```
import pandas as pd
import mysql.connector
import os

# List of CSV files and their corresponding table names
csv_files = [
    ('fraudTest.csv', 'fraudtest')
]

# Connect to the MySQL database
conn = mysql.connector.connect(
    host='127.0.0.1',
    user='root',
    password='Shamayeeta@12',
    database='fraud_detection'
)
cursor = conn.cursor()

# Folder containing the CSV files
folder_path = 'E:/Desktop/Fraud Detection Dataset'

# Function to map pandas dtype to SQL types
def get_sql_type(dtype, column_name=None):
    if column_name and column_name.lower() == 'cc_num':
        return 'VARCHAR(20)'  # to prevent out-of-range error
    if pd.api.types.is_integer_dtype(dtype):
        return 'INT'
    elif pd.api.types.is_float_dtype(dtype):
        return 'FLOAT'
    elif pd.api.types.is_bool_dtype(dtype):
        return 'BOOLEAN'
    elif pd.api.types.is_datetime64_any_dtype(dtype):
        return 'DATETIME'
    else:
        return 'TEXT'

# Process each CSV file
for csv_file, table_name in csv_files:
    file_path = os.path.join(folder_path, csv_file)

    # Read the CSV file into a pandas DataFrame
    df = pd.read_csv(file_path)
    print(f"□ Reading {csv_file} — Shape: {df.shape}")

    # Replace NaN with None
    df = df.where(pd.notnull(df), None)

    # Clean column names
    df.columns = [col.replace(' ', '_').replace('-', '_').replace('.',
'_') for col in df.columns]
```

```
    # Generate the CREATE TABLE SQL statement
    columns = ', '.join([
        f'`{col}` {get_sql_type(df[col].dtype, col)}'
        for col in df.columns
    ])
    create_table_query = f'CREATE TABLE IF NOT EXISTS `{table_name}`
({columns})'
    cursor.execute(create_table_query)
    print(f"□ Table `{table_name}` created or already exists.")

    # Insert rows with error handling
    insert_count = 0
    for idx, row in df.iterrows():
        values = tuple(None if pd.isna(x) else x for x in row)
        placeholders = ', '.join(['%s'] * len(values))
        column_names = ', '.join([f'`{col}`' for col in df.columns])
        sql = f"INSERT INTO `{table_name}` ({column_names}) VALUES
({placeholders})"

        try:
            cursor.execute(sql, values)
            insert_count += 1
        except Exception as e:
            print(f"□ Error inserting row {idx} in {table_name}: {e}")
            print(f"Row data: {row.to_dict()}\n")

    # Commit changes
    conn.commit()
    print(f"□ Inserted {insert_count} rows into `{table_name}`\n")

# Close the connection
conn.close()
print("□ MySQL connection closed.")
```

```
□ Reading fraudTest.csv – Shape: (555719, 23)
□ Table `fraudtest` created or already exists.
□ Inserted 555719 rows into `fraudtest`

□ MySQL connection closed.
```

```
pip install pandas
```

```
Requirement already satisfied: pandas in c:\users\hp\appdata\local\
programs\python\python313\lib\site-packages (2.2.3)
Requirement already satisfied: numpy>=1.26.0 in c:\users\hp\appdata\
local\programs\python\python313\lib\site-packages (from pandas)
(2.1.3)
Requirement already satisfied: python-dateutil>=2.8.2 in c:\users\hp\
appdata\local\programs\python\python313\lib\site-packages (from
pandas) (2.9.0.post0)
```

```
Requirement already satisfied: pytz>=2020.1 in c:\users\hp\appdata\
local\programs\python\python313\lib\site-packages (from pandas)
(2024.2)
Requirement already satisfied: tzdata>=2022.7 in c:\users\hp\appdata\
local\programs\python\python313\lib\site-packages (from pandas)
(2024.2)
Requirement already satisfied: six>=1.5 in c:\users\hp\appdata\local\
programs\python\python313\lib\site-packages (from python-
dateutil>=2.8.2->pandas) (1.16.0)
Note: you may need to restart the kernel to use updated packages.


[notice] A new release of pip is available: 24.2 -> 25.1.1
[notice] To update, run: python.exe -m pip install --upgrade pip
```

```
pip install mysql--connector-python
```

```
Requirement already satisfied: mysql--connector-python in c:\users\hp\
appdata\local\programs\python\python313\lib\site-packages (9.3.0)
Note: you may need to restart the kernel to use updated packages.


[notice] A new release of pip is available: 24.2 -> 25.1.1
[notice] To update, run: python.exe -m pip install --upgrade pip
```

```
pip install matplotlib
```

```
Requirement already satisfied: matplotlib in c:\users\hp\appdata\
local\programs\python\python313\lib\site-packages (3.10.1)
