Credit Card Fraud Detection

This 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.

```python
# 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.

```python
# 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.

```python
# Merge two data sets together and confirm the shape.
df = pd.concat([df_train, df_test],ignore_index=True)
df.shape
```

```
(1852394, 23)
```

Let's review the general information from the data frame '**df**'.

```python
# General info from the Dataframe
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1852394 entries, 0 to 1852393
Data columns (total 23 columns):
 #   Column                 Dtype
---  ------                 -----
 0   Unnamed: 0             int64
 1   trans_date_trans_time  object
 2   cc_num                 int64
 3   merchant               object
 4   category               object
 5   amt                    float64
 6   first                  object
 7   last                   object
 8   gender                 object
 9   street                 object
 10  city                   object
 11  state                  object
 12  zip                    int64
 13  lat                    float64
 14  long                   float64
 15  city_pop               int64
 16  job                    object
 17  dob                    object
 18  trans_num              object
 19  unix_time              int64
 20  merch_lat              float64
 21  merch_long             float64
 22  is_fraud               int64
```

Drop column '**Unnamed: 0**' since it's unnecessary and redundant.

```python
# Drop column 'Unnamed: 0' since it's unnecessary and redundant.
df = df.drop(columns=['Unnamed: 0'])
df.shape
```

```
(1852394, 22)
```

Rename column ‘**dob**’ (Date of Birth) to ‘**birth**’.

```
#   Rename column ‘dob’ (Date of Birth) for simply 'birth'.
df.rename(columns={"dob": "birth"}, inplace=True)
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1852394 entries, 0 to 1852393
Data columns (total 22 columns):
 #   Column                 Dtype
---  ------                 -----
 0   trans_date_trans_time  object
 1   cc_num                 int64
 2   merchant               object
 3   category               object
 4   amt                    float64
 5   first                  object
 6   last                   object
 7   gender                 object
 8   street                 object
 9   city                   object
 10  state                  object
 11  zip                    int64
 12  lat                    float64
 13  long                   float64
 14  city_pop               int64
 15  job                    object
 16  birth                  object
 17  trans_num              object
 18  unix_time              int64
 19  merch_lat              float64
 20  merch_long             float64
 21  is_fraud               int64
```

Let’s confirm if our data frame ‘**df**’ requires extra steps due to missing values and duplicates.

```
# Check for missing values.

missing_data = df.isnull().sum()
print(missing_data)
```

```
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
birth                    0
trans_num                0
unix_time                0
merch_lat                0
merch_long               0
is_fraud                 0
dtype: int64
```

```
# check for duplicates
df.duplicated().sum()
```

```
np.int64(0)
```

Column Transformations.

Now that we’ve cleaned up the column titles and removed unnecessary columns, we can take a closer look at the columns we plan to analyze.

Let’s start by converting categorical columns to lowercase and strip spaces, this will ensure data standardization by preventing variations in capitalization and spaces from being treated as different values (e.g., "Amazon " vs. "amazon"). This transformation improves data quality by reducing inconsistencies in category names and facilitates searches and analysis by enabling accurate comparisons without format discrepancies.

```
# Convert categorical columns to lowercase and strip spaces.
categorical_columns = ['merchant', 'category', 'first', 'last', 'gender', 'street', 'city', 'state', 'job']
df[categorical_columns] = df[categorical_columns].apply(lambda x: x.str.lower().str.strip())
df.head()
```

	trans_date_trans_time	cc_num	merchant	category	amt	first	last	gender	street	city	state	zip	lat	long	city_p
0	2019-01-01 00:00:18	2703186189652095	fraud_rippin, kub and mann	misc_net	4.97	jennifer	banks	f	561 perry cove	moravian falls	nc	28654	36.0788	-81.1781	34
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	orient	wa	99160	48.8878	-118.2105	1
2	2019-01-01 00:00:51	38859492057661	fraud_lind-buckridge	entertainment	220.11	edward	sanchez	m	594 white dale suite 530	malad city	id	83252	42.1808	-112.2620	41
3	2019-01-01 00:01:16	3534093764340240	fraud_kutch, hermiston and farrell	gas_transport	45.00	jeremy	white	m	9443 cynthia court apt. 038	boulder	mt	59632	46.2306	-112.1138	19
4	2019-01-01 00:03:06	375534208663984	fraud_keeling-crist	misc_pos	41.96	tyler	garcia	m	408 bradley rest	doe hill	va	24433	38.4207	-79.4629	

