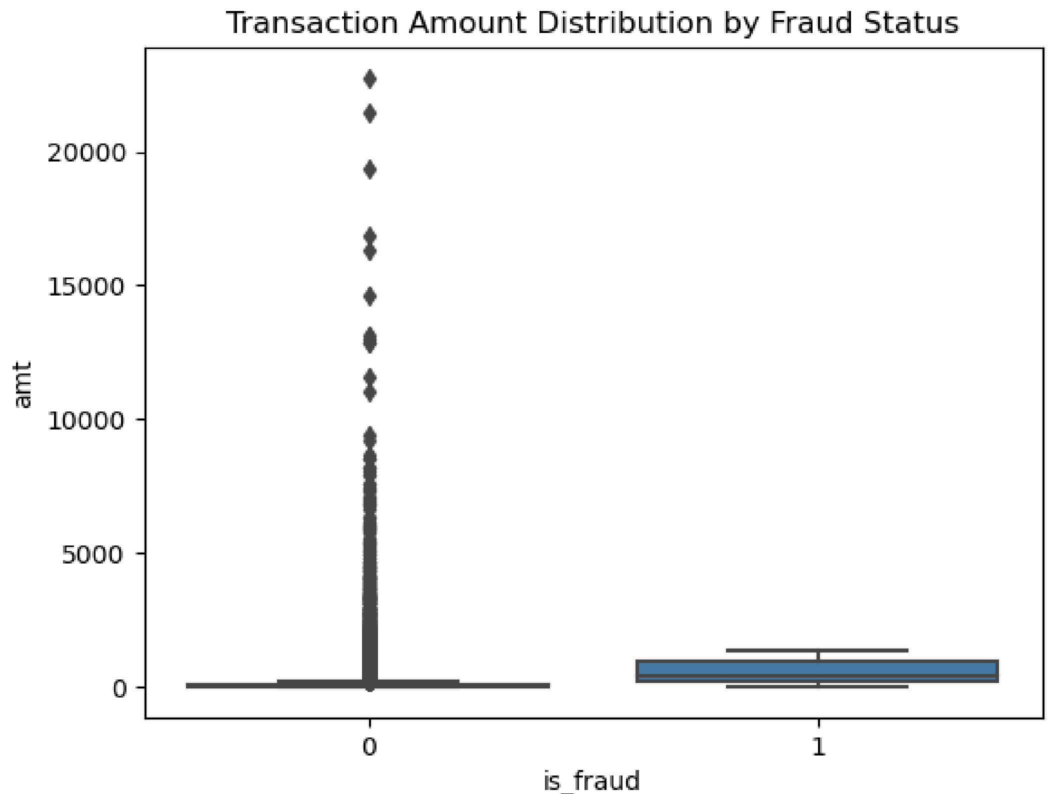
入力 [105]:

```
fraud_rate_by_amt_bin = df_cc_data.groupby('transaction_bins')['is_fraud'].mean()

plt.figure(figsize=(10, 6))
fraud_rate_by_amt_bin.plot(kind='bar', color='orange')
plt.title('Fraud Rate by Transaction Amount Bin')
plt.ylabel('Fraud Rate')
plt.show()
```



