Data Engineering for Fraud ML Project

Description: This script demonstrates the process of setting up a PostgreSQL database in AWS RDS using SQLAlchemy, and then loading a large dataset into the database with batch inserts. It includes data preprocessing and engineering steps such as data loading, database schema creation, and handling potential issues during the data insertion process. Additionally, the script covers database validation by checking the structure and content of the `fraud_detection` table, ensuring the integrity of the data after insertion. Key libraries used include `psycopg2`, `SQLAlchemy`, and `pandas`.

Prerequisites:

- Install necessary libraries: ```bash pip install psycopg2-binary sqlalchemy

In [81]:
```
pip install psycopg2-binary sqlalchemy
```

```
Requirement already satisfied: psycopg2-binary in c:\users\tanch\anaconda3\lib\site-p
ackages (2.9.9)
Requirement already satisfied: sqlalchemy in c:\users\tanch\anaconda3\lib\site-packag
es (1.4.39)
Requirement already satisfied: greenlet!=0.4.17 in c:\users\tanch\anaconda3\lib\site-
packages (from sqlalchemy) (2.0.1)
Note: you may need to restart the kernel to use updated packages.
```

In [97]:
```
import pandas as pd
import psycopg2
from sqlalchemy import create_engine, text
import time
import os
print("Libraries successfully imported")
```

```
Libraries successfully imported
```

In [83]:
```
username = "ChrisTanML"
password = set_environment_variable_pw  # Store securely or use environment variables
host = "fraud-detection-db.c5aqgaeo2bgr.us-east-2.rds.amazonaws.com"  # RDS endpoint
port = "5432"
dbname = "postgres"  # Use default name "postgres", not dbname "fraud-detection-db" fo

# SQLAlchemy connection string
connection_string = f"postgresql://{username}:{password}@{host}:{port}/{dbname}"

# Create an engine instance
try:
    engine = create_engine(connection_string)
    with engine.connect() as connection:
        result = connection.execute("SELECT version();")
        for row in result:
            print(f"Connected successfully! PostgreSQL version: {row[0]}")
except Exception as e:
    print(f"Error: {e}")
```

```
Connected successfully! PostgreSQL version: PostgreSQL 16.3 on x86_64-pc-linux-gnu, c
ompiled by gcc (GCC) 7.3.1 20180712 (Red Hat 7.3.1-12), 64-bit
```

Load the dataset

In [20]:
```
data = pd.read_csv(r'C:\Users\tanch\bank-account-fraud-dataset-neurips-2022\Base.csv')
```

In [21]:
```
data.head()
```

Out[21]:

	fraud_bool	income	name_email_similarity	prev_address_months_count	current_address_months_cou
0	0	0.3	0.986506	-1	
1	0	0.8	0.617426	-1	
2	0	0.8	0.996707	9	
3	0	0.6	0.475100	11	
4	0	0.9	0.842307	-1	

5 rows × 32 columns

In [4]:
```
# Display basic information about the dataset
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000000 entries, 0 to 999999
Data columns (total 32 columns):
 #   Column                             Non-Null Count    Dtype  
---  ------                             --------------    -----  
 0   fraud_bool                         1000000 non-null  int64  
 1   income                             1000000 non-null  float64
 2   name_email_similarity              1000000 non-null  float64
 3   prev_address_months_count          1000000 non-null  int64  
 4   current_address_months_count       1000000 non-null  int64  
 5   customer_age                       1000000 non-null  int64  
 6   days_since_request                 1000000 non-null  float64
 7   intended_balcon_amount             1000000 non-null  float64
 8   payment_type                       1000000 non-null  object 
 9   zip_count_4w                       1000000 non-null  int64  
 10  velocity_6h                        1000000 non-null  float64
 11  velocity_24h                       1000000 non-null  float64
 12  velocity_4w                        1000000 non-null  float64
 13  bank_branch_count_8w               1000000 non-null  int64  
 14  date_of_birth_distinct_emails_4w   1000000 non-null  int64  
 15  employment_status                  1000000 non-null  object 
 16  credit_risk_score                  1000000 non-null  int64  
 17  email_is_free                      1000000 non-null  int64  
 18  housing_status                     1000000 non-null  object 
 19  phone_home_valid                   1000000 non-null  int64  
 20  phone_mobile_valid                 1000000 non-null  int64  
 21  bank_months_count                  1000000 non-null  int64  
 22  has_other_cards                    1000000 non-null  int64  
 23  proposed_credit_limit              1000000 non-null  float64
 24  foreign_request                    1000000 non-null  int64  
 25  source                             1000000 non-null  object 
 26  session_length_in_minutes          1000000 non-null  float64
 27  device_os                          1000000 non-null  object 
 28  keep_alive_session                 1000000 non-null  int64  
 29  device_distinct_emails_8w          1000000 non-null  int64  
 30  device_fraud_count                 1000000 non-null  int64  
 31  month                              1000000 non-null  int64  
dtypes: float64(9), int64(18), object(5)
memory usage: 244.1+ MB
```

