Practice Session 01+02: Data preparation

Data scientists spend a big chunk of their time preparing data and this is one of the first steps in any data mining project. This step is normally called **data preparation**.

The processes of getting an initial understanding of a dataset and preparing it usually go hand-in-hand, and it is critical to perform them well to obtain valid results later. Plus, you can save time and effort by learning how to do proper data preparation.

In this session, we will assume you just received a new dataset and need to do some initial steps with it:

1) Exploratory Data Analysis

- Calculate basis statistics as mean, median, variance, maximum and minimum
- Look at distributions, identify outliers
- Calculate correlations between variables

2) Feature engineering:

- Deal with missing values
- Standardize all numerical columns
- Convert categorical columns to dummy binary variables
- Date and period management
- Feature generation

Tip: This process has several steps. It is tempting to maintain a single variable throughout the entire cleaning process, and do something like `x = x.step1()` then `x = x.step2()`. This will create problems for you because if you go back and re-execute a cell it might fail to operate on already transformed data. A better approach in cases like this where you do not have memory problems, is to do `x1 = x.step1()`, `x2 = x1.step2()` and so on, i.e., create a new variable after each transformation or set of transformations.

(Remove this cell when delivering.)

Author: Roj Gian Gorospe

E-mail: rojgian.gorospe.cics@ust.edu.ph

Date: February 8, 2025

0. The dataset

The dataset, contained in `device_db.csv` is a 10000 registers of mobile device purchases around 2019. **Each record in the dataset describes a customer that buys a new mobile telephone**. The attributes are defined as follows:

1. PURCHASED_DEVICE: the mobile phone bought by the customer

2. DEVICE_VALUE: the cost of the mobile phone bought by the customer
3. LAST_DEVICE_DATE: the date of the previous mobile device purchase
4. DATA_TRAFFIC_MONTH_(1..6): The Mbps of data traffic in the month (-1...-6) used by the customer previous to the mobile device purchase
5. VOICE_TRAFFIC_MONTH_(1..6): The minutes of voice traffic in the month (-1...-6) used by the customer previous to the mobile device purchase
6. BILLING_MONTH_(1..6): Billing (USD) in the month (-1...-6) paid by the customer previous to the mobile device purchase
7. DEVICE_COST_MONTH_(1..6): Monthly cost (USD) associated to the mobile device finance in the month (-1...-6) paid by the customer previous to the mobile device purchase: proportion of owner-occupied units built prior to 1940
8. LINE_ACTIVATION_DATE: Date of the activation of the mobile line by the customer
9. MONTHS_LAST_DEVICE: Number of months of the previous mobile device
10. DURATION_LINE: Number of months since the customer contracted the mobile line
11. PREVIOUS_DEVICE_MODEL: Model of the previous mobile phone
12. PREVIOUS_DEVICE_MANUF: Manufacturer of the previous mobile phone
13. PREVIOUS_DEVICE_BRAND: Brand of the previous mobile phone

This dataset will be used in next practices as recommendation engines.

(Remove this cell when delivering.)

1. Exploratory data analysis

Exploratory Data Analysis (EDA) allows to us to have an understanding of the dataset from a stadistics perspective, i.e., data distribution and correlation between variables. This is crucial to select the most relevant variables for some purpose.

(Remove this cell when delivering.)

```
import pandas as pd
import seaborn as sns
import datetime

import numpy as np
from numpy import array
from numpy import argmax

import matplotlib.pyplot as plt
from matplotlib import pyplot

import sklearn
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import Normalizer
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import LabelEncoder
```

```
from sklearn.preprocessing import OneHotEncoder
from sklearn.model_selection import train_test_split
```

We open the csv file contaning the data using separator ";" and assign to a dataframe (use `read_csv` from the Pandas library).

(Remove this cell when delivering.)

```
# LEAVE AS-IS

input_dataset = pd.read_csv("device_db.csv", sep=",")
```

1.1. Data types and simple statistics

Replace this cell with your code to print the dataset header (column names) and the first five rows of data.

```
input_dataset.head(5)
```

```
                                    PURCHASED_DEVICE  DEVICE_VALUE  \
0  TGLG29162000_LG X210BMW SMARTPHONE PRETO PPB/P...         393.0
1  TGLG29162000_LG X210BMW SMARTPHONE PRETO PPB/P...         345.0
2     TGMO35912000_MOTOROLA XT1922 SMARTPHONE INDIGO         875.0
3  TGLG29162000_LG X210BMW SMARTPHONE PRETO PPB/P...         345.0
4     TGMO35912000_MOTOROLA XT1922 SMARTPHONE INDIGO         609.0

   LAST_DEVICE_CHANGE  DATA_TRAFFIC_MONTH_1  DATA_TRAFFIC_MONTH_2  \
0                 NaN             465.24673             530.80615
1          20170401.0             232.24121             272.25525
2                 NaN             484.62036             264.13843
3          20171001.0            4255.46040             836.11707
4          20190101.0            5014.10300            2659.05150

   DATA_TRAFFIC_MONTH_3  DATA_TRAFFIC_MONTH_4  DATA_TRAFFIC_MONTH_5  \
0             530.80615             781.12646             398.99377
1             272.25525             704.88519             412.71664
2             264.13843             348.50073             380.44156
3             836.11707             691.55640             146.76660
4            2659.05150            2435.03930            2053.97950

   DATA_TRAFFIC_MONTH_6  VOICE_TRAFFIC_MONTH_1  ...
DEVICE_COST_MONTH_3  \
0            1169.39610               47.50000  ...
12.0
1             365.14441                3.70000  ...
0.0
2             250.73566               26.10000  ...
0.0
3             302.49249              175.70000  ...
```

```
6.0
4          1553.11500              383.89999  ...
0.0

   DEVICE_COST_MONTH_4  DEVICE_COST_MONTH_5  DEVICE_COST_MONTH_6  \
0                 12.0                 12.0                 12.0
1                  0.0                  0.0                  0.0
2                  0.0                  0.0                  0.0
3                  6.0                  6.0                  6.0
4                  0.0                  0.0                  0.0

   LINE_ACTIVATION_DATE  MONTHS_LAST_DEVICE  DURATION_LINE  \
0            20041220.0                 NaN          172.0
1            20170405.0                20.0           20.0
2            20040412.0                 NaN          176.0
3            20110825.0                14.0           88.0
4            20140617.0                -1.0           54.0

    PREVIOUS_DEVICE_MODEL                   PREVIOUS_DEVICE_MANUF  \
0            Moto G4 Plus  Motorola Mobility LLC, a Lenovo Company
1  Samsung Galaxy J1 Mini                           Samsung Korea
2     Moto E (2ª Geração)  Motorola Mobility LLC, a Lenovo Company
3                iPhone 6                               Apple Inc
4                  K10a40  Motorola Mobility LLC, a Lenovo Company

   PREVIOUS_DEVICE_BRAND
0               Motorola
1                Samsung
2               Motorola
3                  Apple
4                 Outros

[5 rows x 33 columns]
```

There are many ways of creating a data frame. Above, we created it by reading a file, but one can also create a dataframe from scratch, using an array of dictionaries. Example:

```
countries = []
countries.append({'capital': 'Београд', 'country': 'Република
Србија'})
countries.append({'capital': 'Nairobi', 'country': 'Jamhuri ya
Kenya'})
countries_df = pd.DataFrame(countries, columns=['country', 'capital'])
display(countries_df)
```

(Remove this cell when delivering.)

Create a dataframe named `column_type_df` containing the name of each column, its type and the number of distinct elements in that column. To iterate through the columns of dataframe `df`, use `for column in df.columns`; to determine the type of a column, use `df[column].dtype`; to retrieve the number of distinct elements of that column, use `df[column].nunique()`; to retrieve the size of a column, use `df[column].size`.

(Remove this cell when delivering.)

Replace this cell with your code to create and display a dataframe containing one row per column, and with the following fields: name of the column, type, number of distinct elements, and size. The size of all columns should be equal.

```python
# Create empty array to store the data
data = []

# Loop through each column in the dataset and store the column name,
data type, and number of distinct elements
for column in input_dataset.columns:
  data.append({
    "column_name": column,
    "column_type": input_dataset[column].dtype,
    "distinct_elements": input_dataset[column].nunique()
  })

#Display the data in a DataFrame
column_type_df = pd.DataFrame(data)
display(column_type_df)
```

	column_name	column_type	distinct_elements
0	PURCHASED_DEVICE	object	101
1	DEVICE_VALUE	float64	368
2	LAST_DEVICE_CHANGE	float64	76
3	DATA_TRAFFIC_MONTH_1	float64	7215
4	DATA_TRAFFIC_MONTH_2	float64	7182
5	DATA_TRAFFIC_MONTH_3	float64	7176
6	DATA_TRAFFIC_MONTH_4	float64	7124
7	DATA_TRAFFIC_MONTH_5	float64	7173
8	DATA_TRAFFIC_MONTH_6	float64	7074
9	VOICE_TRAFFIC_MONTH_1	float64	3550
10	VOICE_TRAFFIC_MONTH_2	float64	3346
11	VOICE_TRAFFIC_MONTH_3	float64	3332
12	VOICE_TRAFFIC_MONTH_4	float64	3370
13	VOICE_TRAFFIC_MONTH_5	float64	3530
14	VOICE_TRAFFIC_MONTH_6	float64	2513
15	BILLING_MONTH_1	float64	3810
16	BILLING_MONTH_2	float64	4001
17	BILLING_MONTH_3	float64	3958
18	BILLING_MONTH_4	float64	3988
19	BILLING_MONTH_5	float64	3906
20	BILLING_MONTH_6	float64	3897
21	DEVICE_COST_MONTH_1	float64	292
22	DEVICE_COST_MONTH_2	float64	295

```
23    DEVICE_COST_MONTH_3     float64              303
24    DEVICE_COST_MONTH_4     float64              311
25    DEVICE_COST_MONTH_5     float64              329
26    DEVICE_COST_MONTH_6     float64              336
27   LINE_ACTIVATION_DATE     float64             2546
28      MONTHS_LAST_DEVICE    float64               78
29           DURATION_LINE    float64              283
30  PREVIOUS_DEVICE_MODEL      object              580
31  PREVIOUS_DEVICE_MANUF      object               68
32  PREVIOUS_DEVICE_BRAND      object                5
```

To obtain a **series** (column) from a dataframe you can reference an attribute by name, e.g., `input_dataset.DEVICE_VALUE` returns the series of all device values.

On a series, you can use functions from numpy such as `np.mean`, `np.median`, `np.std`, `np.min` and `np.max`; meanings are self-explanatory. These functions have equivalents `np.nanmean`, `np.nanmedian`, and so on that ignore NaN (not-a-number) values.

To display floats using two decimals, you can use:

```
pd.options.display.float_format = '{:.2f}'.format
```

(Remove this cell when delivering.)

Replace this cell with code to create and display a dataframe containing one row per each column of type `float64` in the input data, and with the following fields: name of the column, mean, median, min, max -- all computed ignoring NaN values.

```
# Create empty array to store the data
temp_data = []

# Loop through each column in the dataset and store the column name,
float64 data types only, and its corresponding mean, median, min, max
computed while ignoring the NaN values

for column in input_dataset.columns:
  if input_dataset[column].dtype == 'float64':
    temp_data.append({
      "column_name": column,
      "column_type": input_dataset[column].dtype,
      "mean": np.nanmean(input_dataset[column]),
      "median": np.nanmedian(input_dataset[column]),
      "min": np.nanmin(input_dataset[column]),
      "max": np.nanmax(input_dataset[column])
    })

#Display the data in a DataFrame
column_type_info_df = pd.DataFrame(temp_data)
pd.options.display.float_format = '{:.2f}'.format
```

```
display(column_type_info_df)
```

```
                 column_name column_type        mean      median
min  \
0               DEVICE_VALUE     float64      750.48      393.00
15.00
1         LAST_DEVICE_CHANGE     float64 20166984.77 20170601.00
20121001.00
2       DATA_TRAFFIC_MONTH_1     float64     3481.83     1208.73
0.00
3       DATA_TRAFFIC_MONTH_2     float64     3649.96     1294.95
0.00
4       DATA_TRAFFIC_MONTH_3     float64     3653.43     1310.67
0.00
5       DATA_TRAFFIC_MONTH_4     float64     3269.44     1176.54
0.00
6       DATA_TRAFFIC_MONTH_5     float64     3673.37     1287.09
0.00
7       DATA_TRAFFIC_MONTH_6     float64     3427.69     1277.12
0.00
8      VOICE_TRAFFIC_MONTH_1     float64      154.85       84.05
0.00
9      VOICE_TRAFFIC_MONTH_2     float64      142.57       74.90
0.00
10     VOICE_TRAFFIC_MONTH_3     float64      141.71       74.40
0.00
11     VOICE_TRAFFIC_MONTH_4     float64      143.15       75.10
0.00
12     VOICE_TRAFFIC_MONTH_5     float64      154.28       82.85
0.00
13     VOICE_TRAFFIC_MONTH_6     float64       84.03        6.20
0.00
14           BILLING_MONTH_1     float64      102.34       94.99      -
128.01
15           BILLING_MONTH_2     float64      104.98       96.43
0.00
16           BILLING_MONTH_3     float64      102.68       96.25
0.00
17           BILLING_MONTH_4     float64      101.99       94.89
0.00
18           BILLING_MONTH_5     float64      102.21       95.29
0.00
19           BILLING_MONTH_6     float64      102.27       94.99
0.00
20       DEVICE_COST_MONTH_1     float64       10.81        0.00
0.00
21       DEVICE_COST_MONTH_2     float64       10.59        0.00
0.00
```

```
22     DEVICE_COST_MONTH_3     float64       11.71        0.00
0.00
23     DEVICE_COST_MONTH_4     float64       11.55        0.00
0.00
24     DEVICE_COST_MONTH_5     float64       12.51        0.00
0.00
25     DEVICE_COST_MONTH_6     float64       12.98        0.00
0.00
26    LINE_ACTIVATION_DATE     float64 20136051.57 20150324.00
19920804.00
27      MONTHS_LAST_DEVICE     float64       25.34       22.00        -
5.00
28           DURATION_LINE     float64       62.37       48.00
0.00

           max
0      9057.00
1  20190501.00
2    127017.59
3    111948.84
4    111948.84
5     87856.41
6    121834.81
7     90550.61
8      4220.10
9      3132.10
10     2992.50
11     3163.30
12     3429.10
13     2129.50
14     1569.10
15     2032.12
16     1741.21
17     1084.82
18      911.72
19     1187.30
20     6440.00
21     1360.00
22     2466.00
23      455.00
24     1258.00
25     1000.00
26 20190416.00
27       78.00
28      320.00
```

The `describe` function can be used to describe a series. To invoke it simply do `input_dataset.DEVICE_VALUE.describe()`

(Remove this cell when delivering.)

