merge

November 3, 2024

[85]:
```
# -*- coding: utf-8 -*-
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
# http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
# implied.
# See the License for the specific language governing permissions and
# limitations under the License.
#
```

1 Always run all imported notebooks when you make a change in some files you imported.

1.0.1 Adam Candrák/Mária Matušisková - 50%/50%

2 Imports

[86]:
```
import import_ipynb
import connections
# import devices
# import normalize
import processes
# import profiles

import pandas as pd
import numpy as np
import seaborn as sns
from sklearn.feature_selection import VarianceThreshold
import matplotlib.pyplot as plt
from numpy import mean
from numpy import std
```

```python
from collections import Counter
from sklearn.model_selection import train_test_split
```

Just for test purposes to check if the import of jupyter notebooks was successful.

[87]:
```python
connections.test()
# devices.test()
processes.test()
# profiles.test()
```

```
Success
Success
```

3 Phase 1 - Exploratory analysis

3.1 2.1 Data Preparation

Change the names of columns of connections dataset:

[88]:
```python
c_connections = connections.connections.rename(columns={
    "c.katana": "facebook",
    "c.android.chrome": "chrome",
    "c.android.gm": "gmail",
    "c.dogalize": "dogalize",
    "c.android.youtube": "youtube",
    "c.updateassist": "updateassist",
    "c.UCMobile.intl": "UCMobile.intl",
    "c.raider": "raider",
    "c.android.vending": "vending",
    "c.UCMobile.x86": "UCMobile.x86",
})
```

Change the names of columns of processes dataset:

[89]:
```python
p_processes = processes.processes.rename(columns={
    "p.katana": "facebook",
    "p.android.chrome": "chrome",
    "p.android.gm": "gmail",
    "p.dogalize": "dogalize",
    "p.android.vending": "vending",
    "p.android.packageinstaller": "packageinstaller",
    "p.system": "system",
    "p.android.documentsui": "documentsui",
    "p.android.settings": "settings",
    "p.android.externalstorage": "externalstorage",
    "p.android.defcontainer": "defcontainer",
    "p.inputmethod.latin": "inputmethod.latin",
```

```
    "p.process.gapps": "gapps",
    "p.simulator": "simulator",
    "p.android.gms": "google mobile services (gms)",
    "p.google": "google",
    "p.olauncher": "olauncher",
    "p.browser.provider": "browser provider",
    "p.notifier": "notifier",
    "p.gms.persistent": "gms.persistent",
})
```

Change type of timestamp to int64 of connections dataset:

[90]:
```
c_connections['ts'] = pd.to_datetime(c_connections['ts']).astype(np.int64)
```

Change type of timestamp to int64 of processes dataset:

[91]:
```
p_processes['ts'] = pd.to_datetime(p_processes['ts']).astype(np.int64)
```

Merge datasets connections and processes:

[92]:
```
new_dataset = pd.merge(c_connections, p_processes, on=['imei', 'ts', 'mwra'])
new_dataset.head()
```

[92]:
```
                    ts                 imei   mwra  facebook_x  chrome_x  \
0  1525514400000000000  3590433799317662188   True    10.99774  11.05477
1  1525514460000000000  3590433799317662394   True    11.08234   9.64636
2  1525514520000000000  3590433799317661834  False    11.49582  12.27416
3  1525514580000000000  8630330696303481289  False    10.50935  11.41774
4  1525514640000000000  8630330696303481149  False    10.25989  14.46448

    gmail_x  dogalize_x   youtube  updateassist  UCMobile.intl  …  \
0   6.03999    12.49767   8.59956      14.00953       52.54470  …
1   8.64167    12.60788   9.84197      38.27736       44.56009  …
2  11.59681    12.99258   9.74923      57.41411       36.83333  …
3  14.43350    12.91018  13.93857      31.57549       41.34296  …
4  14.02728     8.58832  13.04853      49.47100       38.86755  …

   dogalize_y     gapps  simulator  facebook_y  google mobile services (gms)  \
0    95.23250  99.55387   82.64951    55.62534                      43.73958
1    73.67809  55.93619   27.33158    68.28812                      67.18486
2    49.43847  92.96630   54.04233    25.01599                      57.15110
3    71.37356   8.34277   87.09809     5.21806                      98.58641
4    14.58892  27.72954   81.20459    22.42807                      25.06680

     google  olauncher  browser provider  notifier  gms.persistent
0  28.79282    8.22474          73.26391  25.28004        86.66346
1  19.40350   19.26265          58.69464  90.54099        33.10194
2  60.38043   16.88231          55.62452  16.82005        81.58652
3  97.22889   37.30215          68.75315  26.44336        79.98101
```

```
4  73.26831   43.72205          78.80356  16.55350        75.03307

[5 rows x 33 columns]
```