In [5]:
```
# Statistical summary of the dataset
display(data.describe())
```

	fraud_bool	income	name_email_similarity	prev_address_months_count	current_adc
count	1000000.000000	1000000.000000	1000000.000000	1000000.000000	
mean	0.011029	0.562696	0.493694	16.718568	
std	0.104438	0.290343	0.289125	44.046230	
min	0.000000	0.100000	0.000001	-1.000000	
25%	0.000000	0.300000	0.225216	-1.000000	
50%	0.000000	0.600000	0.492153	-1.000000	
75%	0.000000	0.800000	0.755567	12.000000	
max	1.000000	0.900000	0.999999	383.000000	

8 rows × 27 columns

In [22]:
```
# Get the summary statistics
summary = data.describe()

# Identify columns where the minimum value is negative
negative_columns = summary.loc['min'][summary.loc['min'] < 0].index.tolist()

# Display the columns with negative minimum values
print("Columns with negative minimum values:", negative_columns)
```

```
Columns with negative minimum values: ['prev_address_months_count', 'current_address_
months_count', 'intended_balcon_amount', 'velocity_6h', 'credit_risk_score', 'bank_mo
nths_count', 'session_length_in_minutes', 'device_distinct_emails_8w']
```

In [8]:
```
if 'month' in data.columns:
    unique_months = data['month'].unique()
    print("Unique values in the 'month' column:", unique_months)
else:
    print("The 'month' column does not exist in the data.")
```

```
Unique values in the 'month' column: [0 1 2 3 4 5 6 7]
```

Data preprocessing: Replace negative values with the median, encode categorical values, split the data set into training and test sets based on temporal information, and scale the features

In [23]:
```
# Replace negative values with the median of their respective columns
for col in negative_columns:
    median = data[col][data[col] >= 0].median()  # Calculate median ignoring negative
    count_negatives = (data[col] < 0).sum()  # Count negative values
    data[col] = data[col].apply(lambda x: median if x < 0 else x)
    print(f'Column "{col}": {count_negatives} negative values updated to median.')
print("Negative values replaced with medians.")

# Encode categorical features
data = pd.get_dummies(data)
print("Categorical features encoded.")
```

```
# Split the dataset based on temporal information (months 0-5 for training, months 6-7
train_data = data[data['month'] <= 5] # First 6 months (month 0 thru month 5)
test_data = data[data['month'] >= 6] # Last 2 months (month 6 & 7)
print("Dataset split based on temporal information.")

# Split the dataset into training and test sets
X_train, X_test, y_train, y_test = train_test_split(data.drop('fraud_bool', axis=1), d
print("Dataset split into training and test sets.")

# Scale the features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

print("Feature scaling completed.")
```

```
Column "prev_address_months_count": 712920 negative values updated to median.
Column "current_address_months_count": 4254 negative values updated to median.
Column "intended_balcon_amount": 742523 negative values updated to median.
Column "velocity_6h": 44 negative values updated to median.
Column "credit_risk_score": 14445 negative values updated to median.
Column "bank_months_count": 253635 negative values updated to median.
Column "session_length_in_minutes": 2015 negative values updated to median.
Column "device_distinct_emails_8w": 359 negative values updated to median.
Negative values replaced with medians.
Categorical features encoded.
Dataset split based on temporal information.
Dataset split into training and test sets.
Feature scaling completed.
```

In [24]:
```
data.to_csv('data_cleaned.csv', index=False)
print("Dataset saved as 'data_cleaned.csv'")
```

```
Dataset saved as 'data_cleaned.csv'
```