Replace this cell with code to print each column name and then use the `describe` function to print statistics for that column. Include a blank line after each description.

```
# Create empty array to store the data
column_data = []

# Loop through each column in the dataset and store the column name,
float64 data types only, and its corresponding mean, median, min, max
computed while ignoring the NaN values
for column in input_dataset.columns:
  column_data.append({
    "column_name": column,
    "description": input_dataset[column].describe(),
    })

#Display the data in a DataFrame
column_describe_df = pd.DataFrame(column_data)
pd.options.display.float_format = '{:.2f}'.format

display(column_describe_df)
```

```
              column_name
description
0        PURCHASED_DEVICE  count
...
1            DEVICE_VALUE  count    9690.00
mean     750.48
std      979.7...
2      LAST_DEVICE_CHANGE  count       7682.00
mean    20166984.77
std   ...
3     DATA_TRAFFIC_MONTH_1  count     8868.00
mean       3481.83
std        ...
4     DATA_TRAFFIC_MONTH_2  count     8841.00
mean       3649.96
std        ...
5     DATA_TRAFFIC_MONTH_3  count     8846.00
mean       3653.43
std        ...
6     DATA_TRAFFIC_MONTH_4  count    8817.00
mean      3269.44
std       567...
7     DATA_TRAFFIC_MONTH_5  count     8866.00
mean       3673.37
std        ...
8     DATA_TRAFFIC_MONTH_6  count    8535.00
mean      3427.69
```

```
std       588...
9   VOICE_TRAFFIC_MONTH_1  count   8868.00
mean      154.85
std       218.2...
10  VOICE_TRAFFIC_MONTH_2  count   8841.00
mean      142.57
std       200.5...
11  VOICE_TRAFFIC_MONTH_3  count   8846.00
mean      141.71
std       198.5...
12  VOICE_TRAFFIC_MONTH_4  count   8817.00
mean      143.15
std       200.5...
13  VOICE_TRAFFIC_MONTH_5  count   8866.00
mean      154.28
std       210.5...
14  VOICE_TRAFFIC_MONTH_6  count   8535.00
mean       84.03
std       161.3...
15        BILLING_MONTH_1  count   9999.00
mean      102.34
std        67.7...
16        BILLING_MONTH_2  count   9998.00
mean      104.98
std        76.9...
17        BILLING_MONTH_3  count   9992.00
mean      102.68
std        66.6...
18        BILLING_MONTH_4  count   9989.00
mean      101.99
std        64.3...
19        BILLING_MONTH_5  count   9987.00
mean      102.21
std        64.0...
20        BILLING_MONTH_6  count   9979.00
mean      102.27
std        65.4...
21    DEVICE_COST_MONTH_1  count   9999.00
mean       10.81
std        75.8...
22    DEVICE_COST_MONTH_2  count   9998.00
mean       10.59
std        37.0...
23    DEVICE_COST_MONTH_3  count   9992.00
mean       11.71
std        44.4...
24    DEVICE_COST_MONTH_4  count   9989.00
mean       11.55
std        34.1...
```

```
25    DEVICE_COST_MONTH_5  count   9987.00
mean      12.51
std       38.8...
26    DEVICE_COST_MONTH_6  count   9979.00
mean      12.98
std       39.5...
27   LINE_ACTIVATION_DATE  count       9179.00
mean    20136051.57
std   ...
28     MONTHS_LAST_DEVICE  count   7682.00
mean      25.34
std       12.8...
29          DURATION_LINE  count   9179.00
mean      62.37
std       52.0...
30  PREVIOUS_DEVICE_MODEL  count        6169
unique         580
top     ...
31  PREVIOUS_DEVICE_MANUF  count              6169
unique                6...
32  PREVIOUS_DEVICE_BRAND  count        6169
unique         5
top       ...
```

Replace this cell with a brief commentary comparing the previous results for **DURATION_LINE** (time that the customer has had a line) with the ones from the `describe` function.

Indicate all the differences between the statistics that `describe` computed, and the statistics you computed (e.g., missing or extra computations).

```
The custom computations for DURATION_LINE used functions (np.nanmean,
np.nanmedian, np.nanmin, and np.nanmax) which directly provide the
mean, median, minimum, and maximum, ignoring any NaN values. In
contrast, the describe function returns additional statistics
including the count of non-null values, standard deviation, and the
25th and 75th percentiles along with the min, median (50%), and max.

Thus, the differences are:
• Our custom results include only mean, median, min, and max.
• The describe function also reports the non-null count, standard
deviation, and quartile values (25% and 75%), which provide more
insight into the distribution.

Both approaches ignore NaNs, but describe supplies extra context about
the spread and variability of the data.

  Cell In[7], line 1
    The custom computations for DURATION_LINE used functions
(np.nanmean, np.nanmedian, np.nanmin, and np.nanmax) which directly
provide the mean, median, minimum, and maximum, ignoring any NaN
```

```
values. In contrast, the describe function returns additional
statistics including the count of non-null values, standard deviation,
and the 25th and 75th percentiles along with the min, median (50%),
and max.

^
SyntaxError: invalid decimal literal
```

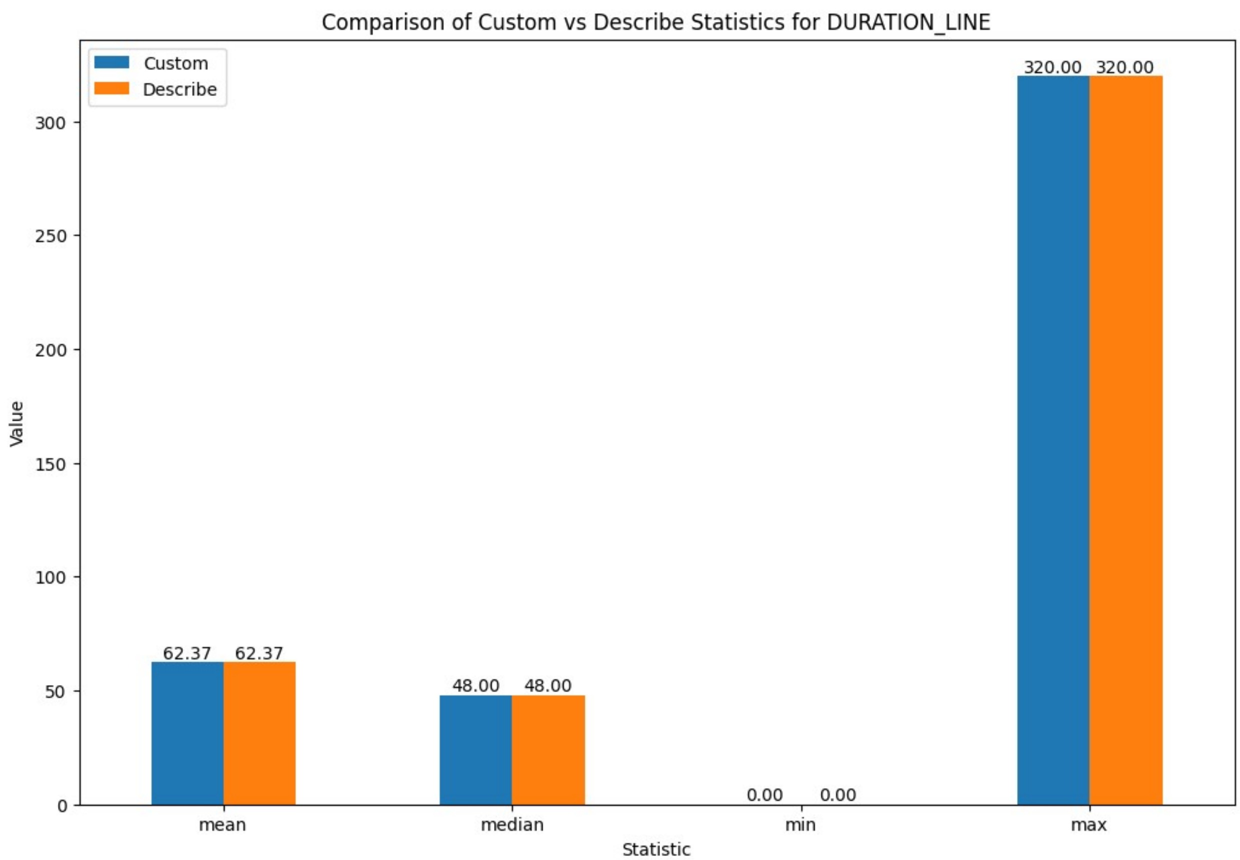
```python
# Compute the count of non-NaN values in DURATION_LINE using two
approaches:
custom_count = input_dataset['DURATION_LINE'].count()
describe_count = input_dataset['DURATION_LINE'].describe()['count']

print("Custom computed count (non-NaN):", custom_count)
print("Describe count:", describe_count)
```

```
Custom computed count (non-NaN): 9179
Describe count: 9179.0
```

```python
# Extract custom statistics for DURATION_LINE from column_type_info_df
custom_stats = column_type_info_df.loc[
  column_type_info_df['column_name'] == 'DURATION_LINE',
  ['mean', 'median', 'min', 'max']
].iloc[0]

# Compute describe statistics from the input_dataset for DURATION_LINE
describe_stats = input_dataset['DURATION_LINE'].describe()
# Rename the '50%' value to 'median'
describe_stats = describe_stats[['mean', 'min', '50%',
'max']].rename({'50%': 'median'})

# Create a comparison dataframe using both
comparison_df = pd.DataFrame({
  'Custom': custom_stats,
  'Describe': describe_stats
}).reindex(['mean', 'median', 'min', 'max'])

print(comparison_df)

ax = comparison_df.plot(kind='bar', rot=0, figsize=(12, 8))
plt.title('Comparison of Custom vs Describe Statistics for
DURATION_LINE')
plt.xlabel('Statistic')
plt.ylabel('Value')

# Annotate each bar with its value
```

```
for container in ax.containers:
  ax.bar_label(container, fmt='%.2f')
plt.show()
```

	Custom	Describe
mean	62.37	62.37
median	48.00	48.00
min	0.00	0.00
max	320.00	320.00



1.2. Inventory of device models

In exploratory data analysis, it is very useful to do an **inventory** or **census** of the possible values of a variable. For us, a census will be a frequency table in which you show the possible values of a variable, and their frequency, in decreasing order of frequency.

(Remove this cell when delivering.)

Replace this cell with code to display a census of PREVIOUS_DEVICE_MODEL and PREVIOUS_DEVICE_BRAND. You should create and display a dataframe in each case.

The most common device model and the most common device brand do not match, why do you think it is so? Replace this cell with an explanation.

```python
# Census for PREVIOUS_DEVICE_MODEL: frequency count sorted in
descending order
census_model =
input_dataset['PREVIOUS_DEVICE_MODEL'].value_counts().reset_index()
census_model.columns = ['PREVIOUS_DEVICE_MODEL', 'Frequency']
display(census_model)

# Census for PREVIOUS_DEVICE_BRAND: frequency count sorted in
descending order
census_brand =
input_dataset['PREVIOUS_DEVICE_BRAND'].value_counts().reset_index()
census_brand.columns = ['PREVIOUS_DEVICE_BRAND', 'Frequency']
display(census_brand)
```

```
       PREVIOUS_DEVICE_MODEL  Frequency
0                   iPhone 7        425
1                   iPhone 6        250
2          Samsung Galaxy J5        243
3                  iPhone 6S        212
4     Samsung Galaxy J1 Mini        204
..                       ...        ...
575         LG Optimus L3 II          1
576  Lenovo S930,Lenovo S939          1
577         Samsung Corby II          1
578                 SGH-U600          1
579              Nokia 1100           1

[580 rows x 2 columns]

  PREVIOUS_DEVICE_BRAND  Frequency
0              Samsung       1877
1               Outros       1592
2                Apple       1548
3             Motorola        638
4                   LG        514
```

```python
# Count unique device model names for iPhone's
unique_iphone_models =
input_dataset[input_dataset['PREVIOUS_DEVICE_MODEL'].str.startswith("i
Phone", na=False)]['PREVIOUS_DEVICE_MODEL'].nunique()

# Count unique device model names for Samsung's
unique_samsung_models =
input_dataset[input_dataset['PREVIOUS_DEVICE_MODEL'].str.startswith("S
amsung", na=False)]['PREVIOUS_DEVICE_MODEL'].nunique()

print("Unique iPhone device models:", unique_iphone_models)
iphone_models =
input_dataset[input_dataset['PREVIOUS_DEVICE_MODEL'].str.startswith("i
Phone", na=False)]['PREVIOUS_DEVICE_MODEL'].unique()
```

```
display(iphone_models)


print("Unique Samsung device models:", unique_samsung_models)
samsung_models =
input_dataset[input_dataset['PREVIOUS_DEVICE_MODEL'].str.startswith("S
amsung", na=False)]['PREVIOUS_DEVICE_MODEL'].unique()
display(samsung_models)
```

```
Unique iPhone device models: 13
```

```
array(['iPhone 6', 'iPhone 6S', 'iPhone 4S', 'iPhone 7', 'iPhone 7
Plus',
       'iPhone 5S', 'iPhone SE', 'iPhone 6S Plus', 'iPhone 6 Plus',
       'iPhone 5C', 'iPhone 4', 'iPhone 5', 'iPhone 3GS'],
dtype=object)
```

```
Unique Samsung device models: 100
```

```
array(['Samsung Galaxy J1 Mini', 'Samsung Galaxy J5',
       'Samsung Galaxy J1 2016', 'Samsung Galaxy S4 Mini',
       'Samsung Galaxy J7', 'Samsung Galaxy Gran Prime 2016',
       'Samsung Galaxy S7 Edge', 'Samsung Galaxy S III Neo Duos',
       'Samsung Galaxy J7 Prime', 'Samsung Galaxy S6',
       'Samsung Galaxy Young 2', 'Samsung Galaxy Grand Neo',
       'Samsung Galaxy A7 2016', 'Samsung Galaxy S7',
       'Samsung Galaxy Pocket 2 Duos', 'Samsung E1207',
       'Samsung Galaxy A9 Pro', 'Samsung Galaxy S5',
       'Samsung Galaxy Note 4', 'Samsung Galaxy A5',
       'Samsung Galaxy SIII', 'Samsung Galaxy Win Duos',
       'Samsung Galaxy Win 2', 'Samsung Galaxy Ace 4 Neo',
       'Samsung Galaxy S5 Mini', 'Samsung S III Mini Refresh',
       'Samsung Galaxy J7 2016', 'Samsung Galaxy J3 2016',
       'Samsung Galaxy J2', 'Samsung Galaxy Gran Prime', 'Samsung
E1086',
       'Samsung Galaxy S5 Neo', 'Samsung Galaxy Gran Prime Duos',
       'Samsung Galaxy A7 2017', 'Samsung Galaxy SII Duos TV',
       'Samsung Galaxy J5 2016', 'Samsung Galaxy S8 Plus',
       'Samsung Galaxy Trend', 'Samsung Galaxy Tab A 8', 'Samsung
E1205',
       'Samsung Galaxy Tab 2 7.0', 'Samsung Galaxy Note III',
       'Samsung Galaxy J5 2017', 'Samsung Galaxy J1',
       'Samsung Duos Basic', 'Samsung Galaxy S6 Edge', 'Samsung Ch@t
222',
       'Samsung E1203', 'Samsung C276', 'Samsung Galaxy Pocket Neo',
       'Samsung Galaxy Fame Lite', 'Samsung Galaxy SIII Mini',
       'Samsung Galaxy S8', 'Samsung Ace IV LTE',
       'Samsung Galaxy Pocket Plus Duos', 'Samsung Galaxy Y',
       'Samsung Galaxy Tab E 7.0 (SM-T116BU)', 'Samsung Galaxy J7
2017',
```

```
       'Samsung Galaxy Gran Prime 4G', 'Samsung Galaxy Y Duos',
       'Samsung Galaxy A3', 'Samsung Galaxy A5 2016',
       'Samsung Galaxy Young Plus Duos TV', 'Samsung Ch@t 333',
       'Samsung Ch@t 322', 'Samsung Galaxy S4 with 4G',
       'Samsung Galaxy Grand Duos', 'Samsung Galaxy A5 2017',
       'Samsung Galaxy S Duos 2', 'Samsung Galaxy Core Plus',
       'Samsung Galaxy Gran 2 Duos TV', 'Samsung Tab 3 7" Lite',
       'Samsung Galaxy A3 2016', 'Samsung Galaxy SIII Slim',
       'Samsung Galaxy Ace 4 Duos', 'Samsung Galaxy Fame',
       'Samsung Galaxy Core 2', 'Samsung Galaxy E5',
       'Samsung Galaxy Tab 7.0 Plus', 'Samsung E2550',
       'Samsung Galaxy K Zoom', 'Samsung Galaxy S III Duos',
       'Samsung Corby II', 'Samsung Galaxy Express', 'Samsung E1195',
       'Samsung Galaxy A7', 'Samsung Galaxy J5 Prime',
       'Samsung Galaxy Tab S2 8', 'Samsung Galaxy Fit',
       'Samsung Galaxy Note 10.1', 'Samsung E2530', 'Samsung Corby',
       'Samsung Galaxy J3 2017', 'Samsung Galaxy S II Lite',
       'Samsung Galaxy Note 5', 'Samsung Galaxy Note II',
       'Samsung Galaxy Mini', 'Samsung Galaxy Fame Duos',
       'Samsung Galaxy S6 Edge+', 'Samsung Ch@t 226 Duos'],
dtype=object)
```

Based on these data validation, the most common device model is not the same as the most common device brand due to the fact that Apple products are scarced and releases phone models rarely -- compared to the vast range of the Samsung brand where they release many models that better fits their target customers.

