Data Cleaning & Transformation

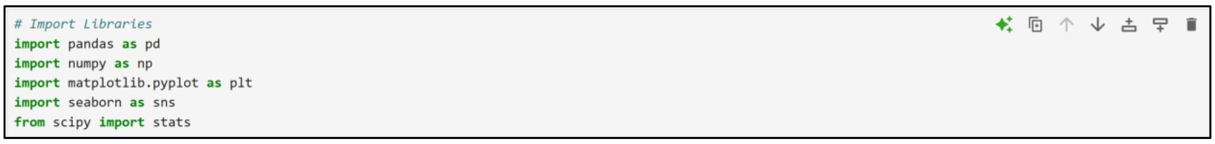
The following dataset simulates credit card transactions, including both legitimate and fraudulent activities, recorded between January 1, 2019, and December 31, 2020. It includes transactions from 1,000 customers across a network of 800 merchants. The data is divided into two subsets: Train and Test, both of which share the same column structure, allowing them to be seamlessly merged into a single dataset. Below is a brief description of each column.

- **Unnamed: 0** – An index column, likely unnecessary for analysis.
- **trans_date_trans_time** – The date and time of the transaction.
- **cc_num** – The credit card number used for the transaction.
- **merchant** – The merchant where the transaction occurred.
- **category** – The category of the transaction (e.g., travel, personal care, health).
- **amt** – The transaction amount in USD.
- **first** – The first name of the cardholder.
- **last** – The last name of the cardholder.
- **gender** – The gender of the cardholder.
- **street** – The street address of the cardholder.
- **city** – The city of the cardholder.
- **state** – The state where the cardholder resides.
- **zip** – The ZIP code of the cardholder.
- **lat** – The latitude of the cardholder's home location.
- **long** – The longitude of the cardholder's home location.
- **city_pop** – The population of the city where the cardholder lives.
- **job** – The job title of the cardholder.
- **dob** – The date of birth of the cardholder.
- **trans_num** – A unique identifier for each transaction.
- **unix_time** – The timestamp of the transaction in Unix format.
- **merch_lat** – The latitude of the merchant's location.
- **merch_long** – The longitude of the merchant's location.
- **is_fraud** – A binary indicator (0 = not fraud, 1 = fraud) for fraudulent transactions.

First, the two datasets are imported and merged into a single, unified dataset. Then, data cleaning and preprocessing are performed to ensure consistency and improve interpretability, providing a clearer understanding of each variable. Additionally, necessary transformations are applied to structure the data more effectively and enhance its analytical value.

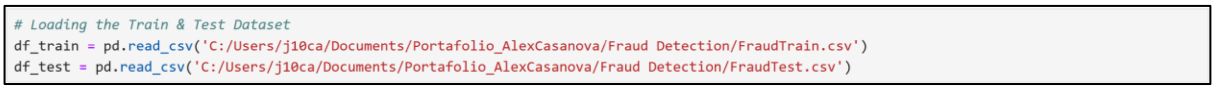
Import & Clean.

Import libraries.



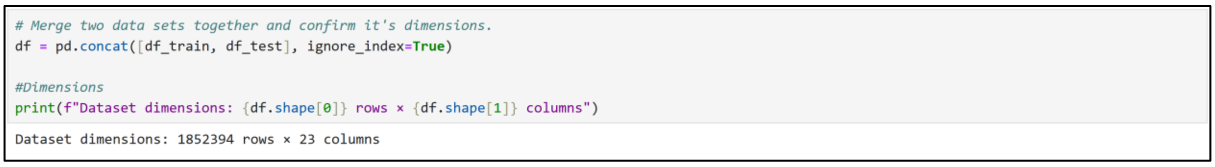
```
# Import Libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from scipy import stats
```

Load the train and test dataset for this fraud dataset.



```
# Loading the Train & Test Dataset
df_train = pd.read_csv('C:/Users/j10ca/Documents/Portafolio_AlexCasanova/Fraud Detection/FraudTrain.csv')
df_test = pd.read_csv('C:/Users/j10ca/Documents/Portafolio_AlexCasanova/Fraud Detection/FraudTest.csv')
```

Merge the two datasets together and confirm the shape.

