dataset2_subject1

January 16, 2025

1 CYBERML - Project

This notebook contains our work for the CYBERML project. We choose the subject **Anomaly detection for tracking attacks** on the **SWaT dataset**.

Our group is composed of :

- Clovis Lechien
- Alexandre Devaux-Rivière
- Florian Segard-Gahery
- Valentin San
- Maël Reynaud

1.0.1 Imports

[1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import mlsecu.data_exploration_utils as deu

from typing import Any
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.ensemble import IsolationForest
from sklearn.neighbors import LocalOutlierFactor

from sklearn.metrics import precision_score, recall_score, f1_score,␣
↪matthews_corrcoef, balanced_accuracy_score, roc_auc_score,␣
↪classification_report, confusion_matrix
from sklearn.model_selection import train_test_split
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

pd.set_option("display.max_columns", None)

%matplotlib inline
```

[2]:
```python
class bcolors:
    HEADER = '\033[95m'
    OKBLUE = '\033[94m'
    OKCYAN = '\033[96m'
    OKGREEN = '\033[92m'
    WARNING = '\033[93m'
    FAIL = '\033[91m'
    ENDC = '\033[0m'
    BOLD = '\033[1m'
    UNDERLINE = '\033[4m'
```

1.0.2 Data exploration / pre processing

Create dataframe

[3]:
```python
path = 'data/swat_newdataset/SWaT.A3_dataset_Jul 19_labelled.xlsx'

df = pd.read_excel(path, skiprows=[0, 2])
df.head()
```

[3]:
```
                     GMT +0  Attack  Label  FIT 101   LIT 101  MV 101  \
0          2019-07-20T04:30:00Z  benign      0      0.0  729.8658       1
1          2019-07-20T04:30:01Z  benign      0      0.0  729.4340       1
2   2019-07-20T04:30:02.004013Z  benign      0      0.0  729.1200       1
3   2019-07-20T04:30:03.004013Z  benign      0      0.0  728.6882       1
4          2019-07-20T04:30:04Z  benign      0      0.0  727.7069       1

   P1_STATE  P101 Status  P102 Status     AIT 201   AIT 202     AIT 203  \
0         3            2            1  142.527557  9.293002  198.077423
1         3            2            1  142.527557  9.293002  198.385025
2         3            2            1  142.527557  9.293002  198.436300
3         3            2            1  142.527557  9.289157  198.667000
4         3            2            1  142.527557  9.289157  198.897720

    FIT 201                                            LS 201  \
0  2.335437  {u'IsSystem': False, u'Name': u'Inactive', u'V…
1  2.335437  {u'IsSystem': False, u'Name': u'Inactive', u'V…
2  2.335437  {u'IsSystem': False, u'Name': u'Inactive', u'V…
3  2.335437  {u'IsSystem': False, u'Name': u'Inactive', u'V…
4  2.335437  {u'IsSystem': False, u'Name': u'Inactive', u'V…

                                            LS 202  \
0  {u'IsSystem': False, u'Name': u'Inactive', u'V…
1  {u'IsSystem': False, u'Name': u'Inactive', u'V…
2  {u'IsSystem': False, u'Name': u'Inactive', u'V…
3  {u'IsSystem': False, u'Name': u'Inactive', u'V…
```

```
4  {u'IsSystem': False, u'Name': u'Inactive', u'V…

                                          LSL 203  \
0  {u'IsSystem': False, u'Name': u'Inactive', u'V…
1  {u'IsSystem': False, u'Name': u'Inactive', u'V…
2  {u'IsSystem': False, u'Name': u'Inactive', u'V…
3  {u'IsSystem': False, u'Name': u'Inactive', u'V…
4  {u'IsSystem': False, u'Name': u'Inactive', u'V…

                                         LSLL 203  MV201  P2_STATE  \
0  {u'IsSystem': False, u'Name': u'Inactive', u'V…      2         2
1  {u'IsSystem': False, u'Name': u'Inactive', u'V…      2         2
2  {u'IsSystem': False, u'Name': u'Inactive', u'V…      2         2
3  {u'IsSystem': False, u'Name': u'Inactive', u'V…      2         2
4  {u'IsSystem': False, u'Name': u'Inactive', u'V…      2         2

   P201 Status  P202 Status  P203 Status  P204 Status  P205 Status  \
0            1            1            2            1            2
1            1            1            2            1            2
2            1            1            2            1            2
3            1            1            2            1            2
4            1            1            2            1            2

   P206 Status  P207 Status  P208 Status   AIT 301     AIT 302     AIT 303  \
0            1            1            1  8.522921  256.431274  143.158966
1            1            1            1  8.522921  256.431274  143.158966
2            1            1            1  8.522921  256.431274  143.158966
3            1            1            1  8.522921  256.431274  143.158966
4            1            1            1  8.522921  256.431274  143.158966

   DPIT 301   FIT 301     LIT 301  MV 301  MV 302  MV 303  MV 304  P3_STATE  \
0  1.190857  0.000512  730.702100       1       1       1       1        99
1  1.190857  0.000512  730.902344       1       1       1       1        99
2  1.190857  0.000512  732.344300       1       1       1       1        99
3  1.190857  0.000512  732.704800       1       1       1       1        99
4  1.190857  0.000512  732.744800       1       1       1       1        99

   P301 Status  P302 Status  AIT 401    AIT 402   FIT 401     LIT 401  \
0            1            1        0  87.951805  0.781740  1000.62805
1            1            1        0  87.823630  0.782380  1000.55115
2            1            1        0  87.798004  0.783021  1000.28200
3            1            1        0  87.695465  0.783021  1000.74341
4            1            1        0  87.618560  0.781228  1000.39734

                                           LS 401  P4_STATE  P401 Status  \
0  {u'IsSystem': False, u'Name': u'Inactive', u'V…         4            2
1  {u'IsSystem': False, u'Name': u'Inactive', u'V…         4            2
```

```
2  {u'IsSystem': False, u'Name': u'Inactive', u'V…        4          2
3  {u'IsSystem': False, u'Name': u'Inactive', u'V…        4          2
4  {u'IsSystem': False, u'Name': u'Inactive', u'V…        4          2

   P402 Status  P403 Status  P404 Status  UV401   AIT 501     AIT 502  \
0            1            1            1      2  7.489618  147.398100
1            1            1            1      2  7.489618  147.398100
2            1            1            1      2  7.489618  147.398100
3            1            1            1      2  7.489618  147.167389
4            1            1            1      2  7.489618  147.090485

      AIT 503    AIT 504   FIT 501   FIT 502   FIT 503   FIT 504  MV 501  \
0  1016.27789  46.065113  0.781594  0.310362  0.623628  0.213432       2
1  1016.27789  45.757500  0.782235  0.315102  0.623628  0.212984       2
2  1016.27789  45.603690  0.782235  0.317023  0.623628  0.212984       2
3  1016.27789  45.603690  0.783133  0.308057  0.623628  0.212792       2
4  1016.27789  45.219173  0.783773  0.303446  0.623628  0.214009       2

   MV 502  MV 503  MV 504  P5_STATE  P501 Status  P502 Status     PIT 501  \
0       2       1       1        12            2            1  167.601257
1       2       1       1        12            2            1  167.601257
2       2       1       1        12            2            1  167.601257
3       2       1       1        12            2            1  167.601257
4       2       1       1        12            2            1  167.601257

    PIT 502     PIT 503  FIT 601  \
0  2.963509  119.921173  0.00032
1  2.963509  119.921173  0.00032
2  2.963509  119.921173  0.00032
3  2.963509  119.921173  0.00032
4  2.963509  119.921173  0.00032

                                            LSH 601  \
0  {u'IsSystem': False, u'Name': u'Active', u'Val…
1  {u'IsSystem': False, u'Name': u'Active', u'Val…
2  {u'IsSystem': False, u'Name': u'Active', u'Val…
3  {u'IsSystem': False, u'Name': u'Active', u'Val…
4  {u'IsSystem': False, u'Name': u'Active', u'Val…

                                            LSH 602  \
0  {u'IsSystem': False, u'Name': u'Active', u'Val…
1  {u'IsSystem': False, u'Name': u'Active', u'Val…
2  {u'IsSystem': False, u'Name': u'Active', u'Val…
3  {u'IsSystem': False, u'Name': u'Active', u'Val…
4  {u'IsSystem': False, u'Name': u'Active', u'Val…

                                            LSH 603  \
```

```
0  {u'IsSystem': False, u'Name': u'Inactive', u'V…
1  {u'IsSystem': False, u'Name': u'Inactive', u'V…
2  {u'IsSystem': False, u'Name': u'Inactive', u'V…
3  {u'IsSystem': False, u'Name': u'Inactive', u'V…
4  {u'IsSystem': False, u'Name': u'Inactive', u'V…

                                            LSL 601  \
0  {u'IsSystem': False, u'Name': u'Inactive', u'V…
1  {u'IsSystem': False, u'Name': u'Inactive', u'V…
2  {u'IsSystem': False, u'Name': u'Inactive', u'V…
3  {u'IsSystem': False, u'Name': u'Inactive', u'V…
4  {u'IsSystem': False, u'Name': u'Inactive', u'V…

                                            LSL 602  \
0  {u'IsSystem': False, u'Name': u'Inactive', u'V…
1  {u'IsSystem': False, u'Name': u'Inactive', u'V…
2  {u'IsSystem': False, u'Name': u'Inactive', u'V…
3  {u'IsSystem': False, u'Name': u'Inactive', u'V…
4  {u'IsSystem': False, u'Name': u'Inactive', u'V…

                                            LSL 603  P6 STATE  P601 Status  \
0  {u'IsSystem': False, u'Name': u'Active', u'Val…        2            1
1  {u'IsSystem': False, u'Name': u'Active', u'Val…        2            1
2  {u'IsSystem': False, u'Name': u'Active', u'Val…        2            1
3  {u'IsSystem': False, u'Name': u'Active', u'Val…        2            1
4  {u'IsSystem': False, u'Name': u'Active', u'Val…        2            1

   P602 Status  P603 Status
0            1            1
1            1            1
2            1            1
3            1            1
4            1            1
```

[4]:
```
df.columns
```

[4]:
```
Index(['GMT +0', 'Attack', 'Label', 'FIT 101', 'LIT 101', 'MV 101', 'P1_STATE',
       'P101 Status', 'P102 Status', 'AIT 201', 'AIT 202', 'AIT 203',
       'FIT 201', 'LS 201', 'LS 202', 'LSL 203', 'LSLL 203', 'MV201',
       'P2_STATE', 'P201 Status', 'P202 Status', 'P203 Status', 'P204 Status',
       'P205 Status', 'P206 Status', 'P207 Status', 'P208 Status', 'AIT 301',
       'AIT 302', 'AIT 303', 'DPIT 301', 'FIT 301', 'LIT 301', 'MV 301',
       'MV 302', 'MV 303', 'MV 304', 'P3_STATE', 'P301 Status', 'P302 Status',
       'AIT 401', 'AIT 402', 'FIT 401', 'LIT 401', 'LS 401', 'P4_STATE',
       'P401 Status', 'P402 Status', 'P403 Status', 'P404 Status', 'UV401',
       'AIT 501', 'AIT 502', 'AIT 503', 'AIT 504', 'FIT 501', 'FIT 502',
       'FIT 503', 'FIT 504', 'MV 501', 'MV 502', 'MV 503', 'MV 504',
```

