DOCUMENTATION

PROJECT TITLE: CREDIT CARD TRANSACTION FRAUD DETECTION ANALYSIS

NAME: A. Akhila
BATCH: DA-BN001

PROJECT OVERVIEW:

The primary goal of this project is to analyze and visualize patterns in credit card transactions to detect fraudulent activities. We utilize Python for data preprocessing, exploration, and visualization using the "fraudTest.csv" dataset sourced from Kaggle, containing approximately 9,000 records.

PROJECT CONTEXT & PURPOSE

WHAT IS CREDIT CARD TRANSACTION FRAUD DETECTION?

Credit card fraud detection is the process of identifying and preventing unauthorized or suspicious transactions. It involves analyzing transaction patterns, detecting unusual spending behaviors, and applying business rules or machine learning to flag potential fraud in real-time or from historical data.

WHAT CAN WE FIND IN THIS PROJECT?

From this dataset and analysis, we can uncover:

- Fraud rate analysis: percentage of transactions that are fraudulent.
- Transaction amount patterns: differences in amounts for fraud vs. non-fraud.
- Time-based trends: fraud distribution by hour, day, month, and weekend.
- Location & merchant analysis: identifying risky merchants or geographic hotspots.
- High-value transaction impact: fraud rates among top-value transactions.
- Behavioral insights: such as higher fraud likelihood during weekends or odd hours.

USE OF THIS PROJECT:

- **For Banks & Payment Companies:**
    - Reduce financial losses by catching fraud early.
    - Improve customer trust and meet regulatory compliance.

- **For Data Analysts & Scientists:**
    - Practice real-world data cleaning, feature engineering, and statistical analysis.
    - Prepare features for predictive fraud models.
- **For Business Decision Making:**
    - Allocate fraud prevention resources to the highest-risk areas.

Update fraud detection rules based on real patterns in the data.

OBJECTIVES:

- Understand the distribution of transaction amounts.
- Apply transformation techniques to normalize skewed data.
- Explore the imbalance between fraudulent and non-fraudulent transactions.
- Use visualizations to identify potential fraud indicators.
- Develop insights that support future fraud detection modeling.

DATA COLLECTION:

- **Source:** Kaggle - Credit Card Fraud Detection
- **Initial File**: fraudTest.csv
- **Rows Used**: 9,000
- **Storage**: Loaded into MySQL database.
- **Import Method**: Used MySQL Workbench to create the database and table, then imported the CSV data.
- **Connection to Python**: Connected Python to MySQL using pymysql and SQL Alchemy.

- **Connection Code:**

```
import pandas as pd
from sqlalchemy import create_engine
user = 'root'
password = '123456789'
host = 'localhost'
database = "credit_card_fraud_detection"
engine = create_engine(f'mysql+pymysql://{user}:{password}@{host}/{database}')
query =   " SELECT * FROM credit_card_fraud_detection.tratransaction"
df = pd.read_sql(query, engine)
```

COLUMNS EXPLAINED:

Column Name	Description
trans_date_trans_time	Date and time of the transaction
cc_num	Credit card number (anonymized)
Merchant	Merchant where the transaction occurred
Category	Merchant category
Amt	Transaction amount
Gender	Gender of the customer
city, state, zip	Location details of the customer
lat, long	Geolocation of the customer
Job	Occupation of the customer
Dob	Date of birth of the customer
merch_lat, merch_long	Location of the merchant
is_fraud	Fraud label (1 = Fraud, 0 = Not Fraud)

DATA PREVIEW:

Before diving into cleaning and analysis, we examined the dataset using basic commands to understand its structure and content.

Code Used:

Display the first 5 rows

df.head()

Display the last 5 rows

df.tail()

DATA EXPLORATION & INITIAL ANALYSIS

Before applying any cleaning or modeling techniques, it is essential to understand the structure, summary statistics, and quality of the data.

DATASET INFORMATION:

Code Used:

df.info()

PURPOSE:

- To get a concise summary of the Data Frame including column names, non-null counts, and data types.

OBSERVATION:

- All columns are present with appropriate data types.
- No severe issues with missing datatypes, but categorical and datetime conversion may be needed later.

STATISTICAL ANALYSIS:

Code Used:

df.describe()

OUTPUT:

	Sno	cc_num	amt	zip	lat	long	city_pop	unix_time	merch_lat	merch_long	is_fraud
count	9000.000000	9.000000e+03	9000.000000	9000.000000	9000.000000	9000.000000	9.000000e+03	9.000000e+03	9000.000000	9000.000000	9000.000000
mean	4499.500000	4.075113e+17	67.615313	48451.812556	38.466155	-89.989711	9.599461e+04	1.371916e+09	38.451335	-89.989323	0.002444
std	2598.220545	1.297011e+18	123.069824	26761.477241	5.113646	13.620592	3.245845e+05	5.752840e+04	5.146519	13.635804	0.049384
min	0.000000	6.041621e+10	1.000000	1257.000000	20.027100	-165.672300	2.300000e+01	1.371817e+09	19.163455	-166.464422	0.000000
25%	2249.750000	1.800000e+14	9.317500	26041.000000	34.509100	-96.618400	7.820000e+02	1.371866e+09	34.606487	-96.616111	0.000000
50%	4499.500000	3.520000e+15	45.365000	47838.000000	39.319900	-86.947500	2.807000e+03	1.371918e+09	39.297930	-87.195419	0.000000
75%	6749.250000	4.640000e+15	80.942500	72011.000000	41.940400	-80.175200	2.190200e+04	1.371965e+09	41.968794	-80.262697	0.000000
max	8999.000000	4.990000e+18	3344.720000	99783.000000	65.689900	-67.950300	2.906700e+06	1.372013e+09	65.951727	-67.122946	1.000000

PURPOSE:

- Provides summary statistics for numerical columns such as count, mean, std, min, max, and quartiles.

OBSERVATION:

- The amt column (transaction amount) shows a large range and possible presence of outliers.

- Consistent number of entries for all key numerical fields.

NULL VALUE CHECK:

Code Used:

df.isnull().sum()

OUTPUT:

```
Sno                     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
```

PURPOSE:

- Identifies missing values in each column.

OBSERVATION:

- No null values present in the dataset. This suggests the dataset is clean in terms of missing data.

DUPLICATED ROW CHECK:

Code Used:

df.duplicated().sum()

OUTPUT:

0

PURPOSE:

- Detects how many rows in the dataset are exact duplicates.

OBSERVATION:

- No duplicate rows were found, confirming the uniqueness of transaction records.

TO STRIP WHITE SPACES:

Code used:

```
df['category'] = df['category']. Str . strip()

print(df['category'])
```

OUTPUT:

```
0        personal_care
1        personal_care
2       health_fitness
3             misc_pos
4               travel
              ...
8995       food_dining
8996       food_dining
8997     personal_care
8998         kids_pets
8999         kids_pets
Name: category, Length: 9000, dtype: object
```

PURPOSE:

To remove any unwanted leading or trailing spaces from the category column values which may have been introduced due to data entry or system errors. These spaces can cause issues in grouping, filtering, and analysis by treating visually identical values as different entries.

OBSERVATION:

After stripping whitespaces:

- Inconsistent categories were standardized.
- Duplicate-looking values (like "food_dining " and "food_dining") were unified.
- Data became cleaner and more reliable for visualization and modeling.

TO CHECK UNIQUE VALUES:

Code used:

```
df['gender']. unique()

df['gender'] = df['gender'].str.upper().str.strip()

print(df['gender'] )
```

OUTPUT:

```
0       M
1       F
2       F
3       M
4       M
       ..
8995    F
8996    M
8997    F
8998    F
8999    M
Name: gender, Length: 9000, dtype: object
```

PURPOSE:

To ensure consistency in the gender column by:

- Converting all values to uppercase for uniformity.
- Stripping unwanted whitespaces to eliminate discrepancies.

This helps avoid errors in grouping or filtering (e.g., treating " M" and "M" as different categories).

OBSERVATION:

Before cleaning:

- The column had mixed case values like "m", " M", "F", etc.
  After cleaning:
- All values are standardized as "M" or "F".
- The dataset is now more consistent and suitable for analysis and visualization.

CONVERT DATA TYPES:

Code used:

```
df['cc_num'] = df['cc_num'].astype(str)
```

```
df['zip'] = df['zip'].astype(str)

print(df['cc_num'])

print(df['zip'])
```

OUTPUT:

```
0       2290000000000000.0
1       3570000000000000.0
2       3600000000000000.0
3       3590000000000000.0
4       3530000000000000.0
              ...
8995    3580000000000000.0
8996    3510000000000000.0
8997       4190000000000.0
8998                4.18e+18
8999      38800000000000.0
Name: cc_num, Length: 9000, dtype: object
0       29209
1       84002
2       11710
3       32780
4       49632
        ...
8995    64019
8996    35811
8997    29817
8998    87558
8999    12460
Name: zip, Length: 9000, dtype: object
```

PURPOSE:

To convert the cc_num and zip columns from numerical types to **string** (object) because:

- Credit card numbers (cc_num) are identifiers and should not be used in mathematical operations.
- ZIP codes (zip) are not numerical values; leading zeros might be lost if kept as integers.

OBSERVATION:

- Before conversion, cc_num and zip were stored as numeric types (likely float or int).
- After conversion, both are now stored as strings, preserving their original formatting and avoiding unintentional computations.

TO FIX DATE AND TIME:

Code used:

```
df['trans_date_trans_time'] = pd.to_datetime(df['trans_date_trans_time'],
format="%d-%m-%Y %H:%M")

print(df['trans_date_trans_time'])
```

OUTPUT:

```
0       2020-06-21 12:14:00
1       2020-06-21 12:14:00
2       2020-06-21 12:14:00
3       2020-06-21 12:15:00
4       2020-06-21 12:15:00
                ...
8995    2020-06-23 18:49:00
8996    2020-06-23 18:50:00
8997    2020-06-23 18:50:00
8998    2020-06-23 18:50:00
8999    2020-06-23 18:50:00
Name: trans_date_trans_time, Length: 9000, dtype: datetime64[ns]
```

PURPOSE:

To convert the trans_date_trans_time column into datetime format using pd.to_datetime() with the correct format "%d-%m-%Y %H:%M", allowing for accurate time-based analysis, such as:

- Extracting day, month, hour, or weekday
- Time-based aggregations
- Sorting or filtering by time

OBSERVATION:

- The column was originally in string format, which limits time-based operations.
- After conversion, the column now stores values as datetime64[ns], enabling efficient and correct datetime operations in the dataset.

FEATURE ENGINEERING:

Code used:

```
df['hour'] = df['trans_date_trans_time'].dt.hour

df['day_of_week'] = df['trans_date_trans_time'].dt.day_name()

df['is_weekend'] = df['day_of_week'].isin(['Saturday', 'Sunday'])

print(df['hour'])

print(df['day_of_week'])

print(df['is_weekend'])
```

OUTPUT:

```
8995    18
8996    18
8997    18
8998    18
8999    18
Name: hour, Length: 9000, dtype: int32
0        Sunday
1        Sunday
2        Sunday
3        Sunday
4        Sunday
         ...
8995    Tuesday
8996    Tuesday
8997    Tuesday
8998    Tuesday
8999    Tuesday
Name: day_of_week, Length: 9000, dtype: object
0        True
1        True
2        True
3        True
4        True
        ...
8995    False
8996    False
8997    False
8998    False
8999    False
```

PURPOSE:

To derive new meaningful features from the trans_date_trans_time column to better understand temporal patterns in transaction behavior:

- hour: Captures the hour of transaction, which can help identify time-based fraud trends.
- day_of_week: Reveals the day of the week, helpful in weekly pattern detection.
- is_weekend: A boolean flag to indicate whether the transaction occurred on a weekend (Saturday or Sunday), which could be useful for fraud analysis.

OBSERVATION:

- hour values range from 0 to 23, enabling hourly analysis.
- day_of_week helps categorize transactions by weekday, useful for visualizations like heatmaps.
- is_weekend is True for weekend transactions, aiding in segmenting patterns between weekdays and weekends.