2. Feature engineering

Feature engineering is the process of extracting valuable features from the data. This requires pre-processing, combining, normalizing, and performing other operations on the values of some features.

(Remove this cell when delivering.)

2.1. Missing values management

Not A Number (NaN) is a generic term to refer to *something that should be a number, but is not*. Usually, the value is either missing completely ("null") or contains the wrong type of object, such as a string or a concept such as infinity.

To find which columns contain NaN values, you can use the isna() function, as explained, e.g., here.

To display a column as percentages in a dataframe, you can use:

```
df['column_name'] = df['column_name'].map('{:,.2%}'.format)
```

(Remove this cell when delivering.)

```
#Get the percentage of NaN values in the dataset
nan_percent = input_dataset.isna().mean() * 100
columns_with_nan = nan_percent[nan_percent > 0]

# Display the columns with NaN values and their respective percentage
nan_columns_df = columns_with_nan.reset_index()
nan_columns_df.columns = ['Column', 'Percentage of NaN']
nan_columns_df['Percentage of NaN'] = nan_columns_df['Percentage of
NaN'] / 100
nan_columns_df['Percentage of NaN'] = nan_columns_df['Percentage of
NaN'].map('{:,.2%}'.format)
display(nan_columns_df)
```

	Column	Percentage of NaN
0	PURCHASED_DEVICE	1.47%
1	DEVICE_VALUE	3.10%
2	LAST_DEVICE_CHANGE	23.18%
3	DATA_TRAFFIC_MONTH_1	11.32%
4	DATA_TRAFFIC_MONTH_2	11.59%
5	DATA_TRAFFIC_MONTH_3	11.54%
6	DATA_TRAFFIC_MONTH_4	11.83%
7	DATA_TRAFFIC_MONTH_5	11.34%
8	DATA_TRAFFIC_MONTH_6	14.65%
9	VOICE_TRAFFIC_MONTH_1	11.32%
10	VOICE_TRAFFIC_MONTH_2	11.59%
11	VOICE_TRAFFIC_MONTH_3	11.54%
12	VOICE_TRAFFIC_MONTH_4	11.83%
13	VOICE_TRAFFIC_MONTH_5	11.34%
14	VOICE_TRAFFIC_MONTH_6	14.65%
15	BILLING_MONTH_1	0.01%
16	BILLING_MONTH_2	0.02%
17	BILLING_MONTH_3	0.08%
18	BILLING_MONTH_4	0.11%
19	BILLING_MONTH_5	0.13%
20	BILLING_MONTH_6	0.21%
21	DEVICE_COST_MONTH_1	0.01%
22	DEVICE_COST_MONTH_2	0.02%
23	DEVICE_COST_MONTH_3	0.08%
24	DEVICE_COST_MONTH_4	0.11%
25	DEVICE_COST_MONTH_5	0.13%
26	DEVICE_COST_MONTH_6	0.21%
27	LINE_ACTIVATION_DATE	8.21%
28	MONTHS_LAST_DEVICE	23.18%
29	DURATION_LINE	8.21%
30	PREVIOUS_DEVICE_MODEL	38.31%
31	PREVIOUS_DEVICE_MANUF	38.31%
32	PREVIOUS_DEVICE_BRAND	38.31%

Replace this cell with your code to print all columns that contain at least one NaN value, and what is the percentage of NaN values in that column. (Create a dataframe with this information, and then display it.)

The way **NaNs** are managed varies according to the meaning of each variable. In some occasions, registers should be removed, filled with other columns or calculated (imputed).

- To delete all rows containing a null value, we can use dropna
- To replace null values, we can use fillna

Please note that these steps should be applied sequentially, i.e., the output of one step should be fed into the next step. You can do, for instance: `df02 = df01.operation(...)` followed by `df03 = df02.operation(...)` and so on.

(Remove this cell when delivering.)

If there is no **PURCHASED_DEVICE**, **DEVICE_VALUE**, or **PREVIOUS_DEVICE_MODEL**, the row is useless to us. Replace this cell with code to remove those rows.

Any NaN value in **DATA_TRAFFIC_MONTH_(1..6)**, **VOICE_TRAFFIC_MONTH_(1..6)**, **BILLING_MONTH_(1..6)**, or **DEVICE_COST_MONTH_(1..6)** should be assumed to be 0. Replace this cell with code to do that imputation.

If there is no **LINE_ACTIVATION_DATE**, we will assume it is equal to **LAST_DEVICE_CHANGE**. Replace this cell with code to do that imputation.

Replace this cell with code to print the header and the first five rows after this processing

```
imputation_dataset = input_dataset.copy()

# Remove rows with missing PURCHASED_DEVICE, DEVICE_VALUE, or
PREVIOUS_DEVICE_MODEL values
imputation_dataset.dropna(subset=['PURCHASED_DEVICE', 'DEVICE_VALUE',
'PREVIOUS_DEVICE_MODEL'], inplace=True)

print("Shape of dataset before removing rows with missing required
columns:", input_dataset.shape)
#print the new shape of the dataframe to verify the removal
print("Shape of dataset after removing rows with missing required
columns:", imputation_dataset.shape)
```

```
Shape of dataset before removing rows with missing required columns:
(10000, 33)
Shape of dataset after removing rows with missing required columns:
(5988, 33)
```

```
# Check for NaNs in the specified columns and print percentages
cols_to_impute = [f"DATA_TRAFFIC_MONTH_{i}" for i in range(1, 7)] + \
         [f"VOICE_TRAFFIC_MONTH_{i}" for i in range(1, 7)] + \
         [f"BILLING_MONTH_{i}" for i in range(1, 7)] + \
```

```python
        [f"DEVICE_COST_MONTH_{i}" for i in range(1, 7)]

# Calculate the percentage of NaN values per column
nan_percentages = imputation_dataset[cols_to_impute].isna().mean() *
100

# Convert the series to a DataFrame for a cleaner display
nan_percentage_df = nan_percentages.reset_index()
nan_percentage_df.columns = ['Column', 'NaN Percentage (%)']

print("NaN percentages per column:")
display(nan_percentage_df)
```

```
NaN percentages per column:
```

	Column	NaN Percentage (%)
0	DATA_TRAFFIC_MONTH_1	11.34
1	DATA_TRAFFIC_MONTH_2	11.51
2	DATA_TRAFFIC_MONTH_3	11.36
3	DATA_TRAFFIC_MONTH_4	11.59
4	DATA_TRAFFIC_MONTH_5	10.99
5	DATA_TRAFFIC_MONTH_6	15.10
6	VOICE_TRAFFIC_MONTH_1	11.34
7	VOICE_TRAFFIC_MONTH_2	11.51
8	VOICE_TRAFFIC_MONTH_3	11.36
9	VOICE_TRAFFIC_MONTH_4	11.59
10	VOICE_TRAFFIC_MONTH_5	10.99
11	VOICE_TRAFFIC_MONTH_6	15.10
12	BILLING_MONTH_1	0.00
13	BILLING_MONTH_2	0.02
14	BILLING_MONTH_3	0.08
15	BILLING_MONTH_4	0.10
16	BILLING_MONTH_5	0.12
17	BILLING_MONTH_6	0.18
18	DEVICE_COST_MONTH_1	0.00
19	DEVICE_COST_MONTH_2	0.02
20	DEVICE_COST_MONTH_3	0.08
21	DEVICE_COST_MONTH_4	0.10
22	DEVICE_COST_MONTH_5	0.12
23	DEVICE_COST_MONTH_6	0.18

```python
# Define the list of columns to impute
cols_to_impute = [f"DATA_TRAFFIC_MONTH_{i}" for i in range(1, 7)] + \
        [f"VOICE_TRAFFIC_MONTH_{i}" for i in range(1, 7)] + \
        [f"BILLING_MONTH_{i}" for i in range(1, 7)] + \
        [f"DEVICE_COST_MONTH_{i}" for i in range(1, 7)]

# Fill NaN values with 0 for the chosen columns in the
imputation_dataset DataFrame
```

```
imputation_dataset[cols_to_impute] =
imputation_dataset[cols_to_impute].fillna(0)

# Optionally, display the count of NaNs in these columns to verify the
imputation worked correctly
print("Number of missing values after imputation:")
print(imputation_dataset[cols_to_impute].isna().sum())
```

```
Number of missing values after imputation:
DATA_TRAFFIC_MONTH_1     0
DATA_TRAFFIC_MONTH_2     0
DATA_TRAFFIC_MONTH_3     0
DATA_TRAFFIC_MONTH_4     0
DATA_TRAFFIC_MONTH_5     0
DATA_TRAFFIC_MONTH_6     0
VOICE_TRAFFIC_MONTH_1    0
VOICE_TRAFFIC_MONTH_2    0
VOICE_TRAFFIC_MONTH_3    0
VOICE_TRAFFIC_MONTH_4    0
VOICE_TRAFFIC_MONTH_5    0
VOICE_TRAFFIC_MONTH_6    0
BILLING_MONTH_1          0
BILLING_MONTH_2          0
BILLING_MONTH_3          0
BILLING_MONTH_4          0
BILLING_MONTH_5          0
BILLING_MONTH_6          0
DEVICE_COST_MONTH_1      0
DEVICE_COST_MONTH_2      0
DEVICE_COST_MONTH_3      0
DEVICE_COST_MONTH_4      0
DEVICE_COST_MONTH_5      0
DEVICE_COST_MONTH_6      0
dtype: int64
```

```
# Check for NaNs in the specified columns and print percentages
cols_to_impute = [f"DATA_TRAFFIC_MONTH_{i}" for i in range(1, 7)] + \
         [f"VOICE_TRAFFIC_MONTH_{i}" for i in range(1, 7)] + \
         [f"BILLING_MONTH_{i}" for i in range(1, 7)] + \
         [f"DEVICE_COST_MONTH_{i}" for i in range(1, 7)]

# Calculate the percentage of NaN values per column
nan_percentages =imputation_dataset[cols_to_impute].isna().mean() *
100

# Convert the series to a DataFrame for a cleaner display
nan_percentage_df = nan_percentages.reset_index()
nan_percentage_df.columns = ['Column', 'NaN Percentage (%)']
```

```
print("NaN percentages per column:")
display(nan_percentage_df)

NaN percentages per column:

                   Column  NaN Percentage (%)
0    DATA_TRAFFIC_MONTH_1                0.00
1    DATA_TRAFFIC_MONTH_2                0.00
2    DATA_TRAFFIC_MONTH_3                0.00
3    DATA_TRAFFIC_MONTH_4                0.00
4    DATA_TRAFFIC_MONTH_5                0.00
5    DATA_TRAFFIC_MONTH_6                0.00
6   VOICE_TRAFFIC_MONTH_1                0.00
7   VOICE_TRAFFIC_MONTH_2                0.00
8   VOICE_TRAFFIC_MONTH_3                0.00
9   VOICE_TRAFFIC_MONTH_4                0.00
10  VOICE_TRAFFIC_MONTH_5                0.00
11  VOICE_TRAFFIC_MONTH_6                0.00
12        BILLING_MONTH_1                0.00
13        BILLING_MONTH_2                0.00
14        BILLING_MONTH_3                0.00
15        BILLING_MONTH_4                0.00
16        BILLING_MONTH_5                0.00
17        BILLING_MONTH_6                0.00
18    DEVICE_COST_MONTH_1                0.00
19    DEVICE_COST_MONTH_2                0.00
20    DEVICE_COST_MONTH_3                0.00
21    DEVICE_COST_MONTH_4                0.00
22    DEVICE_COST_MONTH_5                0.00
23    DEVICE_COST_MONTH_6                0.00

missing_after =
imputation_dataset['LINE_ACTIVATION_DATE'].isna().sum()
print("Number of missing values in LINE_ACTIVATION_DATE before
imputation:", missing_after)

Number of missing values in LINE_ACTIVATION_DATE before imputation:
468


# Replace missing LINE_ACTIVATION_DATE values with values from
LAST_DEVICE_CHANGE
imputation_dataset['LINE_ACTIVATION_DATE'] =
imputation_dataset['LINE_ACTIVATION_DATE'].fillna(imputation_dataset['
LAST_DEVICE_CHANGE'])

# Optionally, verify that no missing values remain in
LINE_ACTIVATION_DATE
missing_after =
imputation_dataset['LINE_ACTIVATION_DATE'].isna().sum()
```

```
print("Number of missing values in LINE_ACTIVATION_DATE after
imputation:", missing_after)
```

```
Number of missing values in LINE_ACTIVATION_DATE after imputation: 124
```

Upon reviewing the still missing values, the issue roots from the values for the 'LAST_DEVICE_CHANGE' sometimes containing NaN values themselves and having an improper date format.

The code below solves the problem, but is commented out since it wasn't instructed to be done.

```
# # Drop rows with NaN in LAST_DEVICE_CHANGE
# imputation_dataset.dropna(subset=['LAST_DEVICE_CHANGE'],
inplace=True)

# # Convert LAST_DEVICE_CHANGE from float to datetime format (%Y%m%d)
# imputation_dataset['LAST_DEVICE_CHANGE'] =
imputation_dataset['LAST_DEVICE_CHANGE'].apply(
#   lambda x: pd.to_datetime(str(int(x)), format='%Y%m%d',
errors='coerce')
# )

# # For LINE_ACTIVATION_DATE, first convert non-missing values from
float to datetime
# imputation_dataset['LINE_ACTIVATION_DATE'] =
imputation_dataset['LINE_ACTIVATION_DATE'].apply(
#   lambda x: pd.to_datetime(str(int(x)), format='%Y%m%d',
errors='coerce') if pd.notna(x) else x
# )

# # Replace missing LINE_ACTIVATION_DATE with LAST_DEVICE_CHANGE
values
# imputation_dataset['LINE_ACTIVATION_DATE'] =
imputation_dataset['LINE_ACTIVATION_DATE'].fillna(imputation_dataset['
LAST_DEVICE_CHANGE'])

# # Verify that no missing values remain in LINE_ACTIVATION_DATE
# missing_after =
imputation_dataset['LINE_ACTIVATION_DATE'].isna().sum()
# print("Number of missing values in LINE_ACTIVATION_DATE after
imputation:", missing_after)
```

If `df` is a dataframe, `df.shape` contains a tuple with the number of rows and the number of columns of the data frame. You should now print something like this:

```
Rows in the original dataset: M
Rows in the new dataset: N ((100*(M-N)/M)% less)
```

(Remove this cell when delivering.)

Replace this cell with code to print the number of rows of the original dataset, the number of rows of the new dataset, and the percentage of rows that were dropped, as well as the names of the columns that still contain NaN values, if any.

```
# Get the number of rows in the original and new datasets
original_rows = input_dataset.shape[0]
new_rows = imputation_dataset.shape[0]

# Compute the percentage of rows dropped
dropped_pct = ((original_rows - new_rows) / original_rows) * 100

print(f"Rows in the original dataset: {original_rows}")
print(f"Rows in the new dataset: {new_rows} ({dropped_pct:.2f}%
less)")

# Identify columns that still contain NaN values in the new dataset
columns_with_nan =
imputation_dataset.columns[imputation_dataset.isna().any()]

if len(columns_with_nan) > 0:
  print("Columns with NaN values:", list(columns_with_nan))
else:
  print("No columns with NaN values remain.")
```

```
Rows in the original dataset: 10000
Rows in the new dataset: 5988 (40.12% less)
Columns with NaN values: ['LAST_DEVICE_CHANGE',
'LINE_ACTIVATION_DATE', 'MONTHS_LAST_DEVICE', 'DURATION_LINE']
```

2.2. Distributions, outliers, and correlations

We will now plot the distributions of some variables and apply some transformations.

- You can use Seaborn library with `kde=False` to create a histogram.
- You can use pandas.DataFrame.plot with `kind='box'` to create a boxplot.