[93]:

```
new_dataset
```

[93]:

```
                         ts                 imei   mwra  facebook_x  chrome_x  \
0      1525514400000000000  3590433799317662188   True    10.99774  11.05477
1      1525514460000000000  3590433799317662394   True    11.08234   9.64636
2      1525514520000000000  3590433799317661834  False    11.49582  12.27416
3      1525514580000000000  8630330696303481289  False    10.50935  11.41774
4      1525514640000000000  8630330696303481149  False    10.25989  14.46448
…                        …                    …    …          …         …
13808  1526408820000000000   863033069630348123   True    13.31200  15.71180
13809  1526408940000000000  3590433799317661594   True    13.52429  13.82647
13810  1526409000000000000  8630330696303481057  False    10.90390   8.77998
13811  1526409060000000000  3590433799317662188   True     8.47958   8.11147
13812  1526409120000000000  3590433799317662204  False     8.96624   7.91019

        gmail_x  dogalize_x   youtube  updateassist  UCMobile.intl  …  \
0       6.03999    12.49767   8.59956      14.00953       52.54470  …
1       8.64167    12.60788   9.84197      38.27736       44.56009  …
2      11.59681    12.99258   9.74923      57.41411       36.83333  …
3      14.43350    12.91018  13.93857      31.57549       41.34296  …
4      14.02728     8.58832  13.04853      49.47100       38.86755  …
…             …           …         …             …            …  …
13808  13.40696    10.90627  15.05067      52.86111       49.79239  …
13809  13.60043    11.46229  17.82659      27.86611       25.41132  …
13810  15.71295    14.06695  14.61081      36.67428       47.42102  …
13811  15.36153     4.71766  12.32035      55.91226       48.17744  …
13812  11.08831    12.05655  12.06833      63.50117       45.87526  …

       dogalize_y     gapps  simulator  facebook_y  \
0        95.23250  99.55387   82.64951    55.62534
1        73.67809  55.93619   27.33158    68.28812
2        49.43847  92.96630   54.04233    25.01599
3        71.37356   8.34277   87.09809     5.21806
4        14.58892  27.72954   81.20459    22.42807
…               …         …          …           …
13808     5.97424  10.90052   64.04619    80.00079
13809     7.86586  66.80404    9.30436    52.73032
13810    72.41636  99.40362    3.27873    90.70094
13811    45.63483  44.30156   83.10716     2.99819
13812    64.41884  36.37121   62.14144    80.34765

       google mobile services (gms)    google  olauncher  browser provider  \
0                          43.73958  28.79282    8.22474          73.26391
```

```
1                        67.18486  19.40350   19.26265           58.69464
2                        57.15110  60.38043   16.88231           55.62452
3                        98.58641  97.22889   37.30215           68.75315
4                        25.06680  73.26831   43.72205           78.80356
…                             …       …         …              …
13808                    78.87558  33.74537   60.31124           89.11008
13809                     9.57537  37.50921   36.65184           97.34896
13810                    51.97068  10.63735   23.57597           81.81312
13811                     1.73479  44.51363   66.19054           76.48819
13812                    67.26413  80.26685   49.46053           58.20929

       notifier  gms.persistent
0      25.28004        86.66346
1      90.54099        33.10194
2      16.82005        81.58652
3      26.44336        79.98101
4      16.55350        75.03307
…           …             …
13808  46.41420        10.20768
13809  38.01707         6.64068
13810  77.31911        61.81489
13811  27.12469         5.60833
13812  19.84458         0.36212

[13813 rows x 33 columns]
```

Data cleaning: Find negative values in the merged dataset:

[94]:
```python
negative_values = new_dataset.select_dtypes(include=[np.number]) < 0

# any for columns and all values in the series of the first any
has_negatives = negative_values.any().any()

if has_negatives:
    print("The dataset has negative values.")
    print(negative_values.any())
else:
    print("No negative values found in the dataset.")
```

```
No negative values found in the dataset.
```

Find NaN values in the merged dataset:

[95]:
```python
has_nan = new_dataset.isnull().values.any()

if has_nan:
    print("The dataset has NaN values.")
    print(new_dataset.isnull().values)
```

```
else:
    print("No NaN values found in the dataset.")
```

```
No NaN values found in the dataset.
```

Find duplicity values in the merged dataset:

[96]:
```
has_duplicity = new_dataset.duplicated().any()

if has_duplicity:
    print("The dataset has duplicity values.")
    print(new_dataset[new_dataset.duplicated()])
    print("Number of duplicate rows:", new_dataset.duplicated().sum())
else:
    print("No duplicity values found in the dataset.")
```

```
No duplicity values found in the dataset.
```

Drop values which are not helpful for further training:

[97]:
```
new_dataset.drop('ts', axis=1, inplace=True)
new_dataset.drop('imei', axis=1, inplace=True)
```

2.1 B - Data integration

Standard Deviation

- detect outliers by standard deviation which spreads data around the mean
- **3x standard deviations (σ) from the mean (μ)**