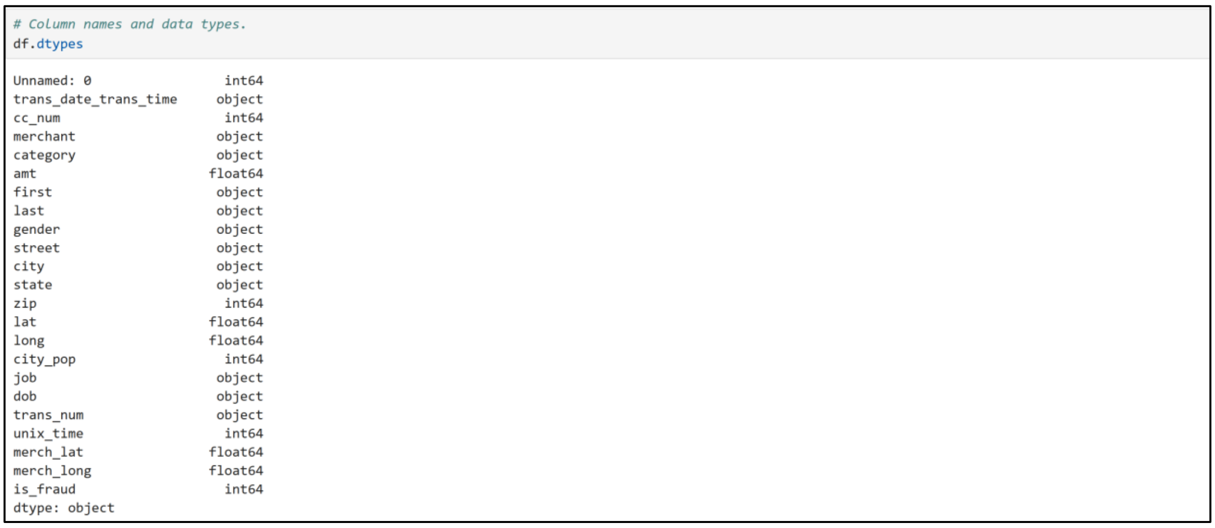


```
# Merge two data sets together and confirm it's dimensions.
df = pd.concat([df_train, df_test], ignore_index=True)

#Dimensions
print(f"Dataset dimensions: {df.shape[0]} rows × {df.shape[1]} columns")
```

```
Dataset dimensions: 1852394 rows × 23 columns
```

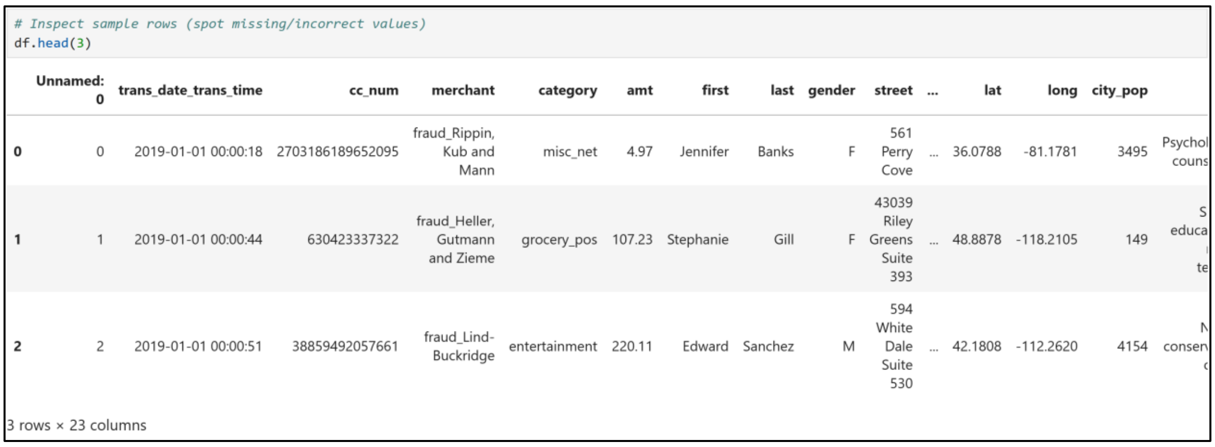
Confirm the name columns and data types for DataFrame '**df**'.



```
# Column names and data types.
df.dtypes
```

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

Displays the first few rows of the DataFrame '**df**'.