```
       'P5_STATE', 'P501 Status', 'P502 Status', 'PIT 501', 'PIT 502',
       'PIT 503', 'FIT 601', 'LSH 601', 'LSH 602', 'LSH 603', 'LSL 601',
       'LSL 602', 'LSL 603', 'P6 STATE', 'P601 Status', 'P602 Status',
       'P603 Status'],
      dtype='object')
```

[5]:
```python
df['Attack'].unique()
```

[5]:
```
array(['benign', 'Spoofing', 'Switch_ON', 'Switch_close', 'Switch_off'],
      dtype=object)
```

The function **map_df_num** allow us to convert the columns which contains json like strings (LS201, LS202, LS203...) to int (0 or 1), because the only value changing in these dictionnaries is *Active* or *Inactive*.

[6]:
```python
def map_df_num(df: pd.DataFrame) -> pd.DataFrame:
    new_df = df.map(lambda x: 0 if isinstance(x, str) and "inactive" in x.
 ↪lower()
                 else 1 if isinstance(x, str) and "active" in x.lower()
                 else x).copy()

    return new_df

df = map_df_num(df)
```

[7]:
```python
df.head()
```

[7]:
```
                   GMT +0  Attack  Label  FIT 101   LIT 101  MV 101  \
0        2019-07-20T04:30:00Z  benign      0      0.0  729.8658       1
1        2019-07-20T04:30:01Z  benign      0      0.0  729.4340       1
2  2019-07-20T04:30:02.004013Z  benign      0      0.0  729.1200       1
3  2019-07-20T04:30:03.004013Z  benign      0      0.0  728.6882       1
4        2019-07-20T04:30:04Z  benign      0      0.0  727.7069       1

   P1_STATE  P101 Status  P102 Status     AIT 201   AIT 202     AIT 203  \
0         3            2            1  142.527557  9.293002  198.077423
1         3            2            1  142.527557  9.293002  198.385025
2         3            2            1  142.527557  9.293002  198.436300
3         3            2            1  142.527557  9.289157  198.667000
4         3            2            1  142.527557  9.289157  198.897720

    FIT 201  LS 201  LS 202  LSL 203  LSLL 203  MV201  P2_STATE  P201 Status  \
0  2.335437       0       0        0         0      2         2            1
1  2.335437       0       0        0         0      2         2            1
2  2.335437       0       0        0         0      2         2            1
3  2.335437       0       0        0         0      2         2            1
4  2.335437       0       0        0         0      2         2            1
```

	P202 Status	P203 Status	P204 Status	P205 Status	P206 Status	\
0	1	2	1	2	1	
1	1	2	1	2	1	
2	1	2	1	2	1	
3	1	2	1	2	1	
4	1	2	1	2	1	

	P207 Status	P208 Status	AIT 301	AIT 302	AIT 303	DPIT 301	\
0	1	1	8.522921	256.431274	143.158966	1.190857	
1	1	1	8.522921	256.431274	143.158966	1.190857	
2	1	1	8.522921	256.431274	143.158966	1.190857	
3	1	1	8.522921	256.431274	143.158966	1.190857	
4	1	1	8.522921	256.431274	143.158966	1.190857	

	FIT 301	LIT 301	MV 301	MV 302	MV 303	MV 304	P3_STATE	\
0	0.000512	730.702100	1	1	1	1	99	
1	0.000512	730.902344	1	1	1	1	99	
2	0.000512	732.344300	1	1	1	1	99	
3	0.000512	732.704800	1	1	1	1	99	
4	0.000512	732.744800	1	1	1	1	99	

	P301 Status	P302 Status	AIT 401	AIT 402	FIT 401	LIT 401	LS 401	\
0	1	1	0	87.951805	0.781740	1000.62805	0	
1	1	1	0	87.823630	0.782380	1000.55115	0	
2	1	1	0	87.798004	0.783021	1000.28200	0	
3	1	1	0	87.695465	0.783021	1000.74341	0	
4	1	1	0	87.618560	0.781228	1000.39734	0	

	P4_STATE	P401 Status	P402 Status	P403 Status	P404 Status	UV401	\
0	4	2	1	1	1	2	
1	4	2	1	1	1	2	
2	4	2	1	1	1	2	
3	4	2	1	1	1	2	
4	4	2	1	1	1	2	

	AIT 501	AIT 502	AIT 503	AIT 504	FIT 501	FIT 502	FIT 503	\
0	7.489618	147.398100	1016.27789	46.065113	0.781594	0.310362	0.623628	
1	7.489618	147.398100	1016.27789	45.757500	0.782235	0.315102	0.623628	
2	7.489618	147.398100	1016.27789	45.603690	0.782235	0.317023	0.623628	
3	7.489618	147.167389	1016.27789	45.603690	0.783133	0.308057	0.623628	
4	7.489618	147.090485	1016.27789	45.219173	0.783773	0.303446	0.623628	

	FIT 504	MV 501	MV 502	MV 503	MV 504	P5_STATE	P501 Status	\
0	0.213432	2	2	1	1	12	2	
1	0.212984	2	2	1	1	12	2	
2	0.212984	2	2	1	1	12	2	

```
3  0.212792       2       2       1       1        12            2
4  0.214009       2       2       1       1        12            2

   P502 Status     PIT 501    PIT 502     PIT 503  FIT 601  LSH 601  LSH 602  \
0            1  167.601257  2.963509  119.921173  0.00032        1        1
1            1  167.601257  2.963509  119.921173  0.00032        1        1
2            1  167.601257  2.963509  119.921173  0.00032        1        1
3            1  167.601257  2.963509  119.921173  0.00032        1        1
4            1  167.601257  2.963509  119.921173  0.00032        1        1

   LSH 603  LSL 601  LSL 602  LSL 603  P6 STATE  P601 Status  P602 Status  \
0        0        0        0        1         2            1            1
1        0        0        0        1         2            1            1
2        0        0        0        1         2            1            1
3        0        0        0        1         2            1            1
4        0        0        0        1         2            1            1

   P603 Status
0            1
1            1
2            1
3            1
4            1
```

Dataset summary

[8]:
```python
def get_summary(df : pd.DataFrame) -> pd.DataFrame:
    df_desc = pd.DataFrame(df.describe(include='all').T)
    df_summary = pd.DataFrame({
        'dtype': df.dtypes,
        'unique':df.nunique().values,
        'missing': df.isna().sum().values,
        'duplicates': df.duplicated().sum(),
        'min': df_desc['min'].values,
        'max': df_desc['max'].values,
        'avg': df_desc['mean'].values,
        'std dev': df_desc['std'].values
    })
    return df_summary
```

[9]:
```python
get_summary(df).style.background_gradient(cmap='viridis_r', low=0.8)
```

[9]:
```
<pandas.io.formats.style.Styler at 0x212eac73620>
```

[10]:
```python
def data_exploration(df : pd.DataFrame) -> None:
    dim = deu.get_nb_of_dimensions(df)
    print(bcolors.HEADER + 'Number of dimensions:' + bcolors.ENDC, dim, '\n')
```

```
    print(bcolors.HEADER + 'Number of rows:' + bcolors.ENDC, deu.
↪get_nb_of_rows(df), '\n')
    print(bcolors.HEADER + 'Column names:' + bcolors.ENDC, deu.
↪get_column_names(df), '\n')
    print(bcolors.HEADER + 'Number column names:' + bcolors.ENDC, deu.
↪get_number_column_names(df), '\n')
    print(bcolors.HEADER + 'Object column names:' + bcolors.ENDC, deu.
↪get_object_column_names(df), '\n')

    for i in range(dim):
        col = df.columns[i]
        if len(deu.get_unique_values(df, col)) > 20:
            print(bcolors.HEADER + f'Unique values of column [{col}]:' +␣
↪bcolors.ENDC, len(deu.get_unique_values(df, col)), "numerical values", '\n')
        else:
            print(bcolors.HEADER + f'Unique values of column [{col}]:' +␣
↪bcolors.ENDC, deu.get_unique_values(df, col), '\n')
```

[11]:
```
data_exploration(df)
```

```
Number of dimensions: 80

Number of rows: 14996

Column names: ['GMT +0', 'Attack', 'Label', 'FIT 101', 'LIT 101', 'MV
101', 'P1_STATE', 'P101 Status', 'P102 Status', 'AIT 201', 'AIT 202', 'AIT 203',
'FIT 201', 'LS 201', 'LS 202', 'LSL 203', 'LSLL 203', 'MV201', 'P2_STATE', 'P201
Status', 'P202 Status', 'P203 Status', 'P204 Status', 'P205 Status', 'P206
Status', 'P207 Status', 'P208 Status', 'AIT 301', 'AIT 302', 'AIT 303', 'DPIT
301', 'FIT 301', 'LIT 301', 'MV 301', 'MV 302', 'MV 303', 'MV 304', 'P3_STATE',
'P301 Status', 'P302 Status', 'AIT 401', 'AIT 402', 'FIT 401', 'LIT 401', 'LS
401', 'P4_STATE', 'P401 Status', 'P402 Status', 'P403 Status', 'P404 Status',
'UV401', 'AIT 501', 'AIT 502', 'AIT 503', 'AIT 504', 'FIT 501', 'FIT 502', 'FIT
503', 'FIT 504', 'MV 501', 'MV 502', 'MV 503', 'MV 504', 'P5_STATE', 'P501
Status', 'P502 Status', 'PIT 501', 'PIT 502', 'PIT 503', 'FIT 601', 'LSH 601',
'LSH 602', 'LSH 603', 'LSL 601', 'LSL 602', 'LSL 603', 'P6 STATE', 'P601
Status', 'P602 Status', 'P603 Status']

Number column names: ['Label', 'FIT 101', 'LIT 101', 'MV 101',
'P1_STATE', 'P101 Status', 'P102 Status', 'AIT 201', 'AIT 202', 'AIT 203', 'FIT
201', 'LS 201', 'LS 202', 'LSL 203', 'LSLL 203', 'MV201', 'P2_STATE', 'P201
Status', 'P202 Status', 'P203 Status', 'P204 Status', 'P205 Status', 'P206
Status', 'P207 Status', 'P208 Status', 'AIT 301', 'AIT 302', 'AIT 303', 'DPIT
301', 'FIT 301', 'LIT 301', 'MV 301', 'MV 302', 'MV 303', 'MV 304', 'P3_STATE',
'P301 Status', 'P302 Status', 'AIT 401', 'AIT 402', 'FIT 401', 'LIT 401', 'LS
401', 'P4_STATE', 'P401 Status', 'P402 Status', 'P403 Status', 'P404 Status',
'UV401', 'AIT 501', 'AIT 502', 'AIT 503', 'AIT 504', 'FIT 501', 'FIT 502', 'FIT
```

```
503', 'FIT 504', 'MV 501', 'MV 502', 'MV 503', 'MV 504', 'P5_STATE', 'P501
Status', 'P502 Status', 'PIT 501', 'PIT 502', 'PIT 503', 'FIT 601', 'LSH 601',
'LSH 602', 'LSH 603', 'LSL 601', 'LSL 602', 'LSL 603', 'P6 STATE', 'P601
Status', 'P602 Status', 'P603 Status']

Object column names: ['GMT +0', 'Attack']

Unique values of column [GMT +0]: 14996 numerical values

Unique values of column [Attack]: ['benign' 'Spoofing' 'Switch_ON'
'Switch_close' 'Switch_off']

Unique values of column [Label]: [0 1]

Unique values of column [FIT 101]: 310 numerical values

Unique values of column [LIT 101]: 4493 numerical values

Unique values of column [MV 101]: [1 0 2]

Unique values of column [P1_STATE]: [3 2]

Unique values of column [P101 Status]: [2 1]

Unique values of column [P102 Status]: [1]

Unique values of column [AIT 201]: 301 numerical values

Unique values of column [AIT 202]: 1484 numerical values

Unique values of column [AIT 203]: 1391 numerical values

Unique values of column [FIT 201]: 361 numerical values

Unique values of column [LS 201]: [0]