Remember to include a title, x-axis label, and y-axis label. All of your plots delivered throughout the course should include these elements. Example:

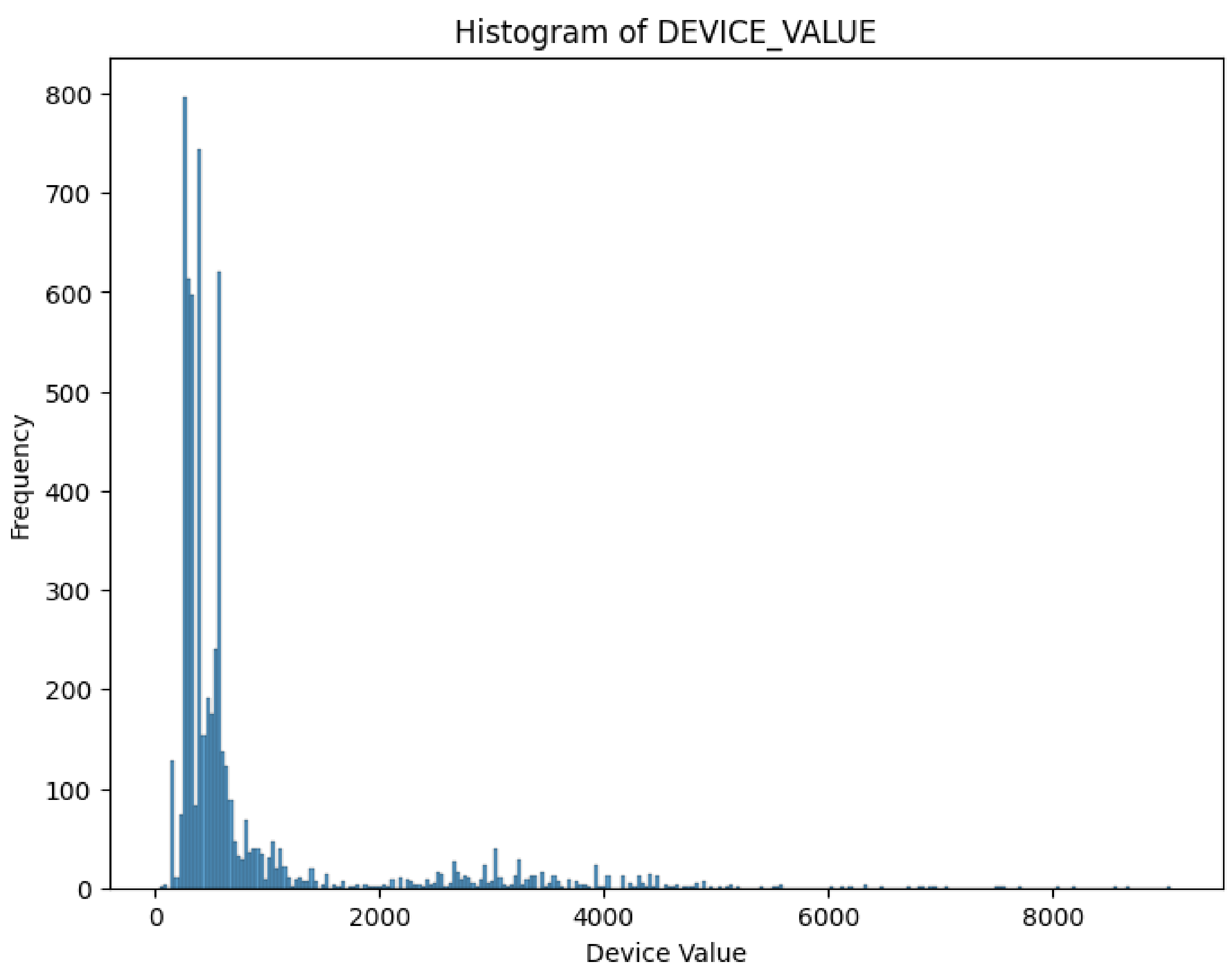
```
ax = sns.histplot(...)
ax.set(title=..., xlabel=..., ylabel=...)
```

(Remove this cell when delivering.)

Replace this cell with code to plot a histogram of **DEVICE_VALUE** and **DURATION_LINE**. Remember to include a title, and labels on the x axis and y axis

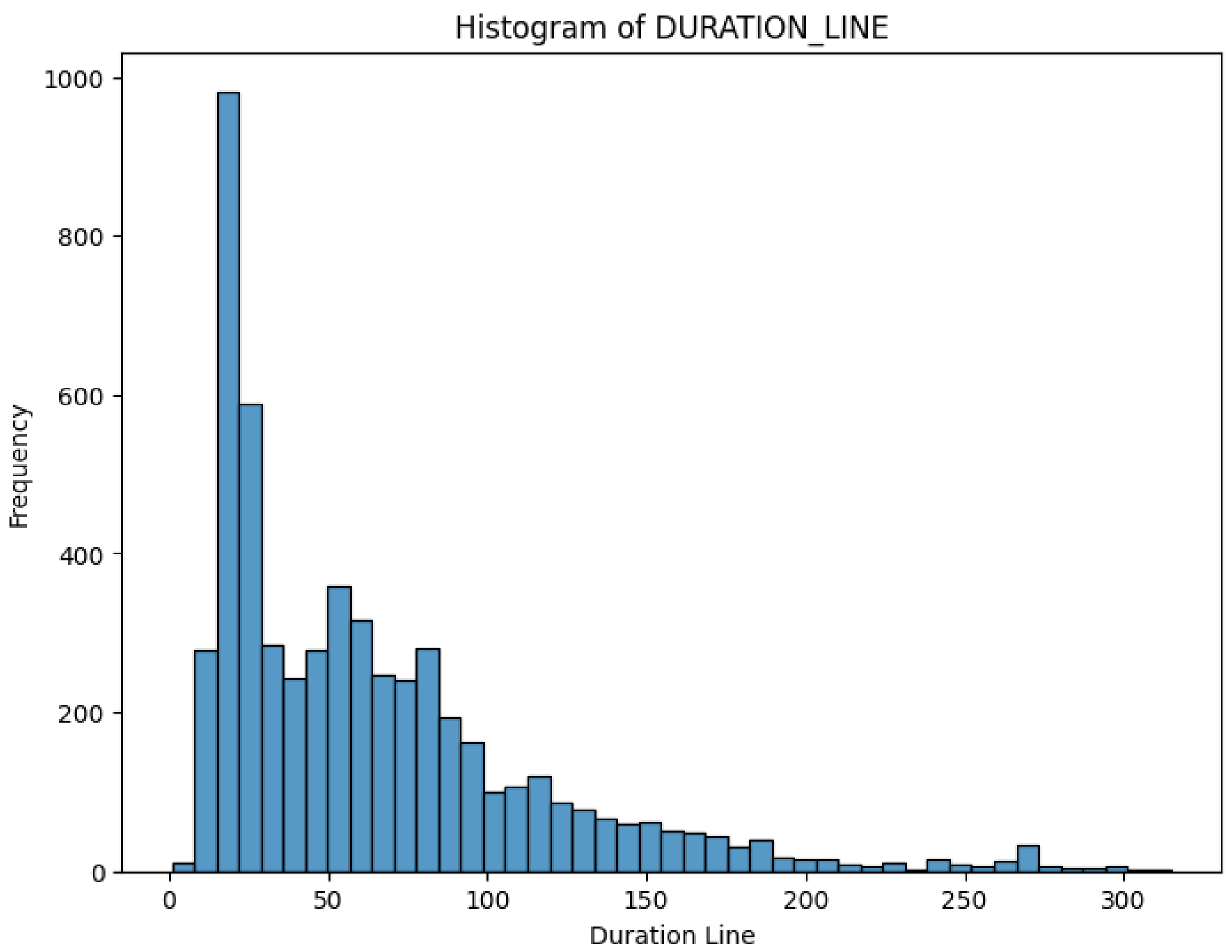
Include after each histogram a markdown cell where you indicate if you recognize any specific distribution (normal, exponential, uniform, ...) or any characteristic of the distribution (unimodal, bimodal).

```
import matplotlib.pyplot as plt

# Histogram for DEVICE_VALUE
plt.figure(figsize=(8, 6))
sns.histplot(data=imputation_dataset, x='DEVICE_VALUE', kde=False)
plt.title('Histogram of DEVICE_VALUE')
plt.xlabel('Device Value')
plt.ylabel('Frequency')
plt.show()
```



The histogram for DEVICE_VALUE suggests that the distribution is unimodal but noticeably right-skewed. Most device values are concentrated towards the lower end, with a long tail reaching into higher values. This shape does not follow a symmetric normal distribution; instead, it appears more similar to a log-normal or an exponential decay, where high-value devices are less common.

```
# Histogram for DURATION_LINE
plt.figure(figsize=(8, 6))
sns.histplot(data=imputation_dataset, x='DURATION_LINE', kde=False)
plt.title('Histogram of DURATION_LINE')
plt.xlabel('Duration Line')
plt.ylabel('Frequency')
plt.show()
```



The histogram for DURATION_LINE shows a unimodal distribution with a clear concentration around its median value. While the bulk of the data clusters in a specific range, there is also evidence of some outliers on the higher end, indicating moderate right-skewness. This distribution does not appear perfectly normal, and the presence of several extreme values suggests that further investigation or possible transformation might be needed if a normality assumption is important.

To be able to see better these histograms when comparing them, you can use:

```
sns.histplot(data=..., bins=20, fill=False)
```

To use logarithmic scale on the X axis or the Y axis, you can use `plt.xscale('log')` or `plt.yscale('log')`.

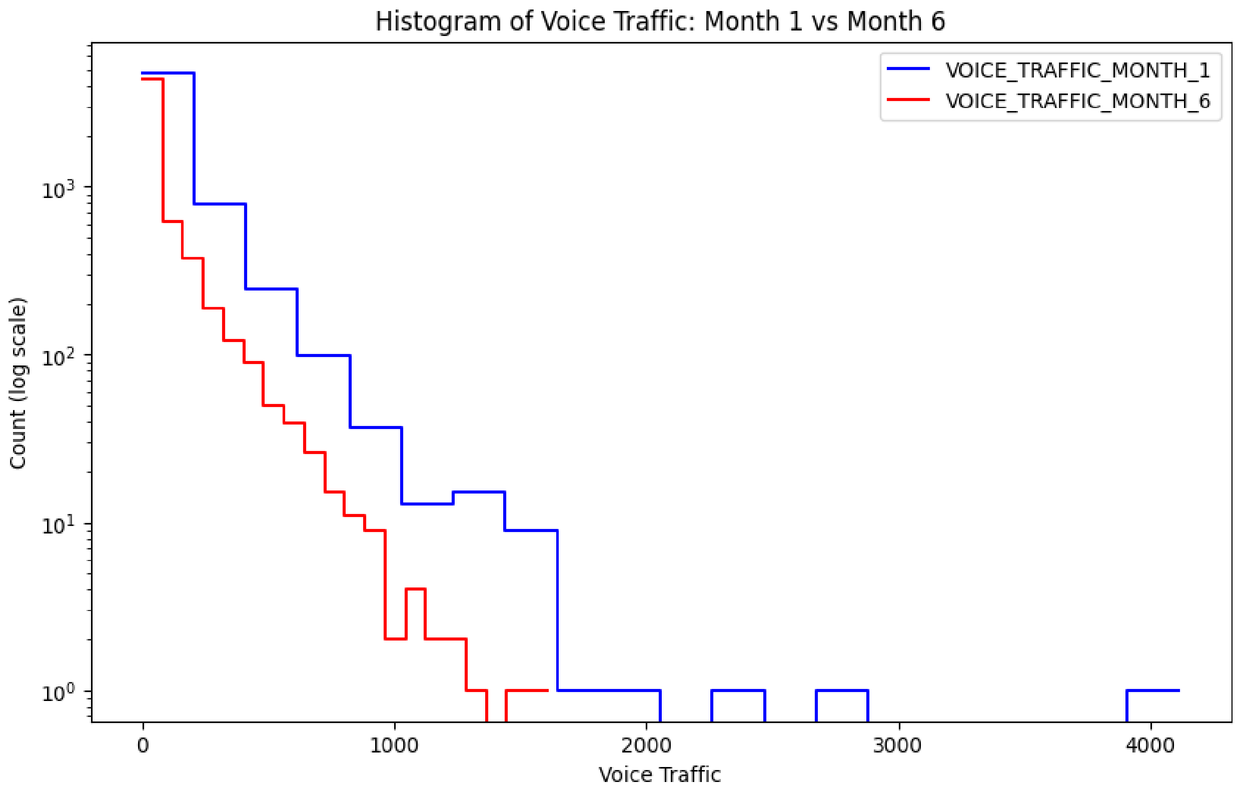
(Remove this cell when delivering.)

Replace this cell with a series of cells with code to plot a histogram comparing **VOICE_TRAFFIC_MONTH_1** against **VOICE_TRAFFIC_MONTH_6**, and **BILLING_MONTH_1** against **BILLING_MONTH_6**. Remember to include a title, labels on the x axis and y axis, and a legend.

Both plots should use logarithmic scale on the y axis

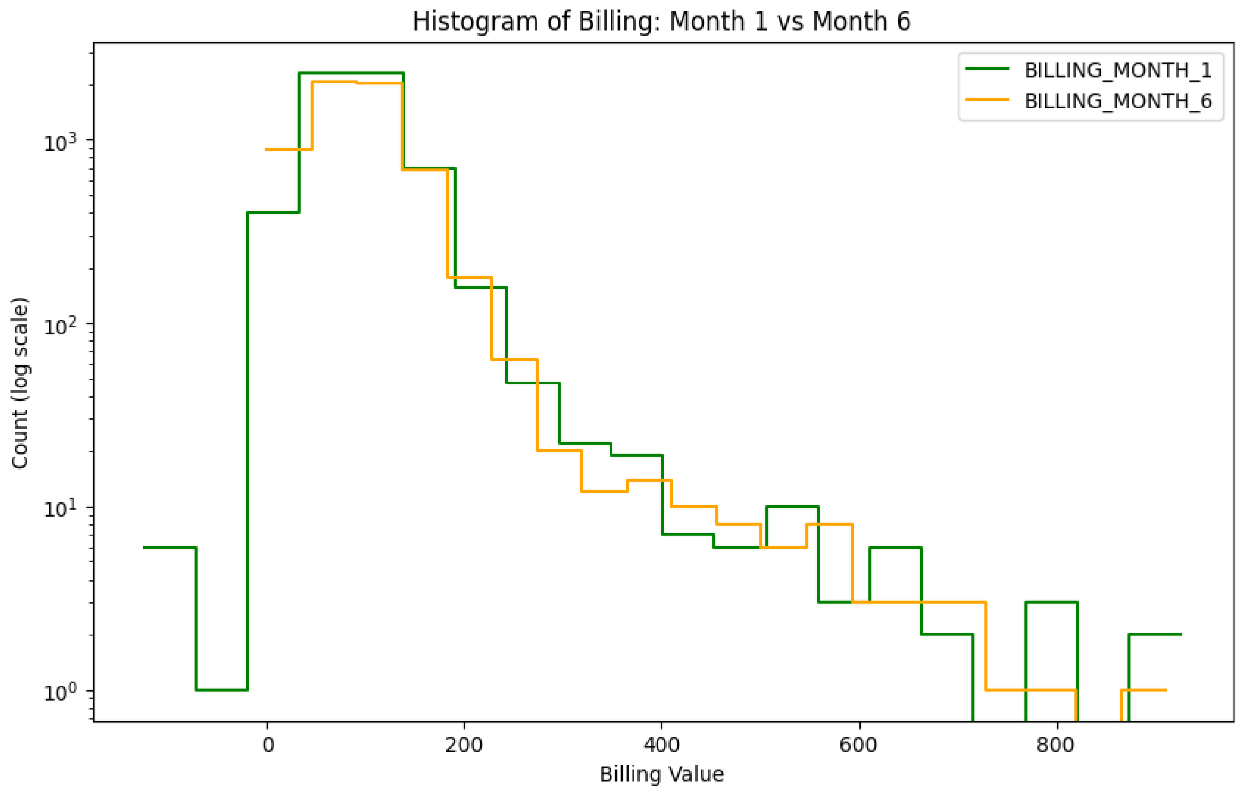
Include after both histograms your comment on the differences between month 1 and month 6.

```
# Plot histogram comparing VOICE_TRAFFIC_MONTH_1 vs
VOICE_TRAFFIC_MONTH_6
plt.figure(figsize=(10, 6))
sns.histplot(data=imputation_dataset, x='VOICE_TRAFFIC_MONTH_1',
bins=20, color='blue', label='VOICE_TRAFFIC_MONTH_1', kde=False,
stat="count", element="step", fill=False)
sns.histplot(data=imputation_dataset, x='VOICE_TRAFFIC_MONTH_6',
bins=20, color='red', label='VOICE_TRAFFIC_MONTH_6', kde=False,
stat="count", element="step", fill=False)
plt.yscale('log')
plt.title('Histogram of Voice Traffic: Month 1 vs Month 6')
plt.xlabel('Voice Traffic')
plt.ylabel('Count (log scale)')
plt.legend()
plt.show()
```



The histogram above shows the distributions of VOICE_TRAFFIC_MONTH_1 and VOICE_TRAFFIC_MONTH_6.
Notice that we used a logarithmic scale on the y axis to better visualize differences in frequency, especially for large and small counts. The overlapping histograms (in blue and red) provide a direct comparison between month 1 and month 6.

```
# Plot histogram comparing BILLING_MONTH_1 vs BILLING_MONTH_6

plt.figure(figsize=(10, 6))
sns.histplot(data=imputation_dataset, x='BILLING_MONTH_1', bins=20,
color='green', label='BILLING_MONTH_1', kde=False, stat="count",
element="step", fill=False)
sns.histplot(data=imputation_dataset, x='BILLING_MONTH_6', bins=20,
color='orange', label='BILLING_MONTH_6', kde=False, stat="count",
element="step", fill=False)
plt.yscale('log')
plt.title('Histogram of Billing: Month 1 vs Month 6')
plt.xlabel('Billing Value')
plt.ylabel('Count (log scale)')
plt.legend()
plt.show()
```



The histogram above compares BILLING_MONTH_1 and BILLING_MONTH_6 distributions. Again, a logarithmic scale is used for the y axis. Comparing the two histograms (green for month 1 and orange for month 6) helps illustrate any shifts or differences in billing values between the first and sixth month.