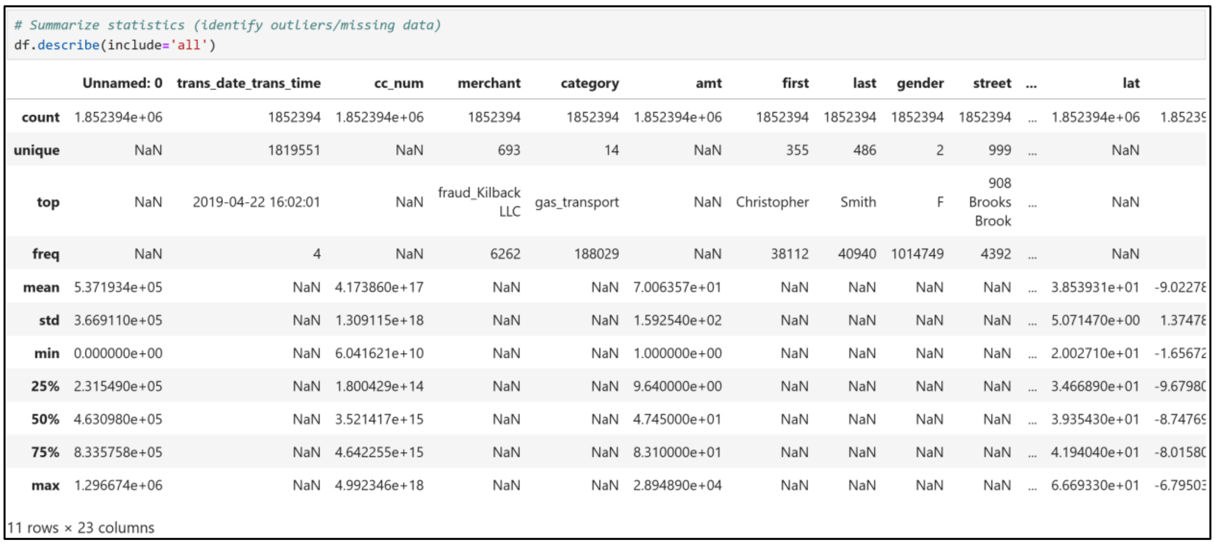


```
# Inspect sample rows (spot missing/incorrect values)
df.head(3)
```

	Unnamed: 0	trans_date_trans_time	cc_num	merchant	category	amt	first	last	gender	street	...	lat	long	city_pop	
0	0	2019-01-01 00:00:18	2703186189652095	fraud_Rippin, Kub and Mann	misc_net	4.97	Jennifer	Banks	F	561 Perry Cove	...	36.0788	-81.1781	3495	Psychol couns
1	1	2019-01-01 00:00:44	630423337322	fraud_Heller, Gutmann and Zieme	grocery_pos	107.23	Stephanie	Gill	F	43039 Riley Greens Suite 393	...	48.8878	-118.2105	149	S educa l te
2	2	2019-01-01 00:00:51	38859492057661	fraud_Lind-Buckridge	entertainment	220.11	Edward	Sanchez	M	594 White Dale Suite 530	...	42.1808	-112.2620	4154	N consen c

3 rows × 23 columns

Generates summary statistics for all columns in DataFrame ‘**df**’.

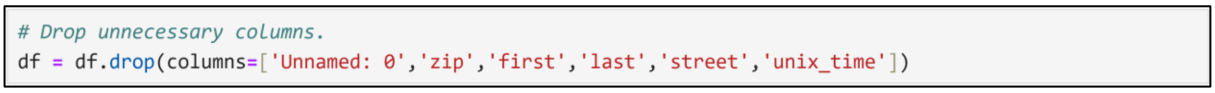


```python
# Summarize statistics (identify outliers/missing data)
df.describe(include='all')
```