Requirement already satisfied: contourpy>=1.0.1 in c:\users\hp\
appdata\local\programs\python\python313\lib\site-packages (from
matplotlib) (1.3.1)
Requirement already satisfied: cycler>=0.10 in c:\users\hp\appdata\
local\programs\python\python313\lib\site-packages (from matplotlib)
(0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in c:\users\hp\
appdata\local\programs\python\python313\lib\site-packages (from
matplotlib) (4.57.0)
Requirement already satisfied: kiwisolver>=1.3.1 in c:\users\hp\
appdata\local\programs\python\python313\lib\site-packages (from
matplotlib) (1.4.8)
Requirement already satisfied: numpy>=1.23 in c:\users\hp\appdata\
local\programs\python\python313\lib\site-packages (from matplotlib)
(2.1.3)
Requirement already satisfied: packaging>=20.0 in c:\users\hp\appdata\
local\programs\python\python313\lib\site-packages (from matplotlib)
(24.2)
Requirement already satisfied: pillow>=8 in c:\users\hp\appdata\local\
programs\python\python313\lib\site-packages (from matplotlib) (11.2.1)
Requirement already satisfied: pyparsing>=2.3.1 in c:\users\hp\
appdata\local\programs\python\python313\lib\site-packages (from
```

```
matplotlib) (3.2.3)
Requirement already satisfied: python-dateutil>=2.7 in c:\users\hp\
appdata\local\programs\python\python313\lib\site-packages (from
matplotlib) (2.9.0.post0)
Requirement already satisfied: six>=1.5 in c:\users\hp\appdata\local\
programs\python\python313\lib\site-packages (from python-
dateutil>=2.7->matplotlib) (1.16.0)
Note: you may need to restart the kernel to use updated packages.


[notice] A new release of pip is available: 24.2 -> 25.1.1
[notice] To update, run: python.exe -m pip install --upgrade pip
```

```
pip install seaborn
```

```
Requirement already satisfied: seaborn in c:\users\hp\appdata\local\
programs\python\python313\lib\site-packages (0.13.2)
Requirement already satisfied: numpy!=1.24.0,>=1.20 in c:\users\hp\
appdata\local\programs\python\python313\lib\site-packages (from
seaborn) (2.1.3)
Requirement already satisfied: pandas>=1.2 in c:\users\hp\appdata\
local\programs\python\python313\lib\site-packages (from seaborn)
(2.2.3)
Requirement already satisfied: matplotlib!=3.6.1,>=3.4 in c:\users\hp\
appdata\local\programs\python\python313\lib\site-packages (from
seaborn) (3.10.1)
Requirement already satisfied: contourpy>=1.0.1 in c:\users\hp\
appdata\local\programs\python\python313\lib\site-packages (from
matplotlib!=3.6.1,>=3.4->seaborn) (1.3.1)
Requirement already satisfied: cycler>=0.10 in c:\users\hp\appdata\
local\programs\python\python313\lib\site-packages (from matplotlib!
=3.6.1,>=3.4->seaborn) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in c:\users\hp\
appdata\local\programs\python\python313\lib\site-packages (from
matplotlib!=3.6.1,>=3.4->seaborn) (4.57.0)
Requirement already satisfied: kiwisolver>=1.3.1 in c:\users\hp\
appdata\local\programs\python\python313\lib\site-packages (from
matplotlib!=3.6.1,>=3.4->seaborn) (1.4.8)
Requirement already satisfied: packaging>=20.0 in c:\users\hp\appdata\
local\programs\python\python313\lib\site-packages (from matplotlib!
=3.6.1,>=3.4->seaborn) (24.2)
Requirement already satisfied: pillow>=8 in c:\users\hp\appdata\local\
programs\python\python313\lib\site-packages (from matplotlib!
=3.6.1,>=3.4->seaborn) (11.2.1)
Requirement already satisfied: pyparsing>=2.3.1 in c:\users\hp\
appdata\local\programs\python\python313\lib\site-packages (from
matplotlib!=3.6.1,>=3.4->seaborn) (3.2.3)
Requirement already satisfied: python-dateutil>=2.7 in c:\users\hp\
appdata\local\programs\python\python313\lib\site-packages (from
matplotlib!=3.6.1,>=3.4->seaborn) (2.9.0.post0)
```

```
Requirement already satisfied: pytz>=2020.1 in c:\users\hp\appdata\
local\programs\python\python313\lib\site-packages (from pandas>=1.2-
>seaborn) (2024.2)
Requirement already satisfied: tzdata>=2022.7 in c:\users\hp\appdata\
local\programs\python\python313\lib\site-packages (from pandas>=1.2-
>seaborn) (2024.2)
Requirement already satisfied: six>=1.5 in c:\users\hp\appdata\local\
programs\python\python313\lib\site-packages (from python-
dateutil>=2.7->matplotlib!=3.6.1,>=3.4->seaborn) (1.16.0)
Note: you may need to restart the kernel to use updated packages.


[notice] A new release of pip is available: 24.2 -> 25.1.1
[notice] To update, run: python.exe -m pip install --upgrade pip
```