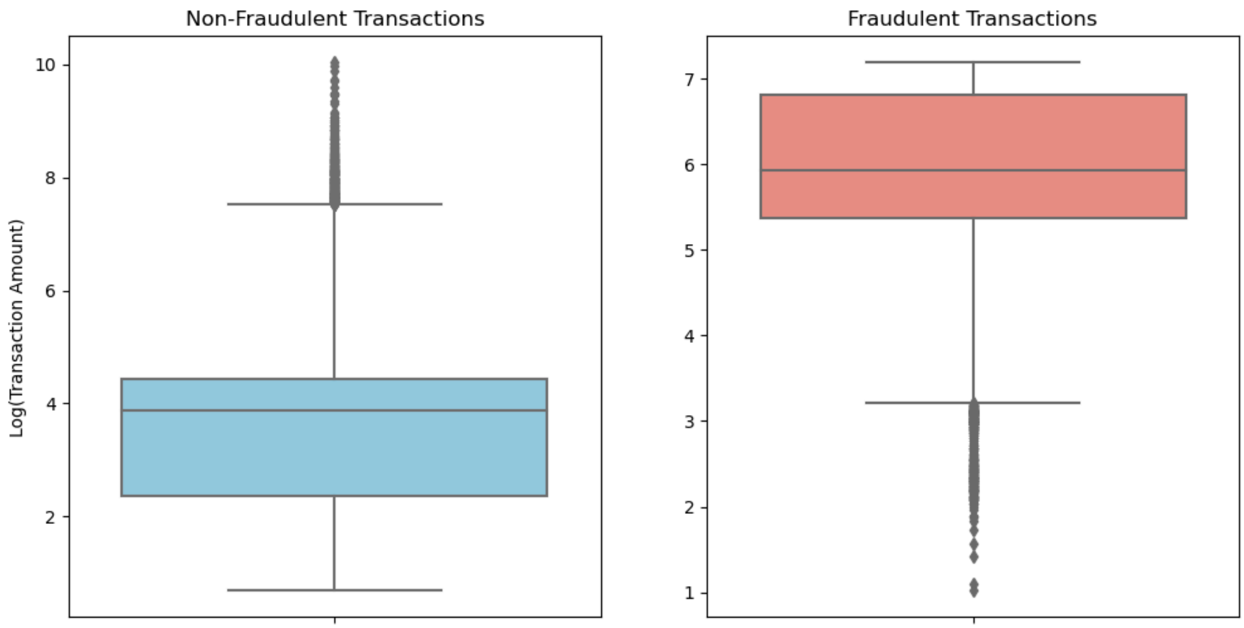
入力 [106]:

```
sns.boxplot(data=df_cc_data, x='is_fraud', y='amt', palette='Set1')
plt.title('Transaction Amount Distribution by Fraud Status')
plt.show()

```



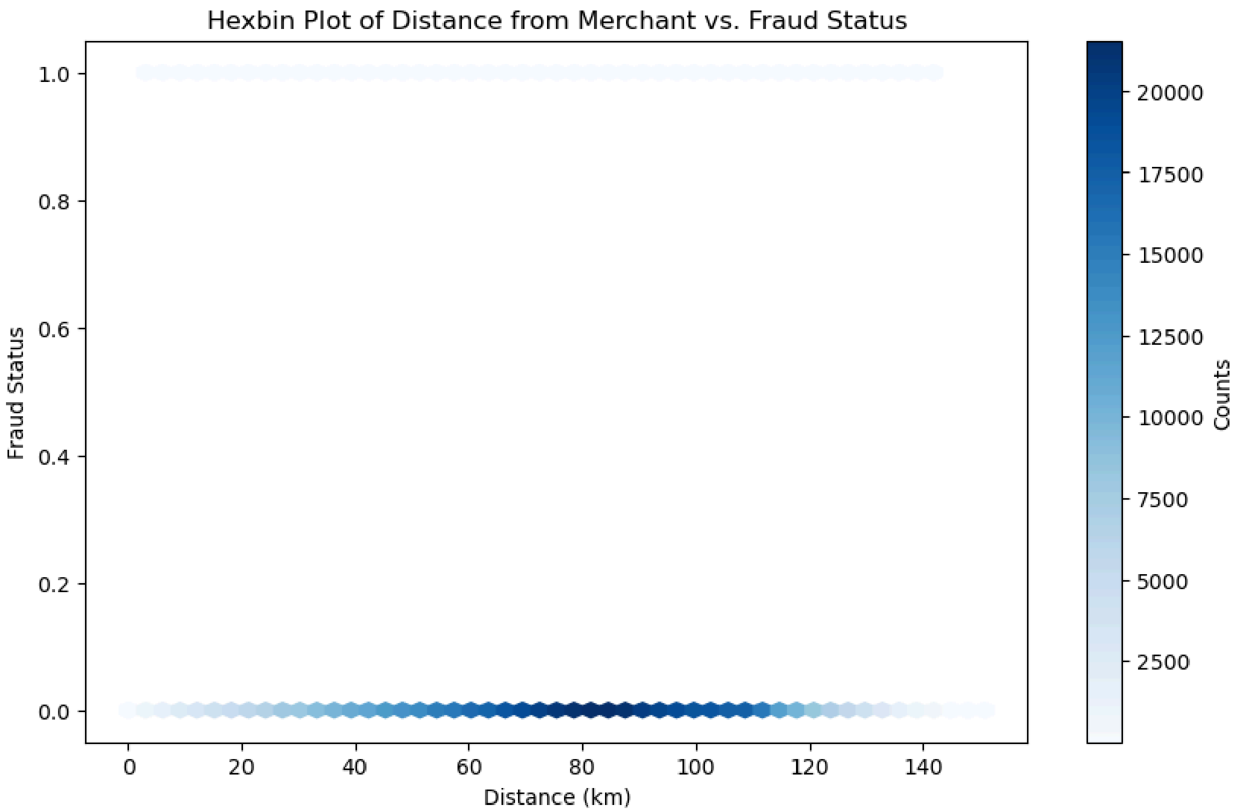
入力 [108]:

```
plt.figure(figsize=(12, 6))

plt.subplot(1, 2, 1)
sns.boxplot(data=df_cc_data[df_cc_data['is_fraud'] == 0], y='log_amt', color='skyblue')
plt.title('Non-Fraudulent Transactions')
plt.ylabel('Log(Transaction Amount)')

plt.subplot(1, 2, 2)
sns.boxplot(data=df_cc_data[df_cc_data['is_fraud'] == 1], y='log_amt', color='salmon')
plt.title('Fraudulent Transactions')
plt.ylabel('')

plt.show()
```



