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

# Set the visualization style and enable inline plotting
sns.set_style("whitegrid")
%matplotlib inline

# --- Load Data ---
# ⚠️ Make sure to use the exact file name you uploaded to Colab
file_path = 'credit_card_transactions.csv'
df = pd.read_csv(file_path)

print("--- Data Head (First 5 Rows) ---")
print(df.head())

print("\n--- Data Info (Data Types and Non-Null Counts) ---")
df.info()

print("\n--- Shape (Rows and Columns) ---")
print(f"Total Rows: {df.shape[0]}, Total Columns: {df.shape[1]}")
```

```
3      Kenneth     Foster  ...  76383.0                Geoscientist
4         Gina     Grimes  ...    606.0              Energy manager

         dob                          trans_num     unix_time  merch_lat  \
0  2/10/1935  309c4bf7fc47b1ddee5ad883bcf634b6  1.354379e+09  42.317313
1  3/16/1964  2ca9bd5df126cc35e541a4f2c2551197  1.354464e+09  41.665873
2  10/9/1973  dda9d800d37a9fc2c086a836d66b0588  1.347661e+09  33.020256
3   4/4/1985  0b902a1e549c98b949444a7557da2403  1.339705e+09  42.397174
4  9/22/1997  1c9f2b574fb9bf860c76ea200252fe05  1.333813e+09  41.361042
```

```
 3   Transaction_Time       3544 non-null   object
 4   cc_num                 3544 non-null   float64
 5   merchant               3544 non-null   object
 6   category               3544 non-null   object
 7   amt                    3544 non-null   object
 8   first                  3544 non-null   object
 9   last                   3544 non-null   object
 10  gender                 3544 non-null   object
 11  street                 3544 non-null   object
 12  city                   3543 non-null   object
 13  state                  3543 non-null   object
 14  zip                    3543 non-null   float64
 15  lat                    3543 non-null   float64
 16  long                   3543 non-null   float64
 17  city_pop               3543 non-null   float64
 18  job                    3543 non-null   object
 19  dob                    3543 non-null   object
 20  trans_num              3543 non-null   object
 21  unix_time              3543 non-null   float64
 22  merch_lat              3543 non-null   float64
 23  merch_long             3543 non-null   float64
 24  is_fraud               3543 non-null   float64
 25  merch_zipcode          2972 non-null   float64
 26  Age                    3543 non-null   float64
dtypes: float64(11), int64(1), object(15)
memory usage: 747.7+ KB

--- Shape (Rows and Columns) ---
Total Rows: 3544, Total Columns: 27
```

```
# A. Drop Unnecessary Columns
# Drop columns that are unique identifiers or not useful for initial EDA (like 'Unn
columns_to_drop = ['Unnamed: 0', 'cc_num', 'first', 'last', 'trans_num']
df = df.drop(columns=columns_to_drop, errors='ignore')
```

```
# B. Handle Missing Values
print("\n--- Missing Values Count Before Cleaning ---")
print(df.isnull().sum())
```

```

--- Missing Values Count Before Cleaning ---
trans_date_trans_time    0
Transaction_Date         0
Transaction_Time         0
merchant                 0
category                 0
amt                      0
gender                   0
street                   0
city                     1
state                    1
zip                      1
lat                      1
long                     1
```

```
city_pop              1
job                   1
dob                   1
unix_time             1
merch_lat             0
merch_long            0
is_fraud              1
merch_zipcode         0
Age                   1
dtype: int64
```

```
# Impute (fill) missing values in crucial numeric columns using the Median
# Assuming 'merch_lat', 'merch_long' are the location columns with NaNs
for col in ['merch_lat', 'merch_long', 'merch_zipcode']:
    if df[col].isnull().any():
        df[col].fillna(df[col].median() if df[col].dtype in ['float64', 'int64'] el
```

```
print(df.columns.tolist())
```

```
['trans_date_trans_time', 'Transaction_Date ', 'Transaction_Time', 'merchant', 'cate
```

```
# C. Data Type Conversion (Date and Time)
# 1. Use the combined column (which was already correct)
df['trans_date_trans_time'] = pd.to_datetime(df['trans_date_trans_time'], errors='c

# 2. Use the correct column name for the date, INCLUDING THE SPACE
df['Transaction_Date '] = pd.to_datetime(df['Transaction_Date '], errors='coerce')
```