	Unnamed: 0	trans_date_trans_time	cc_num	merchant	category	amt	first	last	gender	street	...	lat	
count	1.852394e+06	1852394	1.852394e+06	1852394	1852394	1.852394e+06	1852394	1852394	1852394	1852394	...	1.852394e+06	1.85239
unique	NaN	1819551	NaN	693	14	NaN	355	486	2	999	...	NaN	
top	NaN	2019-04-22 16:02:01	NaN	fraud_Kilback LLC	gas_transport	NaN	Christopher	Smith	F	908 Brooks Brook	...	NaN	
freq	NaN	4	NaN	6262	188029	NaN	38112	40940	1014749	4392	...	NaN	
mean	5.371934e+05	NaN	4.173860e+17	NaN	NaN	7.006357e+01	NaN	NaN	NaN	NaN	...	3.853931e+01	-9.0227
std	3.669110e+05	NaN	1.309115e+18	NaN	NaN	1.592540e+02	NaN	NaN	NaN	NaN	...	5.071470e+00	1.3747
min	0.000000e+00	NaN	6.041621e+10	NaN	NaN	1.000000e+00	NaN	NaN	NaN	NaN	...	2.002710e+01	-1.6567
25%	2.315490e+05	NaN	1.800429e+14	NaN	NaN	9.640000e+00	NaN	NaN	NaN	NaN	...	3.466890e+01	-9.6798
50%	4.630980e+05	NaN	3.521417e+15	NaN	NaN	4.745000e+01	NaN	NaN	NaN	NaN	...	3.935430e+01	-8.7476
75%	8.335758e+05	NaN	4.642255e+15	NaN	NaN	8.310000e+01	NaN	NaN	NaN	NaN	...	4.194040e+01	-8.0158
max	1.296674e+06	NaN	4.992346e+18	NaN	NaN	2.894890e+04	NaN	NaN	NaN	NaN	...	6.669330e+01	-6.7950

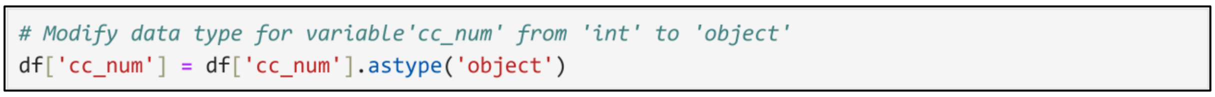
11 rows × 23 columns

Drop unnecessary columns from DataFrame ‘**df**’. Columns such as: Unnamed: 0, zip, first, last, street and unix_time. These columns don’t provide any value to analysis.

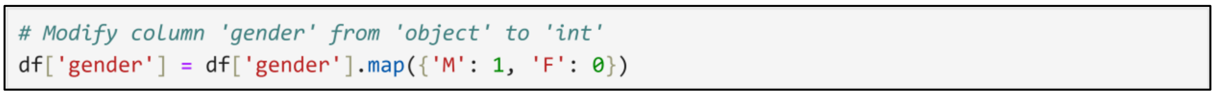


```python
# Drop unnecessary columns.
df = df.drop(columns=['Unnamed: 0','zip','first','last','street','unix_time'])
```

Modify the data types for the ‘cc_num’ and ‘gender’ columns. Convert cc_num to a string (object) to preserve its full numerical value and avoid truncation. Change gender to an integer type with binary values (0 and 1).

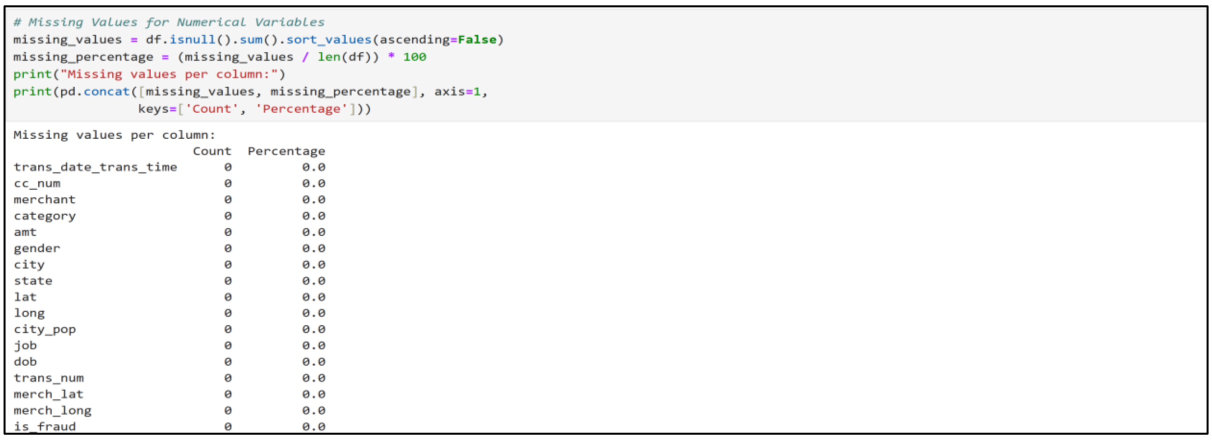


```python
# Modify data type for variable'cc_num' from 'int' to 'object'
df['cc_num'] = df['cc_num'].astype('object')
```



```python
# Modify column 'gender' from 'object' to 'int'
df['gender'] = df['gender'].map({'M': 1, 'F': 0})
```