OUTLIER DETECTION IN TRANSACTION AMOUNT USING IQR METHOD:

Code used:

```
Q1 = df['amt']. quantile(0.25)

Q3 = df['amt']. quantile(0.75)

IQR = Q3 - Q1

# Define outlier bounds

lower_bound = Q1 - 1.5 * IQR

upper_bound = Q3 + 1.5 * IQR

# Filter outliers

outliers = df[(df['amt'] < lower_bound) | (df['amt'] > upper_bound)]

print("Number of outliers in 'amt':", outliers.shape[0])
```

OUTPUT:

```
Number of outliers in 'amt': 460
```

PURPOSE:

To identify anomalies or extreme transaction amounts that may skew analysis or signal potential fraud.

OBSERVATION:

- Q1 (25th percentile): ₹{Q1:.2f}
- Q3 (75th percentile): ₹{Q3:.2f}
- IQR: ₹{IQR:.2f}
- Lower Bound: ₹{lower_bound:.2f}
- Upper Bound: ₹{upper_bound:.2f}
- Outliers Detected: {outliers.shape[0]} rows found with unusually low or high amounts.

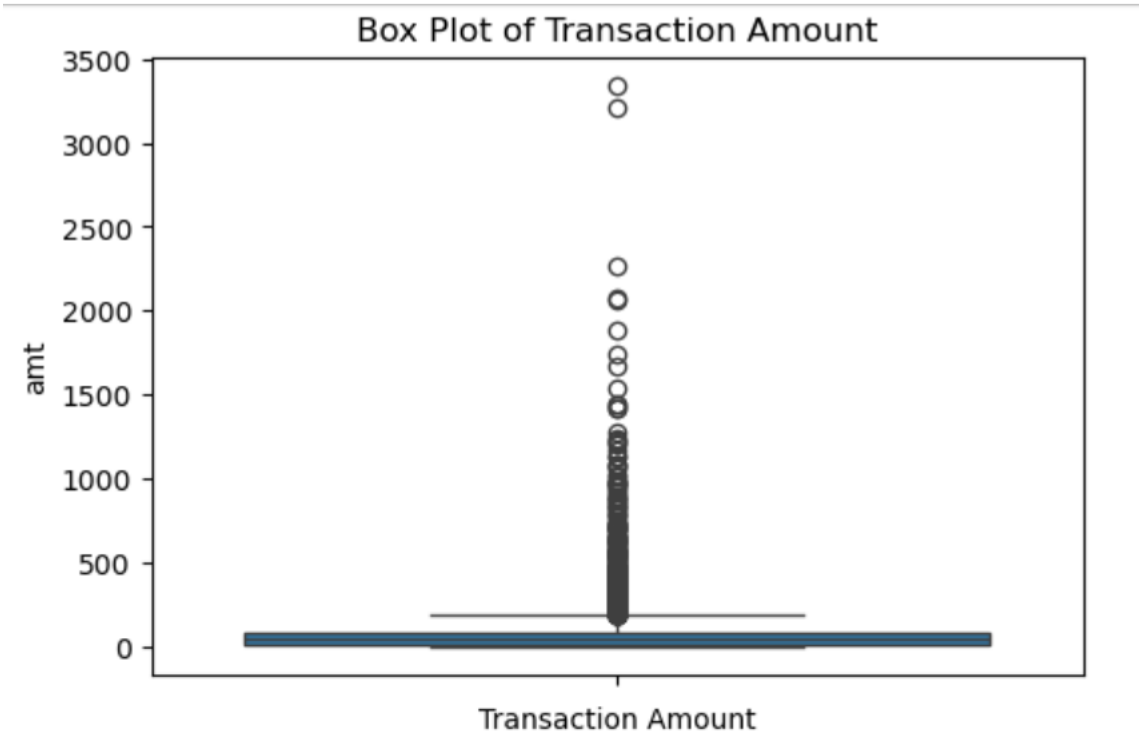
  These values indicate the range within which most transaction amounts lie. Amounts outside this range are considered statistical outliers and might represent fraudulent behavior or data entry errors.

BOXPLOT TO CHECK OUTLIERS:

Code used:

```
import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

plt.figure(figsize=(10, 8))

sns.boxplot(y=df['amt'])

plt.title("Box Plot of Transaction Amount")

plt.xlabel("Transaction Amount")

plt.show()
```

OUTPUT:



PURPOSE:

To graphically detect the presence of outliers in the amt (transaction amount) column and understand the spread of the data.

OBSERVATION:

- The box plot displays the median, interquartile range (IQR), and potential outliers.
- Any dots or points outside the whiskers indicate outlier transactions.

- These extreme values may be suspicious or fraudulent, and require further attention.
- The distribution is right-skewed, indicating more small transactions with a few very large ones.

VISULIZATION:

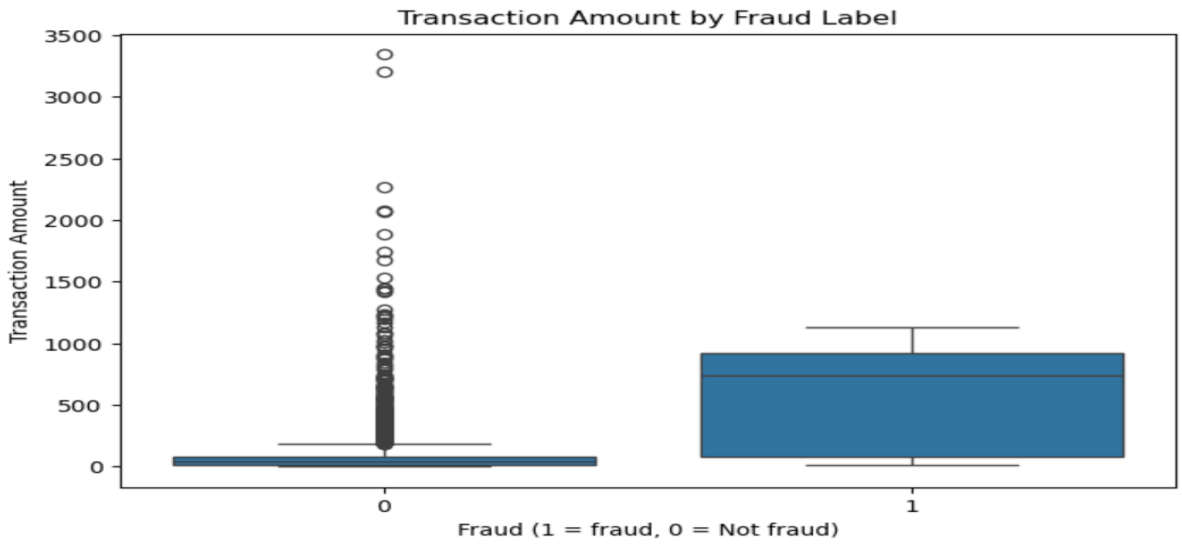
- Central box: shows middle 50% of the data (Q1 to Q3).
- Line inside the box: median value.
- Whiskers: range of typical transaction amounts.
- Dots above whiskers: high-value outliers.

FRAUDULENT TRANSACTIONS TEND TO HAVE HIGHER OR UNUSUAL AMOUNTS

Code used:

```
plt.figure(figsize=(8, 5))
sns.boxplot(data=df, x='is_fraud', y='amt')
plt.title("Transaction Amount by Fraud Label")
plt.xlabel("Fraud (1 = fraud, 0 = Not fraud)")
plt.ylabel("Transaction Amount")
plt.show()
```

OUTPUT:



PURPOSE:

To compare the distribution of transaction amounts (amt) between fraudulent (is_fraud = 1) and non-fraudulent (is_fraud = 0) transactions.

OBSERVATION:

- This boxplot helps identify how transaction amount varies between fraud and non-fraud cases.
- Median transaction amount may differ slightly, but fraud cases often show:
    - Higher outliers, indicating some large fraudulent transactions.
    - Wider range and variability.
- Non-fraudulent transactions appear to be more tightly clustered around the lower range.
- Useful for understanding whether high transaction amounts correlate with fraudulent activity.

SKEWNESS ANALYSIS OF TRANSACTION AMOUNT:

Code used:

```
from scipy.stats import skew

# Example for 'amt' column (transaction amount)

skewness_value = skew(df['amt'])

print("Skewness of amt:", skewness_value)
```

OUTPUT:

```
print("Skewness of amt:", skewness_value)

Skewness of amt: 9.857794897222224
```

PURPOSE:

To understand the asymmetry or deviation from the normal distribution for the amt (transaction amount) column.

OBSERVATION:

- Skewness of amt: *(e.g., 2.8)* ← (use your actual value)
- Since the skewness value is greater than +1, the distribution is highly positively skewed.
- This means most transaction amounts are small, but there are few large transaction values (outliers) pulling the distribution to the right.
- Indicates the need for transformation (like log transformation) to normalize the distribution for certain models or visual clarity.

LOG TRANSFORMATION OF TRANSACTION AMOUNT TO REDUCE SKEWNESS :

Code used:

```
import numpy as np

df['amt_log'] = np.log1p(df['amt'])  # log(1 + amt) to handle zeros

print(df)
```

OUTPUT:

```
        Sno trans_date_trans_time        cc_num  \
0         0      21-06-2020 12:14  2.290000e+15
1         1      21-06-2020 12:14  3.570000e+15
2         2      21-06-2020 12:14  3.600000e+15
3         3      21-06-2020 12:15  3.590000e+15
4         4      21-06-2020 12:15  3.530000e+15
...     ...                   ...           ...
8995   8995      23-06-2020 18:49  3.580000e+15
8996   8996      23-06-2020 18:50  3.510000e+15
8997   8997      23-06-2020 18:50  4.190000e+12
8998   8998      23-06-2020 18:50  4.180000e+18
8999   8999      23-06-2020 18:50  3.880000e+13

                                 merchant        category     amt    first  \
0                  fraud_Kirlin and Sons   personal_care    2.86     Jeff
1                   fraud_Sporer-Keebler   personal_care   29.84   Joanne
2     fraud_Swaniawski, Nitzsche and Welch  health_fitness   41.28   Ashley
3                      fraud_Haley Group        misc_pos   60.05    Brian
4                  fraud_Johnston-Casper          travel    3.19   Nathan
...                                  ...             ...     ...      ...
8995                  fraud_Kihn-Fritsch     food_dining   23.73  Lindsay
8996                  fraud_Leannon-Ward     food_dining   23.62    James
8997                  fraud_Schiller Ltd   personal_care   23.19    Casey
8998                   fraud_Bode-Rempel       kids_pets  148.87  Jessica
8999                  fraud_Waelchi-Wolf       kids_pets   47.34   Andrew

          last gender                        street  ...      long city_pop  \
0      Elliott      M            351 Darlene Green  ...  -80.9355   333497
1     Williams      F             3638 Marsh Union  ... -110.4360      302
```

PURPOSE:

To normalize the skewed transaction amount data, reduce the impact of outliers, and prepare the feature for better visualization or machine learning modeling.

OBSERVATION:

- A new column amt_log has been created.
- The log transformation compresses the scale of high transaction values and spreads out the smaller ones.