[98]:
```
new_dataset.shape
```

[98]:
```
(13813, 31)
```

[99]:
```
# Source: https://www.kaggle.com/code/marcinrutecki/outlier-detection-methods

def StandardDevDetection(data, n, columns):

    outliers_inx = []
    lower = 0
    upper = 0

    for column in columns:
        # Calculate mean and standard derivation of each column
        data_mean, data_std = mean(data[column], axis=0), std(data[column],␣
↪axis=0)
        print('column=', column, 'len=', len(data), 'mean=', data_mean, 'std=',␣
↪data_std)
```

```
        # Divide it to the three outliers in the standard deviations:
        cut_off = data_std * 3
        lower, upper = data_mean - cut_off, data_mean + cut_off
        print('column=', column, 'cutoff=', cut_off, 'lower=', lower, 'upper=',␣
↪upper)

        # Filter the dataframe:
        outliers = data[(data[column] < lower) | (data[column] > upper)].index
        print('Identified outliers:', len(outliers))

        outliers_inx.extend(outliers)

    outliers_inx = Counter(outliers_inx)
    multiple_outliers = list( k for k, v in outliers_inx.items() if v > n )

    data_uppper = data[data[column] > upper]
    data_lower = data[data[column] < lower]
    print('Total number of outliers is:', data_uppper.shape[0] + data_lower.
↪shape[0])

    return multiple_outliers


columns = new_dataset.columns
result = StandardDevDetection(new_dataset, 1, columns)

new_dataset = new_dataset.drop(result, axis = 0).reset_index(drop=True)
```

```
column= mwra len= 13813 mean= 0.6411351625280532 std= 0.4796674534489284
column= mwra cutoff= 1.4390023603467852 lower= -0.7978671978187319 upper=
2.080137522874838
Identified outliers: 0
column= facebook_x len= 13813 mean= 10.960814784623182 std= 2.6473283821315765
column= facebook_x cutoff= 7.94198514639473 lower= 3.0188296382284516 upper=
18.90279993101791
Identified outliers: 2
column= chrome_x len= 13813 mean= 11.539096789256499 std= 2.5207614078254665
column= chrome_x cutoff= 7.5622842234764 lower= 3.976812565780099 upper=
19.1013810127329
Identified outliers: 2
column= gmail_x len= 13813 mean= 12.271741594150438 std= 2.5156002578552674
column= gmail_x cutoff= 7.546800773565803 lower= 4.724940820584635 upper=
19.81854236771624
Identified outliers: 12
column= dogalize_x len= 13813 mean= 10.471127016578587 std= 2.192478595551923
column= dogalize_x cutoff= 6.5774357866557684 lower= 3.893691229922818 upper=
17.048562803234354
```

```
Identified outliers: 20
column= youtube len= 13813 mean= 12.245109354955476 std= 2.5525053045765995
column= youtube cutoff= 7.657515913729799 lower= 4.587593441225677 upper=
19.902625268685274
Identified outliers: 0
column= updateassist len= 13813 mean= 45.93389651994497 std= 12.340017367100165
column= updateassist cutoff= 37.02005210130049 lower= 8.91384441864448 upper=
82.95394862124547
Identified outliers: 5
column= UCMobile.intl len= 13813 mean= 45.81056430319264 std= 12.9849235624614
column= UCMobile.intl cutoff= 38.9547706873842 lower= 6.8557936158084445 upper=
84.76533499057683
Identified outliers: 2
column= raider len= 13813 mean= 49.15474816911605 std= 13.21554230600327
column= raider cutoff= 39.64662691800981 lower= 9.508121251106239 upper=
88.80137508712585
Identified outliers: 5
column= vending_x len= 13813 mean= 49.65063183233186 std= 28.908058893940023
column= vending_x cutoff= 86.72417668182007 lower= -37.07354484948821 upper=
136.37480851415194
Identified outliers: 0
column= UCMobile.x86 len= 13813 mean= 49.779651266198506 std= 28.69411063791361
column= UCMobile.x86 cutoff= 86.08233191374083 lower= -36.302680647542324 upper=
135.86198317993933
Identified outliers: 0
column= packageinstaller len= 13813 mean= 10.993080264243828 std=
2.7314789128261205
column= packageinstaller cutoff= 8.194436738478363 lower= 2.798643525765465
upper= 19.18751700272219
Identified outliers: 0
column= system len= 13813 mean= 11.044334718743212 std= 2.4614338845881534
column= system cutoff= 7.38430165376446 lower= 3.6600330649787516 upper=
18.428636372507672
Identified outliers: 0
column= documentsui len= 13813 mean= 11.042905808296531 std= 2.5169293750495365
column= documentsui cutoff= 7.5507881251486095 lower= 3.4921176831479217 upper=
18.59369393344514
Identified outliers: 0
column= chrome_y len= 13813 mean= 12.192827297473395 std= 2.514829111137564
column= chrome_y cutoff= 7.544487333412691 lower= 4.648339964060703 upper=
19.737314630886086
Identified outliers: 19
column= settings len= 13813 mean= 13.450420093390285 std= 1.7820897501132393
column= settings cutoff= 5.346269250339718 lower= 8.104150843050567 upper=
18.79668934373
Identified outliers: 0
column= gmail_y len= 13813 mean= 12.93598093317889 std= 2.239795948433877
column= gmail_y cutoff= 6.719387845301632 lower= 6.216593087877258 upper=
```

```
19.655368778480522
Identified outliers: 0
column= externalstorage len= 13813 mean= 11.525600619706076 std=
2.5292366512900877
column= externalstorage cutoff= 7.587709953870263 lower= 3.9378906658358126
upper= 19.11331057357634
Identified outliers: 2
column= defcontainer len= 13813 mean= 50.609099905885756 std= 12.465074989936033
column= defcontainer cutoff= 37.3952249698081 lower= 13.213874936077659 upper=
88.00432487569385
Identified outliers: 0
column= vending_y len= 13813 mean= 0.09152040975892277 std= 0.45525791954870326
column= vending_y cutoff= 1.3657737586461098 lower= -1.274253348887187 upper=
1.4572941684050327
Identified outliers: 163
column= inputmethod.latin len= 13813 mean= 50.80920694056324 std=
12.650371121354508
column= inputmethod.latin cutoff= 37.951113364063524 lower= 12.85809357649972
upper= 88.76032030462676
Identified outliers: 0
column= dogalize_y len= 13813 mean= 49.480386747267076 std= 28.96253566924302
column= dogalize_y cutoff= 86.88760700772906 lower= -37.40722026046198 upper=
136.36799375499612
Identified outliers: 0
column= gapps len= 13813 mean= 50.07511253674075 std= 28.852708669670328
column= gapps cutoff= 86.55812600901098 lower= -36.48301347227023 upper=
136.63323854575174
Identified outliers: 0
column= simulator len= 13813 mean= 49.65368691956852 std= 29.02291729915727
column= simulator cutoff= 87.06875189747181 lower= -37.41506497790329 upper=
136.72243881704034
Identified outliers: 0
column= facebook_y len= 13813 mean= 49.97372888583219 std= 28.93853834780385
column= facebook_y cutoff= 86.81561504341155 lower= -36.84188615757937 upper=
136.78934392924373
Identified outliers: 0
column= google mobile services (gms) len= 13813 mean= 50.24779950553826 std=
28.831330512743175
column= google mobile services (gms) cutoff= 86.49399153822952 lower=
-36.24619203269126 upper= 136.7417910437678
Identified outliers: 0
column= google len= 13813 mean= 50.275553065952366 std= 28.794858536663533
column= google cutoff= 86.3845756099906 lower= -36.10902254403824 upper=
136.66012867594299
Identified outliers: 0
column= olauncher len= 13813 mean= 49.81847215232028 std= 29.026895400503307
column= olauncher cutoff= 87.08068620150992 lower= -37.26221404918964 upper=
136.89915835383022
```

```
Identified outliers: 0
column= browser provider len= 13813 mean= 49.872044705712014 std=
28.898834439361433
column= browser provider cutoff= 86.6965033180843 lower= -36.82445861237228
upper= 136.56854802379632
Identified outliers: 0
column= notifier len= 13813 mean= 49.6245631224209 std= 29.066771646099497
column= notifier cutoff= 87.20031493829849 lower= -37.57575181587759 upper=
136.82487806071939
Identified outliers: 0
column= gms.persistent len= 13813 mean= 49.86738886556144 std=
28.846309617754013
column= gms.persistent cutoff= 86.53892885326204 lower= -36.6715399877006 upper=
136.40631771882346
Identified outliers: 0
Total number of outliers is: 0
```