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import mysql.connector
import numpy as np

db = mysql.connector.connect(host="127.0.0.1",
                             username="root",
                             password="Shamayeeta@12",
                             database="fraud_detection")

cur = db.cursor()
```

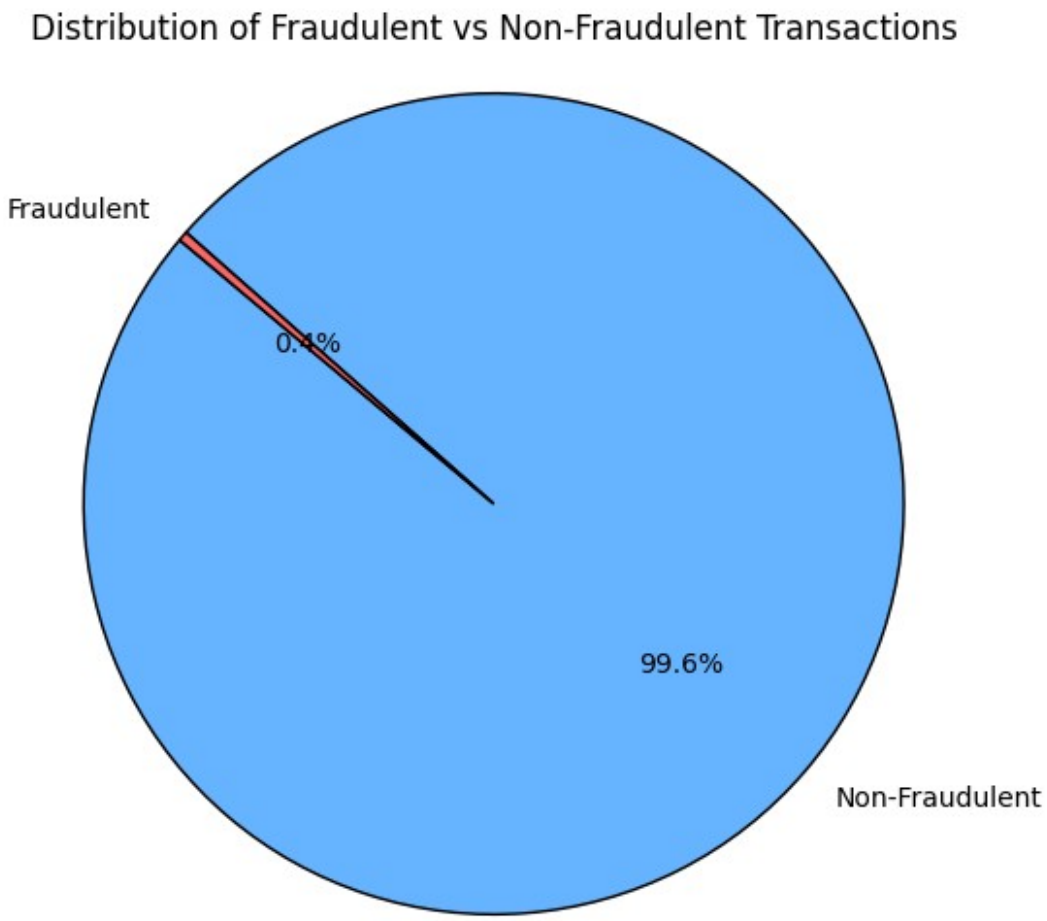
What is the total number of transactions

```
query = """ select count(trans_num) as total_no_of_transactions from
fraudtest"""

cur.execute(query)

data = cur.fetchall()
data
```

```
[(2778595,)]
```

What percentage of transactions are fraudulent vs. non-fraudulent

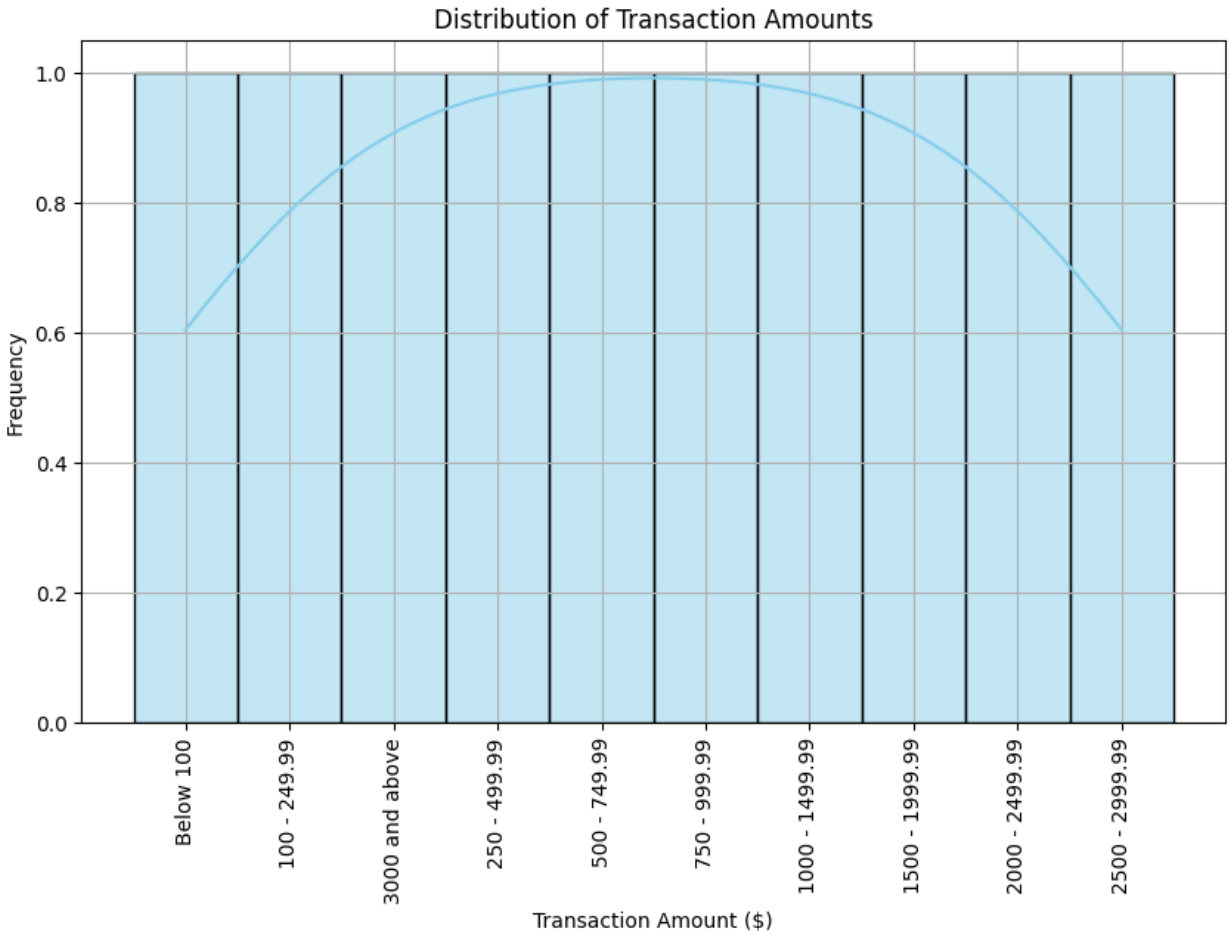
```
query = """ SELECT
    is_fraud,
    COUNT(*) AS transaction_count,
    ROUND(COUNT(*) * 100.0 / (SELECT COUNT(*) FROM fraudtest), 2) AS
percentage
FROM
    fraudtest
GROUP BY
    is_fraud"""