Variables having exponential distribution can be processed and visualized better after transforming them, usually by applying the `log(x+1)` function (we want to avoid zeros, hence the +1).

(Remove this cell when delivering.)

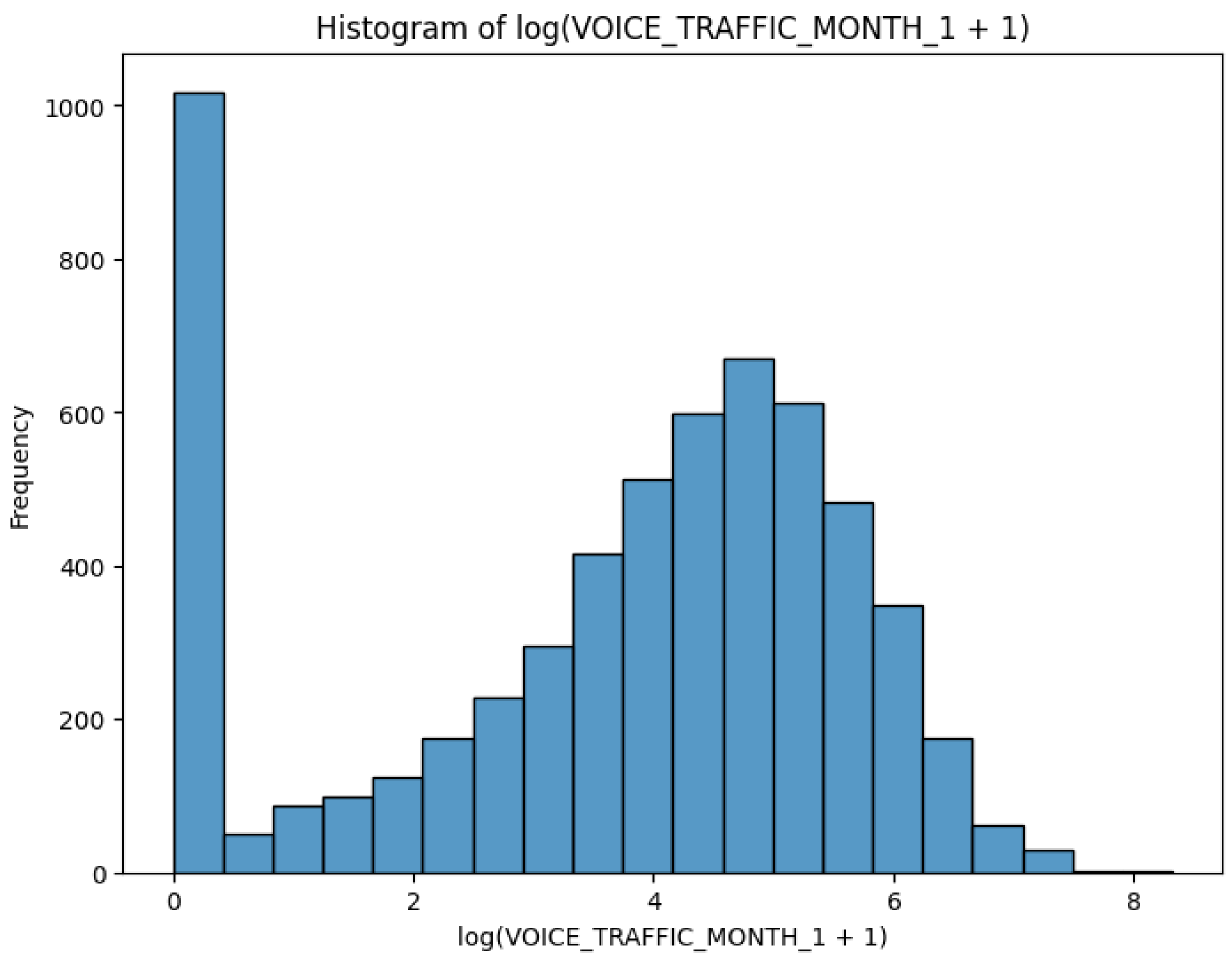
Replace this cell with code to apply **log(x+1)** to **VOICE_TRAFFIC_MONTH_1** and plot its new distribution.

Replace this cell with code to create thre boxplots, each of them for one of the variables **DATA_TRAFFIC_MONTH_6**, **VOICE_TRAFFIC_MONTH_6** and **BILLING_MONTH_6**. Remember to include a title and a label for the y axis.

Replace this cell with a brief commentary indicating which extreme values would you use as threshold for **outliers** in these variables, by looking at these box plots

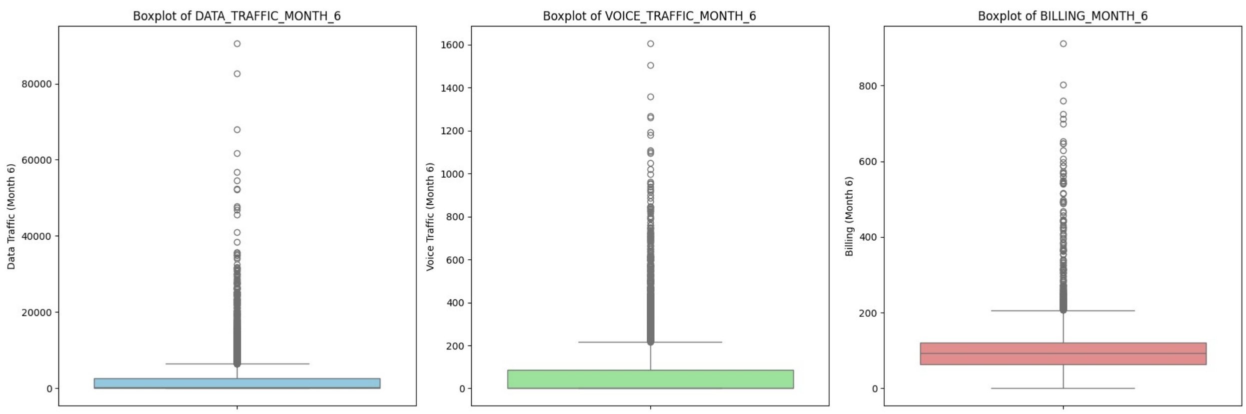
```
# Create a new transformed column using the log(x+1) function
voice_traffic_log =
np.log1p(imputation_dataset['VOICE_TRAFFIC_MONTH_1'])
```

```
# Plot the histogram of the transformed variable
plt.figure(figsize=(8, 6))
sns.histplot(voice_traffic_log, bins=20, kde=False)
plt.title('Histogram of log(VOICE_TRAFFIC_MONTH_1 + 1)')
plt.xlabel('log(VOICE_TRAFFIC_MONTH_1 + 1)')
plt.ylabel('Frequency')
plt.show()
```




```
# Create three boxplots using subplots
import matplotlib.pyplot as plt

# Define the figure size and create a subplot with 3 axes
fig, axes = plt.subplots(1, 3, figsize=(18, 6))

# Boxplot for DATA_TRAFFIC_MONTH_6
sns.boxplot(y=imputation_dataset["DATA_TRAFFIC_MONTH_6"], ax=axes[0],
color='skyblue')
axes[0].set_title("Boxplot of DATA_TRAFFIC_MONTH_6")
axes[0].set_ylabel("Data Traffic (Month 6)")
```

```
# Boxplot for VOICE_TRAFFIC_MONTH_6
sns.boxplot(y=imputation_dataset["VOICE_TRAFFIC_MONTH_6"], ax=axes[1],
color='lightgreen')
axes[1].set_title("Boxplot of VOICE_TRAFFIC_MONTH_6")
axes[1].set_ylabel("Voice Traffic (Month 6)")

# Boxplot for BILLING_MONTH_6
sns.boxplot(y=imputation_dataset["BILLING_MONTH_6"], ax=axes[2],
color='lightcoral')
axes[2].set_title("Boxplot of BILLING_MONTH_6")
axes[2].set_ylabel("Billing (Month 6)")

plt.tight_layout()
plt.show()
```



Based on a visual inspection of the box plots, one typical strategy is to flag values outside the 1.5×IQR (interquartile range) as potential outliers. For these three variables, you might comment as follows:

• For DATA_TRAFFIC_MONTH_6, the bulk of the data is concentrated within a relatively narrow range (with the median around 1,200–1,300) while a few cases reach tens of thousands. In practice, you might set the upper threshold at Q3 + 1.5×IQR (for example, if Q3 ≈ 1,800 and IQR ≈ 800, the threshold would be around 1,800 + 1,200 = 3,000). Values far above that level could be considered outliers.

• For VOICE_TRAFFIC_MONTH_6, the median appears very low (around 6) with many values near zero, but there is a long tail of higher values. Although many observations are low, a few extreme values may exist (for example, if Q3 is around 15 and the IQR is 10, then 15 + 1.5×10 = 30 could serve as an upper threshold). Data points above that threshold would be candidates for further inspection as outliers.

• For BILLING_MONTH_6, the majority of values cluster around the mid-90s (with a median close to 95), but again there are some cases that extend to over 1,000. Using the box plot you could estimate an upper limit (say, if Q3 ≈ 100 and IQR ≈ 10–15, a threshold might be around 115–125). Values beyond this range should be carefully reviewed.

In summary, using the 1.5×IQR rule on these box plots provides a systematic starting point. However, because the distributions are highly skewed, it may be advisable to review and adjust these thresholds in the context of your domain knowledge before deciding which values to treat as outliers.

In this dataset, there are many dependencies between different attributes, e.g., a large voice traffic will probably be associated with a large data traffic, a more expensive bill, and possibly a more expensive device (`DEVICE_VALUE`).

You can use pandas.DataFrame.corr to compute a correlation matrix, and matplotlib.pyplot.matshow to show this graphically.

To compute Pearson correlations, you use:

```
df.corr(method='pearson', numeric_only=True)
```

(Remove this cell when delivering.)

Replace this cell with code to calculate the correlation between all traffic attributes (i.e., voice and data), duration line, billing, device cost and device value. Display the result as a table with rows and columns corresponding to columns, and cells indicating correlations. Display the result as an image using `matshow`

Replace this cell with a brief commentary on the results. Is the billing more correlated, in general, with the data traffic or with the voice traffic?

1. We first build a list of column names including the six months for data and voice traffic, billing, and device cost. We also add "DURATION_LINE" and "DEVICE_VALUE".
2. We then compute the Pearson correlation matrix for these columns.
3. Using display() (available in Jupyter Notebook) the correlation matrix is shown as a table.
4. Lastly, we use plt.matshow() along with plt.xticks() and plt.yticks() to visualize the correlation matrix as an image with a colorbar and labels.

```
# List of columns to include in the correlation analysis
corr_cols = (
  [f"DATA_TRAFFIC_MONTH_{i}" for i in range(1, 7)] +
  [f"VOICE_TRAFFIC_MONTH_{i}" for i in range(1, 7)] +
  [f"BILLING_MONTH_{i}" for i in range(1, 7)] +
  [f"DEVICE_COST_MONTH_{i}" for i in range(1, 7)] +
  ["DURATION_LINE", "DEVICE_VALUE"]
)

# Calculate the Pearson correlation matrix for the selected columns
corr_matrix = imputation_dataset[corr_cols].corr(method='pearson')

# Display the correlation matrix as a table
print("Correlation Matrix:")
display(corr_matrix)

# Plot the correlation matrix using matshow
plt.figure(figsize=(22, 20))
```

```
plt.matshow(corr_matrix)
plt.title("Correlation Matrix of Traffic, Duration, Billing, Device
Cost and Device Value", pad=40)
plt.colorbar()
plt.xticks(range(len(corr_matrix.columns)), corr_matrix.columns,
rotation=90)
plt.yticks(range(len(corr_matrix.columns)), corr_matrix.columns)
plt.show()
```

```
Correlation Matrix:

                       DATA_TRAFFIC_MONTH_1  DATA_TRAFFIC_MONTH_2  \
DATA_TRAFFIC_MONTH_1                   1.00                  0.76
DATA_TRAFFIC_MONTH_2                   0.76                  1.00
DATA_TRAFFIC_MONTH_3                   0.73                  0.97
DATA_TRAFFIC_MONTH_4                   0.70                  0.79
DATA_TRAFFIC_MONTH_5                   0.66                  0.76
DATA_TRAFFIC_MONTH_6                   0.62                  0.69
VOICE_TRAFFIC_MONTH_1                  0.09                  0.08
VOICE_TRAFFIC_MONTH_2                  0.07                  0.08
VOICE_TRAFFIC_MONTH_3                  0.07                  0.08
VOICE_TRAFFIC_MONTH_4                  0.07                  0.07
VOICE_TRAFFIC_MONTH_5                  0.08                  0.08
VOICE_TRAFFIC_MONTH_6                  0.05                  0.03
BILLING_MONTH_1                        0.18                  0.19
BILLING_MONTH_2                        0.17                  0.17
BILLING_MONTH_3                        0.18                  0.17
BILLING_MONTH_4                        0.20                  0.19
BILLING_MONTH_5                        0.19                  0.19
BILLING_MONTH_6                        0.19                  0.20
DEVICE_COST_MONTH_1                    0.01                  0.01
DEVICE_COST_MONTH_2                    0.05                  0.05
DEVICE_COST_MONTH_3                    0.08                  0.06
DEVICE_COST_MONTH_4                    0.08                  0.07
DEVICE_COST_MONTH_5                    0.08                  0.07
DEVICE_COST_MONTH_6                    0.08                  0.07
DURATION_LINE                         -0.01                  0.01
DEVICE_VALUE                           0.12                  0.11

                       DATA_TRAFFIC_MONTH_3  DATA_TRAFFIC_MONTH_4  \
DATA_TRAFFIC_MONTH_1                   0.73                  0.70
DATA_TRAFFIC_MONTH_2                   0.97                  0.79
DATA_TRAFFIC_MONTH_3                   1.00                  0.81
DATA_TRAFFIC_MONTH_4                   0.81                  1.00
DATA_TRAFFIC_MONTH_5                   0.77                  0.83
DATA_TRAFFIC_MONTH_6                   0.70                  0.72
VOICE_TRAFFIC_MONTH_1                  0.08                  0.09
VOICE_TRAFFIC_MONTH_2                  0.08                  0.09
VOICE_TRAFFIC_MONTH_3                  0.08                  0.09
VOICE_TRAFFIC_MONTH_4                  0.07                  0.09
```