Unique values of column [LS 202]: [0]

Unique values of column [LSL 203]: [0]

Unique values of column [LSLL 203]: [0]

Unique values of column [MV201]: [2 0 1]

Unique values of column [P2_STATE]: [2]

Unique values of column [P201 Status]: [1]
```

```
Unique values of column [P202 Status]: [1]

Unique values of column [P203 Status]: [2 1]

Unique values of column [P204 Status]: [1]

Unique values of column [P205 Status]: [2 1]

Unique values of column [P206 Status]: [1]

Unique values of column [P207 Status]: [1]

Unique values of column [P208 Status]: [1]

Unique values of column [AIT 301]: 777 numerical values

Unique values of column [AIT 302]: 915 numerical values

Unique values of column [AIT 303]: 242 numerical values

Unique values of column [DPIT 301]: 637 numerical values

Unique values of column [FIT 301]: 594 numerical values

Unique values of column [LIT 301]: 3468 numerical values

Unique values of column [MV 301]: [1 0 2]

Unique values of column [MV 302]: [1 0 2]

Unique values of column [MV 303]: [1 0 2]

Unique values of column [MV 304]: [1 0 2]

Unique values of column [P3_STATE]: [99  2  4  5  6  7  9 10 14 15 16]

Unique values of column [P301 Status]: [1 2]

Unique values of column [P302 Status]: [1]

Unique values of column [AIT 401]: [0]

Unique values of column [AIT 402]: 881 numerical values

Unique values of column [FIT 401]: 231 numerical values

Unique values of column [LIT 401]: 4496 numerical values
```

```
Unique values of column [LS 401]: [0]

Unique values of column [P4_STATE]: [4]

Unique values of column [P401 Status]: [2 1]

Unique values of column [P402 Status]: [1]

Unique values of column [P403 Status]: [1]

Unique values of column [P404 Status]: [1]

Unique values of column [UV401]: [2 1]

Unique values of column [AIT 501]: 541 numerical values

Unique values of column [AIT 502]: 728 numerical values

Unique values of column [AIT 503]: [1016.27789 1016.18176 1016.05359
1016.342   1016.21381 1016.374
 1016.14972 1016.30994 1016.43811 1016.11768]

Unique values of column [AIT 504]: 226 numerical values

Unique values of column [FIT 501]: 192 numerical values

Unique values of column [FIT 502]: 787 numerical values

Unique values of column [FIT 503]: 137 numerical values

Unique values of column [FIT 504]: 115 numerical values

Unique values of column [MV 501]: [2 0 1]

Unique values of column [MV 502]: [2]

Unique values of column [MV 503]: [1]

Unique values of column [MV 504]: [1]

Unique values of column [P5_STATE]: [12]

Unique values of column [P501 Status]: [2]

Unique values of column [P502 Status]: [1]

Unique values of column [PIT 501]: 310 numerical values
```

```
Unique values of column [PIT 502]: 111 numerical values

Unique values of column [PIT 503]: 267 numerical values

Unique values of column [FIT 601]: 29 numerical values

Unique values of column [LSH 601]: [1 0]

Unique values of column [LSH 602]: [1]

Unique values of column [LSH 603]: [0]

Unique values of column [LSL 601]: [0]

Unique values of column [LSL 602]: [0]

Unique values of column [LSL 603]: [1]

Unique values of column [P6 STATE]: [2]

Unique values of column [P601 Status]: [1 2]

Unique values of column [P602 Status]: [1]

Unique values of column [P603 Status]: [1]
```

The **data_exploration** function allows us to have a look at some stats of the dataset and to print all unique values for all columns. We can see that many columns contains only 1 unique value, which won't be useful for us since no variance => no information.

[12]:
```python
def drop_useless_columns(df: pd.DataFrame) -> pd.DataFrame:
    for col in df.columns:
        if df[col].nunique() <= 1:
            df.drop(col, axis=1, inplace=True)
    return df
```

[13]:
```python
df = drop_useless_columns(df).copy()
df.head()
```

[13]:
```
                         GMT +0  Attack  Label  FIT 101   LIT 101  MV 101  \
0          2019-07-20T04:30:00Z  benign      0      0.0  729.8658       1
1          2019-07-20T04:30:01Z  benign      0      0.0  729.4340       1
2  2019-07-20T04:30:02.004013Z  benign      0      0.0  729.1200       1
3  2019-07-20T04:30:03.004013Z  benign      0      0.0  728.6882       1
4          2019-07-20T04:30:04Z  benign      0      0.0  727.7069       1

   P1_STATE  P101 Status     AIT 201   AIT 202     AIT 203   FIT 201  MV201  \
```

```
0         3            2  142.527557  9.293002  198.077423  2.335437      2
1         3            2  142.527557  9.293002  198.385025  2.335437      2
2         3            2  142.527557  9.293002  198.436300  2.335437      2
3         3            2  142.527557  9.289157  198.667000  2.335437      2
4         3            2  142.527557  9.289157  198.897720  2.335437      2

   P203 Status  P205 Status   AIT 301     AIT 302     AIT 303  DPIT 301  \
0            2            2  8.522921  256.431274  143.158966  1.190857
1            2            2  8.522921  256.431274  143.158966  1.190857
2            2            2  8.522921  256.431274  143.158966  1.190857
3            2            2  8.522921  256.431274  143.158966  1.190857
4            2            2  8.522921  256.431274  143.158966  1.190857

    FIT 301     LIT 301  MV 301  MV 302  MV 303  MV 304  P3_STATE  \
0  0.000512  730.702100       1       1       1       1        99
1  0.000512  730.902344       1       1       1       1        99
2  0.000512  732.344300       1       1       1       1        99
3  0.000512  732.704800       1       1       1       1        99
4  0.000512  732.744800       1       1       1       1        99

   P301 Status    AIT 402   FIT 401     LIT 401  P401 Status  UV401   AIT 501  \
0            1  87.951805  0.781740  1000.62805            2      2  7.489618
1            1  87.823630  0.782380  1000.55115            2      2  7.489618
2            1  87.798004  0.783021  1000.28200            2      2  7.489618
3            1  87.695465  0.783021  1000.74341            2      2  7.489618
4            1  87.618560  0.781228  1000.39734            2      2  7.489618

      AIT 502     AIT 503    AIT 504   FIT 501   FIT 502   FIT 503   FIT 504  \
0  147.398100  1016.27789  46.065113  0.781594  0.310362  0.623628  0.213432
1  147.398100  1016.27789  45.757500  0.782235  0.315102  0.623628  0.212984
2  147.398100  1016.27789  45.603690  0.782235  0.317023  0.623628  0.212984
3  147.167389  1016.27789  45.603690  0.783133  0.308057  0.623628  0.212792
4  147.090485  1016.27789  45.219173  0.783773  0.303446  0.623628  0.214009

   MV 501     PIT 501   PIT 502     PIT 503  FIT 601  LSH 601  P601 Status
0       2  167.601257  2.963509  119.921173  0.00032        1            1
1       2  167.601257  2.963509  119.921173  0.00032        1            1
2       2  167.601257  2.963509  119.921173  0.00032        1            1
3       2  167.601257  2.963509  119.921173  0.00032        1            1
4       2  167.601257  2.963509  119.921173  0.00032        1            1
```

[14]:
```
deu.get_nb_of_dimensions(df)
```

[14]: 47

We dropped 33 unuseful columns (80 -> 47).

Lets count the number of attacks, since the dataset is 'indexed' by time, simply counting the number of Label = 1 is not sufficient.

Here we count the number of attack groups, where a group is composed of one or more multiple attacks ordered consecutively.

[15]:
```python
df_attacks = df[['Attack', 'Label']].copy()

df_attacks['Shifted_Label'] = df_attacks['Label'].shift(fill_value=0)
df_attacks['Group_Start'] = (df_attacks['Label'] == 1) &␣
↪(df_attacks['Shifted_Label'] != 1)
attack_count = df_attacks['Group_Start'].sum()

print(f"Number of attack groups: {attack_count}")
```

```
Number of attack groups: 6
```

[16]:
```python
df_benign = df[['Attack', 'Label']].copy()

df_benign['Shifted_Label'] = df_benign['Label'].shift(fill_value=0)
df_benign['Group_Start'] = (df_benign['Label'] == 0) &␣
↪(df_benign['Shifted_Label'] != 0)
benign_count = df_benign['Group_Start'].sum()

print(f"Number of benign groups: {benign_count}")
```

```
Number of benign groups: 6
```

Let's have a better look on time data over these attacks

[17]:
```python
tmp_df = df.copy()
```

[18]:
```python
tmp_df['GMT +0'] = pd.to_datetime(tmp_df['GMT +0'], format='ISO8601')

tmp_df['Attack_Group'] = (tmp_df['Label'].diff(1) != 0).cumsum()

attacks_only = tmp_df[tmp_df['Label'] == 1].copy()


attacks_only['Attack_Group'] = attacks_only['Attack_Group'].
↪rank(method='dense').astype(int)

attack_summary = (
    attacks_only
    .groupby('Attack_Group')
    .agg({
        'GMT +0': ['min', 'max'],
        'Attack': 'first'
    })
)
```

```
start_time_to_index = attacks_only.reset_index().
  ↪groupby('Attack_Group')['index'].first()
end_time_to_index = attacks_only.reset_index().groupby('Attack_Group')['index'].
  ↪last()
attack_summary.columns = ['Start_Time', 'End_Time', 'Attack_Type']
attack_summary['Start_Time_Index'] = attack_summary.index.
  ↪map(start_time_to_index)
attack_summary['End_Time_Index'] = attack_summary.index.map(end_time_to_index)
attack_summary['Duration'] = attack_summary['End_Time'] -␣
  ↪attack_summary['Start_Time']
attack_summary
```

[18]:

```
                                       Start_Time  \
Attack_Group
1            2019-07-20 07:07:00.005004800+00:00
2            2019-07-20 07:13:27.003005900+00:00
3            2019-07-20 07:25:13.003005900+00:00
4            2019-07-20 07:37:19.002014100+00:00
5               2019-07-20 07:52:25.004013+00:00
6            2019-07-20 08:01:06.002014100+00:00

                                         End_Time   Attack_Type  \
Attack_Group
1            2019-07-20 07:08:45.003005900+00:00      Spoofing
2            2019-07-20 07:17:47.005004800+00:00      Spoofing
3            2019-07-20 07:29:02.002014100+00:00     Switch_ON
4               2019-07-20 07:44:48.004013+00:00     Switch_ON
5               2019-07-20 07:54:48.004013+00:00  Switch_close
6            2019-07-20 08:23:47.002014100+00:00    Switch_off

              Start_Time_Index  End_Time_Index                  Duration
Attack_Group
1                         9416            9521 0 days 00:01:44.998001100
2                         9803           10063 0 days 00:04:20.001998900
3                        10509           10738 0 days 00:03:48.999008200
4                        11235           11684 0 days 00:07:29.001998900
5                        12141           12284           0 days 00:02:23
6                        12662           14023           0 days 00:22:41
```

[19]:

```
def extract_specific_groups(df : pd.DataFrame, id_att : int = 1) -> list[pd.
  ↪DataFrame]:
    if 'Label' not in df.columns:
        raise ValueError("The DataFrame must contain a 'Label' column.")

    df['Group_ID'] = (df['Label'] == id_att).astype(int).diff().fillna(0).ne(0).
  ↪cumsum()
```

```python
    attack_df = df[df['Label'] == id_att]

    attack_groups = [group_df.drop(columns='Group_ID') for _, group_df in␣
↪attack_df.groupby('Group_ID')]

    return attack_groups
```

[20]:

```python
attack_dfs = extract_specific_groups(tmp_df, 1)

for attack_df in attack_dfs:
    print(f"Duration of the attack : {len(attack_df)} seconds")
    print(f"Attack type : {attack_df['Attack'].iloc[0]}")
    display(get_summary(attack_df).style.background_gradient(cmap='viridis_r',␣
↪low=0.8))
    print("\n\n\n===========================================================\n")
```

```
Duration of the attack : 106 seconds
Attack type : Spoofing

<pandas.io.formats.style.Styler at 0x212ec0d9790>



===========================================================

Duration of the attack : 261 seconds
Attack type : Spoofing

<pandas.io.formats.style.Styler at 0x212ecf40950>



===========================================================

Duration of the attack : 230 seconds
Attack type : Switch_ON

<pandas.io.formats.style.Styler at 0x212eade1d60>



===========================================================

Duration of the attack : 450 seconds
Attack type : Switch_ON

<pandas.io.formats.style.Styler at 0x212eabf1d90>
```

```
===========================================================

Duration of the attack : 144 seconds
Attack type : Switch_close
<pandas.io.formats.style.Styler at 0x212e8ac01a0>



===========================================================

Duration of the attack : 1362 seconds
Attack type : Switch_off
<pandas.io.formats.style.Styler at 0x212ec076060>



===========================================================
```

[21]:

```python
benign_dfs = extract_specific_groups(tmp_df, 0)

for benign_df in benign_dfs:
    print(f"Duration of the benign time before an attack : {len(benign_df)} seconds")
    display(get_summary(benign_df).style.background_gradient(cmap='viridis_r', low=0.8))
    print("\n\n\n===========================================================\n")
```

```
Duration of the benign time before an attack : 9416 seconds
<pandas.io.formats.style.Styler at 0x212ec054dd0>



===========================================================

Duration of the benign time before an attack : 281 seconds
<pandas.io.formats.style.Styler at 0x212ec05f950>



===========================================================
```

```
Duration of the benign time before an attack : 445 seconds
<pandas.io.formats.style.Styler at 0x212ec068590>

============================================================

Duration of the benign time before an attack : 496 seconds
<pandas.io.formats.style.Styler at 0x212ec074dd0>

============================================================

Duration of the benign time before an attack : 456 seconds
<pandas.io.formats.style.Styler at 0x212e8ac01a0>

============================================================

Duration of the benign time before an attack : 377 seconds
<pandas.io.formats.style.Styler at 0x212ec077200>

============================================================

Duration of the benign time before an attack : 972 seconds
<pandas.io.formats.style.Styler at 0x212ec07c530>

============================================================
```

1.1 SWaT System Overview

The system is divided into 6 parts, with each part containing the following columns (with the one removed because no variance) :

- P1: Raw Water Storage - Model-Based Monitoring System
  - FIT 101
  - LIT 101
  - MV 101
  - P1_STATE
  - P101 Status
  - P102 Status (**REMOVED**)
- P2: Chemical Dosing - Data-Driven / Model-Based Monitoring System
  - AIT 201
  - AIT 202
  - AIT 203
  - FIT 201
  - LS 201 (**REMOVED**)
  - LS 202 (**REMOVED**)
  - LSL 203 (**REMOVED**)
  - LSLL 203 (**REMOVED**)
  - MV 201
  - P2_STATE (**REMOVED**)
  - P201 Status (**REMOVED**)
  - P202 Status (**REMOVED**)
  - P203 Status
  - P204 Status (**REMOVED**)
  - P205 Status
  - P206 Status (**REMOVED**)
  - P207 Status (**REMOVED**)
  - P208 Status (**REMOVED**)
- P3: Ultra-filtration (UF) - Model-Based Monitoring System
  - AIT 301
  - AIT 302
  - AIT 303
  - DPIT 301
  - FIT 301
  - LIT 301
  - MV 301
  - MV 302
  - MV 303
  - MV 304
  - P3_STATE
  - P301 Status
  - P302 Status (**REMOVED**)
- P4: Dechlorination - Model-Based Monitoring System
  - AIT 401 (**REMOVED**)
  - AIT 402
  - FIT 401
  - LIT 401
  - LS 401 (**REMOVED**)
  - P4_STATE (**REMOVED**)
  - P401 Status

  - P402 Status (**REMOVED**)
  - P403 Status (**REMOVED**)
  - P404 Status (**REMOVED**)
  - UV401
- P5: Reverse Osmosis (RO) - Data-Driven Monitoring System
  - AIT 501
  - AIT 502
  - AIT 503
  - AIT 504
  - FIT 501
  - FIT 502
  - FIT 503
  - FIT 504
  - MV 501
  - MV 502 (**REMOVED**)
  - MV 503 (**REMOVED**)
  - MV 504 (**REMOVED**)
  - P5_STATE (**REMOVED**)
  - P501 Status (**REMOVED**)
  - P502 Status (**REMOVED**)
  - PIT 501
  - PIT 502
  - PIT 503
- P6: RO Permeate transfer, UF backwash - Data-Driven Monitoring System
  - FIT 601
  - LSH 601
  - LSH 602 (**REMOVED**)
  - LSH 603 (**REMOVED**)
  - LSL 601 (**REMOVED**)
  - LSL 602 (**REMOVED**)
  - LSL 603 (**REMOVED**)
  - P6 STATE (**REMOVED**)
  - P601 Status
  - P602 Status (**REMOVED**)
  - P603 Status (**REMOVED**)

Let's isolate each part in a dataframe.

[22]:
```
stamps = df.filter(regex='GMT.*').copy()
attacks = df.filter(regex='Attack').copy()
labels = df.filter(regex='Label').copy()

p1_ = df.filter(regex='P1.*|.*10.*').copy()
p2_ = df.filter(regex='P2.*|.*20.*').copy()
p3_ = df.filter(regex='P3.*|.*30.*').copy()
p4_ = df.filter(regex='P4.*|.*40.*').copy()
p5_ = df.filter(regex='P5.*|.*50.*').copy()
p6_ = df.filter(regex='P6.*|.*60.*').copy()
```

Let's verify that we use all the columns of the previous dataframe

[23]:
```
def check_sum_columns(ref: pd.DataFrame, list_df : list[pd.DataFrame]) -> bool␣
↪| None:
    cumsum = 0
    for df_ in list_df:
        cumsum += deu.get_nb_of_dimensions(df_)
    return cumsum == deu.get_nb_of_dimensions(ref)

check_sum_columns(df, [stamps, attacks, labels, p1_, p2_, p3_, p4_, p5_, p6_])
```

[23]:
```
True
```

Now we define utils functions that will help use prepare / pre-process the data.

[24]:
```
def prepare_dfs(list_df : list[pd.DataFrame]) -> pd.DataFrame:
    df = pd.concat(list_df, axis=1).copy()

    for column in df.columns:
        # proper float format reconstruction
        df[column] = df[column].map(
            lambda x: float(str(x).replace(',', '.'))
            if isinstance(x, str) and x.replace(',', '').replace('.', '', 1).
↪isdigit()
            else x
        )

    df = df.apply(pd.to_numeric, errors='coerce')
    df.fillna(value=0, inplace=True)
    return df
```

Input data normalization:
$z = (x - u)/s$

where: - u: mean - s: standard deviation

[25]:
```
def scale_data(df : pd.DataFrame) -> np.ndarray:
    scaler = StandardScaler()
    scaled_data = scaler.fit_transform(df)

    print(f'Scaled data shape: {scaled_data.shape}')
    return scaled_data
```

Then we compute the principal component analysis of the data to project it to a lower dimensional space.
Notice that the solver is set to `"full"` which means the exact "full" SVD is computed and optionally truncated afterwards.

[26]:
```python
def pca_(scaled_data : np.ndarray, var_retention : float = 0.95) -> tuple[np.
↪array, PCA]:
    pca = PCA(n_components=var_retention, svd_solver="full")
    pca_components = pca.fit_transform(scaled_data)

    print(f'PCA components: {pca_components.shape}')
    return pca_components, pca
```

PCA Data reconstruction with error approximation (L2 error). By reconstructing the data with the pca_components, we can identify outliers, which are potentially anomalies.

[27]:
```python
def metrics_pca(pca : PCA, pca_components: np.array, scaled_data : np.ndarray)␣
↪-> tuple[np.ndarray, np.ndarray, np.ndarray]:
    reconstructed = pca.inverse_transform(pca_components)
    reconstruction_error = np.mean((scaled_data - reconstructed) ** 2, axis=1)

    threshold_pca = np.mean(reconstruction_error) + 3 * np.
↪std(reconstruction_error)
    anomalies_pca = reconstruction_error > threshold_pca
    indexes = (reconstruction_error >  threshold_pca).nonzero()[0]

    print(f'Number of anomalies: {np.sum(anomalies_pca)}')
    return reconstruction_error, threshold_pca, anomalies_pca, indexes
```

1.2 Unsupervised algorithms setup

1. Isolation Forest

[28]:
```python
def metrics_iso_forest(scaled_data : np.ndarray, contamination : float = 0.01,␣
↪random_state : int = 42) -> tuple[np.ndarray, np.ndarray]:
    iso_forest = IsolationForest(contamination=contamination,␣
↪random_state=random_state)

    anomaly_scores = iso_forest.fit_predict(scaled_data)
    anomalies_iforest = anomaly_scores == -1

    return anomaly_scores, anomalies_iforest
```

2. Local Outlier Factor

[29]:
```python
def metrics_lof(scaled_data : np.ndarray, contamination : float = 0.01,␣
↪n_neighbors : int = 20) -> tuple[np.ndarray, np.ndarray]:
    loc_out_factor = LocalOutlierFactor(n_neighbors=n_neighbors,␣
↪contamination=contamination)

    anomaly_scores = loc_out_factor.fit_predict(scaled_data)
    anomalies_lof = anomaly_scores == -1
```

```
    return anomaly_scores, anomalies_lof
```

1.2.1 Plots and metrics

[30]:
```
def plot_reconstruction_error(reconstruction_error: Any, threshold_pca: Any):
    plt.figure(figsize=(12, 6))
    plt.plot(reconstruction_error, label='Reconstruction error', color='blue')
    plt.axhline(threshold_pca, color='red', linestyle='--', label='Anomaly␣
↪threshold')
    plt.title('Reconstruction error with anomaly threshold (PCA)')
    plt.xlabel('Time index')
    plt.ylabel('Error (L2)')
    plt.legend()
    plt.show()


def plot_heatmap(df : pd.DataFrame, size: tuple[int, int] = (30, 15)) -> None:
    plt.figure(figsize=size)
    normal_data = df[df['Anomaly'] == False].iloc[:, :-3].apply(pd.to_numeric,␣
↪errors='coerce').fillna(0)
    anomaly_data = df[df['Anomaly'] == True].iloc[:, :-3].apply(pd.to_numeric,␣
↪errors='coerce').fillna(0)

    data = np.abs(anomaly_data - normal_data.mean())
    data = data.select_dtypes(include=[np.number])

    sns.heatmap(data.T, cmap='coolwarm', cbar=True)
    plt.title('Heatmap of difference between data values at anomaly timestamp␣
↪with mean of sensor on normal timestamps.')
    plt.xlabel('Time Index')
    plt.ylabel('Sensors')
    plt.show()
```

Common anomalies detection overview and metrics

[31]:
```
def check_common_anomalies_3methods(anomalies_pca: np.ndarray,␣
↪anomalies_iforest: np.ndarray, anomalies_lof: np.ndarray) -> None:
    common_anomalies_all = anomalies_pca & anomalies_iforest & anomalies_lof
    common_anomalies_pca_iforest = anomalies_pca & anomalies_iforest
    common_anomalies_pca_lof = anomalies_pca & anomalies_lof
    common_anomalies_iforest_lof = anomalies_iforest & anomalies_lof

    print(f"Total common anomalies detected by all methods: {np.
↪sum(common_anomalies_all)}")
    print(f"Total common anomalies detected by PCA and Isolation Forest: {np.
↪sum(common_anomalies_pca_iforest)}")
```