Continue with an analysis of missing variables for both numerical and categorical variables. This analysis ensures data quality, prevents biased results, and guides appropriate cleaning strategies like imputation or removal, maintaining dataset reliability for accurate analysis and modeling.



```python
# Missing Values for Numerical Variables
missing_values = df.isnull().sum().sort_values(ascending=False)
missing_percentage = (missing_values / len(df)) * 100
print("Missing values per column:")
print(pd.concat([missing_values, missing_percentage], axis=1,
                keys=['Count', 'Percentage']))
```

```
Missing values per column:
                       Count  Percentage
trans_date_trans_time      0         0.0
cc_num                     0         0.0
merchant                   0         0.0
category                   0         0.0
amt                        0         0.0
gender                     0         0.0
city                       0         0.0
state                      0         0.0
lat                        0         0.0
long                       0         0.0
city_pop                   0         0.0
job                        0         0.0
dob                        0         0.0
trans_num                  0         0.0
merch_lat                  0         0.0
merch_long                 0         0.0
is_fraud                   0         0.0
```

```
# Missing Values for Categorical Variables
# Filter categorical columns (object and bool)
cat_columns = df.select_dtypes(include=['object', 'bool']).columns

# Calculate missing values for categorical columns only
missing_cat = df[cat_columns].isnull().sum().sort_values(ascending=False)
missing_cat_percentage = (missing_cat / len(df)) * 100

print("Missing values in categorical columns:")
print(pd.concat([missing_cat, missing_cat_percentage], axis=1,
                keys=['Count', 'Percentage']))
```

```
Missing values in categorical columns:
                       Count  Percentage
trans_date_trans_time      0         0.0
cc_num                     0         0.0
merchant                   0         0.0
category                   0         0.0
city                       0         0.0
state                      0         0.0
job                        0         0.0
dob                        0         0.0
trans_num                  0         0.0
```

Now it is appropriate to proceed with a deduplication to ensure data accuracy, prevent skewed analysis, and optimize performance by eliminating redundant entries that can distort statistical results and waste computational resources.

```
# Deduplication
# Check for exact duplicates
duplicates = df.duplicated()
print(f"Count of exact duplicate rows: {duplicates.sum()}")
```

```
Count of exact duplicate rows: 0
```

Column Transformations.

Now that we've familiarized ourselves with the DataFrame—including its columns and data types—and completed essential preprocessing steps such as removing unnecessary columns, checking for missing values, and identifying duplicates, we can proceed with a detailed analysis of our target columns and perform the necessary transformations.

Proceed to standardize all string columns in the DataFrame ‘**df**’ by converting values to lowercase and trimming whitespace, ensuring consistency for analysis, grouping, or machine learning. It prevents errors from mixed formats (e.g., "USA" vs. "usa") and reduces noise, making the data cleaner and more reliable.

```
# Standardize all string columns in the DataFrame ‘df’
# Select object/string columns
categorical_columns = df.select_dtypes(include=['object', 'string']).columns

# Ensure all values are strings, then clean
df[categorical_columns] = df[categorical_columns].astype(str).apply(lambda x: x.str.lower().str.strip())

df.head(3)
```

	trans_date_trans_time	cc_num	merchant	category	amt	gender	city	state	lat	long	city_pop	job	dob	
0	2019-01-01 00:00:18	2703186189652095	fraud_rippin, kub and mann	misc_net	4.97	0	moravian falls	nc	36.0788	-81.1781	3495	psychologist, counselling	1988-03-09	0b242a
1	2019-01-01 00:00:44	630423337322	fraud_heller, gutmann and zieme	grocery_pos	107.23	0	orient	wa	48.8878	-118.2105	149	special educational needs teacher	1978-06-21	1f7652
2	2019-01-01 00:00:51	38859492057661	fraud_lind-buckridge	entertainment	220.11	1	malad city	id	42.1808	-112.2620	4154	nature conservation officer	1962-01-19	a1a22c