入力 [109]:

```
plt.figure(figsize=(10, 6))
plt.hexbin(df_cc_data['distance_km'], df_cc_data['is_fraud'], gridsize=50, cmap='Blues', mincnt=
plt.colorbar(label='Counts')
plt.title('Hexbin Plot of Distance from Merchant vs. Fraud Status')
plt.xlabel('Distance (km)')
plt.ylabel('Fraud Status')
plt.show()

```



Statistical Analysis

Note: "Statistical tests are performed to determine if there is a significant difference in transaction amounts between fraudulent and non-fraudulent transactions. The t-test assumes normality, while the Mann-Whitney U test is used as a non-parametric alternative."

入力 [111]:

```

fraud = df_cc_data[df_cc_data['is_fraud'] == 1]
non_fraud = df_cc_data[df_cc_data['is_fraud'] == 0]

t_stat, p_value = ttest_ind(fraud['amt'], non_fraud['amt'])
print(f"T-statistic: {t_stat}, P-value: {p_value}")

```

```
T-statistic: 138.1883771290534, P-value: 0.0
```

入力 [112]:

```
from scipy.stats import mannwhitneyu

fraud_amt = df_cc_data[df_cc_data['is_fraud'] == 1]['amt']
non_fraud_amt = df_cc_data[df_cc_data['is_fraud'] == 0]['amt']

stat, p_value = mannwhitneyu(fraud_amt, non_fraud_amt)
print(f"Mann-Whitney U Test: Statistic={stat}, p-value={p_value}")

```

```
Mann-Whitney U Test: Statistic=989380266.5, p-value=0.0
```

入力 [114]:

```
fraud_corr = df_cc_data.corr(numeric_only=True)['is_fraud'].sort_values(ascending=False)
print(fraud_corr)

```

```
is_fraud       1.000000
amt            0.182267
log_amt        0.098374
hour           0.011686
day            0.009365
age            0.007543
lat            0.005863
merch_lat      0.005812
distance_km    0.000233
long          -0.000972
merch_long    -0.001060
cc_num        -0.001540
zip           -0.002271
city_pop      -0.004910
unix_time     -0.013066
week          -0.013446
Unnamed: 0    -0.013892
Name: is_fraud, dtype: float64
```

Note: "Correlation analysis is conducted to quantify the strength of the relationships between various features and the fraud indicator. This can help identify which features are most predictive of fraudulent behavior."

入力 [ ]:

```

```