```
    print(f"Total common anomalies detected by PCA and Local Outlier Factor:␣
↪{np.sum(common_anomalies_pca_lof)}")
    print(f"Total common anomalies detected by Isolation Forest and Local␣
↪Outlier Factor: {np.sum(common_anomalies_iforest_lof)}")


def evaluate_metrics(y_true, y_pred):
    precision = precision_score(y_true, y_pred)
    recall = recall_score(y_true, y_pred)
    f1 = f1_score(y_true, y_pred)
    mcc = matthews_corrcoef(y_true, y_pred)
    balanced_acc = balanced_accuracy_score(y_true, y_pred)
    auc_prc = roc_auc_score(y_true, y_pred)

    return {
        "Precision": precision,
        "Recall": recall,
        "F1-Score": f1,
        "MCC": mcc,
        "Balanced Accuracy": balanced_acc,
        "AUC-PRC": auc_prc
    }
```

1.3 Multi Stage Multi Point (MSMP)

Targets multiple sensors at multiple points in time.

[32]:
```
p1_6 = prepare_dfs([p1_, p2_, p3_, p4_, p5_, p6_])
p1_6.head()
```

[32]:
```
   FIT 101   LIT 101  MV 101  P1_STATE  P101 Status     AIT 201   AIT 202  \
0      0.0  729.8658       1         3            2  142.527557  9.293002
1      0.0  729.4340       1         3            2  142.527557  9.293002
2      0.0  729.1200       1         3            2  142.527557  9.293002
3      0.0  728.6882       1         3            2  142.527557  9.289157
4      0.0  727.7069       1         3            2  142.527557  9.289157

      AIT 203   FIT 201  MV201  P203 Status  P205 Status   AIT 301  \
0  198.077423  2.335437      2            2            2  8.522921
1  198.385025  2.335437      2            2            2  8.522921
2  198.436300  2.335437      2            2            2  8.522921
3  198.667000  2.335437      2            2            2  8.522921
4  198.897720  2.335437      2            2            2  8.522921

      AIT 302     AIT 303  DPIT 301   FIT 301     LIT 301  MV 301  MV 302  \
0  256.431274  143.158966  1.190857  0.000512  730.702100       1       1
1  256.431274  143.158966  1.190857  0.000512  730.902344       1       1
```

```
2  256.431274  143.158966  1.190857  0.000512  732.344300       1       1
3  256.431274  143.158966  1.190857  0.000512  732.704800       1       1
4  256.431274  143.158966  1.190857  0.000512  732.744800       1       1

   MV 303  MV 304  P3_STATE  P301 Status    AIT 402   FIT 401     LIT 401  \
0       1       1        99            1  87.951805  0.781740  1000.62805
1       1       1        99            1  87.823630  0.782380  1000.55115
2       1       1        99            1  87.798004  0.783021  1000.28200
3       1       1        99            1  87.695465  0.783021  1000.74341
4       1       1        99            1  87.618560  0.781228  1000.39734

   P401 Status  UV401   AIT 501     AIT 502     AIT 503    AIT 504   FIT 501  \
0            2      2  7.489618  147.398100  1016.27789  46.065113  0.781594
1            2      2  7.489618  147.398100  1016.27789  45.757500  0.782235
2            2      2  7.489618  147.398100  1016.27789  45.603690  0.782235
3            2      2  7.489618  147.167389  1016.27789  45.603690  0.783133
4            2      2  7.489618  147.090485  1016.27789  45.219173  0.783773

    FIT 502   FIT 503   FIT 504  MV 501     PIT 501   PIT 502     PIT 503  \
0  0.310362  0.623628  0.213432       2  167.601257  2.963509  119.921173
1  0.315102  0.623628  0.212984       2  167.601257  2.963509  119.921173
2  0.317023  0.623628  0.212984       2  167.601257  2.963509  119.921173
3  0.308057  0.623628  0.212792       2  167.601257  2.963509  119.921173
4  0.303446  0.623628  0.214009       2  167.601257  2.963509  119.921173

   FIT 601  LSH 601  P601 Status
0  0.00032        1            1
1  0.00032        1            1
2  0.00032        1            1
3  0.00032        1            1
4  0.00032        1            1
```

[33]:
```python
scaled_data = scale_data(p1_6)
```

```
Scaled data shape: (14996, 44)
```

1.3.1 PCA

[34]:
```python
pca_components, pca = pca_(scaled_data)
```

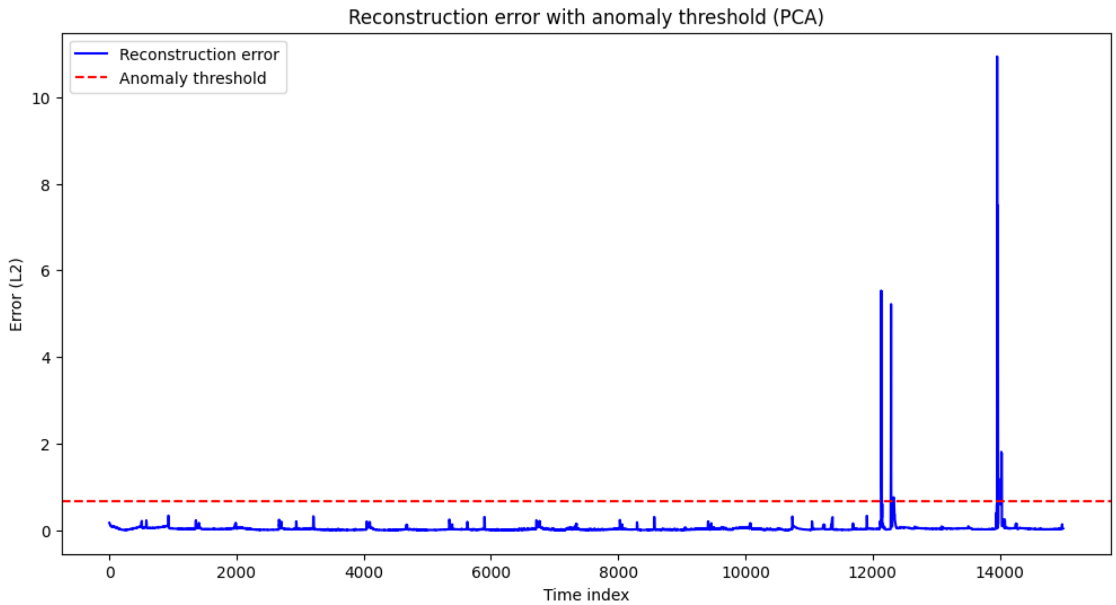
```
PCA components: (14996, 17)
```

As we can see, the PCA algorithm automatically selected 17 principal components.

[35]:
```python
reconstruction_error, threshold_pca, anomalies_pca, indexes  = metrics_pca(pca,␣
↪pca_components, scaled_data)
```

```
Number of anomalies: 54
```

[36]:
```
plot_reconstruction_error(reconstruction_error, threshold_pca)
```



[37]:
```
len(indexes)
```

[37]: 54

There are 54 anomalies among the 3 observed peaks on the reconstructed error.

1.3.2 Isolation Forest

[38]:
```
anomaly_scores_iso, anomalies_iforest = metrics_iso_forest(scaled_data)
```

1.3.3 Local Outlier Factor

[39]:
```
anomaly_scores_lof, anomalies_lof = metrics_lof(scaled_data)
```

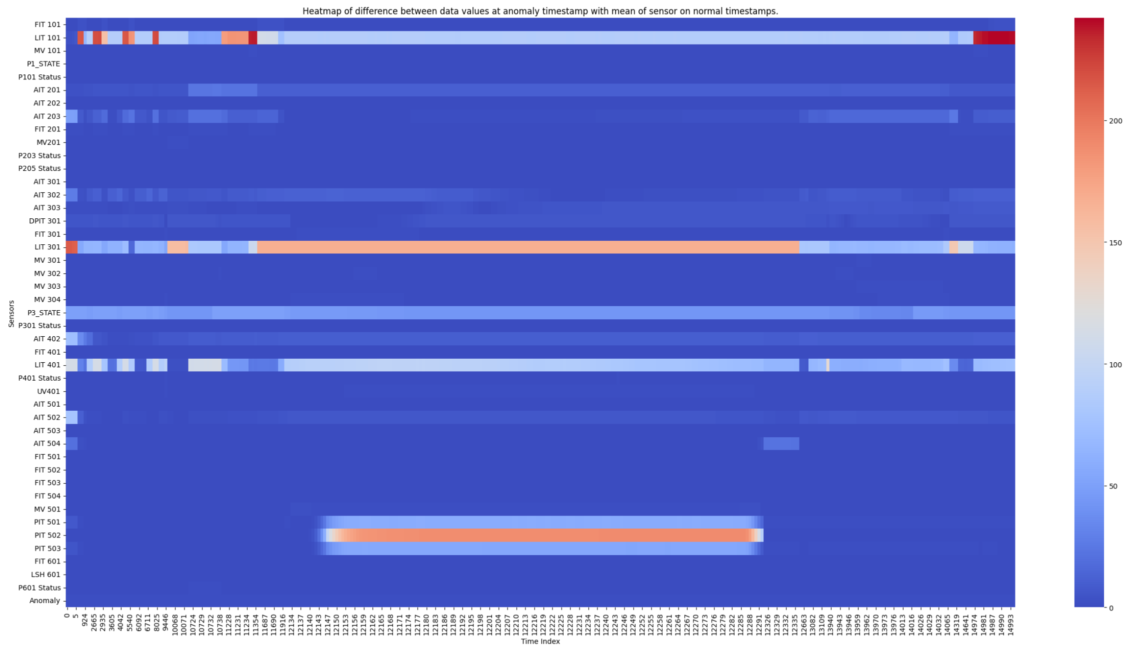
1.3.4 Combining Anomalies

Here we are combining the results of all three methods.

[40]:
```
combined_anomalies = anomalies_pca | anomalies_iforest | anomalies_lof
```

[41]:
```
p1_6['Anomaly'] = combined_anomalies
p1_6['Reconstruction_Error'] = reconstruction_error
p1_6['Isolation_Score'] = anomaly_scores_iso
p1_6['Lof_Score'] = anomaly_scores_lof
```

[42]:
```
plot_heatmap(p1_6)
```



The heatmap provides information about which columns possess the biggest impact on anomaly detection.

We compute the absolute value of the difference between the value of the sensor during an attack and the mean value of the sensor under normal operation.

Temperature at time t:
$|x\_anomaly(t) - x\_mean|$

Here we can see that at a global scale, the sensors which are more likely to have been attacked are the following ones: **LIT 101, LIT 301, P3_STATE, LIT 401, PIT 502, AIT 503.**

[43]:
```
print(f"Total anomalies detected by PCA: {np.sum(anomalies_pca)}")
print(f"Total anomalies detected by Isolation Forest: {np.
  ↪sum(anomalies_iforest)}")
print(f"Total anomalies detected by Local Outlier Factor: {np.
  ↪sum(anomalies_lof)}\n\n")

check_common_anomalies_3methods(anomalies_pca, anomalies_iforest, anomalies_lof)
```

```
Total anomalies detected by PCA: 54
Total anomalies detected by Isolation Forest: 148
Total anomalies detected by Local Outlier Factor: 150