In [26]:
```
data_clean = pd.read_csv('data_cleaned.csv')
```

In [28]:
```
data_clean.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000000 entries, 0 to 999999
Data columns (total 53 columns):
 #   Column                              Non-Null Count    Dtype
---  ------                              --------------    -----
 0   fraud_bool                          1000000 non-null  int64
 1   income                              1000000 non-null  float64
 2   name_email_similarity               1000000 non-null  float64
 3   prev_address_months_count           1000000 non-null  float64
 4   current_address_months_count        1000000 non-null  float64
 5   customer_age                        1000000 non-null  int64
 6   days_since_request                  1000000 non-null  float64
 7   intended_balcon_amount              1000000 non-null  float64
 8   zip_count_4w                        1000000 non-null  int64
 9   velocity_6h                         1000000 non-null  float64
 10  velocity_24h                        1000000 non-null  float64
 11  velocity_4w                         1000000 non-null  float64
 12  bank_branch_count_8w                1000000 non-null  int64
 13  date_of_birth_distinct_emails_4w    1000000 non-null  int64
 14  credit_risk_score                   1000000 non-null  float64
 15  email_is_free                       1000000 non-null  int64
 16  phone_home_valid                    1000000 non-null  int64
 17  phone_mobile_valid                  1000000 non-null  int64
 18  bank_months_count                   1000000 non-null  float64
 19  has_other_cards                     1000000 non-null  int64
 20  proposed_credit_limit               1000000 non-null  float64
 21  foreign_request                     1000000 non-null  int64
 22  session_length_in_minutes           1000000 non-null  float64
 23  keep_alive_session                  1000000 non-null  int64
 24  device_distinct_emails_8w           1000000 non-null  float64
 25  device_fraud_count                  1000000 non-null  int64
 26  month                               1000000 non-null  int64
 27  payment_type_AA                     1000000 non-null  int64
 28  payment_type_AB                     1000000 non-null  int64
 29  payment_type_AC                     1000000 non-null  int64
 30  payment_type_AD                     1000000 non-null  int64
 31  payment_type_AE                     1000000 non-null  int64
 32  employment_status_CA                1000000 non-null  int64
 33  employment_status_CB                1000000 non-null  int64
 34  employment_status_CC                1000000 non-null  int64
 35  employment_status_CD                1000000 non-null  int64
 36  employment_status_CE                1000000 non-null  int64
 37  employment_status_CF                1000000 non-null  int64
 38  employment_status_CG                1000000 non-null  int64
 39  housing_status_BA                   1000000 non-null  int64
 40  housing_status_BB                   1000000 non-null  int64
 41  housing_status_BC                   1000000 non-null  int64
 42  housing_status_BD                   1000000 non-null  int64
 43  housing_status_BE                   1000000 non-null  int64
 44  housing_status_BF                   1000000 non-null  int64
 45  housing_status_BG                   1000000 non-null  int64
 46  source_INTERNET                     1000000 non-null  int64
 47  source_TELEAPP                      1000000 non-null  int64
 48  device_os_linux                     1000000 non-null  int64
 49  device_os_macintosh                 1000000 non-null  int64
 50  device_os_other                     1000000 non-null  int64
 51  device_os_windows                   1000000 non-null  int64
 52  device_os_x11                       1000000 non-null  int64
dtypes: float64(14), int64(39)
memory usage: 404.4 MB
```