VOICE_TRAFFIC_MONTH_5	0.08	0.10
VOICE_TRAFFIC_MONTH_6	0.03	0.03
BILLING_MONTH_1	0.19	0.21
BILLING_MONTH_2	0.18	0.19
BILLING_MONTH_3	0.17	0.19
BILLING_MONTH_4	0.20	0.20
BILLING_MONTH_5	0.19	0.21
BILLING_MONTH_6	0.20	0.22
DEVICE_COST_MONTH_1	0.01	0.01
DEVICE_COST_MONTH_2	0.05	0.05
DEVICE_COST_MONTH_3	0.06	0.07
DEVICE_COST_MONTH_4	0.07	0.07
DEVICE_COST_MONTH_5	0.07	0.07
DEVICE_COST_MONTH_6	0.07	0.07
DURATION_LINE	0.01	-0.02
DEVICE_VALUE	0.12	0.12

	DATA_TRAFFIC_MONTH_5	DATA_TRAFFIC_MONTH_6 \
DATA_TRAFFIC_MONTH_1	0.66	0.62
DATA_TRAFFIC_MONTH_2	0.76	0.69
DATA_TRAFFIC_MONTH_3	0.77	0.70
DATA_TRAFFIC_MONTH_4	0.83	0.72
DATA_TRAFFIC_MONTH_5	1.00	0.80
DATA_TRAFFIC_MONTH_6	0.80	1.00
VOICE_TRAFFIC_MONTH_1	0.09	0.08
VOICE_TRAFFIC_MONTH_2	0.09	0.08
VOICE_TRAFFIC_MONTH_3	0.10	0.08
VOICE_TRAFFIC_MONTH_4	0.09	0.08
VOICE_TRAFFIC_MONTH_5	0.10	0.09
VOICE_TRAFFIC_MONTH_6	0.04	0.03
BILLING_MONTH_1	0.22	0.20
BILLING_MONTH_2	0.20	0.19
BILLING_MONTH_3	0.20	0.19
BILLING_MONTH_4	0.22	0.20
BILLING_MONTH_5	0.21	0.19
BILLING_MONTH_6	0.22	0.20
DEVICE_COST_MONTH_1	0.01	0.01
DEVICE_COST_MONTH_2	0.03	0.05
DEVICE_COST_MONTH_3	0.07	0.09
DEVICE_COST_MONTH_4	0.06	0.07
DEVICE_COST_MONTH_5	0.06	0.07
DEVICE_COST_MONTH_6	0.06	0.08
DURATION_LINE	0.00	0.00
DEVICE_VALUE	0.13	0.12

	VOICE_TRAFFIC_MONTH_1	VOICE_TRAFFIC_MONTH_2 \
DATA_TRAFFIC_MONTH_1	0.09	0.07
DATA_TRAFFIC_MONTH_2	0.08	0.08
DATA_TRAFFIC_MONTH_3	0.08	0.08

```
DATA_TRAFFIC_MONTH_4                       0.09                      0.09
DATA_TRAFFIC_MONTH_5                       0.09                      0.09
DATA_TRAFFIC_MONTH_6                       0.08                      0.08
VOICE_TRAFFIC_MONTH_1                      1.00                      0.82
VOICE_TRAFFIC_MONTH_2                      0.82                      1.00
VOICE_TRAFFIC_MONTH_3                      0.80                      0.98
VOICE_TRAFFIC_MONTH_4                      0.74                      0.83
VOICE_TRAFFIC_MONTH_5                      0.73                      0.81
VOICE_TRAFFIC_MONTH_6                      0.44                      0.51
BILLING_MONTH_1                            0.16                      0.13
BILLING_MONTH_2                            0.13                      0.11
BILLING_MONTH_3                            0.13                      0.12
BILLING_MONTH_4                            0.13                      0.11
BILLING_MONTH_5                            0.13                      0.11
BILLING_MONTH_6                            0.14                      0.12
DEVICE_COST_MONTH_1                       -0.01                     -0.01
DEVICE_COST_MONTH_2                        0.00                      0.00
DEVICE_COST_MONTH_3                        0.01                     -0.00
DEVICE_COST_MONTH_4                        0.00                      0.00
DEVICE_COST_MONTH_5                        0.01                      0.00
DEVICE_COST_MONTH_6                        0.01                      0.00
DURATION_LINE                              0.02                      0.03
DEVICE_VALUE                               0.02                      0.03

                       VOICE_TRAFFIC_MONTH_3
VOICE_TRAFFIC_MONTH_4  ...  \
DATA_TRAFFIC_MONTH_1                       0.07
0.07  ...
DATA_TRAFFIC_MONTH_2                       0.08
0.07  ...
DATA_TRAFFIC_MONTH_3                       0.08
0.07  ...
DATA_TRAFFIC_MONTH_4                       0.09
0.09  ...
DATA_TRAFFIC_MONTH_5                       0.10
0.09  ...
DATA_TRAFFIC_MONTH_6                       0.08
0.08  ...
VOICE_TRAFFIC_MONTH_1                      0.80
0.74  ...
VOICE_TRAFFIC_MONTH_2                      0.98
0.83  ...
VOICE_TRAFFIC_MONTH_3                      1.00
0.85  ...
VOICE_TRAFFIC_MONTH_4                      0.85
1.00  ...
VOICE_TRAFFIC_MONTH_5                      0.82
0.88  ...
VOICE_TRAFFIC_MONTH_6                      0.51
```

```
0.46  ...
BILLING_MONTH_1                                  0.13
0.12  ...
BILLING_MONTH_2                                  0.11
0.11  ...
BILLING_MONTH_3                                  0.13
0.12  ...
BILLING_MONTH_4                                  0.11
0.12  ...
BILLING_MONTH_5                                  0.11
0.12  ...
BILLING_MONTH_6                                  0.12
0.12  ...
DEVICE_COST_MONTH_1                             -0.01                  -
0.01  ...
DEVICE_COST_MONTH_2                              0.01
0.01  ...
DEVICE_COST_MONTH_3                             -0.00
0.01  ...
DEVICE_COST_MONTH_4                              0.00                  -
0.00  ...
DEVICE_COST_MONTH_5                              0.00                  -
0.00  ...
DEVICE_COST_MONTH_6                              0.01
0.01  ...
DURATION_LINE                                    0.03
0.02  ...
DEVICE_VALUE                                     0.03
0.02  ...

                       BILLING_MONTH_5  BILLING_MONTH_6
DEVICE_COST_MONTH_1  \
DATA_TRAFFIC_MONTH_1              0.19             0.19
0.01
DATA_TRAFFIC_MONTH_2              0.19             0.20
0.01
DATA_TRAFFIC_MONTH_3              0.19             0.20
0.01
DATA_TRAFFIC_MONTH_4              0.21             0.22
0.01
DATA_TRAFFIC_MONTH_5              0.21             0.22
0.01
DATA_TRAFFIC_MONTH_6              0.19             0.20
0.01
VOICE_TRAFFIC_MONTH_1             0.13             0.14
-0.01
VOICE_TRAFFIC_MONTH_2             0.11             0.12
-0.01
VOICE_TRAFFIC_MONTH_3             0.11             0.12
```

```
-0.01
VOICE_TRAFFIC_MONTH_4                  0.12              0.12
-0.01
VOICE_TRAFFIC_MONTH_5                  0.13              0.12
-0.01
VOICE_TRAFFIC_MONTH_6                  0.11              0.10
-0.00
BILLING_MONTH_1                        0.78              0.79
0.01
BILLING_MONTH_2                        0.69              0.70
0.01
BILLING_MONTH_3                        0.72              0.75
0.01
BILLING_MONTH_4                        0.80              0.79
0.02
BILLING_MONTH_5                        1.00              0.83
0.00
BILLING_MONTH_6                        0.83              1.00
0.00
DEVICE_COST_MONTH_1                    0.00              0.00
1.00
DEVICE_COST_MONTH_2                    0.04              0.04
0.38
DEVICE_COST_MONTH_3                    0.06              0.06
0.28
DEVICE_COST_MONTH_4                    0.06              0.06
0.33
DEVICE_COST_MONTH_5                    0.05              0.06
0.32
DEVICE_COST_MONTH_6                    0.07              0.07
0.31
DURATION_LINE                          0.09              0.10
-0.00
DEVICE_VALUE                           0.10              0.10
0.08

                       DEVICE_COST_MONTH_2  DEVICE_COST_MONTH_3  \
DATA_TRAFFIC_MONTH_1                  0.05                 0.08
DATA_TRAFFIC_MONTH_2                  0.05                 0.06
DATA_TRAFFIC_MONTH_3                  0.05                 0.06
DATA_TRAFFIC_MONTH_4                  0.05                 0.07
DATA_TRAFFIC_MONTH_5                  0.03                 0.07
DATA_TRAFFIC_MONTH_6                  0.05                 0.09
VOICE_TRAFFIC_MONTH_1                 0.00                 0.01
VOICE_TRAFFIC_MONTH_2                 0.00                -0.00
VOICE_TRAFFIC_MONTH_3                 0.01                -0.00
VOICE_TRAFFIC_MONTH_4                 0.01                 0.01
VOICE_TRAFFIC_MONTH_5                -0.00                 0.00
VOICE_TRAFFIC_MONTH_6                 0.01                 0.00
```

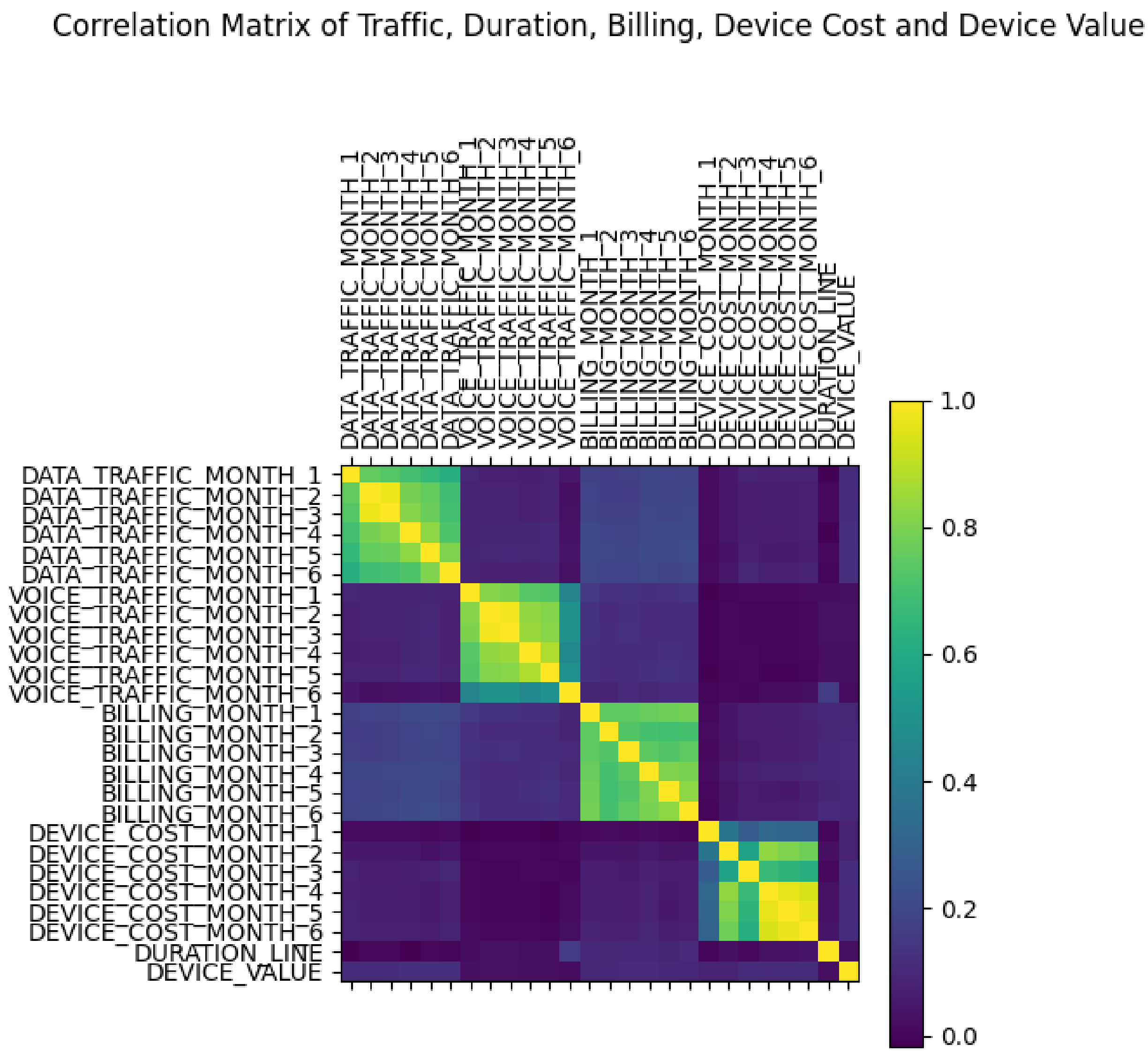
BILLING_MONTH_1	0.05	0.07
BILLING_MONTH_2	0.05	0.06
BILLING_MONTH_3	0.04	0.06
BILLING_MONTH_4	0.06	0.08
BILLING_MONTH_5	0.04	0.06
BILLING_MONTH_6	0.04	0.06
DEVICE_COST_MONTH_1	0.38	0.28
DEVICE_COST_MONTH_2	1.00	0.57
DEVICE_COST_MONTH_3	0.57	1.00
DEVICE_COST_MONTH_4	0.83	0.67
DEVICE_COST_MONTH_5	0.80	0.64
DEVICE_COST_MONTH_6	0.78	0.62
DURATION_LINE	0.03	0.01
DEVICE_VALUE	0.08	0.12

	DEVICE_COST_MONTH_4	DEVICE_COST_MONTH_5 \
DATA_TRAFFIC_MONTH_1	0.08	0.08
DATA_TRAFFIC_MONTH_2	0.07	0.07
DATA_TRAFFIC_MONTH_3	0.07	0.07
DATA_TRAFFIC_MONTH_4	0.07	0.07
DATA_TRAFFIC_MONTH_5	0.06	0.06
DATA_TRAFFIC_MONTH_6	0.07	0.07
VOICE_TRAFFIC_MONTH_1	0.00	0.01
VOICE_TRAFFIC_MONTH_2	0.00	0.00
VOICE_TRAFFIC_MONTH_3	0.00	0.00
VOICE_TRAFFIC_MONTH_4	-0.00	-0.00
VOICE_TRAFFIC_MONTH_5	-0.01	-0.01
VOICE_TRAFFIC_MONTH_6	0.01	0.01
BILLING_MONTH_1	0.07	0.07
BILLING_MONTH_2	0.07	0.07
BILLING_MONTH_3	0.06	0.06
BILLING_MONTH_4	0.08	0.08
BILLING_MONTH_5	0.06	0.05
BILLING_MONTH_6	0.06	0.06
DEVICE_COST_MONTH_1	0.33	0.32
DEVICE_COST_MONTH_2	0.83	0.80
DEVICE_COST_MONTH_3	0.67	0.64
DEVICE_COST_MONTH_4	1.00	0.97
DEVICE_COST_MONTH_5	0.97	1.00
DEVICE_COST_MONTH_6	0.94	0.97
DURATION_LINE	0.03	0.03
DEVICE_VALUE	0.11	0.11

	DEVICE_COST_MONTH_6	DURATION_LINE	DEVICE_VALUE
DATA_TRAFFIC_MONTH_1	0.08	-0.01	0.12
DATA_TRAFFIC_MONTH_2	0.07	0.01	0.11

```
DATA_TRAFFIC_MONTH_3                              0.07             0.01
0.12
DATA_TRAFFIC_MONTH_4                              0.07            -0.02
0.12
DATA_TRAFFIC_MONTH_5                              0.06             0.00
0.13
DATA_TRAFFIC_MONTH_6                              0.08             0.00
0.12
VOICE_TRAFFIC_MONTH_1                             0.01             0.02
0.02
VOICE_TRAFFIC_MONTH_2                             0.00             0.03
0.03
VOICE_TRAFFIC_MONTH_3                             0.01             0.03
0.03
VOICE_TRAFFIC_MONTH_4                             0.01             0.02
0.02
VOICE_TRAFFIC_MONTH_5                             0.00             0.03
0.03
VOICE_TRAFFIC_MONTH_6                             0.02             0.16
0.02
BILLING_MONTH_1                                   0.08             0.10
0.09
BILLING_MONTH_2                                   0.08             0.09
0.10
BILLING_MONTH_3                                   0.07             0.10
0.10
BILLING_MONTH_4                                   0.09             0.10
0.11
BILLING_MONTH_5                                   0.07             0.09
0.10
BILLING_MONTH_6                                   0.07             0.10
0.10
DEVICE_COST_MONTH_1                               0.31            -0.00
0.08
DEVICE_COST_MONTH_2                               0.78             0.03
0.08
DEVICE_COST_MONTH_3                               0.62             0.01
0.12
DEVICE_COST_MONTH_4                               0.94             0.03
0.11
DEVICE_COST_MONTH_5                               0.97             0.03
0.11
DEVICE_COST_MONTH_6                               1.00             0.04
0.11
DURATION_LINE                                     0.04             1.00
0.03
DEVICE_VALUE                                      0.11             0.03
1.00
```

```
[26 rows x 26 columns]

<Figure size 2200x2000 with 0 Axes>
```



2.3. Date management and period calculation

First, we will determine the date of the `LAST_DEVICE_CHANGE` of the last device that was changed in the entire dataset (i.e., the maximum value of the `LAST_DEVICE_CHANGE` column, plus 30 days). We will refer to that date as `latest_change`.