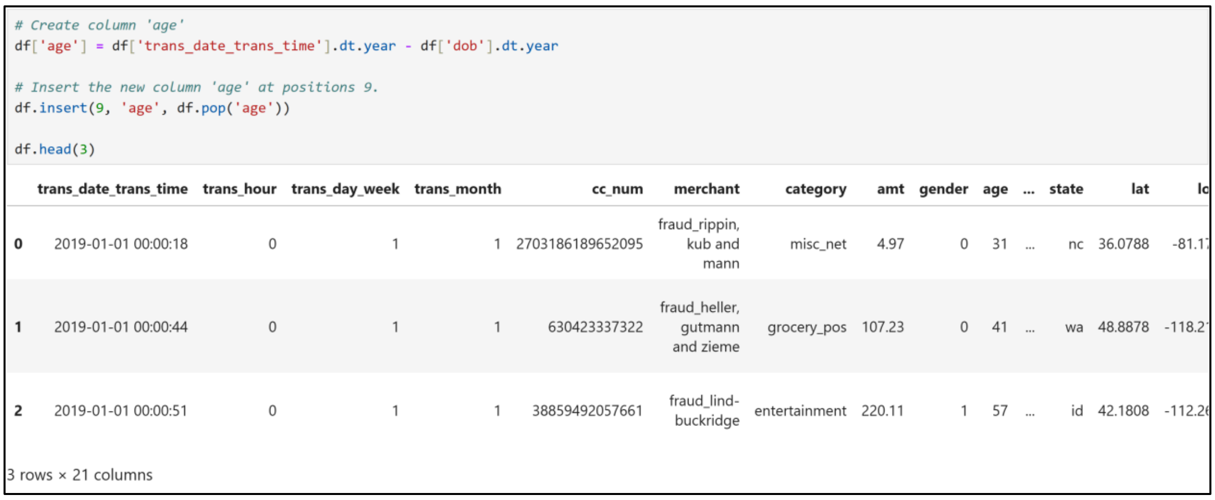
Now convert datetime strings, ‘**trans_date_trans_time**’ and ‘**dob’**, into Pandas datetime objects for proper time-based analysis. Then extracts key temporal features (hour, day of week, month) from the transaction timestamp and inserts them as new columns at strategic positions (columns 1–3) in the DataFrame.

```python
# Transform 'trans_date_trans_time' and 'dob' to datetime
df['trans_date_trans_time'] = pd.to_datetime(df['trans_date_trans_time'])
df['dob'] = pd.to_datetime(df['dob'])

# Extract temporary components
df['trans_hour'] = df['trans_date_trans_time'].dt.hour
df['trans_day_week'] = df['trans_date_trans_time'].dt.dayofweek
df['trans_month'] = df['trans_date_trans_time'].dt.month

# Insert the new columns at positions 1, 2 and 3
df.insert(1, 'trans_hour', df.pop('trans_hour'))
df.insert(2, 'trans_day_week', df.pop('trans_day_week'))
df.insert(3, 'trans_month', df.pop('trans_month'))
```

Now that we’ve converted ‘**trans_date_trans_time**’ and ‘**dob**’ to datetime format, we can efficiently derive the new variable age. This addition will significantly enhance our demographic analysis in the DataFrame ‘**df**’, enabling deeper insights into age-related trends and patterns.



```python
# Create column 'age'
df['age'] = df['trans_date_trans_time'].dt.year - df['dob'].dt.year

# Insert the new column 'age' at positions 9.
df.insert(9, 'age', df.pop('age'))

df.head(3)
```

	trans_date_trans_time	trans_hour	trans_day_week	trans_month	cc_num	merchant	category	amt	gender	age	...	state	lat	lo
0	2019-01-01 00:00:18	0	1	1	2703186189652095	fraud_rippin, kub and mann	misc_net	4.97	0	31	...	nc	36.0788	-81.17
1	2019-01-01 00:00:44	0	1	1	630423337322	fraud_heller, gutmann and zieme	grocery_pos	107.23	0	41	...	wa	48.8878	-118.21
2	2019-01-01 00:00:51	0	1	1	38859492057661	fraud_lind-buckridge	entertainment	220.11	1	57	...	id	42.1808	-112.26