Divide it to the three outliers in the standard deviations:

[100]:
```
new_dataset.shape
```

[100]:
```
(13812, 31)
```

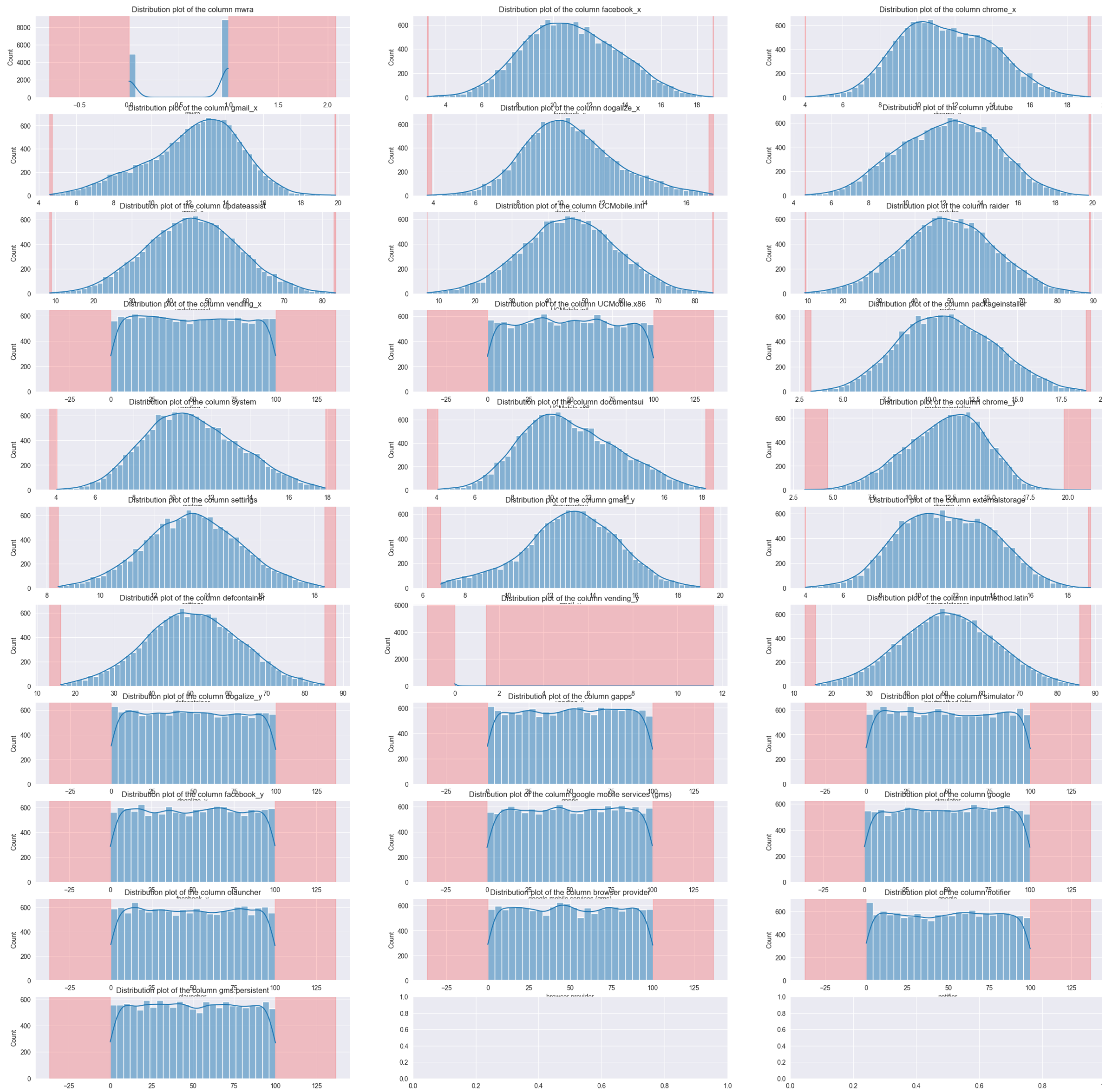
Show data distribution after cut of outlines:

[101]:
```
columns = new_dataset.columns

fig, axes = plt.subplots(nrows=(len(columns)//2) - 4, ncols=3, figsize=(30, 30))

axes = axes.flatten()

for i, col in enumerate(columns):
    data_mean, data_std = mean(new_dataset[col], axis=0), std(new_dataset[col],␣
 ↪axis=0)
    cut_off = data_std * 3
    lower, upper = data_mean - cut_off, data_mean + cut_off
    sns.histplot(new_dataset[col], kde=True, ax=axes[i])
    axes[i].axvspan(xmin = lower,xmax= new_dataset[col].min(),alpha=0.2,␣
 ↪color='red')
    axes[i].axvspan(xmin = upper,xmax= new_dataset[col].max(),alpha=0.2,␣
 ↪color='red')
    axes[i].set_title(f'Distribution plot of the column {col}')
```



2.1 A - Split dataset on Train and Test data

[102]:
```
new_dataset.columns
```

[102]:
```
Index(['mwra', 'facebook_x', 'chrome_x', 'gmail_x', 'dogalize_x', 'youtube',
       'updateassist', 'UCMobile.intl', 'raider', 'vending_x', 'UCMobile.x86',
       'packageinstaller', 'system', 'documentsui', 'chrome_y', 'settings',
       'gmail_y', 'externalstorage', 'defcontainer', 'vending_y',
       'inputmethod.latin', 'dogalize_y', 'gapps', 'simulator', 'facebook_y',
       'google mobile services (gms)', 'google', 'olauncher',
       'browser provider', 'notifier', 'gms.persistent'],
      dtype='object')
```

[103]:
```python
target_column = 'mwra'
mwra = new_dataset[target_column]
data = new_dataset.drop(columns=[target_column], axis=1)
```

[104]:
```python
train_data, test_data, train_mwra, test_mwra = train_test_split(data, mwra,␣
    ↪test_size=0.3, random_state=42)
```

2.1 B Data transformation

One-hot Encoding Binary Columns

[105]:
```python
# Source: IAU week-05

import category_encoders as ce

# create object of BinaryEncoder
ce_binary = ce.BinaryEncoder(cols = ['mwra'])

# fit and transform and you will get the encoded data
ce_binary.fit_transform(train_mwra)
```

[105]:
```
       mwra_0  mwra_1
652         0       1
7157        0       1
6889        1       0
3473        0       1
7855        1       0
…         …       …
5191        0       1
13418       0       1
5390        0       1
860         1       0
7270        1       0

[9668 rows x 2 columns]
```

2.1 C - Scaling

2.1 C - Data normalization

3.2 $x_{normalization} = \frac{x - x_{min}}{x_{max} - x_{min}}$

It is good to use, when in the data is a wide range of values. Which can lead to poor performance of models. Which is our case is true. We compared the data and the range was wild.

[ ]:
```python
# Source: IAU week-05
```

```
from numpy import asarray
from sklearn.preprocessing import MinMaxScaler

# define min max scaler
scaler = MinMaxScaler()

# transform data
train_data = scaler.fit_transform(train_data)
print(train_data)
```

```
[[0.46532684 0.65435847 0.55473366 … 0.53938962 0.49548575 0.62505231]
 [0.60720181 0.69866849 0.52712396 … 0.12748244 0.88609364 0.78160951]
 [0.73288585 0.42310005 0.69813909 … 0.38497812 0.05838268 0.10333648]
 …
 [0.64515966 0.26155698 0.67604713 … 0.78394989 0.04566846 0.05384304]
 [0.82642756 0.7422867  0.78437034 … 0.02611836 0.04168749 0.18057928]
 [0.13805188 0.41466245 0.67943943 … 0.20407831 0.31411581 0.07042059]]
```

The result is in the numpy python format, which is the best format for training because is fast and intuitive compare to normal array.