In [39]:

```
# Create an engine instance and connect to the database
try:
    engine = create_engine(connection_string)

    # Connect to the PostgreSQL database using the engine
    with engine.connect() as connection:
        # Check connection by executing a simple query
        result = connection.execute(text("SELECT version();"))
        for row in result:
            print(f"Connected successfully! PostgreSQL version: {row[0]}")

# SQL query to create the table
        create_table_query = '''
        CREATE TABLE fraud_detection (
            id SERIAL PRIMARY KEY,  -- Unique identifier for each row
            fraud_bool INTEGER NOT NULL,
            income FLOAT8 NOT NULL,
            name_email_similarity FLOAT8 NOT NULL,
            prev_address_months_count FLOAT8 NOT NULL,
            current_address_months_count FLOAT8 NOT NULL,
            customer_age INTEGER NOT NULL,
            days_since_request FLOAT8 NOT NULL,
            intended_balcon_amount FLOAT8 NOT NULL,
            zip_count_4w INTEGER NOT NULL,
            velocity_6h FLOAT8 NOT NULL,
            velocity_24h FLOAT8 NOT NULL,
            velocity_4w FLOAT8 NOT NULL,
            bank_branch_count_8w INTEGER NOT NULL,
            date_of_birth_distinct_emails_4w INTEGER NOT NULL,
            credit_risk_score FLOAT8 NOT NULL,
            email_is_free INTEGER NOT NULL,
            phone_home_valid INTEGER NOT NULL,
            phone_mobile_valid INTEGER NOT NULL,
            bank_months_count FLOAT8 NOT NULL,
            has_other_cards INTEGER NOT NULL,
            proposed_credit_limit FLOAT8 NOT NULL,
            foreign_request INTEGER NOT NULL,
            session_length_in_minutes FLOAT8 NOT NULL,
            keep_alive_session INTEGER NOT NULL,
            device_distinct_emails_8w FLOAT8 NOT NULL,
            device_fraud_count INTEGER NOT NULL,
            month INTEGER NOT NULL,
            payment_type_AA INTEGER NOT NULL,
            payment_type_AB INTEGER NOT NULL,
            payment_type_AC INTEGER NOT NULL,
            payment_type_AD INTEGER NOT NULL,
            payment_type_AE INTEGER NOT NULL,
            employment_status_CA INTEGER NOT NULL,
            employment_status_CB INTEGER NOT NULL,
            employment_status_CC INTEGER NOT NULL,
            employment_status_CD INTEGER NOT NULL,
            employment_status_CE INTEGER NOT NULL,
            employment_status_CF INTEGER NOT NULL,
            employment_status_CG INTEGER NOT NULL,
            housing_status_BA INTEGER NOT NULL,
            housing_status_BB INTEGER NOT NULL,
            housing_status_BC INTEGER NOT NULL,
            housing_status_BD INTEGER NOT NULL,
            housing_status_BE INTEGER NOT NULL,
```

```
            housing_status_BF INTEGER NOT NULL,
            housing_status_BG INTEGER NOT NULL,
            source_INTERNET INTEGER NOT NULL,
            source_TELEAPP INTEGER NOT NULL,
            device_os_linux INTEGER NOT NULL,
            device_os_macintosh INTEGER NOT NULL,
            device_os_other INTEGER NOT NULL,
            device_os_windows INTEGER NOT NULL,
            device_os_x11 INTEGER NOT NULL
        );
        '''

        # Execute the create table query
        connection.execute(text(create_table_query))
        print("Table 'fraud_detection' created successfully.")

        # Verification query to fetch the top ten rows
        verification_query = text("SELECT * FROM fraud_detection;")
        result = connection.execute(verification_query)
        rows = result.fetchall()

except Exception as e:
    print(f"Error: {e}")
finally:
    # It's good practice to dispose of the engine when done
    engine.dispose()
    print("Database connection closed.")
```

```
Connected successfully! PostgreSQL version: PostgreSQL 16.3 on x86_64-pc-linux-gnu, c
ompiled by gcc (GCC) 7.3.1 20180712 (Red Hat 7.3.1-12), 64-bit
Table 'fraud_detection' created successfully.
Database connection closed.
```

