Mishra540_Project_Milestone_03

July 16, 2023

[1]:
```
# DSC540,Summer 2023 - T302 Data Preparation(2237-1)
# Assignment: Project Milestone 03
# Author by:  Debabrata Mishra
# Date: 2023-07-16

# Topic - Credit Card Transactional & Demographic Data
```

1 Milestone 3 Tasks

Perform at least 5 data transformation and/or cleansing steps to your website data. The below examples are not required - they are just potential transformations you could do. If your data doesn't work for these scenarios, complete different transformations. You can do the same transformation multiple times if needed to clean your data. The goal is a clean dataset at the end of the milestone.

```
Replace Headers
Format data into a more readable format
Identify outliers and bad data
Find duplicates
Fix casing or inconsistent values
Conduct Fuzzy Matching
```

Make sure you clearly label each transformation step (Step #1, Step #2, etc.) in your code and describe what it is doing in 1-2 sentences. You can submit a Jupyter Notebook or a PDF of your code. If you submit a .py file you need to also include a PDF or attachment of your results.

2 Cleaning and Formatting Web Source Data

2.1 Web Data

Descrition:

The dataset consists of credit card transactions made by European cardholders in September 2013. The dataset covers a two-day period and contains a total of 200K+ transactions and 31 columns. The dataset is highly imbalanced, with fraud transactions accounting for only 0.172% of the total transactions.

The dataset primarily includes numeric input variables resulting from a PCA (Principal Component Analysis) transformation. Unfortunately, due to confidentiality concerns, the original features and additional background information about the data are not provided. The dataset includes principal components labeled as V1, V2, … V28, which are the outcomes of the PCA transformation. The

'Time' and 'Amount' features are exceptions and have not undergone the PCA transformation. The 'Time' feature represents the number of seconds elapsed between each transaction and the first transaction recorded in the dataset. The 'Amount' feature represents the monetary value of each transaction. The 'Amount' feature can be useful, particularly for approaches involving example-dependent cost-sensitive learning. The response variable, labeled as 'Class,' indicates whether a transaction is fraudulent (1) or not (0).

Link: https://datahub.io/machine-learning/creditcard/r/0.html

As part of Project Milestone 3: I have considered below transformations.

- Converted the date/time to readable format
- Createing new variables which required for future calculations like day, date, month, hour, weekdayfrom transaction date/time field
- Format amount values to 2 decimal points
- Created a amount range which will be utilized to identify the testing txn and BOT attacks
- Drop columns that I won't be use for any of my planned analysis
- Identify outliers using IQR
- check for duplicate and drop those duplicates (if any)
- Missing value check

[2]:
```
#Load the Necessary Libraries

import pandas as pd
import numpy as np
import xlrd
from bs4 import BeautifulSoup
import numpy as np
import datapackage
import matplotlib.pyplot as plt
import seaborn as sns
```

[3]:
```
# To access the Credit card web data source
data_url = 'https://datahub.io/machine-learning/creditcard/datapackage.json'
# to load Data Package into storage
package = datapackage.Package(data_url)
```