2.1 C - Data standardization

3.3 $x_{standardized} = \frac{x-\mu}{\sigma}$

where - μ is the mean of x - σ is the standard deviation of x

It is a z-normalization, it measures variations of values about its mean in the dataset. (in short it measures the range of values)

[ ]:
```
# Source: IAU week-05

from numpy import asarray
from sklearn.preprocessing import StandardScaler

# define standard scaler
scaler = StandardScaler()

# transform data
train_data = scaler.fit_transform(train_data)
print(train_data)
```

2.1 C - Make data distribution more Gaussian :3

Power transformer on random data -> Yeo-Johnson Transform with Linear Regression

- to have more relatively similar to normally distributed

This text is from the week-05: - Replacing the data with the log, square root, or inverse to remove skew - Yeo-Johnson transform (default): works with positive and negative values - Box-Cox transform: only works with strictly positive values - $= -1.0$ is a reciprocal transform. - $= -0.5$ is a reciprocal square root transform.
- $= 0.0$ is a log transform. - $= 0.5$ is a square root transform. - $= 1.0$ is no transform.

[ ]:
```python
# Source: https://www.kaggle.com/code/abhikuks/power-transformers-in-depth-understanding

from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PowerTransformer
from matplotlib import pyplot

plt.figure(figsize=(30, 6))

pt = PowerTransformer()
train_data = pt.fit_transform(train_data)
pyplot.hist(train_data, bins=10)

lr = LinearRegression()
lr.fit(train_data, train_mwra)

# let's make some basic prediction on mwra
predictions = lr.predict(train_data)
print(predictions)

# show trained transformed data in the histogram graph of normalized data
plt.hist(train_data, bins=10, alpha=0.7, label='Transformed Data')
plt.legend()
plt.xlabel('Transformed Feature Value')
plt.ylabel('Frequency')
plt.title('Transformed Training Data')
pyplot.hist(train_data, bins=10)

# show prediction
plt.figure(figsize=(10, 6))
plt.scatter(train_mwra, predictions, alpha=0.6)
plt.plot([train_mwra.min(), train_mwra.max()], [train_mwra.min(), train_mwra.max()], 'k--', lw=2)
plt.xlabel('Actual Values')
plt.ylabel('Predicted Values')
plt.title('Actual vs. Predicted Values')
```

2.1 C - Data discretization

- to reduce the effects of minor observation errors
- minimalize the influence of outliers

Equal-width dicretization

- We chose this discretization as an example, because the data is still wide range between values and thus this could be a solution to cut range into the smaller intervals. It is called equally sized intervals. We might not use this method in further phases tho- we are unsure of how well the model will behave.

[ ]:
```
num_bins = 8

columns = columns.difference(['mwra'])
train_data = pd.DataFrame(train_data, columns=columns)
train_data_binned = pd.DataFrame()

for column in train_data.select_dtypes(include=['float64', 'int64']).columns:
    train_data_binned[f'{column}_binned'] = pd.cut(train_data[column],␣
 ↪bins=num_bins, labels=False)

print(train_data_binned.head())

fig, axes = plt.subplots(nrows=(len(columns)//2) - 5, ncols=3,figsize=(30, 50))
axes = axes.flatten()
i = 0
for col in train_data_binned:
    sns.histplot(train_data_binned[col], kde=True, ax=axes[i], color='blue',␣
 ↪edgecolor='black')
    axes[i].set_title(f'Equal-Width Binned Data for {col}')
    i+= 1
plt.xlabel('Bins')
plt.ylabel('Frequency')
```

2.1 D - Reasons for implementation

Some things are already indicated in the text. However, we decided to merge processes and connections into one dataset since they have common columns and values. They were merged on ts, imei and mwra, these columns they had absolutely identical. In the dataset, there are not anymore ts and imei columns, because they were not adequate for testing purposes and we considered them as unnecessary.

Furthermore, we renamed the columns to better names for reading and better orientation during work. For example, the katana represents Facebook. Why not change it to right away?

After merging for sure, we checked if the data are clean enough for further work.

In our opinion, it is better to cut outliers before splitting the dataset. We also checked the presentation, and it was recommended to do it this way. After that, we split the dataset to train and test into two groups one is for all data and the second is for predicted value mwra. This way it is possible to predict how the mwra will behave.

In conclusion, we prepared data for machine learning by data cleaning (missing values, outliers detection), integration (3x standard deviation, encoding) and transformation (scaling, transformers, discretization). Which ways we picked it are already defined in the specific sections.

2.2 Attribute selection

3.3.1 2.2 A - Comparing feature selection methods

3.3.2 Filter Method

As the first filter attribute selection method, we decided against using variance threshold method, as it did nothing. Instead we will use Mutual Information.

Mutual Information (MI) is a measure of the mutual dependence between two variables. It quantifies how much information knowing one variable reduces the uncertainty about the other. In feature selection, mutual information tells us how informative a feature is about the target variable, regardless of whether the relationship is linear or non-linear.