Note that `LAST_DEVICE_CHANGE` is expressed as a floating point number in the format `YYYYMMDD.0`, for instance 3 of July of 2018 would be `20180703.0`. Convert to integer first, then to string.

As a string, this is formatted according to strptime conventions with format `%Y%m%d`.

Use datetime.datetime.strptime to convert to create object `latest_change` and print it.

Next, add 30 days to that date to obtain object `now` (we will assume we are doing this processing 30 days after the latest device change). Use a `datetime.timedelta` object for that.

Your output should look like this:

```
2019-05-01 00:00:00
2019-05-31 00:00:00
```

(Remove this cell when delivering.)

Replace this cell with code to create and print `latest_change` and `now`.

```
import datetime



# Get the maximum value from the LAST_DEVICE_CHANGE column (ignoring
NaN)
max_change = imputation_dataset['LAST_DEVICE_CHANGE'].max()

# Convert the float value (e.g., 20190501.0) to an integer then to a
string
max_change_str = str(int(max_change))

# Use datetime.datetime.strptime to parse the string into a datetime
object
latest_change = datetime.datetime.strptime(max_change_str, "%Y%m%d")

# Display latest_change
print(latest_change)



2019-05-01 00:00:00

#Add 30 days using datetime.timedelta
thirty_days = datetime.timedelta(days=30)
thirty_days_later = latest_change + thirty_days

# Display the result
print(thirty_days_later)

2019-05-31 00:00:00

print(latest_change)
print(thirty_days_later)
```

```
2019-05-01 00:00:00
2019-05-31 00:00:00
```

Now, obtain the series corresponding to the last device change, you can do it by using pandas.to_datetime as if you were using `strptime`:

```
series_converted = pd.to_datetime(dataframe[column_name], format='%Y%m
%d')
```

Now compute the difference between the now and the series_converted.

Divide that difference by `30 * datetime.timedelta(days=1)` to obtain the difference in periods of 30 days (approximately one month).

Replace the `MONTHS_LAST_DEVICE` column with those differences. You may need to fill the NaN with zeroes, and convert to type `int`.

(Remove this cell when delivering.)

Replace this cell with code that replaces the **MONTHS_LAST_DEVICE** column to be equal to the difference, in periods of 30 days, between **LAST_DEVICE_CHANGE** and the `now` variable.

```
import pandas as pd
import datetime

# Define the now variable. You can use a fixed date or the current
datetime.
# Fixed date example:
now = datetime.datetime(2019, 5, 31)
# Alternatively, to use the current date and time:
# now = datetime.datetime.now()

# Convert LAST_DEVICE_CHANGE values to datetime.
converted_dates = pd.to_datetime(

imputation_dataset['LAST_DEVICE_CHANGE'].astype('Int64').astype(str),
    format='%Y%m%d',
    errors='coerce'
)

# Calculate the difference between now and each date, in periods of 30
days.
month_differences = (now - converted_dates) / pd.Timedelta(days=30)

# Replace MONTHS_LAST_DEVICE with the computed differences.
imputation_dataset['MONTHS_LAST_DEVICE'] =
month_differences.fillna(0).astype(int)

# Display the first few values to verify the update.
print(imputation_dataset['MONTHS_LAST_DEVICE'].head())
```

```
0     0
1    26
2     0
3    20
4     5
Name: MONTHS_LAST_DEVICE, dtype: int32
```

Replace this cell with code to update the **DURATION_LINE** value to be the difference, in days, between **LINE_ACTIVATION_DATE** and the `now` variable. Indicate the average of **DURATION_LINE** -- what is that in years, approximately?

```
# Convert LINE_ACTIVATION_DATE to datetime, handling NaN values
activation_dates = pd.to_datetime(

imputation_dataset['LINE_ACTIVATION_DATE'].fillna(0).astype('Int64').a
stype(str).replace('0', pd.NaT),
      format='%Y%m%d',
      errors='coerce'
)

# Calculate difference in days between now and activation date
days_difference = (now - activation_dates).dt.days

# Update DURATION_LINE with the new values
imputation_dataset['DURATION_LINE'] = days_difference

# Calculate and display the average duration in years (excluding NaN
values)
avg_duration_years =
imputation_dataset['DURATION_LINE'].dropna().mean() / 365
print(f"Average duration: {avg_duration_years:.2f} years")

Average duration: 5.34 years
```

2.4. Standarization and scaling of numerical variables

Scaling a series involves changing the values. Standardization involves ensuring that the mean is 0 and the standard deviation is 1, while min-max scaling requires that the maximum is 1, the minimum is 0, and all remaining values are linearly interpolated.

You can use StandardScaler() to standarize a variable, and MinMaxScaler() to perform min-max scaling.

The following example shows how to use these:

```
test_data = [{'x': -1.0}, {'x': 2.0}, {'x': 3.0}, {'x': 6.0}]
test_df = pd.DataFrame(test_data)
display(test_df)
```

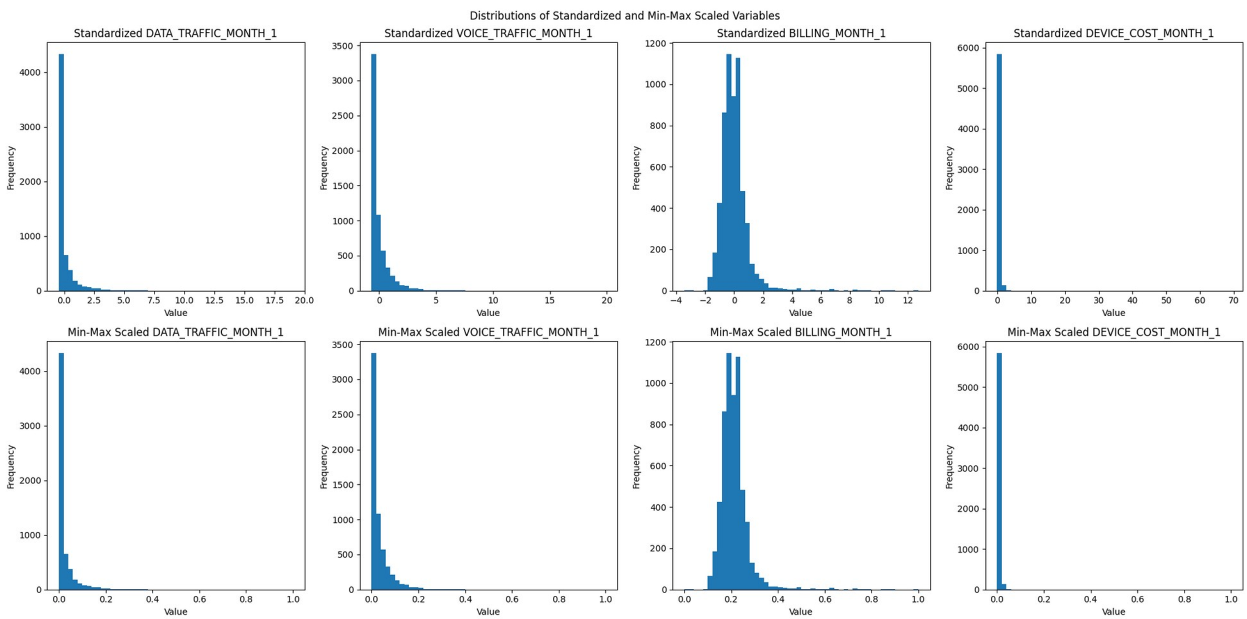
```
test_df['x_standardized'] =
StandardScaler().fit_transform(test_df[['x']])
test_df['x_minmaxscaled'] =
MinMaxScaler().fit_transform(test_df[['x']])
display(test_df)
```

(Remove this cell when delivering.)

Replace this cell with code to standardize and min-max scale the **DATA_TRAFFIC_MONTH_1**, **VOICE_TRAFFIC_MONTH_1**, **BILLING_MONTH_1** and **DEVICE_COST_MONTH_1** columns. Save the results in new colums with the same name followed by **_STANDARD** and **_MINMAX** (e.g., DATA_TRAFFIC_MONTH_1_STAND, DATA_TRAFFIC_MONTH_1_MINMAX). Plot a histogram for each new variable.

```
from sklearn.preprocessing import StandardScaler, MinMaxScaler

# Import required scalers

# Create scalers
std_scaler = StandardScaler()
minmax_scaler = MinMaxScaler()

# List of columns to transform
cols_to_transform = [
  'DATA_TRAFFIC_MONTH_1',
  'VOICE_TRAFFIC_MONTH_1',
  'BILLING_MONTH_1',
  'DEVICE_COST_MONTH_1'
]

# Perform standardization
for col in cols_to_transform:
  new_col = f"{col}_STANDARD"
  imputation_dataset[new_col] =
std_scaler.fit_transform(imputation_dataset[[col]])

# Perform min-max scaling
for col in cols_to_transform:
  new_col = f"{col}_MINMAX"
  imputation_dataset[new_col] =
minmax_scaler.fit_transform(imputation_dataset[[col]])

# Create subplots for histograms
fig, axes = plt.subplots(2, 4, figsize=(20, 10))
fig.suptitle('Distributions of Standardized and Min-Max Scaled
Variables')

# Plot standardized variables
for idx, col in enumerate(cols_to_transform):
  std_col = f"{col}_STANDARD"
```

```
  axes[0, idx].hist(imputation_dataset[std_col], bins=50)
  axes[0, idx].set_title(f'Standardized {col}')
  axes[0, idx].set_xlabel('Value')
  axes[0, idx].set_ylabel('Frequency')

# Plot min-max scaled variables
for idx, col in enumerate(cols_to_transform):
  minmax_col = f"{col}_MINMAX"
  axes[1, idx].hist(imputation_dataset[minmax_col], bins=50)
  axes[1, idx].set_title(f'Min-Max Scaled {col}')
  axes[1, idx].set_xlabel('Value')
  axes[1, idx].set_ylabel('Frequency')

plt.tight_layout()
plt.show()
```



2.5. Convert categorical columns to dummy binary variables

Categorical variables usually need to be transformed into numerical values to apply some machine learning methods.

Use LabelEncoder() to transform a categorical variable to integer values. Example:

```
colors_data = [{'color': 'Blue'}, {'color': 'Red'}, {'color':
'Orange'},
               {'color': 'Blue'}, {'color': 'Orange'}, {'color':
'Blue'}]
colors_df = pd.DataFrame(colors_data, columns=['color'])
```

```
colors_df['colors_int_encoded'] =
LabelEncoder().fit_transform(colors_df['color'])
display(colors_df)
```

(Remove this cell when delivering.)

Create variable **PREVIOUS_DEVICE_BRAND_INT_ENCODED** containing an integer encoding of variable **PREVIOUS_DEVICE_BRAND**.

```
# Create LabelEncoder object
label_encoder = LabelEncoder()

# Create new column with integer encoded brand values
imputation_dataset['PREVIOUS_DEVICE_BRAND_INT_ENCODED'] =
label_encoder.fit_transform(imputation_dataset['PREVIOUS_DEVICE_BRAND'
])

# Display mapping of brands to integers
brand_mapping = dict(zip(label_encoder.classes_,
label_encoder.transform(label_encoder.classes_)))
print("Brand to integer mapping:")
for brand, value in brand_mapping.items():
  print(f"{brand}: {value}")
```

```
Brand to integer mapping:
Apple: 0
LG: 1
Motorola: 2
Outros: 3
Samsung: 4
```

You can use get_dummies() to convert a categorical variable to multiple columns using one-hot encoding. Example:

```
colors_data = [{'color': 'Blue'}, {'color': 'Red'}, {'color':
'Orange'},
               {'color': 'Blue'}, {'color': 'Orange'}, {'color':
'Blue'}]
colors_df = pd.DataFrame(colors_data, columns=['color'])

color_dummies = pd.get_dummies(colors_df['color'], prefix='color_')
colors_df_with_dummies = colors_df.join(color_dummies)
display(colors_df_with_dummies)
```

(Remove this cell when delivering.)

Replace this cell with code to convert **PREVIOUS_DEVICE_MANUF** to dummy binary variables.

```python
# Convert PREVIOUS_DEVICE_MANUF to dummy variables using
pd.get_dummies()
manufacturer_dummies =
pd.get_dummies(imputation_dataset['PREVIOUS_DEVICE_MANUF'],
prefix='manuf')

# Join the dummy variables to the original dataframe
imputation_dataset = imputation_dataset.join(manufacturer_dummies)

# Display the first few rows of the dummy variables
print("Shape of dummy variables dataframe:",
manufacturer_dummies.shape)
print("\nFirst few rows of dummy variables:")
display(manufacturer_dummies.head())
```

```
Shape of dummy variables dataframe: (5988, 67)

First few rows of dummy variables:

   manuf_ASUSTek Computer Inc  manuf_Apple Inc  manuf_BLU Products Inc
\
0                       False            False                   False

1                       False            False                   False

2                       False            False                   False

3                       False             True                   False

4                       False            False                   False


   manuf_Beijing Flyscale Technologies Company Limited  \
0                                              False
1                                              False
2                                              False
3                                              False
4                                              False

   manuf_BlackBerry Limited  manuf_Bullitt Group Limited  \
0                    False                        False
1                    False                        False
2                    False                        False
3                    False                        False
4                    False                        False

   manuf_CT Asia (HK) Ltd  manuf_D-Link Corporation  \
0                   False                     False
1                   False                     False
2                   False                     False
3                   False                     False
```

```
4                   False                    False

   manuf_DG HomTom Group Co Limited  \
0                             False
1                             False
2                             False
3                             False
4                             False

   manuf_DL Comercio e Industria de Produtos Eletronic  ...  \
0                                              False    ...
1                                              False    ...
2                                              False    ...
3                                              False    ...
4                                              False    ...

   manuf_Telit Communications SpA  manuf_Topmax Glory Limited  \
0                           False                       False
1                           False                       False
2                           False                       False
3                           False                       False
4                           False                       False

   manuf_Umi Network Technology Co Limited  manuf_United Mobile  \
0                                    False                False
1                                    False                False
2                                    False                False
3                                    False                False
4                                    False                False

   manuf_United Time Hong Kong Ltd  \
0                            False
1                            False
2                            False
3                            False
4                            False

   manuf_Vikin Communication Technology Co Limited  manuf_Vogtec (H.K)
Co Ltd  \
0                                             False
False
1                                             False
False
2                                             False
False
3                                             False
False
4                                             False
False
```

```
   manuf_Xiaomi Communications Co Ltd  manuf_ZTE Corporation  manuf_u-blox AG
0                               False                  False
False
1                               False                  False
False
2                               False                  False
False
3                               False                  False
False
4                               False                  False
False

[5 rows x 67 columns]
```

2.6. Feature generation

In the current dataset we have a historic of 6 months for data traffic, voice traffic, billing and device cost. Feature generation consists of creating new attributes from the current dataset that can help us to create, e.g., better predictive models.