cur.execute(query)

data = cur.fetchall()
df = pd.DataFrame(data, columns = ["is_fraud", "transaction_count",
"percentage"])


import matplotlib.pyplot as plt

# Plot pie chart
labels = ['Non-Fraudulent', 'Fraudulent']
colors = ['#66b3ff', '#ff6666']  # Blue and red tones

plt.figure(figsize=(6, 6))
plt.pie(df['transaction_count'],
        labels=labels,
        autopct='%1.1f%%',
        colors=colors,
        startangle=140,
        wedgeprops={'edgecolor': 'black'})

plt.title('Distribution of Fraudulent vs Non-Fraudulent Transactions')
plt.axis('equal')  # Ensures pie is a circle
plt.show()
```



What is the distribution of transaction amounts

```
query = """SELECT
    CASE
        WHEN amt < 100 THEN 'Below 100'
        WHEN amt BETWEEN 100 AND 249.99 THEN '100 - 249.99'
        WHEN amt BETWEEN 250 AND 499.99 THEN '250 - 499.99'
        WHEN amt BETWEEN 500 AND 749.99 THEN '500 - 749.99'
        WHEN amt BETWEEN 750 AND 999.99 THEN '750 - 999.99'
        WHEN amt BETWEEN 1000 AND 1499.99 THEN '1000 - 1499.99'
        WHEN amt BETWEEN 1500 AND 1999.99 THEN '1500 - 1999.99'
        WHEN amt BETWEEN 2000 AND 2499.99 THEN '2000 - 2499.99'
        WHEN amt BETWEEN 2500 AND 2999.99 THEN '2500 - 2999.99'
        ELSE '3000 and above'
    END AS amount_range,
    COUNT(*) AS transaction_count
FROM
    fraudtest
GROUP BY
```

```
    amount_range
ORDER BY
    MIN(amt) """

cur.execute(query)

data = cur.fetchall()
df = pd.DataFrame(data, columns = ["amount_range",
"transaction_count"])

plt.figure(figsize=(10, 6))
sns.histplot(df['amount_range'], bins=50, kde=True, color='skyblue')
plt.title('Distribution of Transaction Amounts')
plt.xlabel('Transaction Amount ($)')
plt.ylabel('Frequency')
plt.grid(True)
plt.xticks(rotation = 90)
plt.show()
```



Which customer has the highest number of transactions

```
query = """select first, last, count(*) as trans_count
from fraudtest
group by first, last order by trans_count desc limit 1 """

cur.execute(query)

data = cur.fetchall()
df = pd.DataFrame(data, columns = ["first", "last", "trans_count"])
df
```

	first	last	trans_count
0	Scott	Martin	9825

What is the average transaction amount for fraudulent vs. non-fraudulent transactions

```
query = """ SELECT
    is_fraud,
    COUNT(*) AS transaction_count,
    ROUND(AVG(amt), 2) AS average_amount
FROM
    fraudtest
GROUP BY
    is_fraud """

cur.execute(query)

data = cur.fetchall()
df = pd.DataFrame(data, columns = ["is_fraud", "transaction_count",
"average_amount"])
df
```

	is_fraud	transaction_count	average_amount
0	0	2767870	67.61
1	1	10725	528.36

Top 10 customers with the highest total transaction value

```
query = """ select first, last, round(sum(amt) , 2) as
total_transaction_value
from fraudtest
group by first, last
order by total_transaction_value desc limit 10 """

cur.execute(query)

data = cur.fetchall()
df = pd.DataFrame(data, columns = ["first", "last",
"total_transaction_value"])
df
```

	first	last	total_transaction_value
0	Kristina	Stewart	696005.40
1	Jeffrey	Smith	692070.85
2	Susan	Hardy	642865.45
3	Kristen	Hanson	627009.45
4	Scott	Martin	614109.25
5	Kimberly	Gonzalez	607024.10
6	Jenna	Brooks	598303.85
7	Lauren	Torres	597835.30
8	Joanna	Hudson	590997.65
9	Sharon	Smith	589720.50

How many unique customers are involved in fraudulent transactions

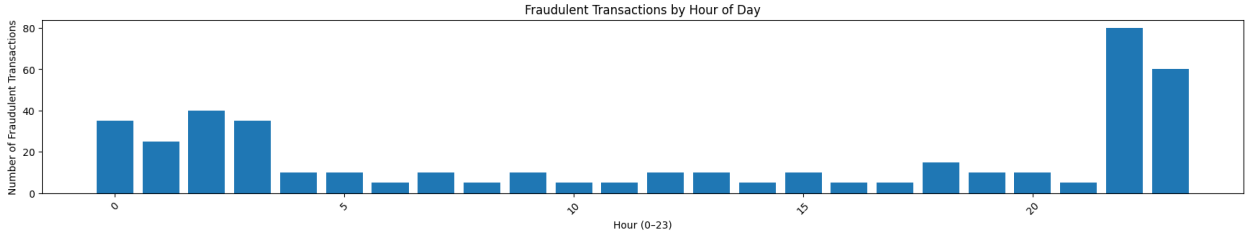
```
query = """ SELECT
    COUNT(DISTINCT first, last) AS unique_fraud_customers
FROM
    fraudtest
WHERE
    is_fraud = 1 """