In [41]:

```
# Query to Check Table Structure
with engine.connect() as connection:
    result = connection.execute(text("SELECT column_name, data_type FROM information_s
    columns = result.fetchall()
    print("fraud_detection table structure:")
    for column in columns:
        print(column)
    engine.dispose()
    print("Database connection closed.")
```

```
fraud_detection table structure:
('id', 'integer')
('fraud_bool', 'integer')
('income', 'double precision')
('name_email_similarity', 'double precision')
('prev_address_months_count', 'double precision')
('current_address_months_count', 'double precision')
('customer_age', 'integer')
('days_since_request', 'double precision')
('intended_balcon_amount', 'double precision')
('zip_count_4w', 'integer')
('velocity_6h', 'double precision')
('velocity_24h', 'double precision')
('velocity_4w', 'double precision')
('bank_branch_count_8w', 'integer')
('date_of_birth_distinct_emails_4w', 'integer')
('credit_risk_score', 'double precision')
('email_is_free', 'integer')
('phone_home_valid', 'integer')
('phone_mobile_valid', 'integer')
('bank_months_count', 'double precision')
('has_other_cards', 'integer')
('proposed_credit_limit', 'double precision')
('foreign_request', 'integer')
('session_length_in_minutes', 'double precision')
('keep_alive_session', 'integer')
('device_distinct_emails_8w', 'double precision')
('device_fraud_count', 'integer')
('month', 'integer')
('payment_type_aa', 'integer')
('payment_type_ab', 'integer')
('payment_type_ac', 'integer')
('payment_type_ad', 'integer')
('payment_type_ae', 'integer')
('employment_status_ca', 'integer')
('employment_status_cb', 'integer')
('employment_status_cc', 'integer')
('employment_status_cd', 'integer')
('employment_status_ce', 'integer')
('employment_status_cf', 'integer')
('employment_status_cg', 'integer')
('housing_status_ba', 'integer')
('housing_status_bb', 'integer')
('housing_status_bc', 'integer')
('housing_status_bd', 'integer')
('housing_status_be', 'integer')
('housing_status_bf', 'integer')
('housing_status_bg', 'integer')
('source_internet', 'integer')
('source_teleapp', 'integer')
('device_os_linux', 'integer')
('device_os_macintosh', 'integer')
('device_os_other', 'integer')
('device_os_windows', 'integer')
('device_os_x11', 'integer')
Database connection closed.
```

In [71]:

```python
# Define the batch size and progress step
batch_size = 1000  # Adjust this value according to your system's capacity
progress_step = 100000
```

```
# Create an engine instance and connect to the database
try:
    engine = create_engine(connection_string)

    # Connect to the PostgreSQL database using the engine
    with engine.connect() as connection:
        # Check connection by executing a simple query
        result = connection.execute(text("SELECT version();"))
        for row in result:
            print(f"Connected successfully! PostgreSQL version: {row[0]}")

        # Truncate the table to remove all existing data
        connection.execute(text("TRUNCATE TABLE fraud_detection;"))
        print("Table fraud_detection has been truncated.")

        # Start timing the data insertion process
        start_time = time.time()

        # Insert data into the PostgreSQL database using batch inserts
        insert_query = '''
        INSERT INTO fraud_detection (
            fraud_bool,
            income,
            name_email_similarity,
            prev_address_months_count,
            current_address_months_count,
            customer_age,
            days_since_request,
            intended_balcon_amount,
            zip_count_4w,
            velocity_6h,
            velocity_24h,
            velocity_4w,
            bank_branch_count_8w,
            date_of_birth_distinct_emails_4w,
            credit_risk_score,
            email_is_free,
            phone_home_valid,
            phone_mobile_valid,
            bank_months_count,
            has_other_cards,
            proposed_credit_limit,
            foreign_request,
            session_length_in_minutes,
            keep_alive_session,
            device_distinct_emails_8w,
            device_fraud_count,
            "month",
            payment_type_AA,
            payment_type_AB,
            payment_type_AC,
            payment_type_AD,
            payment_type_AE,
            employment_status_CA,
            employment_status_CB,
            employment_status_CC,
            employment_status_CD,
            employment_status_CE,
            employment_status_CF,
```

```
            employment_status_CG,
            housing_status_BA,
            housing_status_BB,
            housing_status_BC,
            housing_status_BD,
            housing_status_BE,
            housing_status_BF,
            housing_status_BG,
            source_INTERNET,
            source_TELEAPP,
            device_os_linux,
            device_os_macintosh,
            device_os_other,
            device_os_windows,
            device_os_x11
        ) VALUES (
            %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s
            %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s
            %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s
        );
        '''

        # Split data into batches and insert
        num_batches = len(data_clean) // batch_size + 1
        total_inserted = 0

        for i in range(num_batches):
            batch = data_clean.iloc[i * batch_size:(i + 1) * batch_size]
            connection.execute(text(insert_query), batch.values.tolist()) # Is this th

            # Update the total number of rows inserted
            total_inserted += len(batch)

            # Print progress every 100,000 rows
            if total_inserted % progress_step == 0:
                print(f"{total_inserted} rows inserted successfully.")

        # End timing the data insertion process
        end_time = time.time()
        elapsed_time = end_time - start_time
        print(f"Data inserted successfully in {elapsed_time:.2f} seconds.")

except Exception as e:
    print(f"Error: {e}")
finally:
    # It's good practice to dispose of the engine when done
    engine.dispose()
    print("Database connection closed.")
```

```
Connected successfully! PostgreSQL version: PostgreSQL 16.3 on x86_64-pc-linux-gnu, c
ompiled by gcc (GCC) 7.3.1 20180712 (Red Hat 7.3.1-12), 64-bit
Table fraud_detection has been truncated.
100000 rows inserted successfully.
200000 rows inserted successfully.
300000 rows inserted successfully.
400000 rows inserted successfully.
500000 rows inserted successfully.
600000 rows inserted successfully.
700000 rows inserted successfully.
800000 rows inserted successfully.
900000 rows inserted successfully.
1000000 rows inserted successfully.
Error: dict is not a sequence
Database connection closed.
```