Now, we can split the '**trans_date_trans_time**' column into two columns. One column called '**trans_date**' and a second one called '**trans_time**'.

```
# Split the column into two separate columns.
df[['trans_date', 'trans_time']] = df['trans_date_trans_time'].str.split(' ', expand=True)

# Drop the original column if no longer needed
df = df.drop(columns=['trans_date_trans_time'])

# Insert the new columns at positions 0 and 1
df.insert(0, 'trans_date', df.pop('trans_date'))
df.insert(1, 'trans_time', df.pop('trans_time'))

# Display the updated DataFrame
df.head()
```

	trans_date	trans_time	cc_num	merchant	category	amt	first	last	gender	street	city	state	zip	lat	long	city_
0	2019-01-01	00:00:18	2703186189652095	fraud_Rippin, Kub and Mann	misc_net	4.97	Jennifer	Banks	F	561 Perry Cove	Moravian Falls	NC	28654	36.0788	-81.1781	
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	Orient	WA	99160	48.8878	-118.2105	
2	2019-01-01	00:00:51	38859492057661	fraud_Lind-Buckridge	entertainment	220.11	Edward	Sanchez	M	594 White Dale Suite 530	Malad City	ID	83252	42.1808	-112.2620	
3	2019-01-01	00:01:16	3534093764340240	fraud_Kutch, Hermiston and Farrell	gas_transport	45.00	Jeremy	White	M	9443 Cynthia Court Apt. 038	Boulder	MT	59632	46.2306	-112.1138	
4	2019-01-01	00:03:06	375534208663984	fraud_Keeling-Crist	misc_pos	41.96	Tyler	Garcia	M	408 Bradley Rest	Doe Hill	VA	24433	38.4207	-79.4629	

With new column ‘**trans_date**’ and column ‘**birth**’, we can determine the age of each customer when they executed the transaction. First, we need to convert '**birth**' and '**trans_date**' to datetime then we can create our new column called ‘**age**’.

```
# Convert 'birth' and 'trans_date' to datetime
df['birth'] = pd.to_datetime(df['birth'])
df['trans_date'] = pd.to_datetime(df['trans_date'])

# Create a new column called 'age' to identify the age of the customer once they executed the transaction.
df['age'] = ((df['trans_date'] - df['birth']) / pd.Timedelta(days=365.25)).astype(int)

# Insert column 'age' at positions 8.
df.insert(8, 'age', df.pop('age'))

df.head()
```

	trans_date	trans_time	cc_num	merchant	category	amt	first	last	age	gender	street	city	state	zip	lat	long
0	2019-01-01	00:00:18	2703186189652095	fraud_Rippin, Kub and Mann	misc_net	4.97	Jennifer	Banks	30	F	561 Perry Cove	Moravian Falls	NC	28654	36.0788	-81.1781
1	2019-01-01	00:00:44	630423337322	fraud_Heller, Gutmann and Zieme	grocery_pos	107.23	Stephanie	Gill	40	F	43039 Riley Greens Suite 393	Orient	WA	99160	48.8878	-118.2105
2	2019-01-01	00:00:51	38859492057661	fraud_Lind-Buckridge	entertainment	220.11	Edward	Sanchez	56	M	594 White Dale Suite 530	Malad City	ID	83252	42.1808	-112.2620
3	2019-01-01	00:01:16	3534093764340240	fraud_Kutch, Hermiston and Farrell	gas_transport	45.00	Jeremy	White	51	M	9443 Cynthia Court Apt. 038	Boulder	MT	59632	46.2306	-112.1138
4	2019-01-01	00:03:06	375534208663984	fraud_Keeling-Crist	misc_pos	41.96	Tyler	Garcia	32	M	408 Bradley Rest	Doe Hill	VA	24433	38.4207	-79.4629

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.

```
# Summary statistics for numerical columns.

# Select only numerical columns
df_numeric = df.select_dtypes(include=["number"])

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

Statistical summary for numerical variables:

	cc_num	amt	age	zip	lat	long	city_pop	unix_time	merch_lat	merch_long	is_fraud
count	1.852394e+06	1852394.00	1852394.00	1852394.00	1852394.00	1852394.00	1852394.00	1.852394e+06	1852394.00	1852394.00	1852394.00
mean	4.173860e+17	70.06	45.76	48813.26	38.54	-90.23	88643.67	1.358674e+09	38.54	-90.23	0.01
std	1.309115e+18	159.25	17.41	26881.85	5.07	13.75	301487.62	1.819508e+07	5.11	13.76	0.07
min	6.041621e+10	1.00	13.00	1257.00	20.03	-165.67	23.00	1.325376e+09	19.03	-166.67	0.00
25%	1.800429e+14	9.64	32.00	26237.00	34.67	-96.80	741.00	1.343017e+09	34.74	-96.90	0.00
50%	3.521417e+15	47.45	44.00	48174.00	39.35	-87.48	2443.00	1.357089e+09	39.37	-87.44	0.00
75%	4.642255e+15	83.10	57.00	72042.00	41.94	-80.16	20328.00	1.374581e+09	41.96	-80.25	0.00
max	4.992346e+18	28948.90	96.00	99921.00	66.69	-67.95	2906700.00	1.388534e+09	67.51	-66.95	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 **45.76 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.