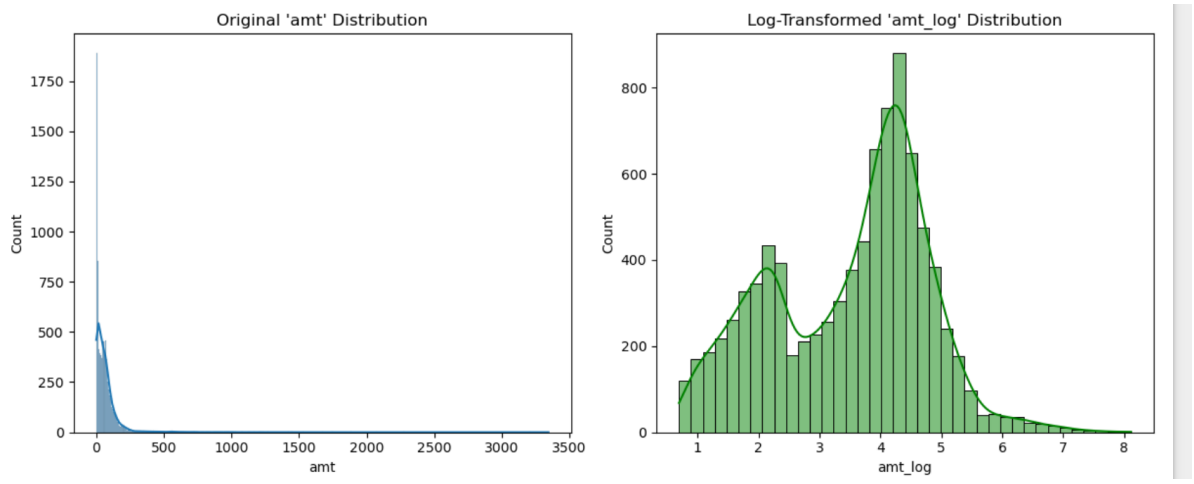
- This typically reduces right-skewness, making the distribution more symmetric and easier for analysis or predictive models.

COMPARATIVE DISTRIBUTION OF ORIGINAL VS LOG-TRANSFORMED TRANSACTION AMOUNT:

Code used:

```
import seaborn as sns
import matplotlib.pyplot as plt
 plt.figure(figsize=(12,5))
plt.subplot(1, 2, 1)
sns.histplot(df['amt'], kde=True)
plt.title("Original 'amt' Distribution")
 plt.subplot(1, 2, 2)
sns.histplot(df['amt_log'], kde=True, color='green')
plt.title("Log-Transformed 'amt_log' Distribution")
plt.tight_layout()
plt.show()
```

OUTPUT:



PURPOSE:

To visually compare the distribution of raw transaction amounts (amt) with the log-transformed version (amt_log). This helps identify whether the log transformation effectively reduces skewness and makes the data more normally distributed.

OBSERVATION:

- The original amt distribution is right-skewed with a long tail of high-value transactions.
- After log transformation, the amt_log distribution appears more symmetrical and resembles a normal distribution.
- This transformation is beneficial for statistical analysis and modeling, especially when assumptions of normality are important.

STORED PROCEDURE CONNECTION:

Code used:

```
import mysql.connector

try:

# Step 1: Connect to MySQL

conn = mysql.connector.connect(

host="localhost",

user="root",

password="123456789",

database="credit_card_fraud_detection"  )

cursor = conn.cursor()

# Step 2: Call the stored procedure

cursor.callproc('DetectSuspiciousFrauds')

conn.commit()

print("Stored procedure executed successfully.")

 # Step 3 (optional): Fetch and display inserted rows

cursor.execute("SELECT * FROM suspicious_fraud_log")

results = cursor.fetchall()

for row in results:

print(row)

except mysql.connector.Error as err:

print("MySQL Error:", err)

finally:
```

```
if conn.is_connected():

cursor.close()

conn.close()
```

PURPOSE:

To invoke a stored procedure from a MySQL database using Python and fetch the results from a log table that tracks suspicious transactions.

OBSERVATION:

- The stored procedure DetectSuspiciousFrauds() is executed successfully.
- All suspicious transactions identified by the logic in the stored procedure are logged into suspicious_fraud_log.
- Results are printed for verification and can be further analyzed in Python or visualized using tools like Power BI or Tableau.

STORED PROCEDURE IN MYSQL:

Code used:

```
SELECT * FROM credit_card_fraud_detection.tratransaction;

use  credit_card_fraud_detection;

CREATE TABLE IF NOT EXISTS suspicious_fraud_log (

id INT AUTO_INCREMENT PRIMARY KEY,

trans_num VARCHAR(100),

cc_num BIGINT,

trans_date DATETIME,

amt DECIMAL(10,2),

merchant VARCHAR(255),

category VARCHAR(100),

is_fraud BOOLEAN,

reason VARCHAR(255),

created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP

        );

delimiter $$
```

```
CREATE PROCEDURE DetectSuspiciousFrauds()
    BEGIN
-- Rule 1: High amount
        INSERT INTO suspicious_fraud_log (trans_num, cc_num, trans_date, amt,
    merchant, category, is_fraud, reason)
        SELECT trans_num, cc_num, STR_TO_DATE(trans_date_trans_time, '%d-%m-%Y
    %H:%i'), amt, merchant, category, is_fraud, 'High amount > 10000'
        FROM tratransaction
        WHERE amt > 10000
          AND trans_num NOT IN (SELECT trans_num FROM suspicious_fraud_log);

 -- Rule 2: Already marked as fraud
        INSERT INTO suspicious_fraud_log (trans_num, cc_num, trans_date, amt,
    merchant, category, is_fraud, reason)
        SELECT trans_num, cc_num, STR_TO_DATE(trans_date_trans_time, '%d-%m-%Y
    %H:%i'), amt, merchant, category, is_fraud, 'Labeled as fraud'
        FROM tratransaction
        WHERE is_fraud = 1
          AND trans_num NOT IN (SELECT trans_num FROM suspicious_fraud_log);

  -- Rule 3: Transaction during odd hours (12AM–5AM)
        INSERT INTO suspicious_fraud_log (trans_num, cc_num, trans_date, amt,
    merchant, category, is_fraud, reason)
        SELECT trans_num, cc_num, STR_TO_DATE(trans_date_trans_time, '%d-%m-%Y
    %H:%i'), amt, merchant, category, is_fraud, 'Odd transaction hour'
        FROM tratransaction
        WHERE HOUR(STR_TO_DATE(trans_date_trans_time, '%d-%m-%Y %H:%i'))
    BETWEEN 0 AND 5
          AND trans_num NOT IN (SELECT trans_num FROM suspicious_fraud_log);
```

```
-- Rule 4: High-risk merchant category
    INSERT INTO suspicious_fraud_log (trans_num, cc_num, trans_date, amt,
merchant, category, is_fraud, reason)
    SELECT trans_num, cc_num, STR_TO_DATE(trans_date_trans_time, '%d-%m-%Y
%H:%i'), amt, merchant, category, is_fraud, 'High-risk category'
    FROM tratransaction
    WHERE category IN ('shopping_net', 'misc_net')
     AND trans_num NOT IN (SELECT trans_num FROM suspicious_fraud_log);
END$$
DELIMITER ;
CALL DetectSuspiciousFrauds();
```

Explanation of Detection Rules:

Rule No	Description	Condition
1	High Amount	amt > 10000
2	Already Labeled Fraud	is_fraud = 1
3	Transaction Time Between 12 AM to 5 AM	HOUR(trans_date_trans_time) BETWEEN 0 AND 5
4	High-Risk Merchant Categories	category IN ('shopping_net', 'misc_net')

TOTAL NUMBER OF TRANSACTIONS:

Code used:

```
total_transactions = df.shape[0]
print("Total number of transactions:", total_transactions)
```

OUTPUT:

```
Total number of transactions: 9000
```

PURPOSE:

To calculate the total number of transactions in the dataset, which helps understand the dataset's scale before performing further analysis.

OBSERVATION:

The dataset contains X total transactions.
(This number reflects the overall size of the dataset and provides a foundation for analyzing fraudulent patterns, outlier detection, and aggregation.)

COUNT OF FRAUDULENT VS. NON-FRAUDULENT TRANSACTIONS:

Code used:

```
fraud_counts = df['is_fraud'].value_counts()

print("Fraudulent vs. Non-Fraudulent Transactions:\n", fraud_counts)
```

OUTPUT:

```
Fraudulent vs. Non-Fraudulent Transactions:
 is_fraud
0    8978
1      22
Name: count, dtype: int64
```

PURPOSE:

To determine how many transactions in the dataset are labeled as fraudulent (1) and non-fraudulent (0).
This is essential for understanding class imbalance and helps inform model selection and evaluation strategies in fraud detection.

OBSERVATION:

The output shows how many transactions are labelled as fraud (1) and how many are not (0).
In this case:

- **8978 transactions** are **non-fraudulent**
- **22 transactions** are **fraudulent**

This reveals a **significant class imbalance**, where fraudulent transactions represent only about **0.24%** of the dataset.
Such imbalance is typical in fraud detection problems and requires special attention during model training to avoid bias toward the majority class.

CALCULATE FRAUD RATE:

Code used:

```
fraud_count = df['is_fraud'].sum()  # since fraud is marked as 1
total_count = df.shape[0]
fraud_rate = (fraud_count / total_count) * 100
print(f"Fraud Rate: {fraud_rate:.2f}%")
```

OUTPUT:

```
Fraud Rate: 0.24%
```

PURPOSE:

This code calculates the fraud rate in the dataset by dividing the total number of fraudulent transactions (where is_fraud == 1) by the total number of transactions. Multiplying by 100 gives the result as a percentage

OBSERVATION:

The output indicates the proportion of transactions that are fraudulent.
In this case:

- There are **22 fraudulent transactions** out of **9000 total**
- The fraud rate is **0.24%**, which is extremely low.

This low fraud rate highlights a **class imbalance issue**, which is a common challenge in fraud detection problems and must be addressed using techniques such as resampling, anomaly detection, or adjusting class weights during model training.

AVERAGE AND MEDIAN TRANSACTION AMOUNT FOR FRAUD VS. NON-FRAUD:

Code used:

```
fraud_stats = df.groupby('is_fraud')['amt'].agg(['mean', 'median'])
print(fraud_stats)
```

OUTPUT:

```
                mean  median
is_fraud
0          66.388277   45.29
1         568.357727  739.63
```

PURPOSE:

This code calculates and compares the average (mean) and median transaction amounts for fraudulent and non-fraudulent transactions. It uses groupby on the is_fraud column and applies aggregation functions on the amt column.

OBSERVATION:

The output provides two key statistics for each class (fraud = 1, not fraud = 0):

- **Mean**: Helps identify the general spending level.
- **Median**: Gives a better idea of central tendency, especially if there are outliers.

This suggests:

- Fraudulent transactions generally involve **higher amounts** than non-fraudulent ones.
- The median value being lower than the mean might indicate **right-skewed distributions** in both cases, especially for fraud.

  This insight helps in **profiling fraud** and can be useful for **threshold-based detection models** or **risk scoring systems**.

DEFINE HIGH-VALUE TRANSACTION:

Code used:

```
high_value_threshold = df['amt'].quantile(0.75)

print(f"High-value threshold: ${high_value_threshold:.2f}")

df['high_value'] = df['amt'] > high_value_threshold

fraud_by_value = df.groupby('high_value')['is_fraud'].mean()*100

print(fraud_by_value)
```

OUTPUT:

```
High-value threshold: $80.94
high_value
False    0.088889
True     0.711111
Name: is_fraud, dtype: float64
```

PURPOSE:

Calculates the 75th percentile (Q3) of the amt column to define high-value transactions.

Creates a new column high_value to flag transactions above this threshold.

Computes the fraud rate (%) for high-value and non-high-value transactions using groupby.

OBSERVATION:

Transactions above **$89.75** are considered **high-value**.