[4]:
```
# to load only tabular data
resources = package.resources
for resource in resources:
    if resource.tabular:
        df_web_data = pd.read_csv(resource.descriptor['path'])
        print (df_web_data)
```

```
            Time         V1         V2        V3        V4        V5  \
0            0.0  -1.359807  -0.072781  2.536347  1.378155 -0.338321
```

```
1            0.0   1.191857   0.266151  0.166480  0.448154  0.060018
2            1.0  -1.358354  -1.340163  1.773209  0.379780 -0.503198
3            1.0  -0.966272  -0.185226  1.792993 -0.863291 -0.010309
4            2.0  -1.158233   0.877737  1.548718  0.403034 -0.407193
…            …         …         …         …         …         …
284802  172786.0 -11.881118  10.071785 -9.834783 -2.066656 -5.364473
284803  172787.0  -0.732789  -0.055080  2.035030 -0.738589  0.868229
284804  172788.0   1.919565  -0.301254 -3.249640 -0.557828  2.630515
284805  172788.0  -0.240440   0.530483  0.702510  0.689799 -0.377961
284806  172792.0  -0.533413  -0.189733  0.703337 -0.506271 -0.012546

                V6        V7        V8        V9  …       V21       V22  \
0         0.462388  0.239599  0.098698  0.363787  … -0.018307  0.277838
1        -0.082361 -0.078803  0.085102 -0.255425  … -0.225775 -0.638672
2         1.800499  0.791461  0.247676 -1.514654  …  0.247998  0.771679
3         1.247203  0.237609  0.377436 -1.387024  … -0.108300  0.005274
4         0.095921  0.592941 -0.270533  0.817739  … -0.009431  0.798278
…              …         …         …         …  …         …         …
284802 -2.606837 -4.918215  7.305334  1.914428  …  0.213454  0.111864
284803  1.058415  0.024330  0.294869  0.584800  …  0.214205  0.924384
284804  3.031260 -0.296827  0.708417  0.432454  …  0.232045  0.578229
284805  0.623708 -0.686180  0.679145  0.392087  …  0.265245  0.800049
284806 -0.649617  1.577006 -0.414650  0.486180  …  0.261057  0.643078

                V23       V24       V25       V26       V27       V28  Amount  \
0         -0.110474  0.066928  0.128539 -0.189115  0.133558 -0.021053  149.62
1          0.101288 -0.339846  0.167170  0.125895 -0.008983  0.014724    2.69
2          0.909412 -0.689281 -0.327642 -0.139097 -0.055353 -0.059752  378.66
3         -0.190321 -1.175575  0.647376 -0.221929  0.062723  0.061458  123.50
4         -0.137458  0.141267 -0.206010  0.502292  0.219422  0.215153   69.99
…               …         …         …         …         …         …       …
284802    1.014480 -0.509348  1.436807  0.250034  0.943651  0.823731    0.77
284803    0.012463 -1.016226 -0.606624 -0.395255  0.068472 -0.053527   24.79
284804   -0.037501  0.640134  0.265745 -0.087371  0.004455 -0.026561   67.88
284805   -0.163298  0.123205 -0.569159  0.546668  0.108821  0.104533   10.00
284806    0.376777  0.008797 -0.473649 -0.818267 -0.002415  0.013649  217.00

        Class
0         '0'
1         '0'
2         '0'
3         '0'
4         '0'
…         …
284802    '0'
284803    '0'
284804    '0'
284805    '0'
```

```
284806    '0'

[284807 rows x 31 columns]
            Time         V1         V2        V3        V4        V5  \
0            0.0  -1.359807  -0.072781  2.536347  1.378155 -0.338321
1            0.0   1.191857   0.266151  0.166480  0.448154  0.060018
2            1.0  -1.358354  -1.340163  1.773209  0.379780 -0.503198
3            1.0  -0.966272  -0.185226  1.792993 -0.863291 -0.010309
4            2.0  -1.158233   0.877737  1.548718  0.403034 -0.407193
…            …         …          …         …         …         …
284802  172786.0 -11.881118  10.071785 -9.834783 -2.066656 -5.364473
284803  172787.0  -0.732789  -0.055080  2.035030 -0.738589  0.868229
284804  172788.0   1.919565  -0.301254 -3.249640 -0.557828  2.630515
284805  172788.0  -0.240440   0.530483  0.702510  0.689799 -0.377961
284806  172792.0  -0.533413  -0.189733  0.703337 -0.506271 -0.012546

              V6        V7        V8        V9  …       V21       V22  \
0       0.462388  0.239599  0.098698  0.363787  … -0.018307  0.277838
1      -0.082361 -0.078803  0.085102 -0.255425  … -0.225775 -0.638672
2       1.800499  0.791461  0.247676 -1.514654  …  0.247998  0.771679
3       1.247203  0.237609  0.377436 -1.387024  … -0.108300  0.005274
4       0.095921  0.592941 -0.270533  0.817739  … -0.009431  0.798278
…            …         …         …         …  …         …         …
284802 -2.606837 -4.918215  7.305334  1.914428  …  0.213454  0.111864
284803  1.058415  0.024330  0.294869  0.584800  …  0.214205  0.924384
284804  3.031260 -0.296827  0.708417  0.432454  …  0.232045  0.578229
284805  0.623708 -0.686180  0.679145  0.392087  …  0.265245  0.800049
284806 -0.649617  1.577006 -0.414650  0.486180  …  0.261057  0.643078

              V23       V24       V25       V26       V27       V28  Amount  \
0       -0.110474  0.066928  0.128539 -0.189115  0.133558 -0.021053  149.62
1        0.101288 -0.339846  0.167170  0.125895 -0.008983  0.014724    2.69
2        0.909412 -0.689281 -0.327642 -0.139097 -0.055353 -0.059752  378.66
3       -0.190321 -1.175575  0.647376 -0.221929  0.062723  0.061458  123.50
4       -0.137458  0.141267 -0.206010  0.502292  0.219422  0.215153   69.99
…            …         …         …         …         …         …  …
284802  1.014480 -0.509348  1.436807  0.250034  0.943651  0.823731    0.77
284803  0.012463 -1.016226 -0.606624 -0.395255  0.068472 -0.053527   24.79
284804 -0.037501  0.640134  0.265745 -0.087371  0.004455 -0.026561   67.88
284805 -0.163298  0.123205 -0.569159  0.546668  0.108821  0.104533   10.00
284806  0.376777  0.008797 -0.473649 -0.818267 -0.002415  0.013649  217.00

        Class
0         '0'
1         '0'
2         '0'
3         '0'
4         '0'
```

```
…       …
284802    '0'
284803    '0'
284804    '0'
284805    '0'
284806    '0'

[284807 rows x 31 columns]
```