From Lab study material file- week-05: The concept of MI is linked to information theory and **information entropy** ($\mathcal{H}$). The unit of information depends on the base of the logarithm. If the base is 2, the most used, then the information is measured in *bits*. **MI is non-negative and symmetric.**

$\mathcal{H}(X) = -\int dx\ \mu(x)\ log\mu(x)$

$I(X,Y) = -\int\int dx\ dy\ \mu(x,y)\ log\frac{\mu(x,y)}{\mu_x(x)\ \mu_y(y)}$ For discrete variables, $H(X)$ is calculated as: #### $H(X) = -\sum_i P(x_i) log P(x_i)$

MI can be equivalently expressed as the amount of uncertainty in X minus the amount of uncertainty in X after Y is known, denoted as

$I(X;Y) = H(X) - H(X|Y)$ The entropy of X after observing values of Y is formulated by #### $H(X|Y) = -\sum_j P(y_j) \sum_i P(x_i|y_j) \log_2 P(x_i|y_j)$

where - $P(x_i)$ is the prior probabilities for all values of X and - $P(x_i|y_i)$ is the posterior probabilities of X given the values of Y.

(Adam note: Ngl plne nechapem tymto matematickym hieroglyfom)

[ ]:
```
from sklearn.feature_selection import mutual_info_classif
num_of_features = 10

selector = mutual_info_classif(train_data, train_mwra)
scores = pd.Series(selector, index=train_data.columns).
  ↪sort_values(ascending=False)
mi_selected_features = scores[:num_of_features].index

print("The top " + num_of_features.__str__() + " features according to F-value␣
  ↪method are: ")
```

```
print(mi_selected_features)
```

As the second filter method, we decided to use **F-value** method since our data is continuous.

F-value source: Lab study material file- week-05

The correlation between each regressor and the target is computed by

3.3.3 $F_{value} = \frac{variance_{dataset_1}}{variance_{dataset_2}} = \frac{\sigma_1^2}{\sigma_2^2}$

It is converted to an F score then to a p-value.

[ ]:
```
from sklearn.feature_selection import SelectKBest, f_classif
num_of_features = 10

selector = SelectKBest(score_func=f_classif, k=num_of_features)
selector.fit(train_data, train_mwra)

# get scores and p-values
f_scores = pd.Series(selector.scores_, index=train_data.columns)
f_scores_sorted = f_scores.sort_values(ascending=False)
f_selected_features = f_scores_sorted.index[:num_of_features]

print(f"\nThe top {num_of_features} features according to F-value method are:")
print(f_selected_features)
```

3.3.4 Wrapper

These methods evaluate subsets of features by actually training a model, making them more accurate for specific algorithms.

For showcasing wrapper method we chose RFE (Recursive feature elimination)

RFE is a wrapper method that uses the built-in importance of the estimator attributes (in this case **RandomForest**) to generate the ratings. RFE is able to account for interactions between attributes, but is computationally more demanding than filter methods (note: look at the execution time of this cell lol).

[83]:
```
from sklearn.feature_selection import SelectKBest, RFE
from sklearn.ensemble import RandomForestClassifier
num_of_features = 10
model = RandomForestClassifier()

#rfe
rfe = RFE(estimator=model, n_features_to_select=num_of_features)
tmp = rfe.fit_transform(train_data, train_mwra)

rfe_selected_features = rfe.get_support(indices=True)

rfe_selected_features = train_data.columns[rfe_selected_features]
```

```
final_rf = RandomForestClassifier()
final_rf.fit(train_data[rfe_selected_features], train_mwra)

# get and sort features by importance
rf_importance = pd.Series(final_rf.feature_importances_,␣
 ↪index=rfe_selected_features)
rfe_selected_features = rf_importance.sort_values(ascending=False).index

print(f"\nThe top {num_of_features} features according to RFE method are:")
print(rfe_selected_features)
```

```
The top 10 features according to RFE method are:
Index(['gmail_x', 'gapps', 'facebook_x', 'chrome_x', 'browser provider',
       'google', 'chrome_y', 'gms.persistent', 'UCMobile.intl',
       'UCMobile.x86'],
      dtype='object')
```