Total common anomalies detected by all methods: 6
```

```
Total common anomalies detected by PCA and Isolation Forest: 6
Total common anomalies detected by PCA and Local Outlier Factor: 28
Total common anomalies detected by Isolation Forest and Local Outlier Factor: 7
```

Let's compare our unsupervised predictions with true labels.

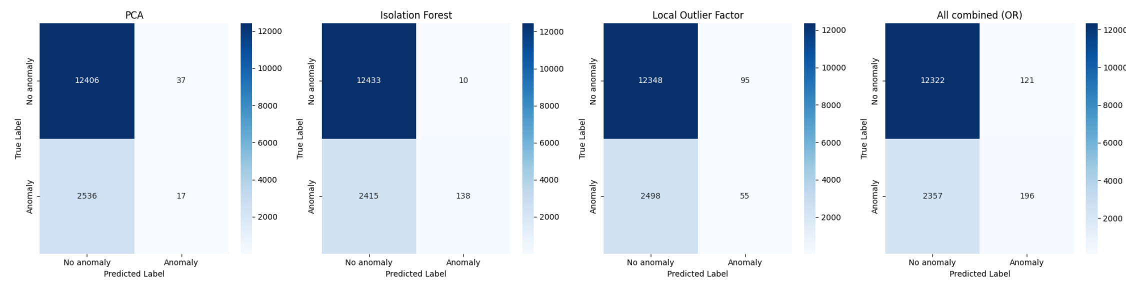
[44]:
```python
true_labels = prepare_dfs([labels])

print("Classification report:")
print(classification_report(true_labels, p1_6['Anomaly']))
```

```
Classification report:
              precision    recall  f1-score   support

           0       0.84      0.99      0.91     12443
           1       0.62      0.08      0.14      2553

    accuracy                           0.83     14996
   macro avg       0.73      0.53      0.52     14996
weighted avg       0.80      0.83      0.78     14996
```

In terms of pure anomaly detection, the results are pretty low, but we will see after that this is enough to detect attacks.

[45]:
```python
cm1 = confusion_matrix(true_labels, anomalies_pca)
cm2 = confusion_matrix(true_labels, anomalies_iforest)
cm3 = confusion_matrix(true_labels, anomalies_lof)
cm4 = confusion_matrix(true_labels, p1_6['Anomaly'])
```

[46]:
```python
fig, axes = plt.subplots(1, 4, figsize=(20, 5))

# Plot each confusion matrix
for ax, cm, title in zip(
    axes,
    [cm1, cm2, cm3, cm4],
    ['PCA', 'Isolation Forest', 'Local Outlier Factor', 'All combined (OR)']
):
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['No␣
↪anomaly', 'Anomaly'], yticklabels=['No anomaly', 'Anomaly'], ax=ax)
    ax.set_title(title)
    ax.set_xlabel('Predicted Label')
    ax.set_ylabel('True Label')

# Adjust layout for better appearance
plt.tight_layout()
plt.show()
```



As we can see, the isolation forest > local outlier factor > pca, in term of anomaly detection.

[47]:
```python
attacks_dict = {(i, attack_summary.iloc[i]["Attack_Type"]): [] for i in range(len(attack_summary))}
for i, line in enumerate(attack_summary.iterrows()):
  key = list(attacks_dict.keys())[i]
  for j in range(len(anomalies_pca)):
    if anomalies_pca[j] and j >= line[1]["Start_Time_Index"] and j <= line[1]["End_Time_Index"]:
      attacks_dict[key].append(("pca", j))

  for j in range(len(anomalies_iforest)):
    if anomalies_iforest[j] and j >= line[1]["Start_Time_Index"] and j <= line[1]["End_Time_Index"]:
      attacks_dict[key].append(("iforest", j))

  for j in range(len(anomalies_lof)):
    if anomalies_lof[j] and j >= line[1]["Start_Time_Index"] and j <= line[1]["End_Time_Index"]:
      attacks_dict[key].append(("lof", j))

for k, v in attacks_dict.items():
  print(f'On attack {k[0]} of type {k[1]}, which timestamps range from {attack_summary.iloc[k[0]]["Start_Time_Index"]} to {attack_summary.iloc[k[0]]["End_Time_Index"]}, we found these anomilies : {v}')
```

```
On attack 0 of type Spoofing, which timestamps range from 9416 to 9521, we found
these anomilies : [('lof', 9446)]
On attack 1 of type Spoofing, which timestamps range from 9803 to 10063, we
found these anomilies : []
On attack 2 of type Switch_ON, which timestamps range from 10509 to 10738, we
found these anomilies : [('lof', 10723), ('lof', 10724), ('lof', 10726), ('lof',
10727), ('lof', 10729), ('lof', 10730), ('lof', 10731), ('lof', 10732), ('lof',
10736), ('lof', 10737), ('lof', 10738)]
On attack 3 of type Switch_ON, which timestamps range from 11235 to 11684, we
found these anomilies : [('lof', 11352), ('lof', 11353), ('lof', 11354)]
On attack 4 of type Switch_close, which timestamps range from 12141 to 12284, we
```

```
found these anomilies : [('pca', 12141), ('pca', 12142), ('pca', 12143), ('pca',
12144), ('iforest', 12147), ('iforest', 12148), ('iforest', 12149), ('iforest',
12150), ('iforest', 12151), ('iforest', 12152), ('iforest', 12153), ('iforest',
12154), ('iforest', 12155), ('iforest', 12156), ('iforest', 12157), ('iforest',
12158), ('iforest', 12159), ('iforest', 12160), ('iforest', 12161), ('iforest',
12162), ('iforest', 12163), ('iforest', 12164), ('iforest', 12165), ('iforest',
12166), ('iforest', 12167), ('iforest', 12168), ('iforest', 12169), ('iforest',
12170), ('iforest', 12171), ('iforest', 12172), ('iforest', 12173), ('iforest',
12174), ('iforest', 12175), ('iforest', 12176), ('iforest', 12177), ('iforest',
12178), ('iforest', 12179), ('iforest', 12180), ('iforest', 12181), ('iforest',
12182), ('iforest', 12183), ('iforest', 12184), ('iforest', 12185), ('iforest',
12186), ('iforest', 12187), ('iforest', 12188), ('iforest', 12189), ('iforest',
12190), ('iforest', 12191), ('iforest', 12192), ('iforest', 12193), ('iforest',
12194), ('iforest', 12195), ('iforest', 12196), ('iforest', 12197), ('iforest',
12198), ('iforest', 12199), ('iforest', 12200), ('iforest', 12201), ('iforest',
12202), ('iforest', 12203), ('iforest', 12204), ('iforest', 12205), ('iforest',
12206), ('iforest', 12207), ('iforest', 12208), ('iforest', 12209), ('iforest',
12210), ('iforest', 12211), ('iforest', 12212), ('iforest', 12213), ('iforest',
12214), ('iforest', 12215), ('iforest', 12216), ('iforest', 12217), ('iforest',
12218), ('iforest', 12219), ('iforest', 12220), ('iforest', 12221), ('iforest',
12222), ('iforest', 12223), ('iforest', 12224), ('iforest', 12225), ('iforest',
12226), ('iforest', 12227), ('iforest', 12228), ('iforest', 12229), ('iforest',
12230), ('iforest', 12231), ('iforest', 12232), ('iforest', 12233), ('iforest',
12234), ('iforest', 12235), ('iforest', 12236), ('iforest', 12237), ('iforest',
12238), ('iforest', 12239), ('iforest', 12240), ('iforest', 12241), ('iforest',
12242), ('iforest', 12243), ('iforest', 12244), ('iforest', 12245), ('iforest',
12246), ('iforest', 12247), ('iforest', 12248), ('iforest', 12249), ('iforest',
12250), ('iforest', 12251), ('iforest', 12252), ('iforest', 12253), ('iforest',
12254), ('iforest', 12255), ('iforest', 12256), ('iforest', 12257), ('iforest',
12258), ('iforest', 12259), ('iforest', 12260), ('iforest', 12261), ('iforest',
12262), ('iforest', 12263), ('iforest', 12264), ('iforest', 12265), ('iforest',
12266), ('iforest', 12267), ('iforest', 12268), ('iforest', 12269), ('iforest',
12270), ('iforest', 12271), ('iforest', 12272), ('iforest', 12273), ('iforest',
12274), ('iforest', 12275), ('iforest', 12276), ('iforest', 12277), ('iforest',
12278), ('iforest', 12279), ('iforest', 12280), ('iforest', 12281), ('iforest',
12282), ('iforest', 12283), ('iforest', 12284), ('lof', 12143), ('lof', 12144),
('lof', 12145), ('lof', 12244)]
On attack 5 of type Switch_off, which timestamps range from 12662 to 14023, we
found these anomilies : [('pca', 13959), ('pca', 13960), ('pca', 13961), ('pca',
13962), ('pca', 13968), ('pca', 13969), ('pca', 13970), ('pca', 13971), ('pca',
13972), ('pca', 13973), ('pca', 13974), ('pca', 13977), ('pca', 13978), ('lof',
12662), ('lof', 12663), ('lof', 12664), ('lof', 13081), ('lof', 13082), ('lof',
13083), ('lof', 13108), ('lof', 13109), ('lof', 13110), ('lof', 13507), ('lof',
13940), ('lof', 13941), ('lof', 13942), ('lof', 13943), ('lof', 13944), ('lof',
13945), ('lof', 13946), ('lof', 13949), ('lof', 13950), ('lof', 13959), ('lof',
13960), ('lof', 13961), ('lof', 13969), ('lof', 13970), ('lof', 13971), ('lof',
13972), ('lof', 13973), ('lof', 13974), ('lof', 13975), ('lof', 13976), ('lof',
13977), ('lof', 13978), ('lof', 14013), ('lof', 14014), ('lof', 14015), ('lof',
```

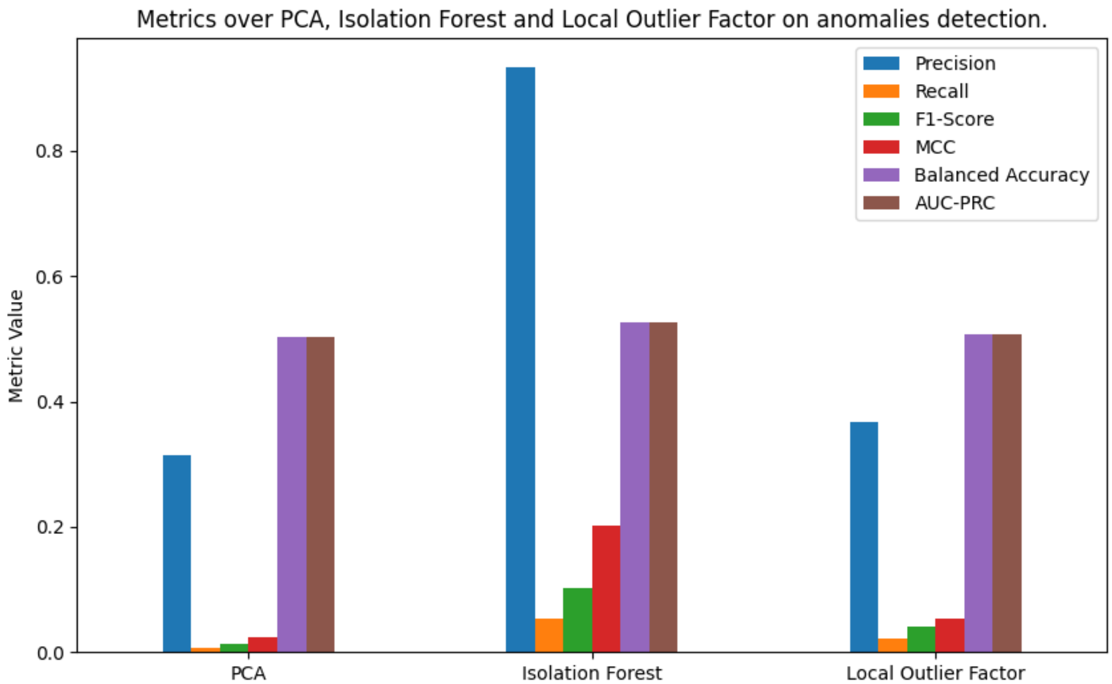
```
14016)]
```

The accuracy is mid, the anomaly aren't all detected, but it seems to be enough to detect an occuring attack, as we can see **we detect 5 out of 6 of them.**

[48]:
```
metrics_pca_dict = evaluate_metrics(true_labels, anomalies_pca)
metrics_iso_forest_dict = evaluate_metrics(true_labels, anomalies_iforest)
metrics_lof_dict = evaluate_metrics(true_labels, anomalies_lof)

results = pd.DataFrame([metrics_pca_dict, metrics_iso_forest_dict,␣
↪metrics_lof_dict],
                       index=["PCA", "Isolation Forest", "Local Outlier␣
↪Factor"])

print(results)
```

```
                      Precision    Recall  F1-Score       MCC  \
PCA                    0.314815  0.006659  0.013042  0.023124
Isolation Forest       0.932432  0.054054  0.102184  0.202463
Local Outlier Factor   0.366667  0.021543  0.040696  0.052531