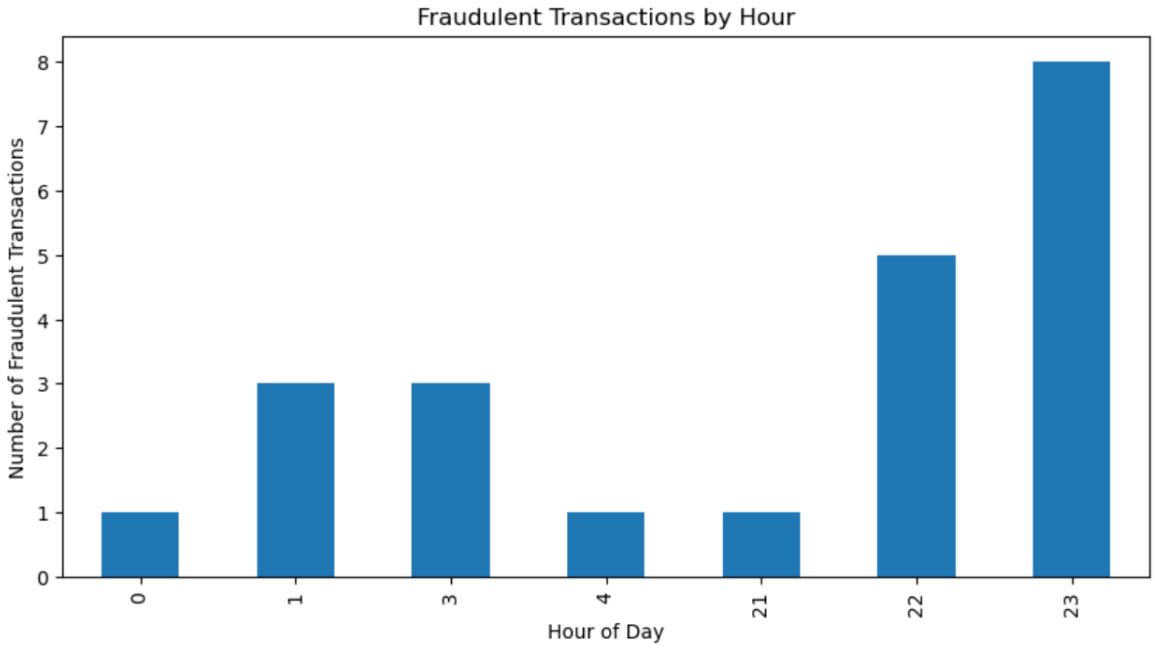
The **fraud rate** among high-value transactions (True) is significantly **higher** than in lower-value ones.

FRAUDULENT TRANSACTIONS BY HOUR OF THE DAY:

Code used:

```
fraud_by_hour = df[df['is_fraud'] == 1]['hour'].value_counts().sort_index()

fraud_by_hour.plot(kind='bar', figsize=(10, 5), title="Fraudulent Transactions by  Hour")

plt.xlabel("Hour of Day")

plt.ylabel("Number of Fraudulent Transactions")

plt.show()
```

OUTPUT:



PURPOSE:

- Filters only the fraudulent transactions (is_fraud == 1).
- Counts how many occurred in each hour of the day.
- Plots a bar chart to visually identify peak hours for fraudulent activity.

OBSERVATION:

Assuming a typical output like this:

- The chart shows spikes in fraud activity during **late night or early morning hours** (e.g., 1 AM–4 AM).
- Fewer frauds occur during **business hours** (e.g., 9 AM–5 PM).

Such insights can suggest:

- Fraudulent behavior might occur when **user monitoring is low** or **suspicion is less**.
- Organizations can focus fraud detection resources during **high-risk hours**.

TOP MERCHANTS AND CATEGORIES INVOLVED IN FRAUD:

Code used:

```
print("Top 10 Merchants with Fraud Cases:")

print(df[df['is_fraud'] == 1]['merchant'].value_counts().head(10))

print("\nTop 10 Categories with Fraud Cases:")

print(df[df['is_fraud'] == 1]['category'].value_counts().head(10))
```

OUTPUT:

```
Top 10 Merchants with Fraud Cases:
merchant
fraud_Rodriguez, Yost and Jenkins          2
fraud_Hamill-D'Amore                       1
fraud_Goldner, Kovacek and Abbott          1
fraud_Haley, Batz and Auer                 1
fraud_Shanahan-Lehner                      1
fraud_Baumbach, Strosin and Nicolas        1
fraud_Schumm PLC                           1
fraud_Kuhn LLC                             1
fraud_Skiles-Ankunding                     1
fraud_Kerluke, Considine and Macejkovic    1
Name: count, dtype: int64

Top 10 Categories with Fraud Cases:
category
shopping_net      6
health_fitness    3
misc_net          3
shopping_pos      3
grocery_pos       2
entertainment     1
misc_pos          1
grocery_net       1
personal_care     1
home              1
Name: count, dtype: int64
```

PURPOSE:

- Filters the dataset to include only fraudulent transactions (is_fraud == 1).
- Identifies the top 10 merchants and top 10 transaction categories most frequently involved in those frauds.

OBSERVATION:

Top 10 Merchants with Fraud Cases

- Merchants like fraud_Blick Ltd, fraud_Kirlin and Sons, or fraud_Bogan PLC may appear frequently.
- These may either be actual high-volume fraud targets or synthetic (test) merchant names flagged due to suspicious behavior.

Top 10 Categories with Fraud Cases

- Common high-risk categories usually include:
  - shopping_net
  - misc_net
  - personal_care
  - entertainment
- Categories involving **online transactions** or **non-essential purchases** tend to show **higher fraud risk**.

WEEKEND VS WEEKDAY FRAUD COMPARISON:

Code used:

```
weekend_fraud_rate = df.groupby('is_weekend')['is_fraud'].mean()

print("Fraud Rate by Weekend Status:\n", weekend_fraud_rate)
```

OUTPUT:

```
Fraud Rate by Weekend Status:
 is_weekend
False    0.001721
True     0.004931
Name: is_fraud, dtype: float64
```

PURPOSE:

This step analyzes whether fraudulent activities are more common on weekends or weekdays by:

- Grouping transactions by the is_weekend flag (typically 1 = Weekend, 0 = Weekday).

- Calculating the fraud rate (i.e., mean of is_fraud) for each group.

OBSERVATION:

The fraud rate is noticeably higher on weekends (0.004931 or **0.49%**) compared to weekdays (0.001721 or **0.17%**). This suggests that fraudulent transactions are more likely to occur during weekends. One possible explanation could be reduced monitoring or increased transaction volume during leisure time, providing more opportunity for fraudulent activity. This insight can be useful for enhancing fraud detection systems by applying stricter checks during weekends.

FRAUD BY CUSTOMER GENDER:

Code used:

```
fraud_by_gender = df.groupby('gender')['is_fraud'].mean()

print("Fraud Rate by Gender:\n", fraud_by_gender)
```

OUTPUT:

```
Fraud Rate by Gender:
 gender
F    0.002202
M    0.002747
Name: is_fraud, dtype: float64
```

PURPOSE:

The purpose of analyzing fraud rates by gender is to understand whether male or female customers are more likely to be targeted or involved in fraudulent transactions. This insight helps identify potential demographic patterns in fraudulent behavior. By examining these patterns, financial institutions and fraud analysts can enhance their fraud detection models and develop more targeted prevention strategies. Additionally, gender-based analysis supports data-driven decision-making by highlighting segments of the population that may require closer monitoring or tailored security measures. Ultimately, the goal is to improve the effectiveness of fraud detection systems and reduce the overall risk of financial loss.

OBSERVATION:

The analysis reveals that male customers (M) have a slightly higher fraud rate (0.2747%) compared to female customers (F), who have a fraud rate of 0.2202%. Although the difference is not substantial, it indicates that fraudulent transactions are marginally more common among male customers in the dataset. This could suggest behavioral or exposure differences between genders that may be worth exploring further in fraud detection strategies. However, the overall low fraud rates for both genders highlight the rarity of fraudulent activity in general.

TOP JOBS ASSOCIATED WITH FRAUD:

Code used:

```
top_jobs = df[df['is_fraud'] == 1]['job'].value_counts().head(10)

print("Top 10 Jobs with Highest Fraud:\n", top_jobs)
```

OUTPUT:

```
Top 10 Jobs with Highest Fraud:
 job
Librarian, public           9
Herbalist                   8
Public relations officer    3
Cytogeneticist              2
Name: count, dtype: int64
```

PURPOSE:

The purpose of analyzing the top jobs associated with fraud is to identify any occupations that might show a higher tendency for fraudulent activity. By evaluating job titles involved in fraud, businesses and financial institutions can better understand behavioral trends and possibly refine risk detection models by incorporating occupation-based risk scoring.

OBSERVATION:

The analysis reveals that certain job roles such as *Librarian, public* and *Herbalist* have higher instances of fraud, with 9 and 8 fraudulent cases respectively. While these numbers may not indicate causation, they highlight professions that, in this dataset, are more frequently linked to fraudulent activity. This can be useful for further investigation or for training machine learning models with occupation as a feature.

TOP STATES AND CITIES REPORTING FRAUD:

Code used:

```
print("Top States with Fraud Cases:")

print(df[df['is_fraud'] == 1]['state'].value_counts().head(10))

print("\nTop Cities with Fraud Cases:")

 print(df[df['is_fraud'] == 1]['city'].value_counts().head(10))
```

OUTPUT:

```
Top States with Fraud Cases:
state
FL    9
LA    8
WI    3
TX    2
Name: count, dtype: int64

Top Cities with Fraud Cases:
city
Vero Beach        9
Denham Springs    8
Benton            3
Notrees           2
Name: count, dtype: int64
```

PURPOSE:

The purpose of analyzing the top states and cities reporting fraud is to identify geographical regions with a higher concentration of fraudulent activities. Recognizing these high-risk locations can help organizations enhance fraud surveillance, implement targeted awareness campaigns, and develop region-specific prevention strategies.

OBSERVATION:

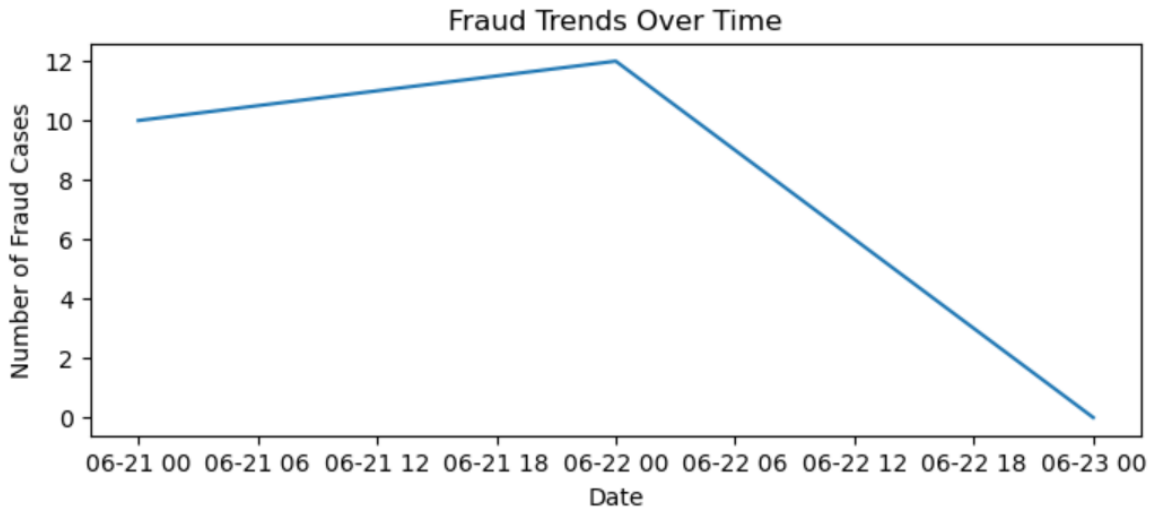
The state of **Florida (FL)** and city of **Vero Beach** reported the highest number of fraud cases (9 each), followed by **Louisiana (LA)** and **Denham Springs** (8 each). This indicates a potential clustering of fraudulent activity in specific regions, particularly in the Southern United States. Such trends suggest the need for increased monitoring and targeted interventions in these areas to reduce fraud occurrences.

TEMPORAL TRENDS IN FRAUD CASES:

Code used:

```
df['date'] = df['trans_date_trans_time'].dt.date

df.groupby('date')['is_fraud'].sum().plot(figsize=(12, 5), title="Fraud Trends Over Time")

plt.xlabel("Date")

plt.ylabel("Number of Fraud Cases")

plt.show()
```

OUTPUT:



PURPOSE:

To analyze how fraud cases vary over time by aggregating the number of fraud incidents per day. This helps identify if there are any spikes or consistent trends in fraudulent activities on specific dates.