[5]:
```
# 01: Convert time to a readable format
df_web_data["Time"] = pd.to_datetime(df_web_data["Time"], unit="s")
df_web_data.head()
```

[5]:
```
                 Time        V1        V2        V3        V4        V5  \
0 1970-01-01 00:00:00 -1.359807 -0.072781  2.536347  1.378155 -0.338321
1 1970-01-01 00:00:00  1.191857  0.266151  0.166480  0.448154  0.060018
2 1970-01-01 00:00:01 -1.358354 -1.340163  1.773209  0.379780 -0.503198
3 1970-01-01 00:00:01 -0.966272 -0.185226  1.792993 -0.863291 -0.010309
4 1970-01-01 00:00:02 -1.158233  0.877737  1.548718  0.403034 -0.407193

         V6        V7        V8        V9  …       V21       V22       V23  \
0  0.462388  0.239599  0.098698  0.363787  … -0.018307  0.277838 -0.110474
1 -0.082361 -0.078803  0.085102 -0.255425  … -0.225775 -0.638672  0.101288
2  1.800499  0.791461  0.247676 -1.514654  …  0.247998  0.771679  0.909412
3  1.247203  0.237609  0.377436 -1.387024  … -0.108300  0.005274 -0.190321
4  0.095921  0.592941 -0.270533  0.817739  … -0.009431  0.798278 -0.137458

        V24       V25       V26       V27       V28  Amount  Class
0  0.066928  0.128539 -0.189115  0.133558 -0.021053  149.62    '0'
1 -0.339846  0.167170  0.125895 -0.008983  0.014724    2.69    '0'
2 -0.689281 -0.327642 -0.139097 -0.055353 -0.059752  378.66    '0'
3 -1.175575  0.647376 -0.221929  0.062723  0.061458  123.50    '0'
4  0.141267 -0.206010  0.502292  0.219422  0.215153   69.99    '0'

[5 rows x 31 columns]
```

[6]:
```
# 02: Convert amount to a float with two decimal places

df_web_data['Amount'] = np.round(df_web_data['Amount'], 2)
df_web_data.head()
```