cur.execute(query)

data = cur.fetchall()
df = pd.DataFrame(data, columns = ["unique_fraud_customers"])
df
```

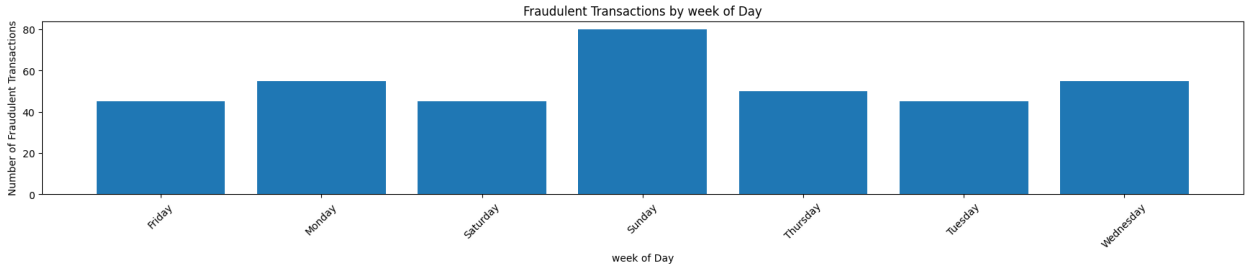
```
   unique_fraud_customers
0                     218
```

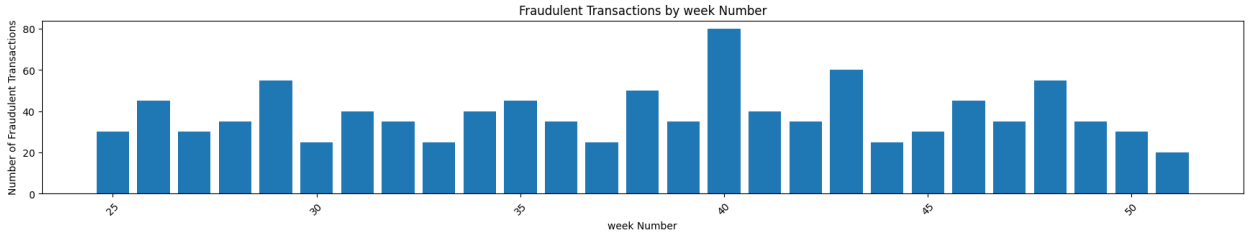
Do fraud transactions occur more during specific time periods (hour/day/week)

```
query = """ SELECT
    HOUR(trans_date_trans_time) AS hour_of_day,
    DAYNAME(trans_date_trans_time) AS day_of_week,
    WEEK(trans_date_trans_time) AS week_number,
    COUNT(*) AS fraud_count
FROM
    fraudtest
WHERE
    is_fraud = 1
GROUP BY
    hour_of_day, day_of_week, week_number
ORDER BY
    hour_of_day, day_of_week, week_number """


cur.execute(query)

data = cur.fetchall()
df = pd.DataFrame(data, columns = ["hour_of_day", "day_of_week",
"week_number", "fraud_count"])

plt.figure(figsize=(18, 12))

plt.subplot(3, 1, 1)
plt.bar(df["hour_of_day"], df["fraud_count"])
plt.xticks(rotation = 45)
plt.title('Fraudulent Transactions by Hour of Day')
plt.xlabel('Hour (0–23)')
plt.ylabel('Number of Fraudulent Transactions')


plt.subplot(3, 1, 2)
plt.bar(df["day_of_week"], df["fraud_count"])
plt.xticks(rotation = 45)
plt.xlabel('week of Day')
plt.ylabel('Number of Fraudulent Transactions')
plt.title('Fraudulent Transactions by week of Day')


plt.subplot(3, 1, 3)
plt.bar(df["week_number"], df["fraud_count"])
```

```
plt.xticks(rotation = 45)
plt.xlabel('week Number')
plt.ylabel('Number of Fraudulent Transactions')
plt.title('Fraudulent Transactions by week Number')
(plt.tight_layout(pad=3))
plt.show()
```







Are there specific customers who are repeatedly involved in fraud

```
query = """ SELECT
    first, last,
    COUNT(*) AS fraud_transaction_count
FROM
    fraudtest
WHERE
    is_fraud = 1
GROUP BY
    first, last
HAVING
    COUNT(*) > 1
ORDER BY
```

```
    fraud_transaction_count DESC """

cur.execute(query)

data = cur.fetchall()
df = pd.DataFrame(data, columns = ["first", "last",
"fraud_transaction_count"])
df
```

```
         first       last  fraud_transaction_count
0         Mary   Williams                       95
1         Gina     Grimes                       90
2         Mary   Humphrey                       80
3    Elizabeth      Mckee                       80
4          Ana     Howell                       80
..         ...        ...                      ...
213  Christine    Leblanc                       15
214     Brooke      Smith                       10
215       Erin     Chavez                       10
216    Barbara       Lowe                       10
217      Janet  Carpenter                       10

[218 rows x 3 columns]
```

What is the relationship between transaction amount and likelihood of fraud