```
/tmp/ipython-input-3929491084.py:3: UserWarning: Could not infer format, so each ele
  df['trans_date_trans_time'] = pd.to_datetime(df['trans_date_trans_time'], errors='
```

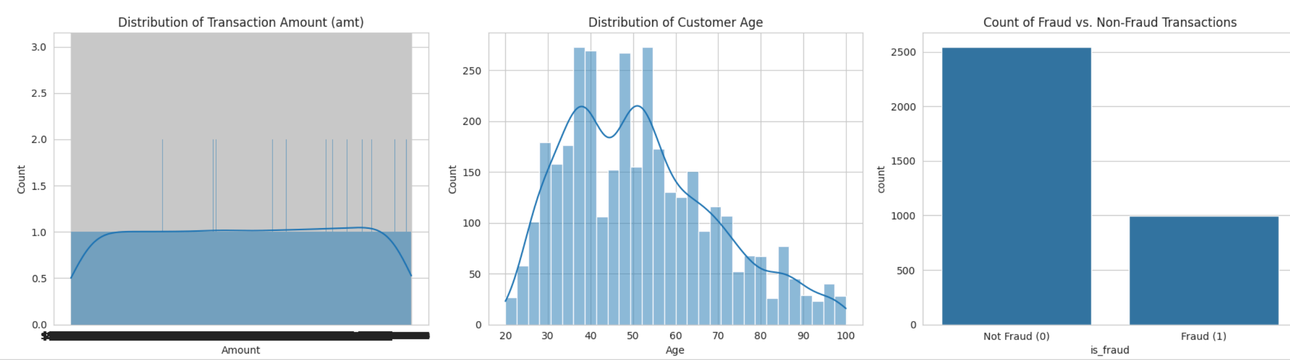
```
# 1. Fill NaN values with 0
df['is_fraud'] = df['is_fraud'].fillna(0)

# 2. Now convert to integer (this will succeed)
df['is_fraud'] = df['is_fraud'].astype('int')

# Or combine steps:
# df['is_fraud'] = df['is_fraud'].fillna(0).astype('int')
```

```
print("\n--- 1. Summary Statistics for Numeric Variables ---")
print(df.describe())
# Observation 1: Check for extreme values (Outliers) in 'amt'.

print("\n--- 2. 'is_fraud' Value Counts (Target Variable Analysis) ---")
fraud_counts = df['is_fraud'].value_counts()
print(fraud_counts)
```

```
# Observation 2: Check for Imbalanced Data (Fraud count is typically much lower tha

print("\n--- 3. Top 10 'category' Counts ---")
print(df['category'].value_counts().head(10))
# Observation 3: Identify the most frequent transaction categories.
```

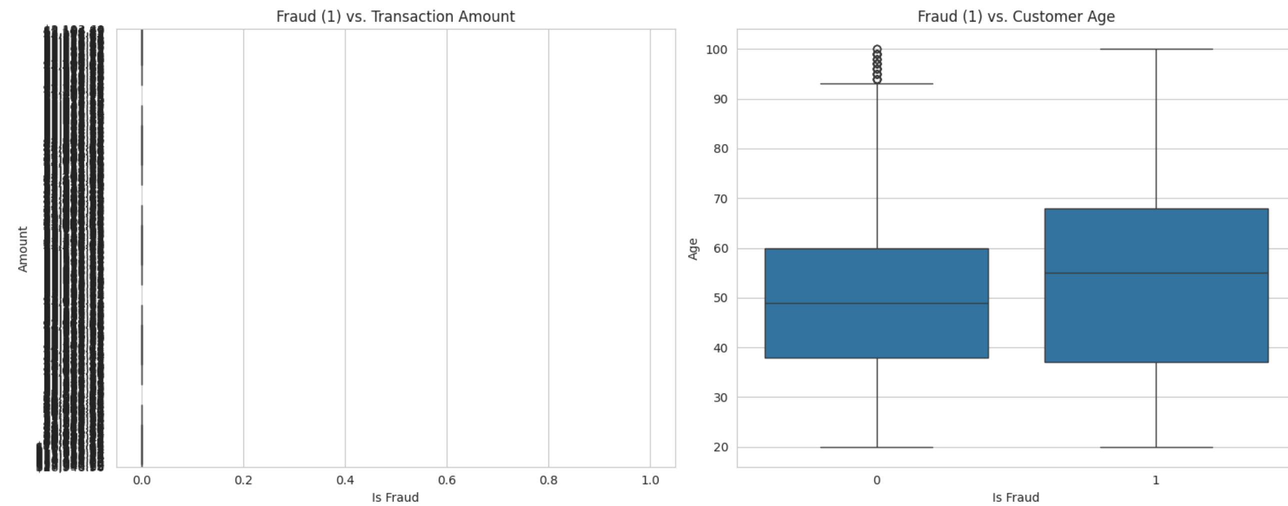
```
--- 1. Summary Statistics for Numeric Variables ---
                 trans_date_trans_time               Transaction_Date  \
count                             3544                           3544
mean   2019-08-13 09:40:19.672686336  2019-08-12 17:30:20.316027136
min              2019-01-01 02:54:00            2019-01-01 00:00:00
25%              2019-05-05 17:26:00            2019-05-05 00:00:00
50%              2019-08-17 22:58:30            2019-08-17 00:00:00
75%              2019-12-02 19:33:00            2019-12-02 00:00:00
max              2020-03-10 13:37:00            2020-03-10 00:00:00
std                              NaN                            NaN