So to recap everything…

[84]:
```
num_of_features = 10

print("Columns selected by every method: ")
mi_set = set(mi_selected_features)
f_set = set(f_selected_features)
rfe_set = set(rfe_selected_features)

common_features = mi_set.intersection(f_set, rfe_set)
print("\nFeatures selected by all three methods:")
for feature in common_features:
    print(f"- {feature}")

comparison_df = pd.DataFrame({
    'Feature': list(set(list(mi_selected_features) + list(f_selected_features)␣
 ↪+ list(rfe_selected_features))),
    'Mutual Information': False,
    'F-Value': False,
    'RFE': False
})

comparison_df.loc[comparison_df['Feature'].isin(mi_selected_features), 'Mutual␣
 ↪Information'] = True
comparison_df.loc[comparison_df['Feature'].isin(f_selected_features),␣
 ↪'F-Value'] = True
comparison_df.loc[comparison_df['Feature'].isin(rfe_selected_features), 'RFE']␣
 ↪= True
```

```python
################
# help with stylization by Claude.ai
################

def color_boolean(val):
    if isinstance(val, bool):
        color = '#90EE90' if val else '#FFB6C1'  # Light green if True, light␣
↪red if False
        return f'background-color: {color}'
    return ''

# Apply styling to the DataFrame
styled_df = (comparison_df.style
             .map(color_boolean, subset=['Mutual Information', 'F-Value',␣
↪'RFE'])
             .set_properties(**{
    'text-align': 'center',
    'border': '1px solid gray',
    'padding': '8px'
})
             .set_properties(subset=['Feature'], **{
    'text-align': 'left',
    'font-weight': 'bold',
    'background-color': '#F0F8FF'  # Light blue background for feature names
})
             .set_table_styles([
    {'selector': 'th',
     'props': [('background-color', '#4682B4'),  # Steel blue headers
               ('color', 'white'),
               ('font-weight', 'bold'),
               ('text-align', 'center'),
               ('padding', '8px'),
               ('border', '1px solid gray')]},
    {'selector': 'caption',
     'props': [('caption-side', 'top'),
               ('font-size', '16px'),
               ('font-weight', 'bold'),
               ('color', '#2F4F4F'),  # Dark slate gray
               ('padding', '8px')]}
])
             .set_caption('Feature Selection Methods Comparison'))

# Display the styled DataFrame
display(styled_df)

# sets of features unique to each method
mi_unique = mi_set - (f_set | rfe_set)
```

```
f_unique = f_set - (mi_set | rfe_set)
rfe_unique = rfe_set - (mi_set | f_set)

print("\nFeatures unique to each method:")
print("Mutual Information only:", mi_unique if mi_unique else "None")
print("F-Value only:", f_unique if f_unique else "None")
print("RFE only:", rfe_unique if rfe_unique else "None")

################
# source: Claude.ia
################

def create_ranking_comparison(mi_features, f_features, rfe_features):
    # Create a dictionary to store rankings
    rankings = {}

    # Add rankings for each method (1 being highest importance)
    for i, feature in enumerate(mi_features):
        if feature not in rankings:
            rankings[feature] = {'MI_rank': i + 1}
        else:
            rankings[feature]['MI_rank'] = i + 1

    for i, feature in enumerate(f_features):
        if feature not in rankings:
            rankings[feature] = {'F_rank': i + 1}
        else:
            rankings[feature]['F_rank'] = i + 1

    for i, feature in enumerate(rfe_features):
        if feature not in rankings:
            rankings[feature] = {'RFE_rank': i + 1}
        else:
            rankings[feature]['RFE_rank'] = i + 1

    # Convert to DataFrame
    ranking_df = pd.DataFrame.from_dict(rankings, orient='index')

    # Fill NaN with 0 (features not selected by a method)
    ranking_df = ranking_df.fillna(0)

    # Calculate average rank (excluding zeros)
    ranking_df['Avg_Rank'] = ranking_df.replace(0, np.nan).mean(axis=1)

    # Count methods that selected each feature
    ranking_df['Methods_Count'] = (ranking_df[['MI_rank', 'F_rank', 
 ↪'RFE_rank']] != 0).sum(axis=1)
```

```
    # Sort by Methods_Count (descending) and then by Avg_Rank (ascending)
    ranking_df = ranking_df.sort_values(['Methods_Count', 'Avg_Rank'],␣
↪ascending=[False, True])

    return ranking_df

# Create the ranking comparison
ranking_comparison = create_ranking_comparison(
    mi_selected_features,
    f_selected_features,
    rfe_selected_features
)

ranking_comparison
```

```
Columns selected by every method:

Features selected by all three methods:
- UCMobile.intl
- browser provider
- gmail_x
- chrome_x
- gapps
- gms.persistent
- chrome_y
- facebook_x

<pandas.io.formats.style.Styler at 0x7fbec1f12e80>


Features unique to each method:
Mutual Information only: None
F-Value only: None
RFE only: {'google', 'UCMobile.x86'}
```

[84]:

	MI_rank	F_rank	RFE_rank	Avg_Rank	Methods_Count
gmail_x	1.0	1.0	1.0	1.000000	3
gapps	2.0	2.0	2.0	2.000000	3
facebook_x	3.0	3.0	3.0	3.000000	3
chrome_x	4.0	4.0	4.0	4.000000	3
browser provider	7.0	5.0	5.0	5.666667	3
chrome_y	5.0	7.0	7.0	6.333333	3
UCMobile.intl	6.0	8.0	9.0	7.666667	3
gms.persistent	9.0	10.0	8.0	9.000000	3
inputmethod.latin	8.0	6.0	0.0	7.000000	2
facebook_y	10.0	9.0	0.0	9.500000	2
google	0.0	0.0	6.0	6.000000	1

```
UCMobile.x86             0.0      0.0       10.0  10.000000                1
```

From our observations, we can see that Mutual Information and F-value selected the same features. There is also a big overlap with RFE that selected eight of the features that the other two methods selected. While MI and F-value selected facebook.y and inputmethod.latin RFE selected UCMobile.x86 and google- clearly showing that these features have more complex relationships with mwra.

Furthermore, the one feature consistently considered as the most important is gmail_x followed by gapps, facebook_x and chrome_x.