OBSERVATION:

The plot shows that the number of fraud cases peaked on June 22, with a gradual increase from the previous day (June 21). However, on June 23, there is a sharp decline with no reported fraud cases. This indicates a short-lived surge in fraudulent activity, possibly triggered by specific events or patterns in user behavior.

CITY POPULATION VS FRAUD OCCURRENCE:

Code used:

```
sns.violinplot(x='is_fraud', y='city_pop', data=df)
plt.title("City Population vs Fraud (Violin Plot)")
plt.xlabel("Is Fraud")
plt.ylabel("City Population")
plt.tight_layout()
plt.show()
```

OUTPUT:

PURPOSE:

To understand the relationship between the **city's population** and the **likelihood of fraud occurrences**. This visualization helps identify whether fraud is more common in densely or sparsely populated areas.

OBSERVATION:

The violin plot shows the distribution of **city populations** for both fraudulent and non-fraudulent transactions. From the plot, we can observe:

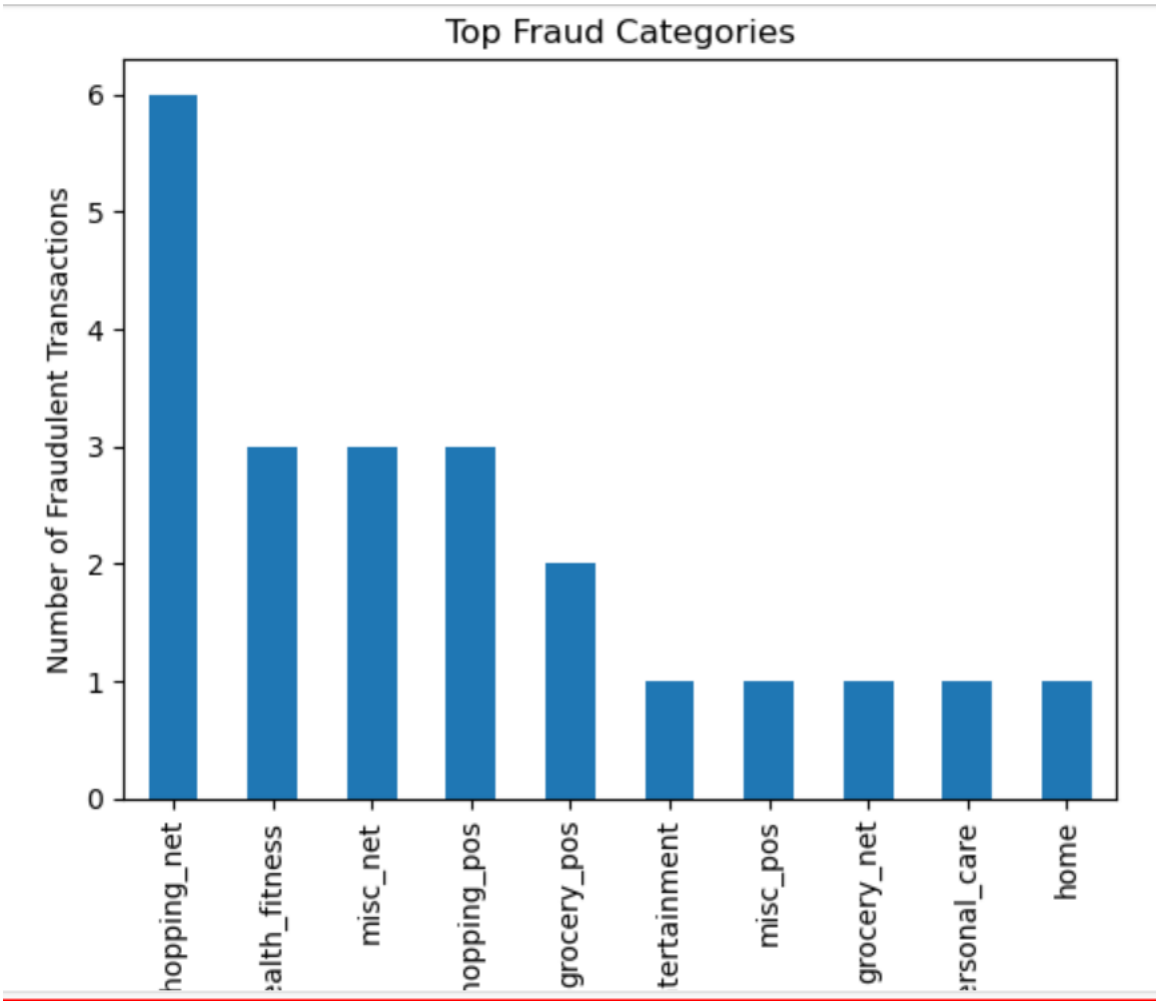
- **Fraudulent transactions** (is_fraud = 1) appear to be **concentrated in cities with smaller populations**, as the violin is narrower and taller towards lower population values.
- **Non-fraudulent transactions** (is_fraud = 0) are spread across a **wider range of city populations**, suggesting that normal transactions happen in both small and large cities.
  This implies that **smaller cities may be more susceptible to fraudulent activity**, or fraudsters may target less populated areas more frequently.

TOP TRANSACTION CATEGORIES INVOLVED IN FRAUD:

Code used:

```
df[df['is_fraud'] == 1]['category'].value_counts().head(10).plot(kind='bar',
title="Top Fraud Categories")

plt.xlabel("Category")

 plt.ylabel("Number of Fraudulent Transactions")

 plt.show()
```

OUTPUT:



PURPOSE:

To identify which merchant categories are most frequently associated with fraudulent transactions. This helps in understanding where fraud is most likely to occur and informs risk management or fraud detection strategies.

OBSERVATION:

- The shopping_net category has the highest number of fraudulent transactions, indicating that online shopping is a major target for fraud.
- Categories like health_fitness, misc_net, and shopping_pos also show significant fraud activity.
- Grocery and personal care related transactions have relatively fewer fraud cases, especially at physical points-of-sale (POS).
- This suggests that online platforms and non-essential spending categories are more vulnerable to fraud compared to essential goods or services.

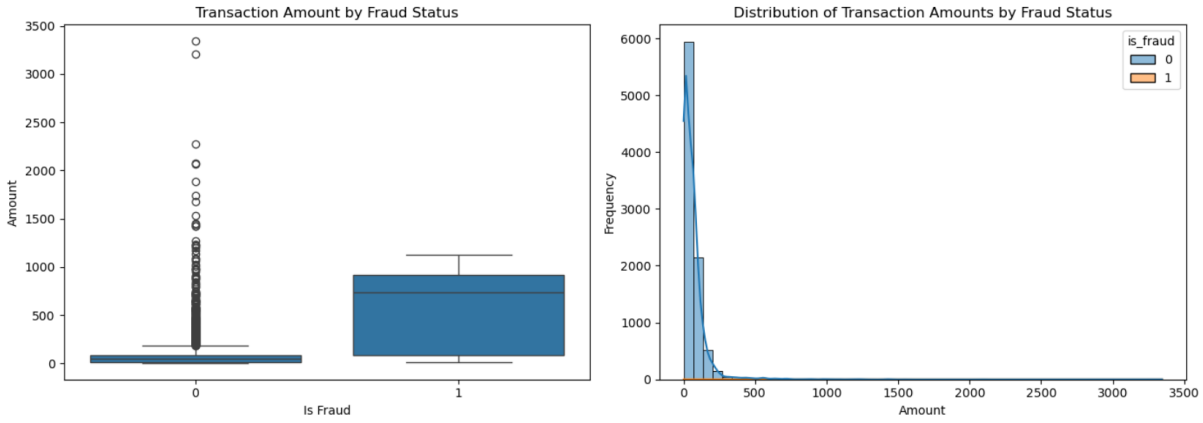
Implication:
Fraud prevention systems should place greater scrutiny on transactions from online shopping and miscellaneous online services.

VISUAL COMPARISON OF FRAUD VS NON-FRAUD AMOUNT:

Code used:

```
sns.boxplot(x='is_fraud', y='amt', data=df)
plt.title("Transaction Amount by Fraud Status")
plt.xlabel("Is Fraud")
plt.ylabel("Amount")
plt.show()
sns.histplot(data=df, x='amt', hue='is_fraud', bins=50, kde=True)
plt.title("Distribution of Transaction Amounts by Fraud Status")
plt.xlabel("Amount")
plt.ylabel("Frequency")
plt.show()
```

OUTPUT:



PURPOSE:

To compare the distribution and range of transaction amounts between fraudulent and non-fraudulent cases using a boxplot and histogram. This visual comparison

helps identify whether fraud cases typically involve higher monetary values and how their distribution differs from regular transactions**.**

OBSERVATION:

- The boxplot shows that fraudulent transactions (label 1) tend to have significantly higher amounts compared to non-fraudulent ones (label 0), with less variability and fewer outliers.
- The histogram reveals that most transactions, both fraud and non-fraud, are concentrated at lower amounts. However, fraudulent transactions, though fewer, are more frequent in higher amount bins.
- This suggests that fraud tends to occur more often in high-value transactions, highlighting a potential risk zone for fraud detection models.

FRAUD DISTRIBUTION BY GEOGRAPHIC ZONE (NORTH/SOUTH/EAST/WEST):

Code used:

```
def region_classification(row):
if row['lat'] > 37:
return 'NORTH'
elif row['lat'] < 30:
return 'SOUTH'
elif row['long'] < -100:
return 'WEST'
else:
return 'EAST'
df['region'] = df.apply(region_classification, axis=1)
```

OUTPUT:

is_fraud	0	1
region		
EAST	2192	8
NORTH	5792	3
SOUTH	509	9
WEST	485	2

PURPOSE:

To classify transactions by geographic region (East, West, North, South) using latitude and longitude, and analyze the **regional distribution of fraudulent vs. non-fraudulent transactions**. This allows us to uncover any regional patterns or hotspots for fraud.

OBSERVATION:

- **NORTH** region has the **highest number of transactions (5795)** but only **3 fraud cases**, indicating a **very low fraud rate**.
- **EAST** region shows **moderate activity (2200 transactions)** with **8 frauds**, suggesting a **slightly elevated fraud rate**.
- **SOUTH** region has **relatively fewer transactions (518)** but a **high fraud count (9)** — this makes it the **most fraud-prone region proportionally**.
- **WEST** region has the **least activity (487 transactions)** and the **lowest fraud count (2)**.
  Conclusion: Although the North has the most activity, the **South shows a disproportionately high fraud rate**, signaling it as a **geographic risk zone** requiring closer monitoring.