                zip          lat         long      city_pop     unix_time  \
count   3543.000000  3543.000000  3543.000000  3.543000e+03  3.543000e+03
mean   48414.403613    38.230561   -90.004914  1.090972e+05  1.344840e+09
min     1257.000000    20.027100  -165.672300  2.300000e+01  1.325386e+09
25%    24986.000000    34.309100   -96.798000  1.078000e+03  1.336233e+09
50%    46366.000000    39.150500   -86.696600  4.533000e+03  1.345245e+09
75%    72476.000000    41.566600   -79.827450  3.537100e+04  1.354477e+09
max    99783.000000    66.693300   -67.950300  2.906700e+06  1.362923e+09
std    27313.939253     5.214455    14.085445  3.230181e+05  1.040366e+07

          merch_lat   merch_long     is_fraud  merch_zipcode          Age
count  3544.000000  3544.000000  3544.000000    3544.000000  3543.000000
mean     38.232212   -89.999580     0.281321   46658.448646    51.705052
min      19.057322  -165.658110     0.000000    1007.000000    20.000000
25%      34.437641   -96.861416     0.000000   28415.750000    38.000000
50%      39.215872   -86.964352     0.000000   48169.000000    50.000000
75%      41.666649   -79.800289     1.000000   63961.750000    63.000000
max      67.510267   -67.067585     1.000000   99361.000000   100.000000
std       5.242195    14.088653     0.449707   23984.569692    17.710362

--- 2. 'is_fraud' Value Counts (Target Variable Analysis) ---
is_fraud
0    2547
1     997
Name: count, dtype: int64

--- 3. Top 10 'category' Counts ---
category
shopping_net    1567
shopping_pos    1174
travel           380
misc_pos         216
misc_net         207
Name: count, dtype: int64
```

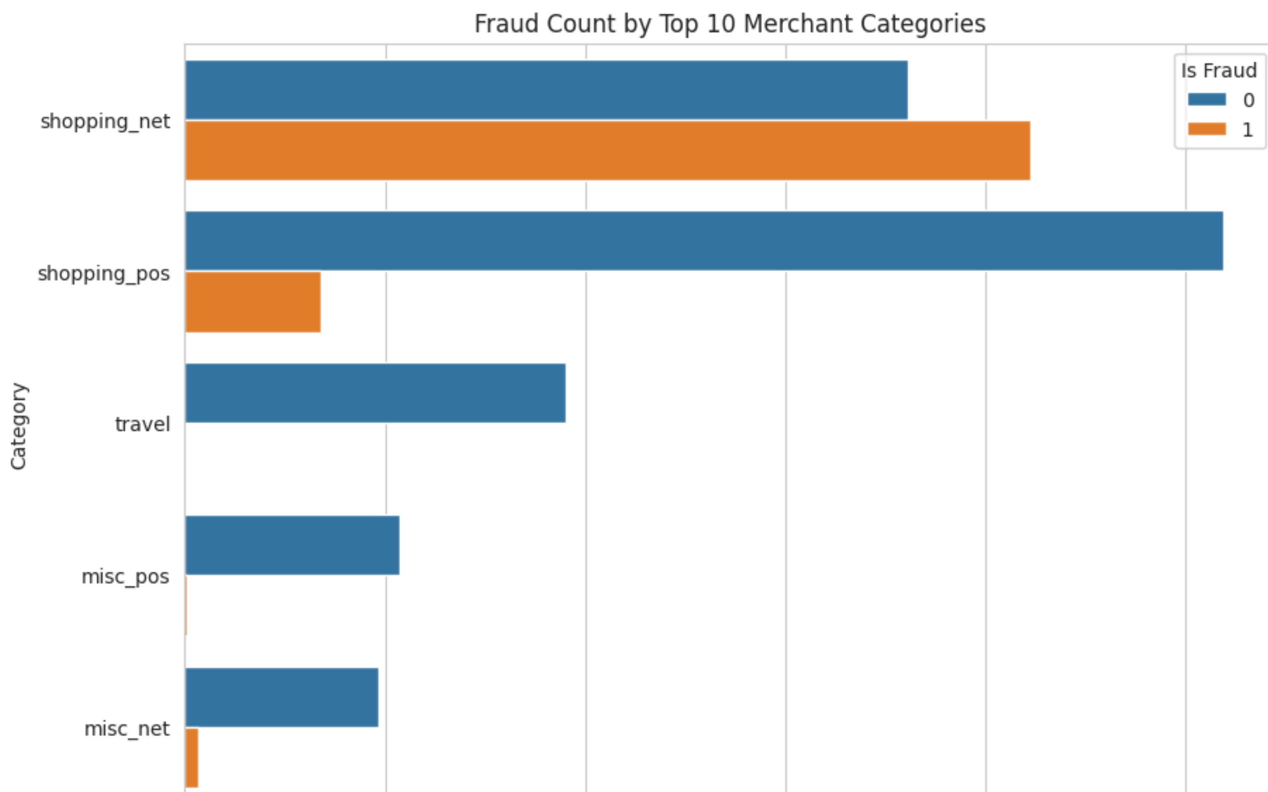
```
plt.figure(figsize=(18, 5))