[6]:
```
                 Time        V1        V2        V3        V4        V5  \
0 1970-01-01 00:00:00 -1.359807 -0.072781  2.536347  1.378155 -0.338321
1 1970-01-01 00:00:00  1.191857  0.266151  0.166480  0.448154  0.060018
2 1970-01-01 00:00:01 -1.358354 -1.340163  1.773209  0.379780 -0.503198
3 1970-01-01 00:00:01 -0.966272 -0.185226  1.792993 -0.863291 -0.010309
4 1970-01-01 00:00:02 -1.158233  0.877737  1.548718  0.403034 -0.407193
```

```
          V6        V7        V8        V9  …       V21       V22       V23  \
0  0.462388  0.239599  0.098698  0.363787  … -0.018307  0.277838 -0.110474
1 -0.082361 -0.078803  0.085102 -0.255425  … -0.225775 -0.638672  0.101288
2  1.800499  0.791461  0.247676 -1.514654  …  0.247998  0.771679  0.909412
3  1.247203  0.237609  0.377436 -1.387024  … -0.108300  0.005274 -0.190321
4  0.095921  0.592941 -0.270533  0.817739  … -0.009431  0.798278 -0.137458

        V24       V25       V26       V27       V28  Amount  Class
0  0.066928  0.128539 -0.189115  0.133558 -0.021053  149.62    '0'
1 -0.339846  0.167170  0.125895 -0.008983  0.014724    2.69    '0'
2 -0.689281 -0.327642 -0.139097 -0.055353 -0.059752  378.66    '0'
3 -1.175575  0.647376 -0.221929  0.062723  0.061458  123.50    '0'
4  0.141267 -0.206010  0.502292  0.219422  0.215153   69.99    '0'

[5 rows x 31 columns]
```

[7]:
```
# 03 : Create a new field  amount_range
bin_edges = [0,1,5,25,50, 100,200,300,400,500,999,1000,50000]
bin_labels = [ '02: 000.01 - 001.00', '03: 001.01 - 005.00','04: 005.01 - 025.
 ↪00','05: 025.01 - 050.00 ','06: 050.01 - 100.00'
              ,'07: 100.01 - 200.00', '08: 200.01 - 300.00', '09: 300.01 - 400.
 ↪00', '10: 400.01 - 500.00'
              ,'11: 500.01 - 999.00', '12: 999.01 - 1000.00', '13: 1000.
 ↪01-50000.00']

#df_web_data['amount_range'] = pd.cut(df_web_data['Amount'], bins=bin_edges,␣
 ↪labels=bin_labels)

df_web_data['amount_range'] = np.where(df_web_data['Amount'] == 0, '01: 000.00␣
 ↪- 000.00', pd.cut(df_web_data['Amount']
                                                                              ␣
 ↪                 , bins=bin_edges
                                                                              ␣
 ↪                 , labels=bin_labels))

df_web_data.head()
```

[7]:
```
                 Time        V1        V2        V3        V4        V5  \
0 1970-01-01 00:00:00 -1.359807 -0.072781  2.536347  1.378155 -0.338321
1 1970-01-01 00:00:00  1.191857  0.266151  0.166480  0.448154  0.060018
2 1970-01-01 00:00:01 -1.358354 -1.340163  1.773209  0.379780 -0.503198
3 1970-01-01 00:00:01 -0.966272 -0.185226  1.792993 -0.863291 -0.010309
4 1970-01-01 00:00:02 -1.158233  0.877737  1.548718  0.403034 -0.407193

          V6        V7        V8        V9  …       V22       V23       V24  \
0  0.462388  0.239599  0.098698  0.363787  …  0.277838 -0.110474  0.066928
1 -0.082361 -0.078803  0.085102 -0.255425  … -0.638672  0.101288 -0.339846
```

```
2  1.800499  0.791461  0.247676 -1.514654  …  0.771679  0.909412 -0.689281
3  1.247203  0.237609  0.377436 -1.387024  …  0.005274 -0.190321 -1.175575
4  0.095921  0.592941 -0.270533  0.817739  …  0.798278 -0.137458  0.141267

        V25       V26       V27       V28  Amount  Class         amount_range
0  0.128539 -0.189115  0.133558 -0.021053  149.62    '0'  07: 100.01 - 200.00
1  0.167170  0.125895 -0.008983  0.014724    2.69    '0'  03: 001.01 - 005.00
2 -0.327642 -0.139097 -0.055353 -0.059752  378.66    '0'  09: 300.01 - 400.00
3  0.647376 -0.221929  0.062723  0.061458  123.50    '0'  07: 100.01 - 200.00
4 -0.206010  0.502292  0.219422  0.215153   69.99    '0'  06: 050.01 - 100.00

[5 rows x 32 columns]
```

[8]:
```
# 04: Create a new field is_fraud based on class value
class_mapping = {"'0'": 0, "'1'": 1}
df_web_data['is_fraud'] = df_web_data['Class'].map(class_mapping)
df_web_data.head()
```