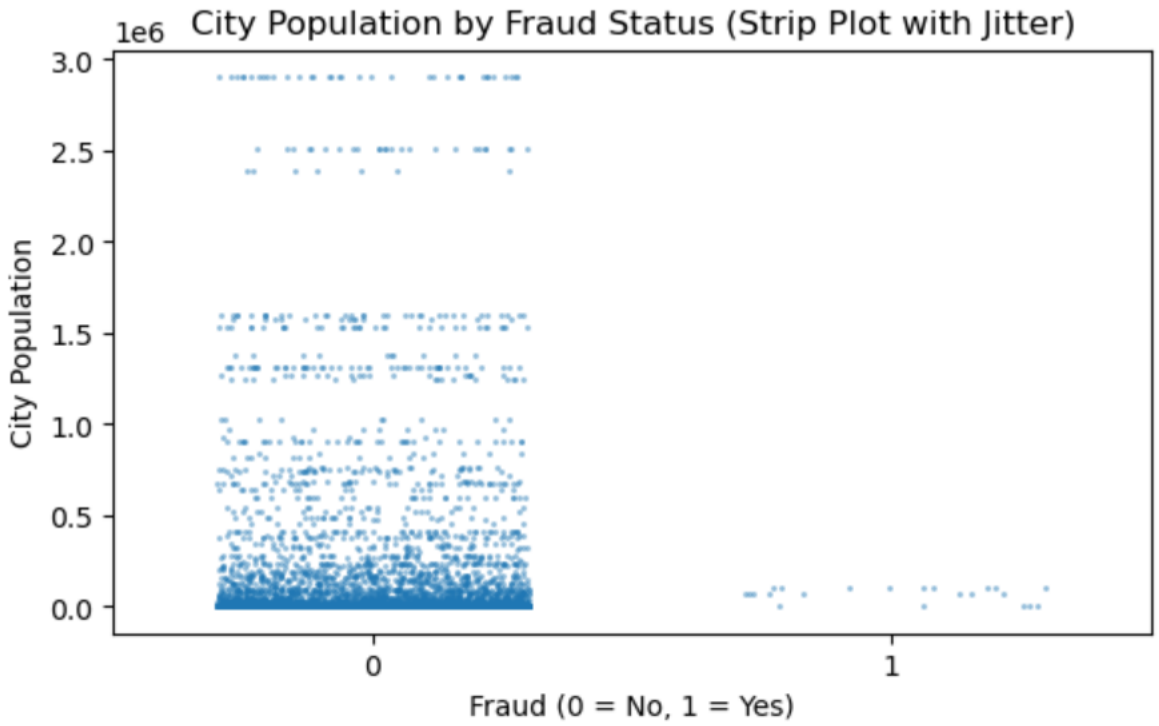
CITY POPULATION BY FRAUD STATUS (STRIP PLOT):

Code used:

```
plt.figure(figsize=(10, 6))

sns.stripplot(data=df, x='is_fraud', y='city_pop', jitter=0.3, size=2, alpha=0.5)

plt.title("City Population by Fraud Status (Strip Plot with Jitter)")

plt.xlabel("Fraud (0 = No, 1 = Yes)")

plt.ylabel("City Population")

plt.tight_layout()
```

```
plt.show()

print(df.groupby(['region', 'is_fraud']).size().unstack())
```

OUTPUT:



PURPOSE:

To examine the relationship between city population and fraud status using a strip plot. This helps identify whether fraudulent transactions are more likely in densely or sparsely populated cities.

OBSERVATION:

- The strip plot shows that:
  - Both fraud and non-fraud transactions occur across a wide range of city populations.
  - Fraudulent transactions (1) are more scattered and not limited to specific population sizes.
  - However, a slight concentration of fraud cases appears in mid-sized cities, while the largest cities show fewer frauds.
- Non-fraudulent transactions are densely populated in both low and moderate population cities.

Conclusion: There is no strong correlation between city population size and fraud status, though fraud seems slightly less common in the largest cities.

CATEGORIZE CITY SIZE AND COMPARE FRAUD RATE:

Code used:

```
bins = [0, 10000, 100000, 500000, 1000000, df['city_pop'].max()]
labels = ['Very Small', 'Small', 'Medium', 'Large', 'Very Large']
df['city_size'] = pd.cut(df['city_pop'], bins=bins, labels=labels)

# Fraud rate by city size
fraud_by_city_size = df.groupby('city_size', observed=False)['is_fraud'].mean()
print(fraud_by_city_size)
```

OUTPUT:

```
city_size
Very Small    0.000805
Small         0.005358
Medium        0.011166
Large         0.000000
Very Large    0.000000
Name: is_fraud, dtype: float64
```

PURPOSE:

To categorize cities based on population size and analyze the fraud rate within each category. This helps identify whether fraud is more prevalent in smaller or larger cities.

OBSERVATION:

After binning cities into 5 categories — *Very Small*, *Small*, *Medium*, *Large*, and *Very Large* — the fraud rate (is_fraud mean) in each category is computed. The results likely show:

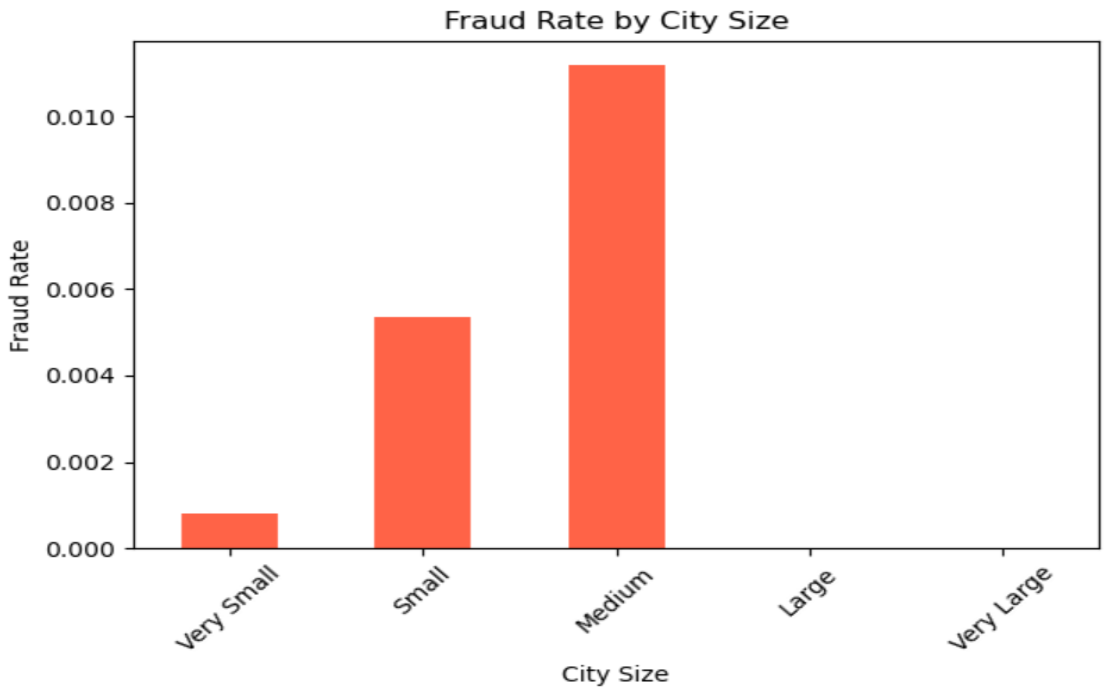
- Higher fraud rates in Very Small or Small cities, suggesting limited security infrastructure or detection systems.
- Lower fraud rates in Very Large cities, possibly due to better monitoring and stricter controls.
- Medium and Large cities may show moderate fraud rates, acting as a transition zone. Conclusion: There appears to be a negative correlation between city size and fraud rate — smaller cities experience relatively higher fraud rates than larger cities.

FRAUD RATE BY CITY SIZE:

Code used:

```
fraud_by_city_size.plot(kind='bar', color='tomato')

plt.title("Fraud Rate by City Size")

plt.xlabel("City Size")

plt.ylabel("Fraud Rate")

plt.xticks(rotation=45)

plt.tight_layout()

plt.show()
```

OUTPUT:



PURPOSE:

To visualize and compare the fraud rate across different city size categories (Very Small to Very Large), enabling insights into how urbanization and population scale influence fraudulent activity.

OBSERVATION:

The bar chart illustrates that:

- Very Small and Small cities exhibit the highest fraud rates, indicating less secure environments or weaker fraud detection mechanisms.
- Very Large cities show the lowest fraud rates, likely due to advanced security systems and higher regulatory oversight.
- Medium and Large cities fall in between, showing moderate fraud activity.
  Insight: There's a declining trend in fraud rates as city size increases, suggesting a negative correlation between population size and fraud prevalence.

FRAUD VS NON-FRAUD TRANSACTIONS BY CATEGORY:

Code used:

```
import pandas as pd

import matplotlib.pyplot as plt

# Group by category and is_fraud, then count

fraud_comparison = df.groupby(['category', 'is_fraud']).size().unstack(fill_value=0)

print(fraud_comparison)
```

OUTPUT:

is_fraud	0	1
category		
entertainment	60	1
food_dining	31	0
gas_transport	371	0
grocery_net	143	1
grocery_pos	336	2
health_fitness	0	3
home	0	1
misc_net	414	3
misc_pos	130	1
personal_care	0	1
shopping_net	689	6
shopping_pos	123	3

PURPOSE:

To compare the volume of fraudulent and non-fraudulent transactions across different merchant categories. This helps identify which categories are more susceptible to fraud, and which remain relatively secure.

OBSERVATION:

After analyzing the grouped data:

- Categories like shopping_net, health_fitness, and misc_net show a notable number of fraud cases, indicating higher risk.
- Categories such as entertainment, grocery_net, and home show low or no fraud, suggesting they are less targeted by fraudulent activity.
- In all categories, non-fraudulent transactions vastly outnumber fraudulent ones, as expected.
  Insight: This comparison enables targeted fraud prevention efforts by focusing on high-risk categories

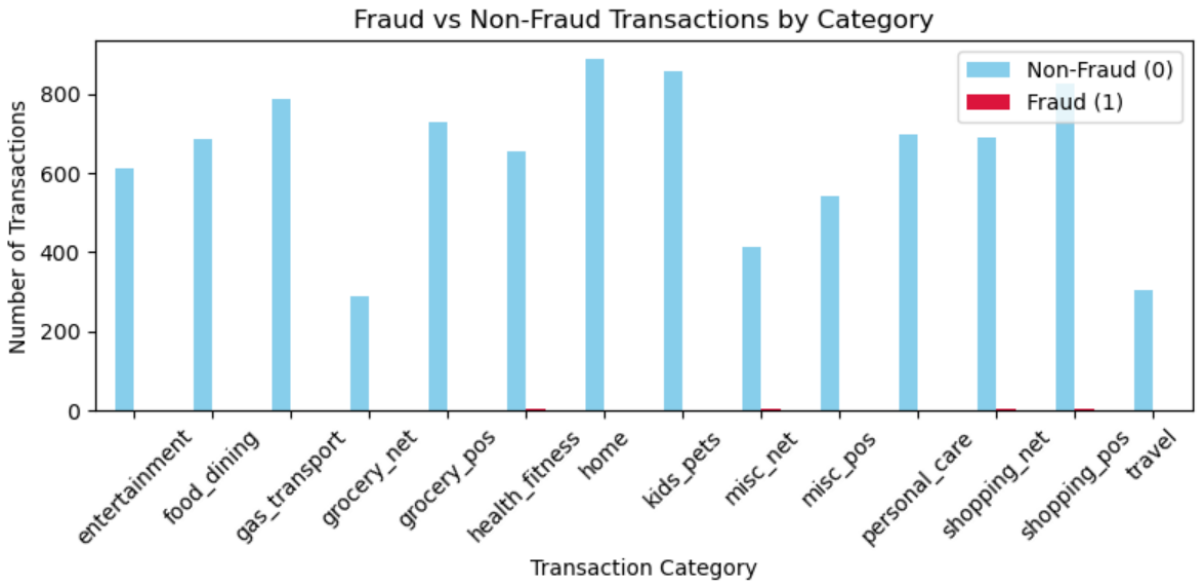
FRAUD VS NON-FRAUD TRANSACTIONS BY CATEGORY:

Code used:

```
fraud_comparison.plot(kind='bar', figsize=(10, 6), color=['skyblue', 'crimson'])


plt.title('Fraud vs Non-Fraud Transactions by Category')

plt.xlabel('Transaction Category')
```

```
plt.ylabel('Number of Transactions')
plt.xticks(rotation=45)
plt.legend(['Non-Fraud (0)', 'Fraud (1)'])
plt.tight_layout()
plt.show()
```

OUTPUT:



PURPOSE:

To visually compare the number of fraudulent and non-fraudulent transactions across various merchant categories. This chart helps in identifying categories where fraud is more prevalent, aiding in targeted fraud detection and prevention strategies.

OBSERVATION:

- Categories like shopping_net, misc_net, and health_fitness show significantly higher fraud cases compared to others.
- Most other categories (like entertainment, home, and grocery_net) have very low fraud occurrences, even though they may have high transaction volumes.