```
query = """ SELECT
    CASE
        WHEN amt < 100 THEN 'Below 100'
        WHEN amt BETWEEN 100 AND 249.99 THEN '100 - 249.99'
        WHEN amt BETWEEN 250 AND 499.99 THEN '250 - 499.99'
        WHEN amt BETWEEN 500 AND 749.99 THEN '500 - 749.99'
        WHEN amt BETWEEN 750 AND 999.99 THEN '750 - 999.99'
        WHEN amt BETWEEN 1000 AND 1499.99 THEN '1000 - 1499.99'
        WHEN amt BETWEEN 1500 AND 1999.99 THEN '1500 - 1999.99'
        WHEN amt BETWEEN 2000 AND 2499.99 THEN '2000 - 2499.99'
        WHEN amt BETWEEN 2500 AND 2999.99 THEN '2500 - 2999.99'
        ELSE '3000 and above'
    END AS amount_range,

    COUNT(*) AS total_transactions,
    SUM(CASE WHEN is_fraud = 1 THEN 1 ELSE 0 END) AS
fraud_transactions,
    ROUND(SUM(CASE WHEN is_fraud = 1 THEN 1 ELSE 0 END) * 100.0 /
COUNT(*), 2) AS fraud_rate_percent
```

```
FROM
    fraudtest
GROUP BY
    amount_range
ORDER BY
    MIN(amt) """


cur.execute(query)

data = cur.fetchall()
df = pd.DataFrame(data, columns = ["amount_range",
"total_transactions", "fraud_transactions", "fraud_rate_percent"])
df
```

	amount_range	total_transactions	fraud_transactions	fraud_rate_percent
0	Below 100	2278590	2400	0.11
1	100 - 249.99	422680	415	0.10
2	3000 and above	895	0	0.00
3	250 - 499.99	46930	2730	5.82
4	500 - 749.99	15475	750	4.85
5	750 - 999.99	7000	3050	43.57
6	1000 - 1499.99	4880	1380	28.28
7	1500 - 1999.99	1240	0	0.00
8	2000 - 2499.99	545	0	0.00
9	2500 - 2999.99	360	0	0.00

Are certain job categories more likely to be involved in fraud

```
query = """ SELECT
    job,
    COUNT(*) AS total_transactions,
    SUM(CASE WHEN is_fraud = 1 THEN 1 ELSE 0 END) AS fraud_count,
    ROUND(100.0 * SUM(CASE WHEN is_fraud = 1 THEN 1 ELSE 0 END) /
```

```
COUNT(*), 2) AS fraud_rate_percentage
FROM fraudtest
GROUP BY job
ORDER BY fraud_rate_percentage DESC """


cur.execute(query)

data = cur.fetchall()
df = pd.DataFrame(data, columns = ["job", "total_transactions",
"fraud_count", "fraud_rate_percentage"])
df
```

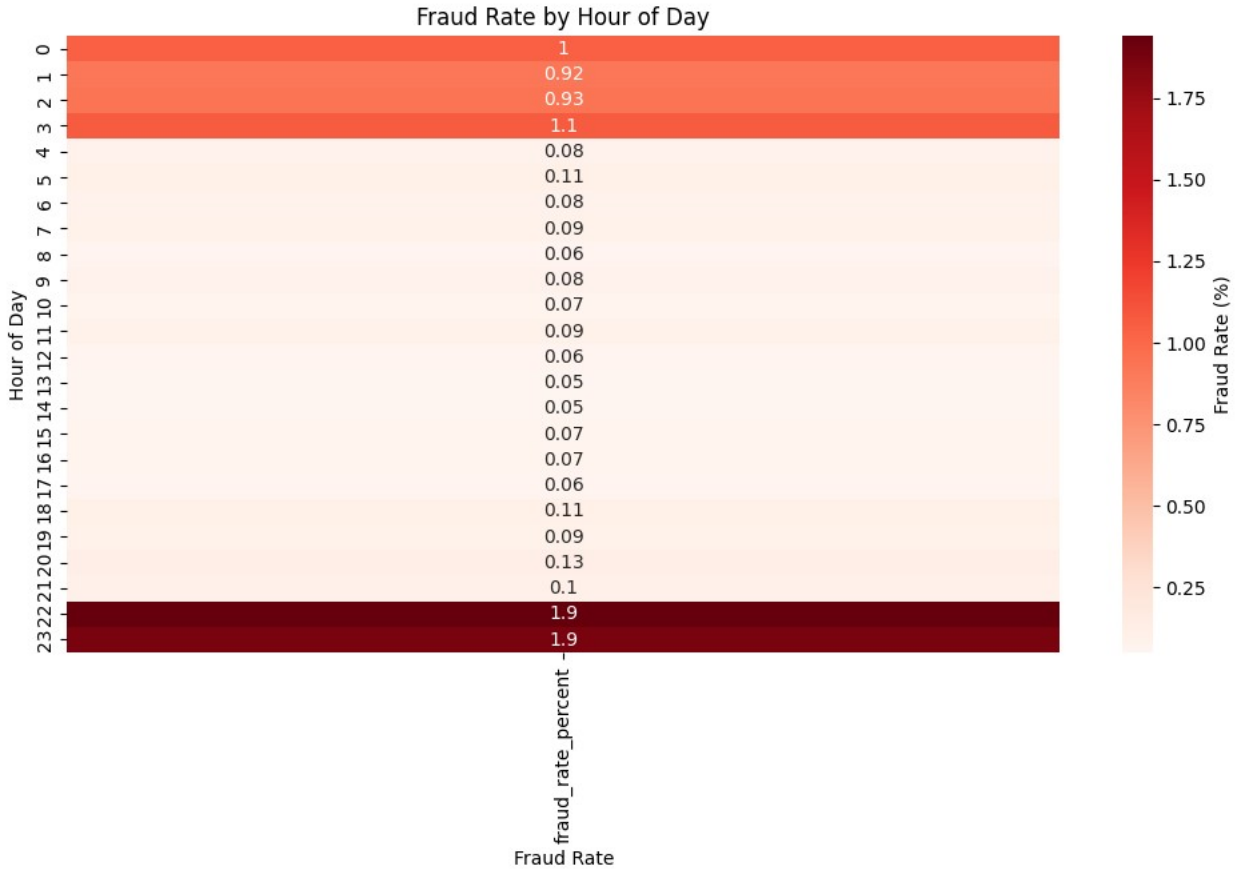
	job	total_transactions	fraud_count	\
0	Engineer, water	40	40	
1	Operational investment banker	55	55	
2	Software engineer	55	55	
3	Horticultural consultant	1065	60	
4	Accountant, chartered certified	1085	60	
..	...	...	...	
473	Administrator, arts	1005	0	
474	Occupational therapist	1095	0	
475	Solicitor, Scotland	1100	0	
476	Sports administrator	2060	0	
477	Artist	1095	0	

	fraud_rate_percentage
0	100.00
1	100.00
2	100.00
3	5.63
4	5.53
..	...
473	0.00
474	0.00
475	0.00
476	0.00
477	0.00