[8]:
```
                 Time        V1        V2        V3        V4        V5  \
0 1970-01-01 00:00:00 -1.359807 -0.072781  2.536347  1.378155 -0.338321
1 1970-01-01 00:00:00  1.191857  0.266151  0.166480  0.448154  0.060018
2 1970-01-01 00:00:01 -1.358354 -1.340163  1.773209  0.379780 -0.503198
3 1970-01-01 00:00:01 -0.966272 -0.185226  1.792993 -0.863291 -0.010309
4 1970-01-01 00:00:02 -1.158233  0.877737  1.548718  0.403034 -0.407193

         V6        V7        V8        V9  …       V23       V24       V25  \
0  0.462388  0.239599  0.098698  0.363787  … -0.110474  0.066928  0.128539
1 -0.082361 -0.078803  0.085102 -0.255425  …  0.101288 -0.339846  0.167170
2  1.800499  0.791461  0.247676 -1.514654  …  0.909412 -0.689281 -0.327642
3  1.247203  0.237609  0.377436 -1.387024  … -0.190321 -1.175575  0.647376
4  0.095921  0.592941 -0.270533  0.817739  … -0.137458  0.141267 -0.206010

        V26       V27       V28  Amount  Class         amount_range  is_fraud
0 -0.189115  0.133558 -0.021053  149.62    '0'  07: 100.01 - 200.00         0
1  0.125895 -0.008983  0.014724    2.69    '0'  03: 001.01 - 005.00         0
2 -0.139097 -0.055353 -0.059752  378.66    '0'  09: 300.01 - 400.00         0
3 -0.221929  0.062723  0.061458  123.50    '0'  07: 100.01 - 200.00         0
4  0.502292  0.219422  0.215153   69.99    '0'  06: 050.01 - 100.00         0

[5 rows x 33 columns]
```

[9]:
```
# 05: Create variables from trasnaction date/time for future calculations

df_web_data['year']=df_web_data['Time'].dt.year
df_web_data['month']=df_web_data['Time'].dt.strftime('%b')
df_web_data['month']=df_web_data['Time'].dt.month
df_web_data['day']=df_web_data['Time'].dt.day
```

```
df_web_data['hour']=df_web_data['Time'].dt.hour
df_web_data['weekday']=df_web_data['Time'].dt.strftime('%a')
df_web_data['dayofYear']=df_web_data['Time'].dt.dayofyear
```

[10]:
```
# Size before duplicate check
print("Size of the dataset before to duplicate check: ",df_web_data.shape)
```

```
Size of the dataset before to duplicate check:  (284807, 39)
```

[11]:
```
# 06 : Identify any duplicate rows
df_web_data_duplicates = df_web_data[df_web_data.duplicated(subset=df_web_data.
  ↪columns[:-1], keep=False)]
print(df_web_data_duplicates)
```

```
                     Time        V1        V2        V3        V4        V5  \
32     1970-01-01 00:00:26 -0.529912  0.873892  1.347247  0.145457  0.414209
33     1970-01-01 00:00:26 -0.529912  0.873892  1.347247  0.145457  0.414209
34     1970-01-01 00:00:26 -0.535388  0.865268  1.351076  0.147575  0.433680
35     1970-01-01 00:00:26 -0.535388  0.865268  1.351076  0.147575  0.433680
112    1970-01-01 00:01:14  1.038370  0.127486  0.184456  1.109950  0.441699
…                      …       …       …       …       …       …
283485 1970-01-02 23:40:27 -1.457978  1.378203  0.811515 -0.603760 -0.711883
284190 1970-01-02 23:50:33 -2.667936  3.160505 -3.355984  1.007845 -0.377397
284191 1970-01-02 23:50:33 -2.667936  3.160505 -3.355984  1.007845 -0.377397
284192 1970-01-02 23:50:33 -2.691642  3.123168 -3.339407  1.017018 -0.293095
284193 1970-01-02 23:50:33 -2.691642  3.123168 -3.339407  1.017018 -0.293095

               V6        V7        V8        V9  …  Amount  Class  \
32       0.100223  0.711206  0.176066 -0.286717  …    6.14    '0'
33       0.100223  0.711206  0.176066 -0.286717  …    6.14    '0'
34       0.086983  0.693039  0.179742 -0.285642  …    1.77    '0'
35       0.086983  0.693039  0.179742 -0.285642  …    1.77    '0'
112      0.945283 -0.036715  0.350995  0.118950  …    1.18    '0'
…             …       …       …       …  …     …     …
283485  -0.471672 -0.282535  0.880654  0.052808  …   11.93    '0'
284190  -0.109730 -0.667233  2.309700 -1.639306  …   55.66    '0'
284191  -0.109730 -0.667233  2.309700 -1.639306  …   55.66    '0'
284192  -0.167054 -0.745886  2.325616 -1.634651  …   36.74    '0'
284193  -0.167054 -0.745886  2.325616 -1.634651  …   36.74    '0'

                  amount_range  is_fraud  year  month  day  hour  weekday  \
32       04: 005.01 - 025.00          0  1970      1    1     0      Thu
33       04: 005.01 - 025.00          0  1970      1    1     0      Thu
34       03: 001.01 - 005.00          0  1970      1    1     0      Thu
35       03: 001.01 - 005.00          0  1970      1    1     0      Thu
112      03: 001.01 - 005.00          0  1970      1    1     0      Thu
…                        …       …    …       …  …   …       …
283485   04: 005.01 - 025.00          0  1970      1    2    23      Fri
```

```
284190   06: 050.01 - 100.00          0  1970      1    2    23      Fri
284191   06: 050.01 - 100.00          0  1970      1    2    23      Fri
284192  05: 025.01 - 050.00          0  1970      1    2    23      Fri
284193  05: 025.01 - 050.00          0  1970      1    2    23      Fri

        dayofYear
32              1
33              1
34              1
35              1
112             1
…             …
283485          2
284190          2
284191          2
284192          2
284193          2

[1854 rows x 39 columns]
```