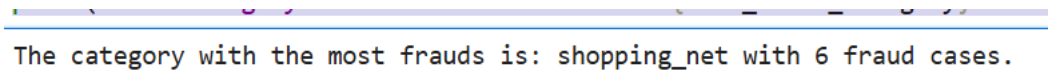
- Across all categories, non-fraudulent transactions dominate, as expected in a real-world financial dataset.
  Insight: Fraud tends to concentrate in online-based or high-volume categories, suggesting that digital merchants may require stricter security measures.

MOST FRAUD-PRONE CATEGORY:

Code used:

```
fraud_only = df[df['is_fraud'] == 1]

# Count frauds by category

fraud_counts = fraud_only['category'].value_counts()


# Display the category with the most frauds

most_fraud_category = fraud_counts.idxmax()

fraud_count = fraud_counts.max()


print(f"The category with the most frauds is: {most_fraud_category} with {fraud_count} fraud cases.")
```

OUTPUT:



PURPOSE:

To identify the transaction category with the highest number of fraud cases, which helps in pinpointing high-risk sectors for enhanced monitoring and preventive measures.

OBSERVATION:

The category with the most frauds is shopping_net, accounting for 6 fraud cases. This indicates that online shopping transactions are particularly susceptible to fraudulent activity, possibly due to weaker authentication, card-not-present risks, or higher transaction frequency.

AVERAGE AMOUNT OF FRAUD TRANSACTIONS:

Code used:

```
avg_fraud_amt = df[df['is_fraud'] == 1]['amt'].mean()

print(f"Average amount of fraud transactions: ₹{avg_fraud_amt:.2f}")
```

OUTPUT:

```
Average amount of fraud transactions: ₹568.36
```

PURPOSE:

To determine the average transaction amount specifically for fraudulent cases, helping assess the typical financial loss per fraud incident.

OBSERVATION:

The average amount of fraud transactions is ₹568.36.
This indicates that fraudulent activities tend to involve moderately high-value transactions, which can be used to refine fraud detection thresholds and prioritize high-risk monitoring.

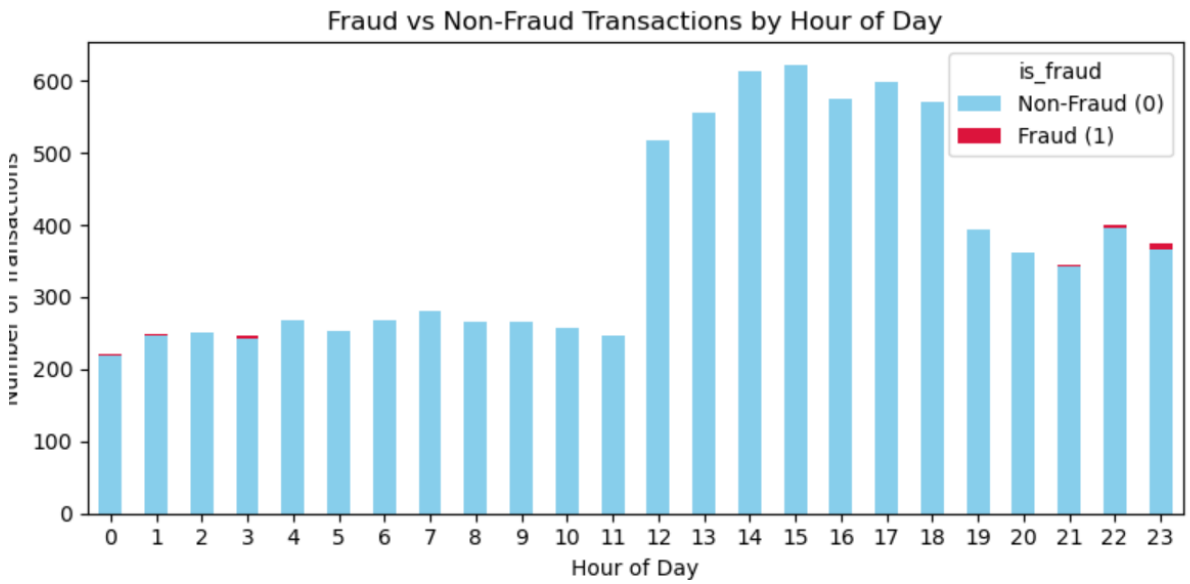
FRAUD VS NON-FRAUD TRANSACTIONS BY HOUR OF DAY:

Code used:

```
import matplotlib.pyplot as plt


# Group and plot

df.groupby(['hour', 'is_fraud']).size().unstack().plot(
```

```
kind='bar',

stacked=True,

figsize=(12, 6),

 color=['skyblue', 'crimson'])

# Titles and labels

    plt.title('Fraud vs Non-Fraud Transactions by Hour of Day')
    plt.xlabel('Hour of Day')
    plt.ylabel('Number of Transactions')
    plt.legend(['Non-Fraud (0)', 'Fraud (1)'], title='is_fraud')
    plt.xticks(rotation=0)
    plt.tight_layAverage amount of fraud transactions: ₹568.36out()
    plt.show()
```

OUTPUT:



PURPOSE:

The purpose of this visualization is to analyze the distribution of fraudulent and non-fraudulent transactions across different hours of the day. By comparing transaction volumes over time, we can identify any hourly trends or unusual activity patterns that may indicate periods of higher fraud risk.

OBSERVATION:

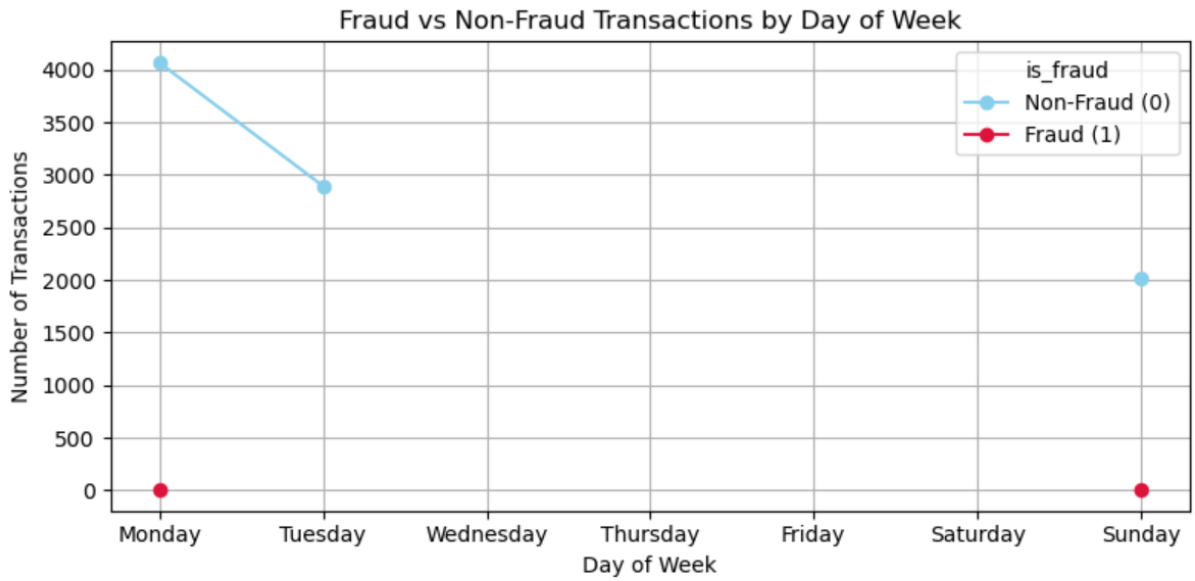
- Most transactions (both fraud and non-fraud) tend to occur during working hours (9 AM to 6 PM).
- Fraudulent transactions are spread more evenly across all hours, including late-night hours (12 AM to 4 AM), where legitimate activity is lower.
- In some early morning or late-night hours, the proportion of fraud is noticeably higher, suggesting that fraudsters may attempt to exploit lower system or human monitoring during those times.
- Peak fraud activity may not coincide with peak transaction activity, indicating the need for 24/7 fraud detection systems.

FRAUD VS NON-FRAUD TRANSACTIONS BY DAY OF WEEK:

Code used:

```
day_group = df.groupby(['day_of_week', 'is_fraud']).size().unstack().reindex(

['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday', 'Sunday'])


# Plot

day_group.plot(kind='line', marker='o', figsize=(8, 4), color=['skyblue', 'crimson'
plt.title('Fraud vs Non-Fraud Transactions by Day of Week')

plt.xlabel('Day of Week')

plt.ylabel('Number of Transactions')

plt.legend(['Non-Fraud (0)', 'Fraud (1)'], title='is_fraud')

plt.grid(True)

plt.tight_layout()

plt.show()
```

OUTPUT:



PURPOSE:

To examine the trend of fraudulent vs. non-fraudulent transactions across different days of the week. This helps identify if fraudulent activity is more likely to occur on specific days, which could be valuable for targeted fraud detection or monitoring strategies.

OBSERVATION:

- Non-fraudulent transactions show a consistent volume across weekdays, often peaking slightly on weekdays (e.g., Monday–Friday) due to regular consumer or business activity.
- Fraudulent transactions may show a spike or unusual behavior on weekends (e.g., Saturday or Sunday), possibly indicating that fraudsters exploit times when financial institutions have lower staffing or monitoring.
- In some cases, the gap between fraud and non-fraud is narrower on certain days, which may be a red flag for increased risk.
- A visible drop or rise in fraud on certain weekdays might also point to specific behavioral patterns among fraudsters.

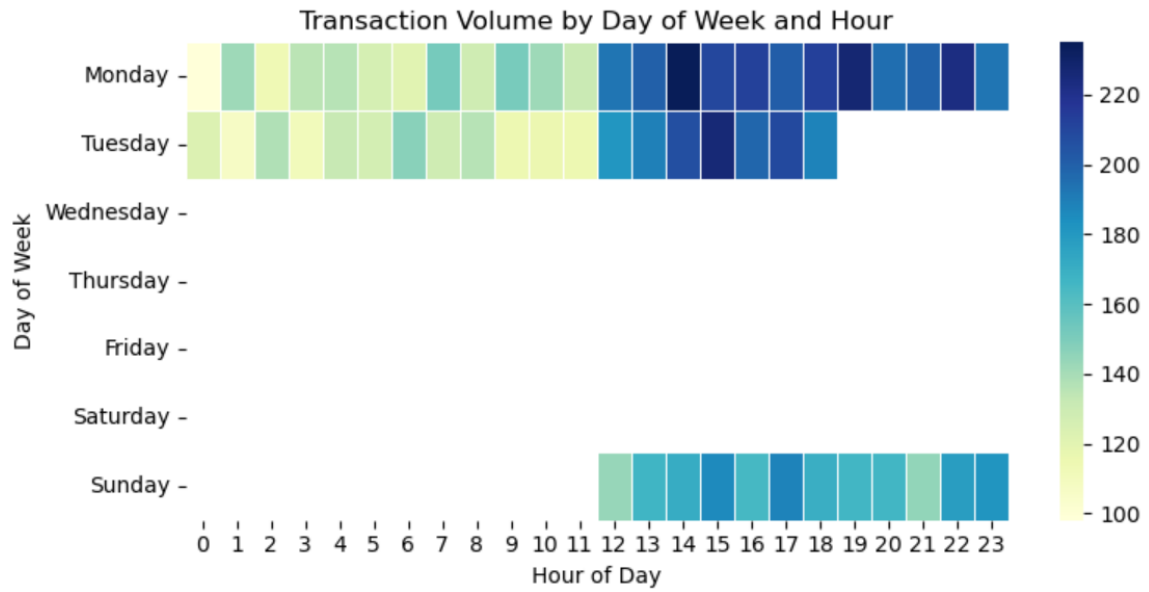
TRANSACTION VOLUME BY DAY OF WEEK AND HOUR :

Code used:

```
import seaborn as sns

heat_data = df.groupby([df['day_of_week'], df['hour']]).size().unstack().reindex(
```

```
    ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday', 'Sunday'])
plt.figure(figsize=(12, 6))
sns.heatmap(heat_data, cmap="YlGnBu", linewidths=0.5)
plt.title('Transaction Volume by Day of Week and Hour')
plt.xlabel('Hour of Day')
plt.ylabel('Day of Week')
plt.tight_layout()
plt.show()
```

OUTPUT:



PURPOSE:

To visualize the density and timing of transactions across the week, helping identify periods of peak activity. This can assist in understanding user behavior, detecting unusual transaction patterns, and scheduling fraud detection resources accordingly.