                      Balanced Accuracy   AUC-PRC
PCA                            0.501843  0.501843
Isolation Forest               0.526625  0.526625
Local Outlier Factor           0.506954  0.506954
```

[49]:
```
results.plot(kind='bar', figsize=(10, 6), title='Metrics over PCA, Isolation␣
↪Forest and Local Outlier Factor on anomalies detection.')
plt.ylabel('Metric Value')
plt.xticks(rotation=0)
plt.legend(loc='upper right')
plt.show()
```



1.4 Single Stage Multi Point (SSMP)

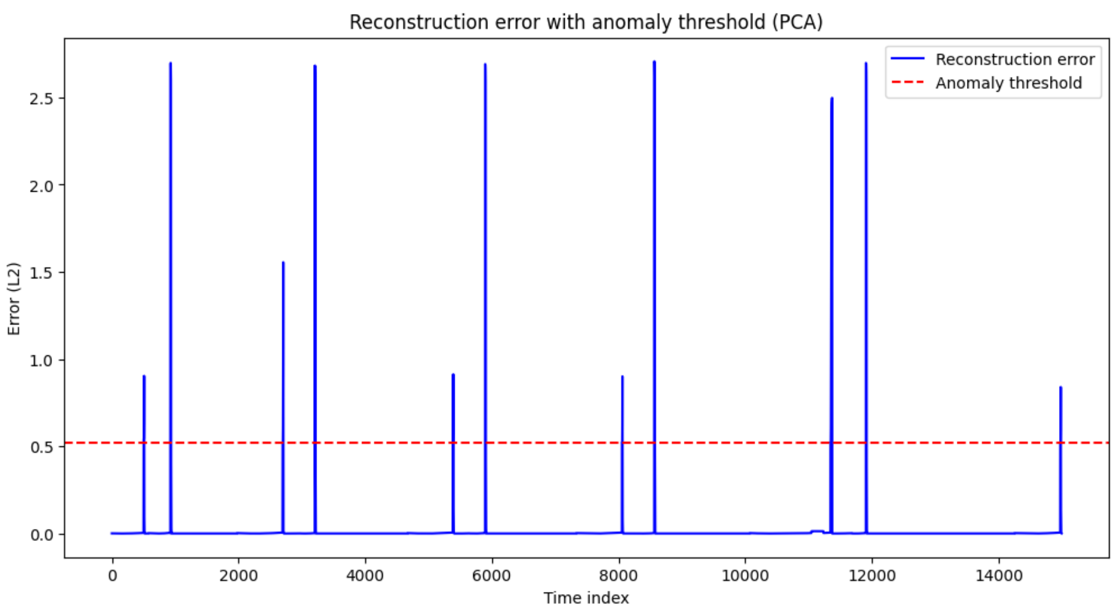
Targets multiple sensors at a single point in time.

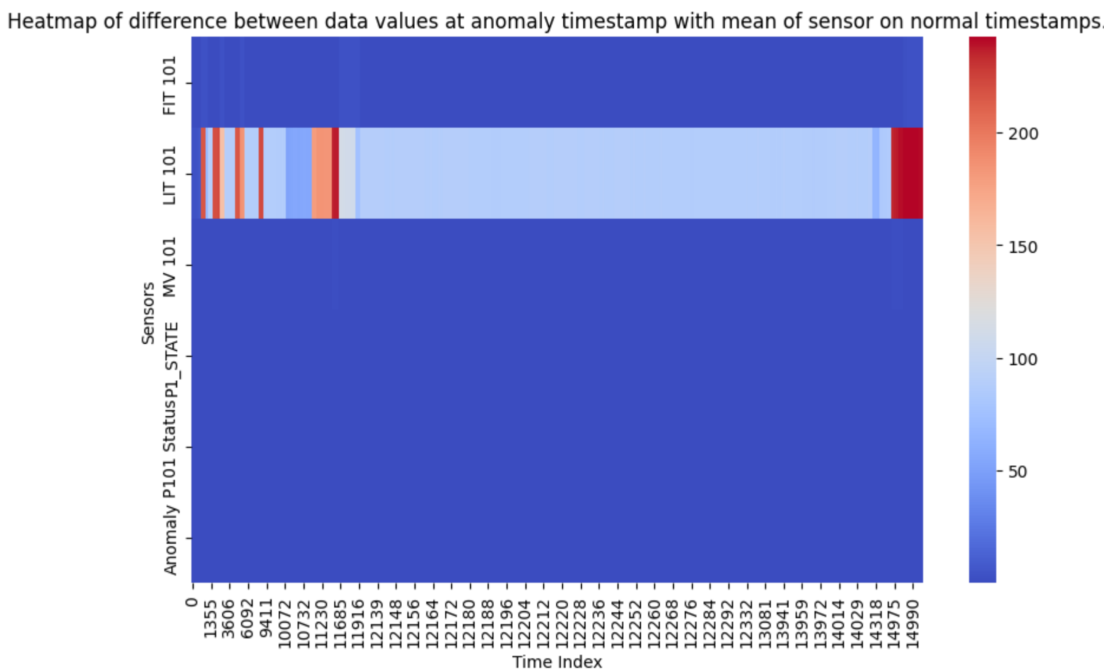
1.4.1 SSMP - P1

[50]:
```
p1 = prepare_dfs([p1_])
scaled_data = scale_data(p1)

pca_components, pca = pca_(scaled_data)
reconstruction_error, threshold_pca, anomalies_pca, indexes = metrics_pca(pca,␣
↪pca_components, scaled_data)

anomaly_scores_iso, anomalies_iforest = metrics_iso_forest(scaled_data)

anomaly_scores_lof, anomalies_lof = metrics_lof(scaled_data)

combined_anomalies_p1_or = anomalies_pca | anomalies_iforest | anomalies_lof

print("\nConfusion matrix :\n")
print(confusion_matrix(labels, combined_anomalies_p1_or))

p1['Anomaly'] = combined_anomalies
p1['Reconstruction_Error'] = reconstruction_error
p1['Isolation_Score'] = anomaly_scores_iso
```

```
p1['Lof_Score'] = anomaly_scores_lof

plot_reconstruction_error(reconstruction_error, threshold_pca)
plot_heatmap(p1, (10, 6))

print(f"Total anomalies detected by PCA: {np.sum(anomalies_pca)}")
print(f"Total anomalies detected by Isolation Forest: {np.
 ↪sum(anomalies_iforest)}")
print(f"Total anomalies detected by Local Outlier Factor: {np.
 ↪sum(anomalies_lof)}\n\n")

check_common_anomalies_3methods(anomalies_pca, anomalies_iforest, anomalies_lof)
```

```
Scaled data shape: (14996, 5)
PCA components: (14996, 4)
Number of anomalies: 105

Confusion matrix :

[[12204   239]
 [ 2523    30]]
```





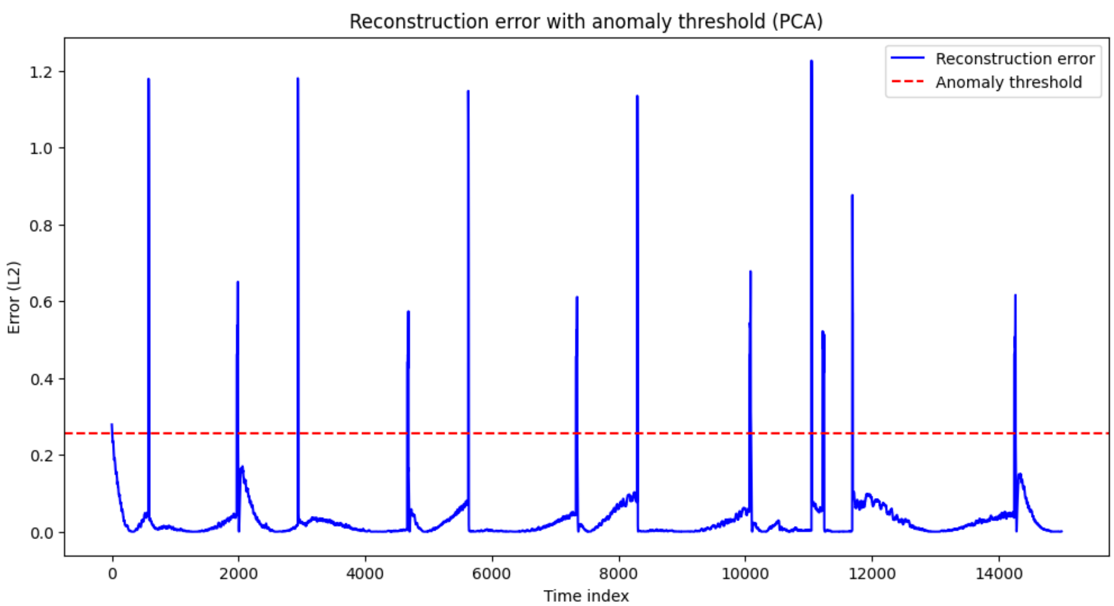

```
Total anomalies detected by PCA: 105
Total anomalies detected by Isolation Forest: 150
Total anomalies detected by Local Outlier Factor: 150


Total common anomalies detected by all methods: 32
Total common anomalies detected by PCA and Isolation Forest: 93
Total common anomalies detected by PCA and Local Outlier Factor: 33
Total common anomalies detected by Isolation Forest and Local Outlier Factor: 42
```

1.4.2 SSMP - P2

[51]:
```
p2 = prepare_dfs([p2_])
scaled_data = scale_data(p2)

pca_components, pca = pca_(scaled_data)
reconstruction_error, threshold_pca, anomalies_pca, indexes = metrics_pca(pca,␣
↪pca_components, scaled_data)

anomaly_scores_iso, anomalies_iforest = metrics_iso_forest(scaled_data)

anomaly_scores_lof, anomalies_lof = metrics_lof(scaled_data)

combined_anomalies_p2_or = anomalies_pca | anomalies_iforest | anomalies_lof
```

```
print("\nConfusion matrix :\n")
print(confusion_matrix(labels, combined_anomalies_p2_or))

p2['Anomaly'] = combined_anomalies
p2['Reconstruction_Error'] = reconstruction_error
p2['Isolation_Score'] = anomaly_scores_iso
p2['Lof_Score'] = anomaly_scores_lof

plot_reconstruction_error(reconstruction_error, threshold_pca)
plot_heatmap(p2, (10, 6))

print(f"Total anomalies detected by PCA: {np.sum(anomalies_pca)}")
print(f"Total anomalies detected by Isolation Forest: {np.
↪sum(anomalies_iforest)}")
print(f"Total anomalies detected by Local Outlier Factor: {np.
↪sum(anomalies_lof)}\n\n")

check_common_anomalies_3methods(anomalies_pca, anomalies_iforest, anomalies_lof)
```