```
[478 rows x 4 columns]
```

Create a heatmap showing fraud rates by time of day

```
query = """ SELECT
    HOUR(trans_date_trans_time) AS hour_of_day,
    COUNT(*) AS total_transactions,
    SUM(is_fraud) AS fraud_transactions,
    ROUND(SUM(is_fraud) / COUNT(*) * 100, 2) AS fraud_rate_percent
FROM
    fraudtest
GROUP BY
    HOUR(trans_date_trans_time)
ORDER BY
    hour_of_day """


cur.execute(query)

data = cur.fetchall()
df = pd.DataFrame(data, columns = ["hour_of_day",
"total_transactions", "fraud_transactions", "fraud_rate_percent"])

df['fraud_rate_percent'] = pd.to_numeric(df['fraud_rate_percent'],
errors='coerce')
plt.figure(figsize=(12, 6))
sns.heatmap(df[['fraud_rate_percent']], annot=True, cmap='Reds',
cbar_kws={'label': 'Fraud Rate (%)'})
plt.title("Fraud Rate by Hour of Day")
plt.xlabel("Fraud Rate")
plt.ylabel("Hour of Day")
plt.xticks(rotation=90)
plt.show()
```



Which destination accounts receive the most fraudulent funds

```
query = """ SELECT
    merchant,
    COUNT(*) AS fraud_transaction_count,
    SUM(amt) AS total_fraud_amount
FROM
    fraudtest
WHERE
    is_fraud = 1
GROUP BY
    merchant
ORDER BY
    total_fraud_amount DESC
LIMIT 10 """


cur.execute(query)

data = cur.fetchall()
```

```
df = pd.DataFrame(data, columns = ["merchant",
"fraud_transaction_count", "total_fraud_amount"])
df
```

```
                                       merchant  fraud_transaction_count
\
0         fraud_Mosciski, Ziemann and Farrell                         90

1                          fraud_Lemke-Gutmann                         90

2  fraud_Romaguera, Cruickshank and Greenholt                         90

3         fraud_Heathcote, Yost and Kertzmann                         85

4                        fraud_Bashirian Group                         75

5                             fraud_Boyer PLC                         75

6                             fraud_Kuhic LLC                         70

7                           fraud_Medhurst PLC                         75

8                          fraud_Heathcote LLC                         75

9       fraud_Altenwerth, Cartwright and Koss                         70


   total_fraud_amount
0        89442.651062
1        88335.900269
2        88136.450195
3        87226.650391
4        79123.900146
5        72831.000209
6        72588.999939
7        71891.799927
8        71868.099670
9        68734.100037
```

Are there any suspicious patterns of transactions between the same sender and receiver repeatedly

```
query = """ SELECT
    cc_num,
    merchant,
    COUNT(*) AS total_transactions,
```

```
    SUM(is_fraud) AS fraud_transactions,
    ROUND(AVG(amt), 2) AS avg_amount,
    MIN(trans_date_trans_time) AS first_txn_time,
    MAX(trans_date_trans_time) AS last_txn_time,
    TIMESTAMPDIFF(HOUR, MIN(trans_date_trans_time),
MAX(trans_date_trans_time)) AS hours_between_first_and_last
FROM
    fraudtest
GROUP BY
    cc_num, merchant
HAVING
    total_transactions > 5 AND
    hours_between_first_and_last < 24 -- transactions packed into a
short time
ORDER BY
    fraud_transactions DESC, total_transactions DESC """


cur.execute(query)

data = cur.fetchall()
df = pd.DataFrame(data, columns = ["cc_num", "merchant",
"total_transactions", "fraud_transactions", "avg_amount",
"first_txn_time", "last_txn_time", "hours_between_first_and_last"])
df
```

```
                   cc_num                                    merchant
\
0            676173792455  fraud_Romaguera, Cruickshank and Greenholt

1     4725837176265195730           fraud_Goldner, Kovacek and Abbott

2        4119762878330989                         fraud_Donnelly PLC

3        6011104316292105                      fraud_Turner and Sons

4         341283058448499                    fraud_McDermott-Weimann

...                   ...                                       ...