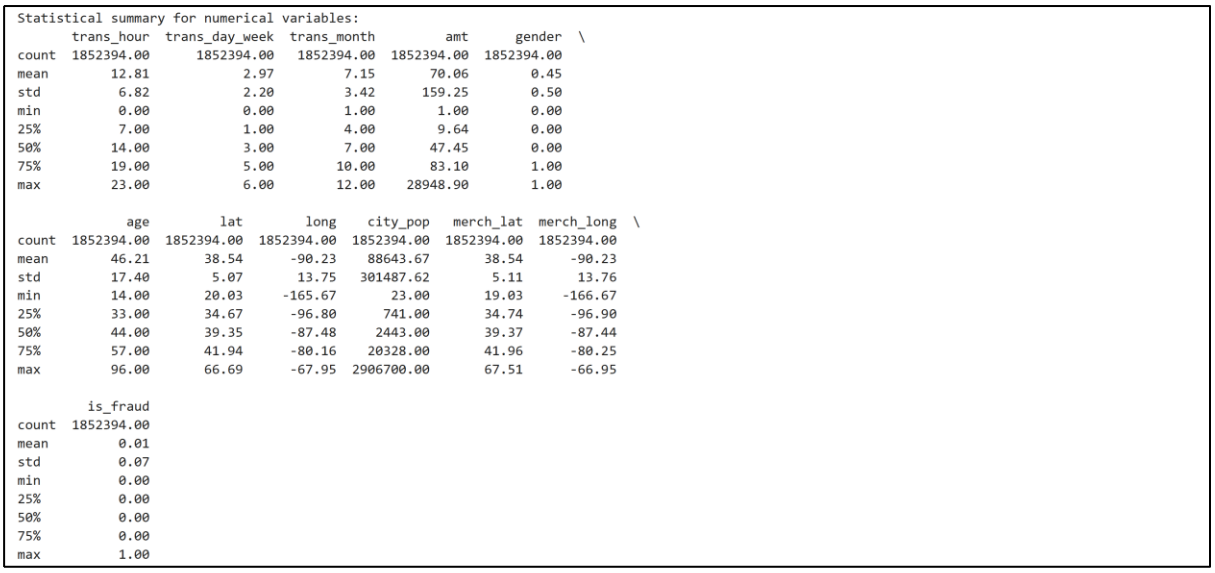
3 rows × 21 columns

In the final steps of our data cleaning and transformation project, we will generate a statistical summary of the numerical columns in the DataFrame ‘**df**’. This analysis will provide deeper insights into the data, helping us better understand its characteristics before further analysis.

```python
# Summary statistics for numerical columns

# Select only numerical columns
df_numeric = df.select_dtypes(include=['int','float'])

# Print the statistical summary with max two decimals for clarity
print("\nStatistical summary for numerical variables:")
print(df_numeric.describe().round(2))
```



```
Statistical summary for numerical variables:
       trans_hour  trans_day_week  trans_month         amt      gender  \
count  1852394.00      1852394.00   1852394.00  1852394.00  1852394.00
mean        12.81            2.97         7.15       70.06        0.45
std          6.82            2.20         3.42      159.25        0.50
min          0.00            0.00         1.00        1.00        0.00
25%          7.00            1.00         4.00        9.64        0.00
50%         14.00            3.00         7.00       47.45        0.00
75%         19.00            5.00        10.00       83.10        1.00
max         23.00            6.00        12.00    28948.90        1.00

              age         lat        long    city_pop   merch_lat  merch_long  \
count  1852394.00  1852394.00  1852394.00  1852394.00  1852394.00  1852394.00
mean        46.21       38.54      -90.23    88643.67       38.54      -90.23
std         17.40        5.07       13.75   301487.62        5.11       13.76
min         14.00       20.03     -165.67       23.00       19.03     -166.67
25%         33.00       34.67      -96.80      741.00       34.74      -96.90
50%         44.00       39.35      -87.48     2443.00       39.37      -87.44
75%         57.00       41.94      -80.16    20328.00       41.96      -80.25
max         96.00       66.69      -67.95  2906700.00       67.51      -66.95

         is_fraud
count  1852394.00
mean         0.01
std          0.07
min          0.00
25%          0.00
50%          0.00
75%          0.00
max          1.00
```

Summary statistics of numerical columns:

- The average transaction is **$70.06**, but there's a large spread with a maximum transaction of **$28,948.90**.
- The average cardholder age is **46.21 years**, ranging from **13 to 96 years**.
- The median city population is **88,643.67**, indicating that many transactions happen in smaller towns.
- Fraud cases are quite rare, with a mean value of **0.01**, indicating only about **1%** of transactions are fraudulent.