[12]:
```
# 07 : Drop all the duplicate rows in the dataset
df_web_data = df_web_data.drop_duplicates()
print("Size of the dataset after dropping the duplicate rows: ",df_web_data.
  ↪shape)
```

```
Size of the dataset after dropping the duplicate rows:  (283726, 39)
```

[13]:
```
# 08: Identify any missing values
missing_values = df_web_data.isnull().sum().sum()
print("Missing Values:\n", missing_values)
```

```
Missing Values:
 0
```

[14]:
```
# 09 : Find the Outilier for the Amount

# Calculate summary statistics for the transaction amount column
amount_stats = df_web_data['Amount'].describe()

# Calculate the interquartile range (IQR)
Q1 = amount_stats['25%']
Q3 = amount_stats['75%']
IQR = Q3 - Q1

# Find the lower and upper bounds for outliers
lower_bound = Q1 - (1.5 * IQR)
upper_bound = Q3 + (1.5 * IQR)
```

```
# Identify the rows with transaction amounts outside the bounds
outliers = df_web_data[(df_web_data['Amount'] < lower_bound) |␣
 ↪(df_web_data['Amount'] > upper_bound)]

# Print the number of outliers found
print("Number of outliers found: ", len(outliers))

# Remove the outliers

df_web_data_no_outliers = df_web_data[(df_web_data['Amount'] >= lower_bound) & ␣
 ↪(df_web_data['Amount'] <= upper_bound)]

print("Size of the dataset after removal of outliers: ",␣
 ↪len(df_web_data_no_outliers))
```

```
Number of outliers found:  31685
Size of the dataset after removal of outliers:  252041
```