(Remove this cell when delivering.)

Replace this cell with code to create from the 6 months of **DATA_TRAFFIC_MONTH_[1-6]**, **VOICE_TRAFFIC_MONTH_[1-6]**, **BILLING_MONTH_[1-6]** and **DEVICE_COST_MONTH_[1-6]**, new columns with the mean, maximum, minimum, range (i.e., difference between maximum and minimum) for each element. For instance, column **DATA_TRAFFIC_MEAN** should contain the average of these six numbers: **DATA_TRAFFIC_MONTH_1**, **DATA_TRAFFIC_MONTH_2**, ..., **DATA_TRAFFIC_MONTH_6**.

```python
# List of base column names
base_columns = ['DATA_TRAFFIC', 'VOICE_TRAFFIC', 'BILLING', 'DEVICE_COST']

# For each base column
for base in base_columns:
  # Get the 6 monthly columns
  columns = [f"{base}_MONTH_{i}" for i in range(1, 7)]

  # Calculate statistics across the 6 months
  imputation_dataset[f"{base}_MEAN"] = imputation_dataset[columns].mean(axis=1)
  imputation_dataset[f"{base}_MAX"] = imputation_dataset[columns].max(axis=1)
  imputation_dataset[f"{base}_MIN"] = imputation_dataset[columns].min(axis=1)
  imputation_dataset[f"{base}_RANGE"] = imputation_dataset[f"{base}_MAX"] - imputation_dataset[f"{base}_MIN"]
```

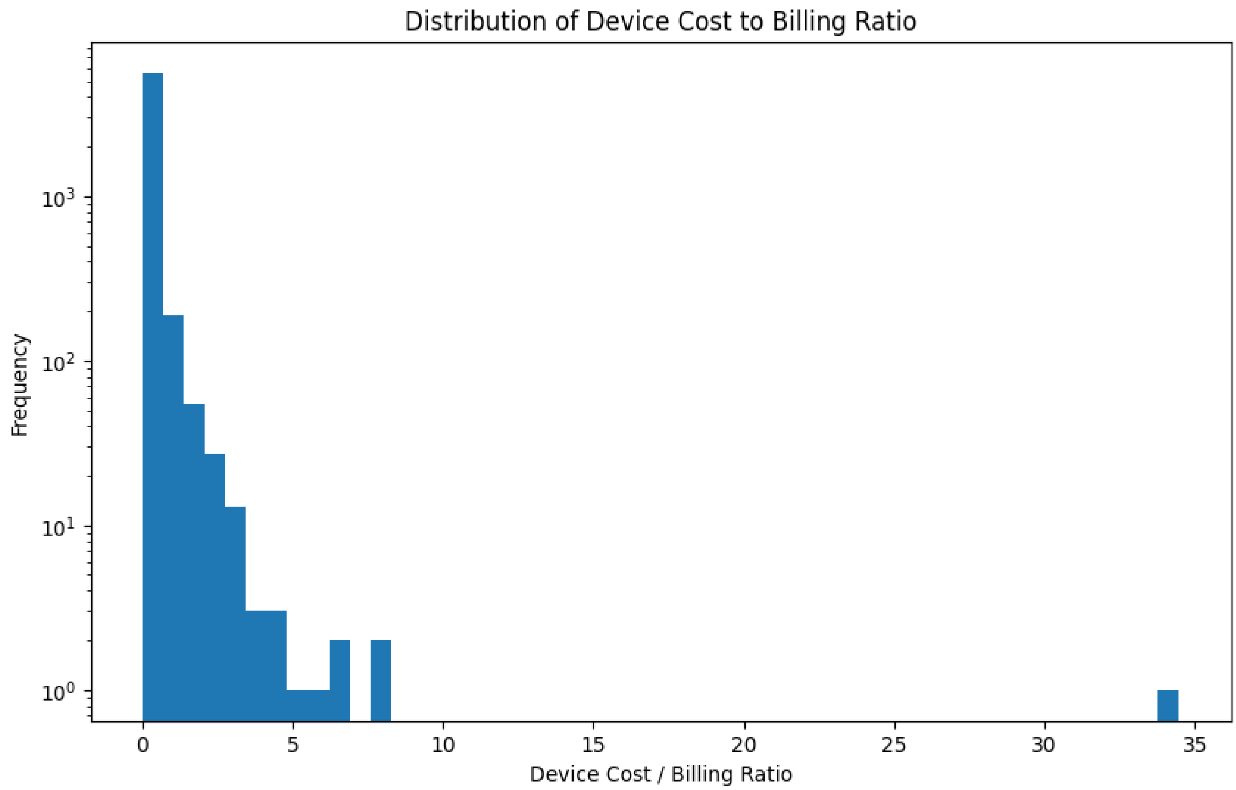
```python
# Display the new columns
new_columns = [col for col in imputation_dataset.columns if any(x in
col for x in ['_MEAN', '_MAX', '_MIN', '_RANGE'])]
display(imputation_dataset[new_columns].head())
```

```
   DATA_TRAFFIC_MONTH_1_MINMAX  VOICE_TRAFFIC_MONTH_1_MINMAX  \
0                         0.00                          0.01
1                         0.00                          0.00
2                         0.00                          0.01
3                         0.04                          0.04
4                         0.04                          0.09

   BILLING_MONTH_1_MINMAX  DEVICE_COST_MONTH_1_MINMAX
DATA_TRAFFIC_MEAN  \
0                    0.20                        0.00
646.06
1                    0.16                        0.00
376.58
2                    0.23                        0.00
332.10
3                    0.18                        0.00
1178.09
4                    0.22                        0.00
2729.06

   DATA_TRAFFIC_MAX  DATA_TRAFFIC_MIN  DATA_TRAFFIC_RANGE
VOICE_TRAFFIC_MEAN  \
0           1169.40            398.99              770.40
40.72
1            704.89            232.24              472.64
3.07
2            484.62            250.74              233.88
114.10
3           4255.46            146.77             4108.69
185.30
4           5014.10           1553.12             3460.99
63.98

   VOICE_TRAFFIC_MAX  VOICE_TRAFFIC_MIN  VOICE_TRAFFIC_RANGE
BILLING_MEAN  \
0              79.70              21.80                57.90
92.96
1               4.90               0.50                 4.40
49.44
2             218.70              26.10               192.60
121.78
3             231.20             119.00               112.20
58.22
4             383.90               0.00               383.90
109.70
```

```
   BILLING_MAX  BILLING_MIN  BILLING_RANGE  DEVICE_COST_MEAN  
DEVICE_COST_MAX  \
0       107.93        85.00          22.93             12.00  
12.00   
1        56.56        47.00           9.56              0.00  
0.00   
2       129.14       113.77          15.37              0.00  
0.00   
3        60.93        55.99           4.94              6.00  
6.00   
4       110.69       107.99           2.70              0.00  
0.00   

   DEVICE_COST_MIN  DEVICE_COST_RANGE  
0            12.00               0.00  
1             0.00               0.00  
2             0.00               0.00  
3             6.00               0.00  
4             0.00               0.00  
```

Replace this cell with code create an additional column **DEVICE_COST_TO_BILLING_RATIO** containing the ratio between **DEVICE_COST_MEAN** and **BILLING_MEAN** and plot its distribution.

```
# Calculate ratio between DEVICE_COST_MEAN and BILLING_MEAN
imputation_dataset['DEVICE_COST_TO_BILLING_RATIO'] = 
imputation_dataset['DEVICE_COST_MEAN'] / 
imputation_dataset['BILLING_MEAN']

# Plot the distribution of the ratio
plt.figure(figsize=(10, 6))
plt.hist(imputation_dataset['DEVICE_COST_TO_BILLING_RATIO'], bins=50)
plt.title('Distribution of Device Cost to Billing Ratio')
plt.xlabel('Device Cost / Billing Ratio')
plt.ylabel('Frequency')

# Use log scale on y-axis to better visualize the distribution
plt.yscale('log')
plt.show()
```



Replace this cell with a brief commentary on the distribution of the variable **DEVICE_COST_TO_BILLING_RATIO**. Can you recognize its distribution?

```
# The distribution of DEVICE_COST_TO_BILLING_RATIO shows a highly
right-skewed (positively skewed) distribution, with the following key
characteristics:

# 1. Most ratios are concentrated near zero, indicating that device
costs are typically much lower than billing amounts for the majority
of customers

# 2. There is a long right tail extending to higher ratio values,
representing cases where device costs are comparable to or exceed
billing amounts

# 3. The use of a logarithmic scale on the y-axis helps visualize the
full range of frequencies, revealing that the distribution
approximately follows a log-normal pattern

# 4. There appear to be some outliers with very high ratios, likely
representing special cases where device costs were unusually high
relative to billing
```

2.7. Text parsing/processing

In machine learning, text processing is a very useful tool that can be used to improve datasets. In some use cases, for instance customer care applications using digital channels as Whatsapp, Facebook, etc..., data scientist teams mainly work with text data.

One of the text processing technique is to extract concrete words or tokens from a sentence or documents. Regular expressions are a great tool to extract data trough these patterns.

In this dataset, note that **PURCHASED_DEVICE** is a variable that is formed by a "**device_code**"+"**_"+"manufacture name"+" "+"device model**". We want to split this variable into its components.

Tip: use str.split to separate a string into several parts.

(Remove this cell when delivering.)

Replace this cell with code to use the **PURCHASED_DEVICE** variable to create 3 new columns with the following variables names: **PURCHASED_DEVICE_CODE**, **PURCHASED_DEVICE_MANUFACTURER** and **PURCHASED_DEVICE_MODEL**.

Replace this cell with code to create two tables: one with the number of devices per manufacturer in **PURCHASED_DEVICE_MANUFACTURER** and one with the number of devices per manufacturer in **PREVIOUS_DEVICE_MANUF**.

2.8. Splitting and sampling a dataset

Splitting and sampling dataset are techniques that distribute the original dataset in n-parts. One of the most interesting application of these tools is to separate the dataset to train and test a machine learning model. Meanwhile sampling guarantees same type of data (i.e. distributions), splitting will separate the dataset with the ratio we need. Usually, 80%-20% or 70%-30% splitting ratios are the most common used.

Once again, Sklearn library helps to us to cover this necessity through the function sklearn.model_selection.train_test_split which splits a dataset into two parts, which usually will be used for training and testing.

(Remove this cell when delivering.)

Replace this cell with code to split the dataset in two separate datasets: one with 70% of the rows and the other with 30% of rows

Replace this cell with code to compute the main statistics (mean, standard deviation, min, max, 25%, 50%, 75%) for the variables **DATA_TRAFFIC_MONTH_1**, **VOICE_TRAFFIC_MONTH_1** and **BILLING_MONTH_1** in both training and testing parts of the dataset.

Replace this cell with a brief commentary indicating if you find these statistics match between the two splits, or do not match between them.

3. Comparing iPhone and Samsung J series users

Finally, find some features that are different between users of an Apple iPhone and users of a Samsung J series phone (this includes J410G, J610G, J415G, and all other models by Samsung that start with a *J*).

(Remove this cell when delivering.)

Replace this cell with code to create two dataframes: one with all the attributes of Apple iPhone users and one with all the attributes of Samsung J series users.

Replace this cell with code to compare some variables between the two datasets. Consider 2 or 3 variables, plot together the histograms of each variable in both datasets (including a legend).

Replace this cell with a brief commentary on the differences you found between these two groups of users.

DELIVER (individually)

Remember to read the section on "delivering your code" in the course evaluation guidelines.

Deliver a zip file containing:

- This notebook

Extra points available

For more learning and extra points, remember what you learned in machine learning and create a simple decision tree model having as input variables:

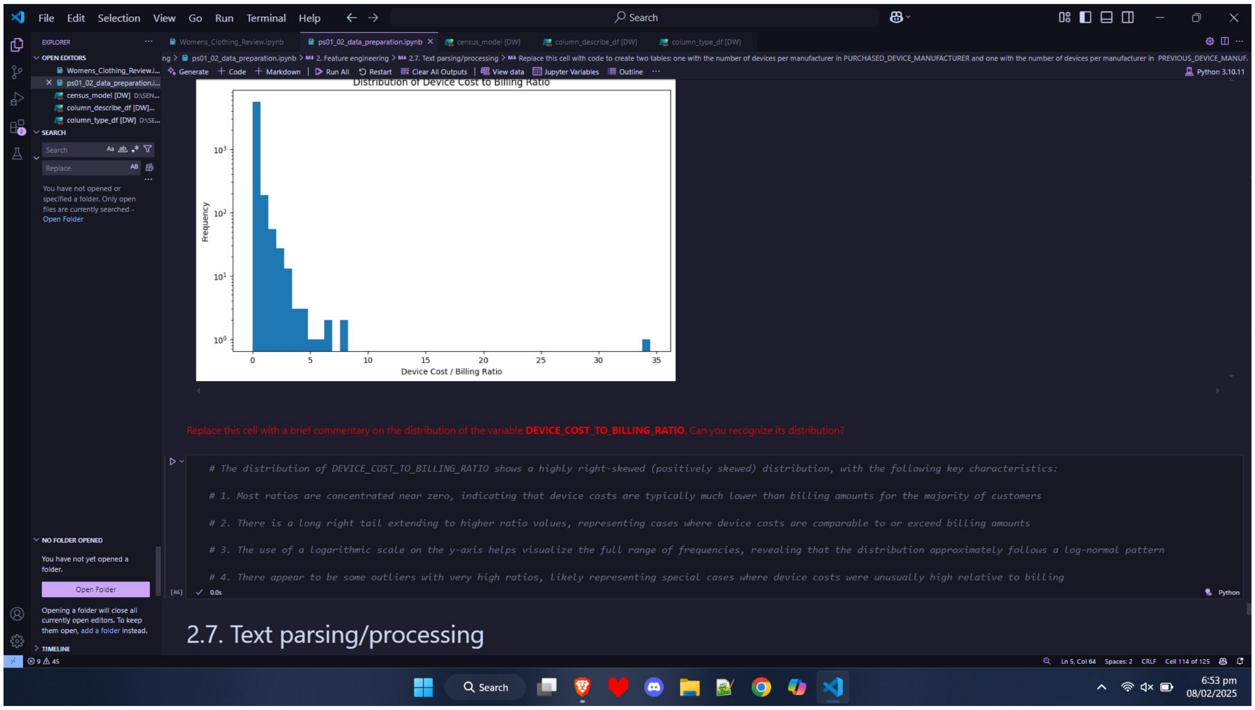
1. PREVIOUS_DEVICE_MODEL
2. PREVIOUS_DEVICE_BRAND
3. MONTHS_LAST_DEVICE

And as output variable `PURCHASED_DEVICE_MANUFACTURER`. Measure the accuracy of this 3-variables model. Then, add two more variables, of your own choice, that improve the classification accuracy. Measure the accuracy of this 5-variables model.

Note: if you go for the extra points, add `<font size="+2" color="blue">Additional results: model purchased device</font>` at the top of your notebook.

(Remove this cell when delivering.)

I hereby declare that, except for the code provided by the course instructors, all of my code, report, and figures were produced by myself.


Distribution of Device Cost to Billing Ratio
Frequency
Device Cost / Billing Ratio
Replace this cell with a brief commentary on the distribution of the variable DEVICE_COST_TO_BILLING_RATIO. Can you recognize its distribution?
The distribution of DEVICE_COST_TO_BILLING_RATIO shows a highly right-skewed (positively skewed) distribution, with the following key characteristics:
1. Most ratios are concentrated near zero, indicating that device costs are typically much lower than billing amounts for the majority of customers
2. There is a long right tail extending to higher ratio values, representing cases where device costs are comparable to or exceed billing amounts
3. The use of a logarithmic scale on the y-axis helps visualize the full range of frequencies, revealing that the distribution approximately follows a log-normal pattern
4. There appear to be some outliers with very high ratios, likely representing special cases where device costs were unusually high relative to billing
2.7. Text parsing/processing