1052        4683638447911                         fraud_Pollich LLC

1053       30518206766474              fraud_Hauck, Dietrich and Funk

1054     4710792708725663         fraud_Langosh, Wintheiser and Hyatt

1055     4509142395811241                      fraud_Abbott-Steuber

1056      213163860545705                      fraud_Hilpert-Conroy
```

```
      total_transactions fraud_transactions  avg_amount
first_txn_time  \
0                     10                  10      973.05  2020-07-31
23:08:01
1                     10                  10      334.88  2020-08-04
01:14:34
2                     10                  10      815.56  2020-08-25
22:36:22
3                     10                  10      873.04  2020-08-28
01:00:31
4                     10                  10      303.55  2020-09-05
02:34:32
...                   ...                 ...         ...
...
1052                  10                   0       75.71  2020-12-31
12:32:05
1053                  10                   0       39.59  2020-12-31
12:46:58
1054                  10                   0       63.19  2020-12-31
13:16:21
1055                  10                   0      115.20  2020-12-31
13:53:07
1056                  10                   0       50.61  2020-12-31
21:32:58

             last_txn_time  hours_between_first_and_last
0     2020-07-31 23:49:08                             0
1     2020-08-04 03:53:50                             2
2     2020-08-25 23:52:04                             1
3     2020-08-28 02:33:26                             1
4     2020-09-05 22:29:56                            19
...                   ...                           ...
1052  2020-12-31 14:17:54                             1
1053  2020-12-31 21:51:22                             9
1054  2020-12-31 14:32:41                             1
1055  2020-12-31 21:49:55                             7
1056  2020-12-31 23:21:30                             1

[1057 rows x 8 columns]
```

Using SQL window functions: Are there rapid successive transactions from the same user (could be bot/fraud behavior)

```
query = """ SELECT *
FROM (
```

```
    SELECT
        cc_num,
        trans_date_trans_time,
        amt,
        is_fraud,
        LAG(trans_date_trans_time) OVER (PARTITION BY cc_num ORDER BY
trans_date_trans_time) AS prev_time,
        TIMESTAMPDIFF(SECOND,
            LAG(trans_date_trans_time) OVER (PARTITION BY cc_num ORDER
BY trans_date_trans_time),
            trans_date_trans_time
        ) AS time_diff_seconds
    FROM fraudTest
) AS sub
WHERE time_diff_seconds IS NOT NULL AND time_diff_seconds < 60
ORDER BY cc_num, trans_date_trans_time
 """

cur.execute(query)

data = cur.fetchall()
df = pd.DataFrame(data, columns = ["cc_num", "trans_date_trans_time",
"amt", "is_fraud", "prev_time", "time_diff_seconds"])
df
```

	cc_num	trans_date_trans_time	amt	is_fraud	\
0	180011453250192	2020-06-21 15:41:32	42.32	0	
1	180011453250192	2020-06-21 15:41:32	42.32	0	
2	180011453250192	2020-06-21 15:41:32	42.32	0	
3	180011453250192	2020-06-21 15:41:32	42.32	0	
4	180011453250192	2020-06-21 17:33:11	60.11	0	
...	...	...	...	...	
2225089	676372984911	2020-12-31 20:58:25	211.32	0	
2225090	676372984911	2020-12-31 21:01:02	4.44	0	
2225091	676372984911	2020-12-31 21:01:02	4.44	0	
2225092	676372984911	2020-12-31 21:01:02	4.44	0	
2225093	676372984911	2020-12-31 21:01:02	4.44	0	

	prev_time	time_diff_seconds
0	2020-06-21 15:41:32	0
1	2020-06-21 15:41:32	0
2	2020-06-21 15:41:32	0
3	2020-06-21 15:41:32	0
4	2020-06-21 17:33:11	0
...	...	...
2225089	2020-12-31 20:58:25	0
2225090	2020-12-31 21:01:02	0
2225091	2020-12-31 21:01:02	0
2225092	2020-12-31 21:01:02	0
2225093	2020-12-31 21:01:02	0

```
[2225094 rows x 6 columns]
```

What is the average fraud amount by age group

```
query = """ SELECT
    CASE
        WHEN TIMESTAMPDIFF(YEAR, dob, CURDATE()) < 20 THEN 'Below 20'
        WHEN TIMESTAMPDIFF(YEAR, dob, CURDATE()) BETWEEN 20 AND 29
THEN '20-29'
        WHEN TIMESTAMPDIFF(YEAR, dob, CURDATE()) BETWEEN 30 AND 39
THEN '30-39'
        WHEN TIMESTAMPDIFF(YEAR, dob, CURDATE()) BETWEEN 40 AND 49
THEN '40-49'
        WHEN TIMESTAMPDIFF(YEAR, dob, CURDATE()) BETWEEN 50 AND 59
THEN '50-59'
        WHEN TIMESTAMPDIFF(YEAR, dob, CURDATE()) BETWEEN 60 AND 69
THEN '60-69'
        ELSE '70+'
    END AS age_group,
    COUNT(*) AS fraud_transactions,
    ROUND(AVG(amt), 2) AS avg_fraud_amount
FROM fraudtest
WHERE is_fraud = 1
GROUP BY age_group
ORDER BY age_group """


cur.execute(query)

data = cur.fetchall()
df = pd.DataFrame(data, columns = [ "age_group", "fraud_transactions",
"avg_fraud_amount"])
df
```

	age_group	fraud_transactions	avg_fraud_amount
0	20-29	815	610.10
1	30-39	2085	510.35
2	40-49	1930	462.22
3	50-59	2135	499.89
4	60-69	1710	582.76
5	70+	2050	560.71

What are the top 5 states with the highest number of fraudulent transactions

```
query = """ SELECT
    state,
    COUNT(*) AS fraud_count
FROM
    fraudtest
WHERE
    is_fraud = 1
GROUP BY
    state
ORDER BY
    fraud_count DESC
LIMIT 5 """


cur.execute(query)

data = cur.fetchall()
df = pd.DataFrame(data, columns = [ "state", "fraud_count"])
df
```

	state	fraud_count
0	NY	875
1	PA	570
2	TX	565
3	CA	380
4	IL	380