# 1. Distribution of Transaction Amount ('amt')
plt.subplot(1, 3, 1)
sns.histplot(df['amt'], bins=50, kde=True)
plt.title('Distribution of Transaction Amount (amt)')
plt.xlabel('Amount')

# 2. Distribution of Customer Age ('Age')
plt.subplot(1, 3, 2)
sns.histplot(df['Age'], bins=30, kde=True)
plt.title('Distribution of Customer Age')
plt.xlabel('Age')

# 3. Target Variable Count
plt.subplot(1, 3, 3)
sns.countplot(x='is_fraud', data=df)
plt.title('Count of Fraud vs. Non-Fraud Transactions')
plt.xticks([0, 1], ['Not Fraud (0)', 'Fraud (1)'])
plt.xlabel('is_fraud')

plt.tight_layout()
plt.show()

# Write Observations for each plot here.
```



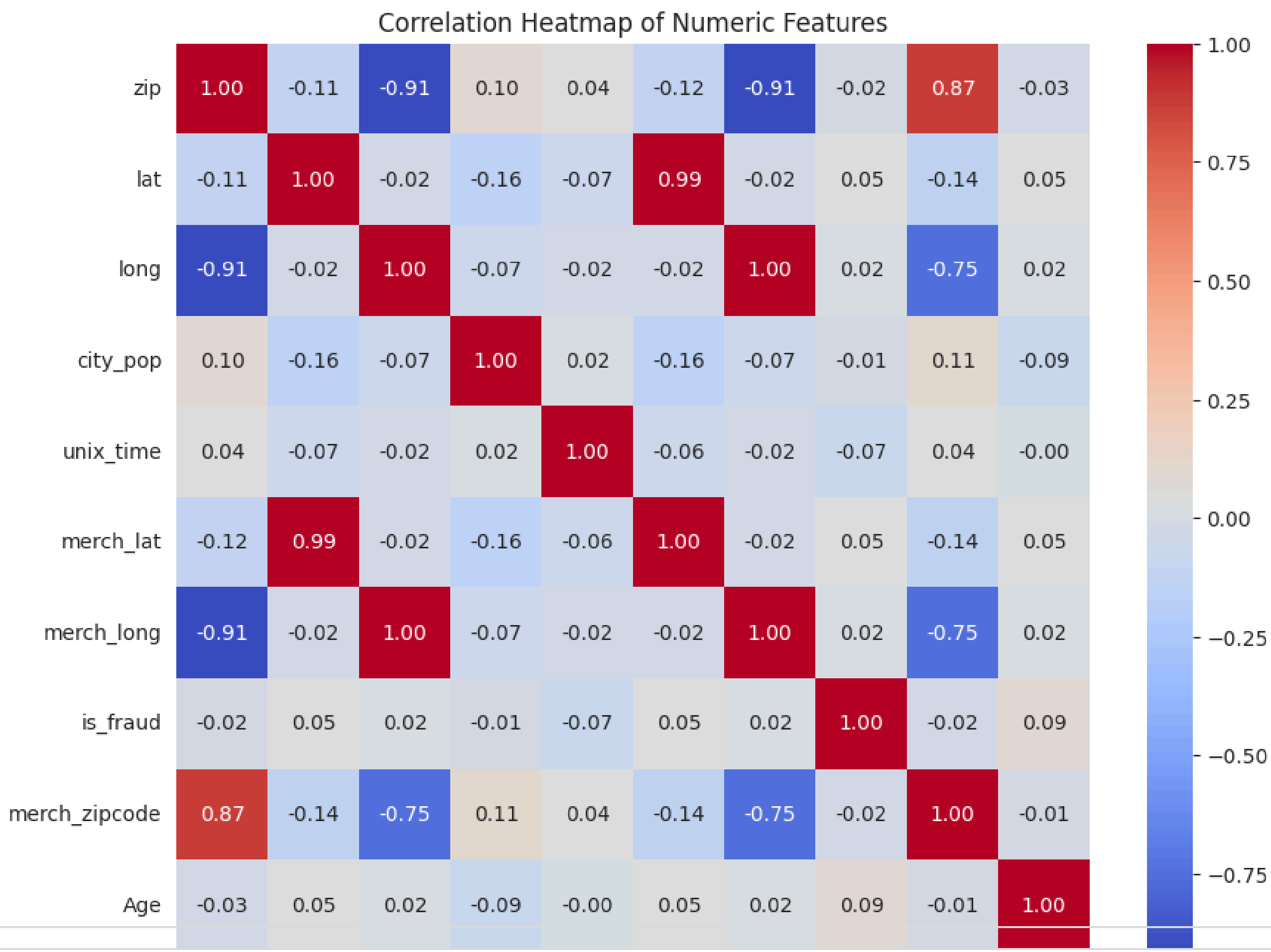

```
plt.figure(figsize=(15, 6))

# 1. Fraud vs. Transaction Amount (Boxplot)
plt.subplot(1, 2, 1)
sns.boxplot(x='is_fraud', y='amt', data=df)
plt.title('Fraud (1) vs. Transaction Amount')
plt.ylim(0, 500) # Limit Y-axis to focus on the majority of data, excluding extreme
plt.xlabel('Is Fraud')
plt.ylabel('Amount')

# 2. Fraud vs. Age (Boxplot)
plt.subplot(1, 2, 2)
sns.boxplot(x='is_fraud', y='Age', data=df)
plt.title('Fraud (1) vs. Customer Age')
plt.xlabel('Is Fraud')
```

```
plt.ylabel('Age')

plt.tight_layout()
plt.show()

# Write Observations: Compare the median and spread of 'amt' and 'Age' between the

# 3. Fraud Count by Category
plt.figure(figsize=(10, 7))