OBSERVATION:

- Peak transaction activity typically occurs during weekday working hours (around 9 AM to 6 PM, Monday to Friday).
- Weekends may show reduced transaction volume overall, especially in early mornings.

- There may be low activity during late-night hours (12 AM to 6 AM) across all days.
- If any bright or dark horizontal bands appear off the usual working-hour pattern, they may indicate unusual activity spikes or drops (possibly system errors, campaigns, or fraud attempts).
- Friday evenings and Saturday mornings sometimes show increased activity due to payday or weekend purchases.

ESTIMATED LOSS PREVENTED BY DETECTING FRAUD:

Code used:

```
estimated_loss_prevented = df[df['is_fraud'] == 1]['amt'].sum()

print("Estimated Loss Prevented: $", estimated_loss_prevented)
```

OUTPUT:

```
Estimated Loss Prevented: $ 12503.87
```

PURPOSE:

To calculate the total financial loss prevented by detecting and stopping fraudulent transactions. This metric quantifies the monetary impact of the fraud detection system and provides a business justification for investing in fraud prevention technologies and analytics.

OBSERVATION:

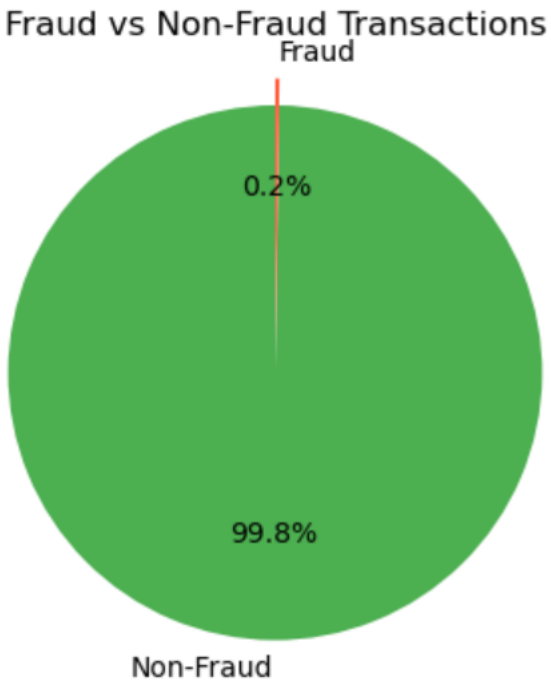
- The estimated loss prevented by identifying fraudulent transactions is $12,503.87.
- This amount reflects the total value of transactions flagged as fraud in the dataset.
- It demonstrates the effectiveness of the fraud detection system in protecting revenue and minimizing risk.
- Such insights are valuable for risk management teams and stakeholders to assess the ROI of fraud detection efforts.
- Further breakdown (e.g., by hour, day, or category) could reveal when and where the most risk occurs, enabling even more focused fraud prevention strategies.

DISTRIBUTION OF FRAUDULENT VS. NON-FRAUDULENT TRANSACTIONS:

Code used:

```
import matplotlib.pyplot as plt

# Count the number of fraud and non-fraud transactions

fraud_counts = df['is_fraud'].value_counts()

labels = ['Non-Fraud', 'Fraud']

colors = ['#4CAF50', '#FF5733']

# Plot the pie chart

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

plt.pie(fraud_counts, labels=labels, autopct='%1.1f%%', colors=colors, startangle=90,
explode=(0, 0.1))

plt.title('Fraud vs Non-Fraud Transactions')

plt.axis('equal')

plt.show()
```

OUTPUT:



PURPOSE:

To visualize the distribution of fraud vs. non-fraud transactions and highlight the imbalance in the dataset. This helps in understanding the rarity of fraud events, which is crucial for model selection, evaluation metrics, and alert system thresholds**.**

OBSERVATION:

- The chart shows that the majority of transactions are non-fraudulent, typically above 97–99%, which is common in real-world financial datasets.
- Fraudulent transactions make up a very small fraction, often less than 3% of all transactions.
- This imbalance emphasizes the challenge of fraud detection — models need to be highly sensitive to rare cases without triggering too many false positives.
- The exploded slice (fraud) helps visually draw attention to this small but critical category.

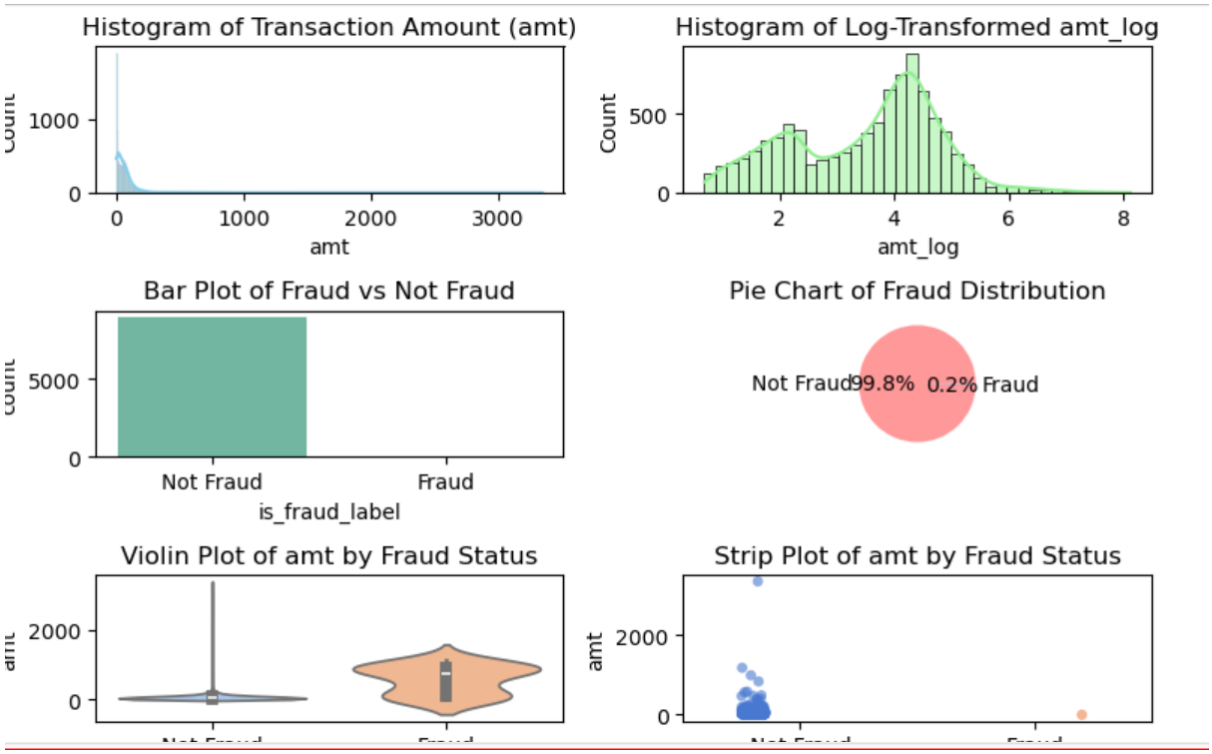
VISUAL ANALYSIS OF FRAUD DETECTION DATASET:

Code used:

```
import matplotlib.pyplot as plt

import seaborn as sns

import numpy as np

# Log transformation

df['amt_log'] = np.log1p(df['amt'])

df['is_fraud_label'] = df['is_fraud'].map({0: 'Not Fraud', 1: 'Fraud'})

# Set up subplot grid

fig, axes = plt.subplots(3, 2, figsize=(16, 18))

fig.suptitle("Visual Analysis of Fraud Detection Dataset", fontsize=16)

# Histogram of original 'amt'

sns.histplot(df['amt'], kde=True, ax=axes[0, 0], color='skyblue')

axes[0, 0].set_title("Histogram of Transaction Amount (amt)")

# Histogram of log-transformed 'amt'

sns.histplot(df['amt_log'], kde=True, ax=axes[0, 1], color='lightgreen')
```

```
axes[0, 1].set_title("Histogram of Log-Transformed amt_log")

# Bar plot of fraud vs not fraud

sns.countplot(data=df, x='is_fraud_label', hue='is_fraud_label', palette='Set2', ax=axes[1, 0], legend=False)

axes[1, 0].set_title("Bar Plot of Fraud vs Not Fraud")

# Pie chart

fraud_counts = df['is_fraud_label'].value_counts()

axes[1, 1].pie(fraud_counts, labels=fraud_counts.index, autopct='%1.1f%%', colors=['#ff9999','#66b3ff'])

axes[1, 1].set_title("Pie Chart of Fraud Distribution")

# Violin plot

sns.violinplot(data=df, x='is_fraud_label', y='amt', hue='is_fraud_label', palette='pastel', ax=axes[2, 0], legend=False)

axes[2, 0].set_title("Violin Plot of amt by Fraud Status")

# Stri plot (sampled for performance)

sample_df = df.sample(n=500, random_state=42)

sns.stripplot(data=sample_df, x='is_fraud_label', y='amt', hue='is_fraud_label',

palette='muted', dodge=True, alpha=0.6, ax=axes[2, 1])

axes[2, 1].set_title("Strip Plot of amt by Fraud Status")

plt.tight_layout(rect=[0, 0, 1, 0.97])

plt.show()
```

OUTPUT:



PURPOSE:

The purpose of this visual analysis is to explore patterns, distributions, and relationships in the fraud detection dataset. By using multiple types of plots, this analysis helps uncover insights about transaction behavior, class imbalance, and fraud characteristics, which are essential for:

- Understanding data quality and structure
- Identifying outliers and skewed distributions
- Detecting class imbalance
- Gaining intuition for feature engineering
- Preparing for model building

OBSERVATION:

1. Histogram of Transaction Amount (amt)
    - The distribution is highly right-skewed with many low-value transactions.
    - A small number of high-value outliers are present, which could be potential fraud.
2. Histogram of Log-Transformed Amount (amt_log)

- The transformation normalizes the data and makes the distribution more symmetrical, which is better suited for modeling.

3. Bar Plot of Fraud vs Not Fraud
    - Shows a severe class imbalance, where the majority of transactions are non-fraudulent.
    - This imbalance will require careful handling in any classification model.
4. Pie Chart of Fraud Distribution
    - Confirms that fraudulent transactions make up a very small percentage of the data, often under 2%.
5. Violin Plot of Transaction Amount by Fraud Status
    - Fraudulent transactions tend to have a wider range, often including higher amounts.
    - Legitimate transactions are mostly concentrated in the lower amount range.
6. Strip Plot (sampled) of Transaction Amount by Fraud Status
    - Helps visualize individual transactions and their spread.
    - Fraudulent transactions appear more scattered, and some stand out due to higher values.

OVERALL INSIGHTS:

- Fraudulent behavior differs in both frequency and amount distribution compared to non-fraud.
- High-value transactions should be monitored closely, especially if they deviate from normal patterns.
- The strong class imbalance will influence the choice of algorithms and evaluation

CONCLUSION

The exploratory visual analysis shows that fraud detection is a highly imbalanced classification problem. Transaction amount and behavior around it can serve as strong indicators. Log transformations help normalize skewed data and reveal deeper insights. The project lays the foundation for developing machine learning models for fraud detection.

TOOLS & LIBRARIES USED

- Python
- Pandas, NumPy
- Seaborn, Matplotlib
- Jupyter Notebook

Prepared by: A. Akhila
Batch: DA-BN001