[15]:
```
# 10 : Drop colmuns those are not required

# Drop the columns that are not needed and create a new DataFrame without those␣
 ↪columns
df_web_data_final = df_web_data_no_outliers.
 ↪drop(columns=df_web_data_no_outliers.filter(like='V').columns)

# Display the new DataFrame with the unnecessary columns dropped
print(df_web_data_final)
```

```
                     Time  Amount Class         amount_range  is_fraud  year  \
0      1970-01-01 00:00:00  149.62   '0'  07: 100.01 - 200.00         0  1970
1      1970-01-01 00:00:00    2.69   '0'  03: 001.01 - 005.00         0  1970
3      1970-01-01 00:00:01  123.50   '0'  07: 100.01 - 200.00         0  1970
4      1970-01-01 00:00:02   69.99   '0'  06: 050.01 - 100.00         0  1970
5      1970-01-01 00:00:02    3.67   '0'  03: 001.01 - 005.00         0  1970
…                      …       …   …   …                  …       …   …
284801 1970-01-02 23:59:45    2.69   '0'  03: 001.01 - 005.00         0  1970
284802 1970-01-02 23:59:46    0.77   '0'  02: 000.01 - 001.00         0  1970
284803 1970-01-02 23:59:47   24.79   '0'  04: 005.01 - 025.00         0  1970
284804 1970-01-02 23:59:48   67.88   '0'  06: 050.01 - 100.00         0  1970
284805 1970-01-02 23:59:48   10.00   '0'  04: 005.01 - 025.00         0  1970

        month  day  hour weekday  dayofYear
0           1    1     0     Thu          1
1           1    1     0     Thu          1
3           1    1     0     Thu          1
4           1    1     0     Thu          1
5           1    1     0     Thu          1
…       …  …    …     …       …
```

```
284801      1    2    23     Fri          2
284802      1    2    23     Fri          2
284803      1    2    23     Fri          2
284804      1    2    23     Fri          2
284805      1    2    23     Fri          2

[252041 rows x 11 columns]
```

[16]:
```
# Overview of the structure and characteristics
print(df_web_data_final.info())
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 252041 entries, 0 to 284805
Data columns (total 11 columns):
 #   Column        Non-Null Count   Dtype
---  ------        --------------   -----
 0   Time          252041 non-null  datetime64[ns]
 1   Amount        252041 non-null  float64
 2   Class         252041 non-null  object
 3   amount_range  252041 non-null  object
 4   is_fraud      252041 non-null  int64
 5   year          252041 non-null  int64
 6   month         252041 non-null  int64
 7   day           252041 non-null  int64
 8   hour          252041 non-null  int64
 9   weekday       252041 non-null  object
 10  dayofYear     252041 non-null  int64
dtypes: datetime64[ns](1), float64(1), int64(6), object(3)
memory usage: 23.1+ MB
None
```

[17]:
```
# Use value_counts() to get range-wise counts
range_wise_counts = df_web_data['amount_range'].value_counts()

# Display the range-wise counts
print(range_wise_counts.sort_index())
```

```
01: 000.00 - 000.00      1808
02: 000.01 - 001.00     28523
03: 001.01 - 005.00     38416
04: 005.01 - 025.00     80893
05: 025.01 - 050.00     40508
06: 050.01 - 100.00     37179
07: 100.01 - 200.00     27629
08: 200.01 - 300.00     10758
09: 300.01 - 400.00      5521
10: 400.01 - 500.00      3382
11: 500.01 - 999.00      6040
12: 999.01 - 1000.00      134
```

```
13: 1000.01-50000.00     2935
Name: amount_range, dtype: int64
```

[18]:
```
# Use value_counts() to get range-wise counts
range_wise_fraud_counts = df_web_data.groupby('amount_range')['is_fraud'].sum()

# Display the range-wise counts
print(range_wise_fraud_counts.sort_index())
```

```
amount_range
01: 000.00 - 000.00      25
02: 000.01 - 001.00     146
03: 001.01 - 005.00      40
04: 005.01 - 025.00      54
05: 025.01 - 050.00      28
06: 050.01 - 100.00      55
07: 100.01 - 200.00      43
08: 200.01 - 300.00      21
09: 300.01 - 400.00      20
10: 400.01 - 500.00       7
11: 500.01 - 999.00      25
12: 999.01 - 1000.00      0
13: 1000.01-50000.00      9
Name: is_fraud, dtype: int64
```

[19]:
```
concat_df = pd.concat([range_wise_counts, range_wise_fraud_counts], axis=1)
print(concat_df.sort_index())
```

```
                     amount_range  is_fraud
01: 000.00 - 000.00          1808        25
02: 000.01 - 001.00         28523       146
03: 001.01 - 005.00         38416        40
04: 005.01 - 025.00         80893        54
05: 025.01 - 050.00         40508        28
06: 050.01 - 100.00         37179        55
07: 100.01 - 200.00         27629        43
08: 200.01 - 300.00         10758        21
09: 300.01 - 400.00          5521        20
10: 400.01 - 500.00          3382         7
11: 500.01 - 999.00          6040        25
12: 999.01 - 1000.00          134         0
13: 1000.01-50000.00          2935        9
```