In [89]:

```python
try:
    engine = create_engine(connection_string)

    with engine.connect() as connection:
        # Execute the query to count the number of columns
        result = connection.execute(text(
            "SELECT COUNT(*) FROM information_schema.columns WHERE table_name = 'fraud
        ))

        column_count = result.scalar()
        print(f"Number of columns in the 'fraud_detection' table: {column_count}")

        # Query to check the table structure
        result = connection.execute(text(
            "SELECT column_name, data_type FROM information_schema.columns WHERE table
        ))
        columns = result.fetchall()

        print("fraud_detection table structure:")
        for column in columns:
            print(column)

        # Query to count the total number of rows
        result = connection.execute(text(
            "SELECT COUNT(*) FROM fraud_detection;"
        ))
        row_count = result.scalar()  # fetch the single value (count) from the result

        print(f"Total number of rows in fraud_detection table: {row_count}")

         # Query to count the total number of duplicates
        fetch_duplicates = "SELECT id, COUNT(*) FROM fraud_detection GROUP BY id HAVIN
        result = connection.execute(text(fetch_duplicates))
        columns = result.fetchall()

        print("duplicates:")
        for column in columns:
            print(column)

except Exception as e:
    print(f"Error: {e}")
finally:
```

```
    engine.dispose()
    print("Database connection closed.")
```

```
Number of columns in the 'fraud_detection' table: 54
fraud_detection table structure:
('id', 'integer')
('fraud_bool', 'integer')
('income', 'double precision')
('name_email_similarity', 'double precision')
('prev_address_months_count', 'double precision')
('current_address_months_count', 'double precision')
('customer_age', 'integer')
('days_since_request', 'double precision')
('intended_balcon_amount', 'double precision')
('zip_count_4w', 'integer')
('velocity_6h', 'double precision')
('velocity_24h', 'double precision')
('velocity_4w', 'double precision')
('bank_branch_count_8w', 'integer')
('date_of_birth_distinct_emails_4w', 'integer')
('credit_risk_score', 'double precision')
('email_is_free', 'integer')
('phone_home_valid', 'integer')
('phone_mobile_valid', 'integer')
('bank_months_count', 'double precision')
('has_other_cards', 'integer')
('proposed_credit_limit', 'double precision')
('foreign_request', 'integer')
('session_length_in_minutes', 'double precision')
('keep_alive_session', 'integer')
('device_distinct_emails_8w', 'double precision')
('device_fraud_count', 'integer')
('month', 'integer')
('payment_type_aa', 'integer')
('payment_type_ab', 'integer')
('payment_type_ac', 'integer')
('payment_type_ad', 'integer')
('payment_type_ae', 'integer')
('employment_status_ca', 'integer')
('employment_status_cb', 'integer')
('employment_status_cc', 'integer')
('employment_status_cd', 'integer')
('employment_status_ce', 'integer')
('employment_status_cf', 'integer')
('employment_status_cg', 'integer')
('housing_status_ba', 'integer')
('housing_status_bb', 'integer')
('housing_status_bc', 'integer')
('housing_status_bd', 'integer')
('housing_status_be', 'integer')
('housing_status_bf', 'integer')
('housing_status_bg', 'integer')
('source_internet', 'integer')
('source_teleapp', 'integer')
('device_os_linux', 'integer')
('device_os_macintosh', 'integer')
('device_os_other', 'integer')
('device_os_windows', 'integer')
('device_os_x11', 'integer')
Total number of rows in fraud_detection table: 1000000
duplicates:
fraud_detection table structure:
```

```
(1000000,)
Database connection closed.
```