# Plot the top 10 categories to keep the visualization clear
top_10_categories = df['category'].value_counts().index[:10]
sns.countplot(y='category', hue='is_fraud', data=df[df['category'].isin(top_10_cate
              order=top_10_categories)
plt.title('Fraud Count by Top 10 Merchant Categories')
plt.ylabel('Category')
plt.legend(title='Is Fraud')
plt.show()

# Write Observations: Which categories have the highest absolute count of fraud tra
```




```
ıata for correlation calculation
_dtypes(include=np.number)
ımeric_df.corr()

. 8))
 correlation values on the heatmap
ı_matrix, annot=True, cmap='coolwarm', fmt=".2f")
 Heatmap of Numeric Features')


.ook for strong correlations (close to 1 or -1) between any two variables.
```




```
print("--- Summary of Key EDA Findings ---")

# 1. Data Quality and Preparation:
print(f"* Missing Data: All missing values in key columns ('merch_lat', 'merch_long

# 2. Target Variable (is_fraud):
print(f"* Imbalance Issue: The data is highly imbalanced, with only {fraud_counts[1

# 3. Key Trends and Patterns:
print("* Transaction Amount Distribution: The 'amt' distribution is heavily skewed
print("* Fraud vs. Amount: [State your observation from the boxplot, e.g., Fraud tr
```