```
Scaled data shape: (14996, 7)
PCA components: (14996, 3)
Number of anomalies: 191

Confusion matrix :

[[12197   246]
 [ 2482    71]]
```




```
Total anomalies detected by PCA: 191
Total anomalies detected by Isolation Forest: 150
Total anomalies detected by Local Outlier Factor: 150


Total common anomalies detected by all methods: 29
Total common anomalies detected by PCA and Isolation Forest: 111
Total common anomalies detected by PCA and Local Outlier Factor: 49
Total common anomalies detected by Isolation Forest and Local Outlier Factor: 43
```

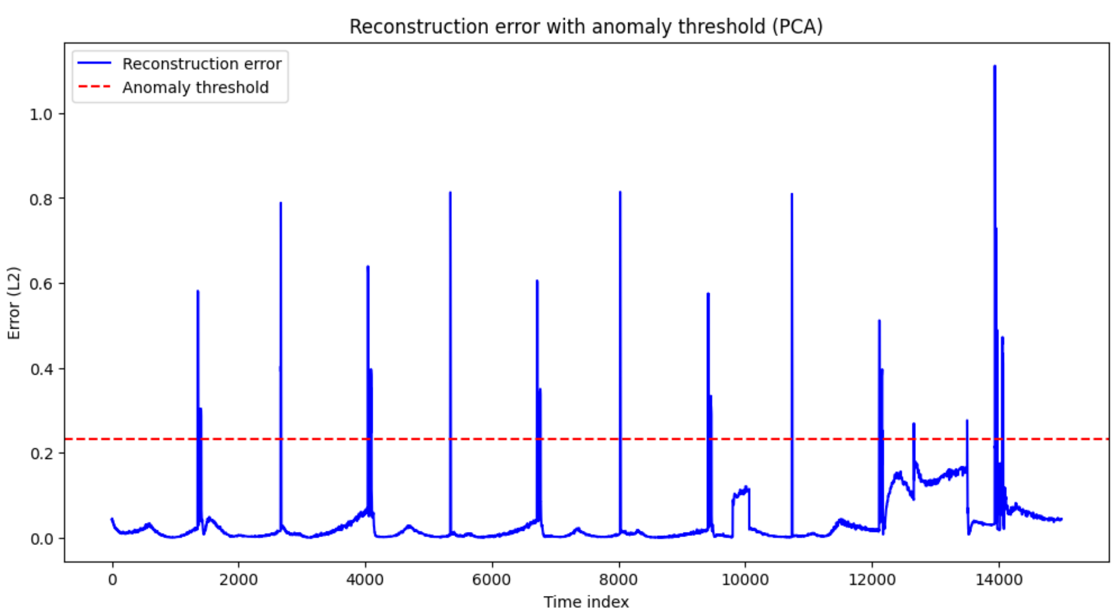
1.4.3 SSMP - P3

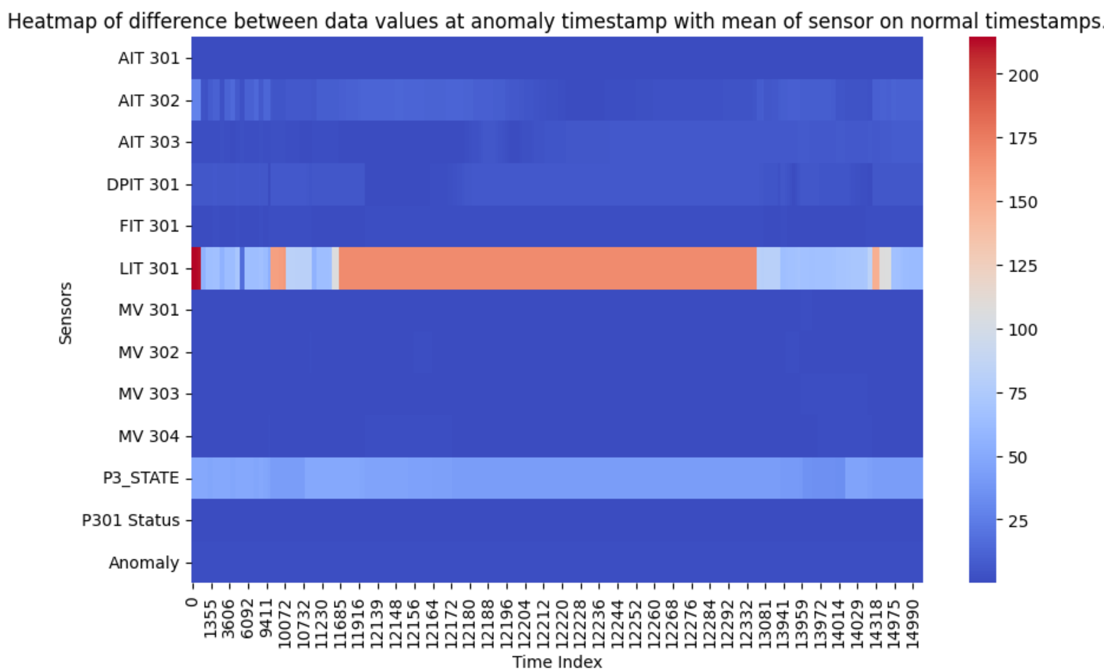
[52]:

```
p3 = prepare_dfs([p3_])
scaled_data = scale_data(p3)

pca_components, pca = pca_(scaled_data)
reconstruction_error, threshold_pca, anomalies_pca, indexes = metrics_pca(pca,
  ↪pca_components, scaled_data)

anomaly_scores_iso, anomalies_iforest = metrics_iso_forest(scaled_data)

anomaly_scores_lof, anomalies_lof = metrics_lof(scaled_data)

combined_anomalies_p3_or = anomalies_pca | anomalies_iforest | anomalies_lof
```

```python
print("\nConfusion matrix :\n")
print(confusion_matrix(labels, combined_anomalies_p3_or))

p3['Anomaly'] = combined_anomalies
p3['Reconstruction_Error'] = reconstruction_error
p3['Isolation_Score'] = anomaly_scores_iso
p3['Lof_Score'] = anomaly_scores_lof

plot_reconstruction_error(reconstruction_error, threshold_pca)
plot_heatmap(p3, (10, 6))

print(f"Total anomalies detected by PCA: {np.sum(anomalies_pca)}")
print(f"Total anomalies detected by Isolation Forest: {np.
 ↪sum(anomalies_iforest)}")
print(f"Total anomalies detected by Local Outlier Factor: {np.
 ↪sum(anomalies_lof)}\n\n")

check_common_anomalies_3methods(anomalies_pca, anomalies_iforest, anomalies_lof)
```

```
Scaled data shape: (14996, 12)
PCA components: (14996, 7)
Number of anomalies: 213

Confusion matrix :

[[12200   243]
 [ 2401   152]]
```





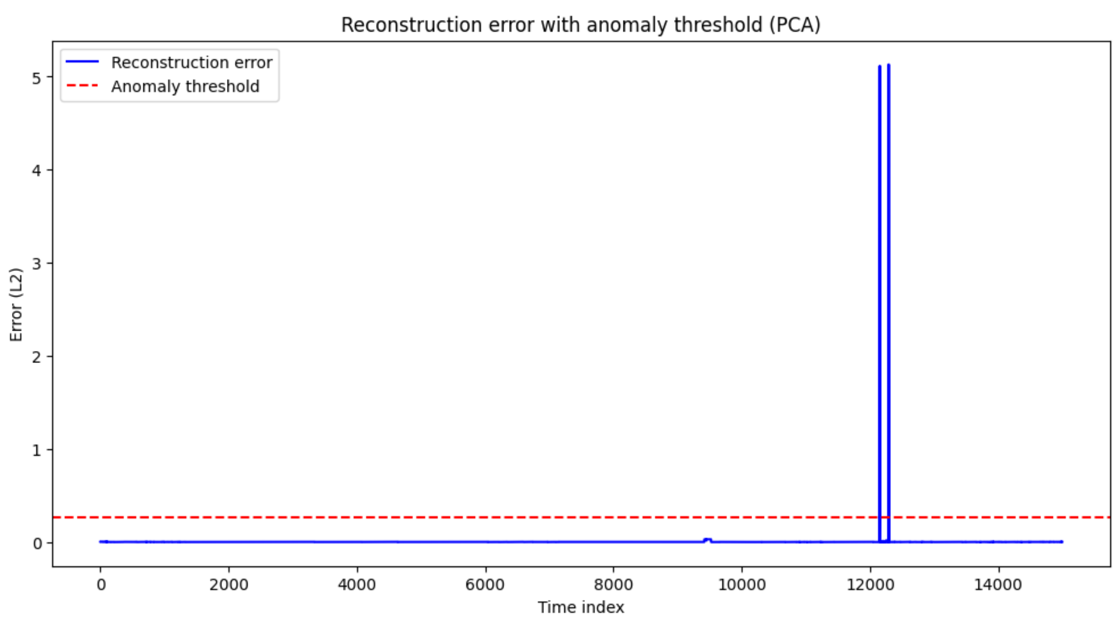

```
Total anomalies detected by PCA: 213
Total anomalies detected by Isolation Forest: 150
Total anomalies detected by Local Outlier Factor: 150


Total common anomalies detected by all methods: 22
Total common anomalies detected by PCA and Isolation Forest: 52
Total common anomalies detected by PCA and Local Outlier Factor: 56
Total common anomalies detected by Isolation Forest and Local Outlier Factor: 32
```

1.4.4 SSMP - P4

[53]:
```
p4 = prepare_dfs([p4_])
scaled_data = scale_data(p4)

pca_components, pca = pca_(scaled_data)
reconstruction_error, threshold_pca, anomalies_pca, indexes = metrics_pca(pca,
    pca_components, scaled_data)

anomaly_scores_iso, anomalies_iforest = metrics_iso_forest(scaled_data)

anomaly_scores_lof, anomalies_lof = metrics_lof(scaled_data)

combined_anomalies_p4_or = anomalies_pca | anomalies_iforest | anomalies_lof
```

```
print("\nConfusion matrix :\n")
print(confusion_matrix(labels, combined_anomalies_p4_or))

p4['Anomaly'] = combined_anomalies
p4['Reconstruction_Error'] = reconstruction_error
p4['Isolation_Score'] = anomaly_scores_iso
p4['Lof_Score'] = anomaly_scores_lof

plot_reconstruction_error(reconstruction_error, threshold_pca)
plot_heatmap(p4, (10, 6))

print(f"Total anomalies detected by PCA: {np.sum(anomalies_pca)}")
print(f"Total anomalies detected by Isolation Forest: {np.
 ↪sum(anomalies_iforest)}")
print(f"Total anomalies detected by Local Outlier Factor: {np.
 ↪sum(anomalies_lof)}\n\n")

check_common_anomalies_3methods(anomalies_pca, anomalies_iforest, anomalies_lof)
```

```
Scaled data shape: (14996, 5)
PCA components: (14996, 4)
Number of anomalies: 12

Confusion matrix :

[[12329   114]
 [ 2374   179]]
```



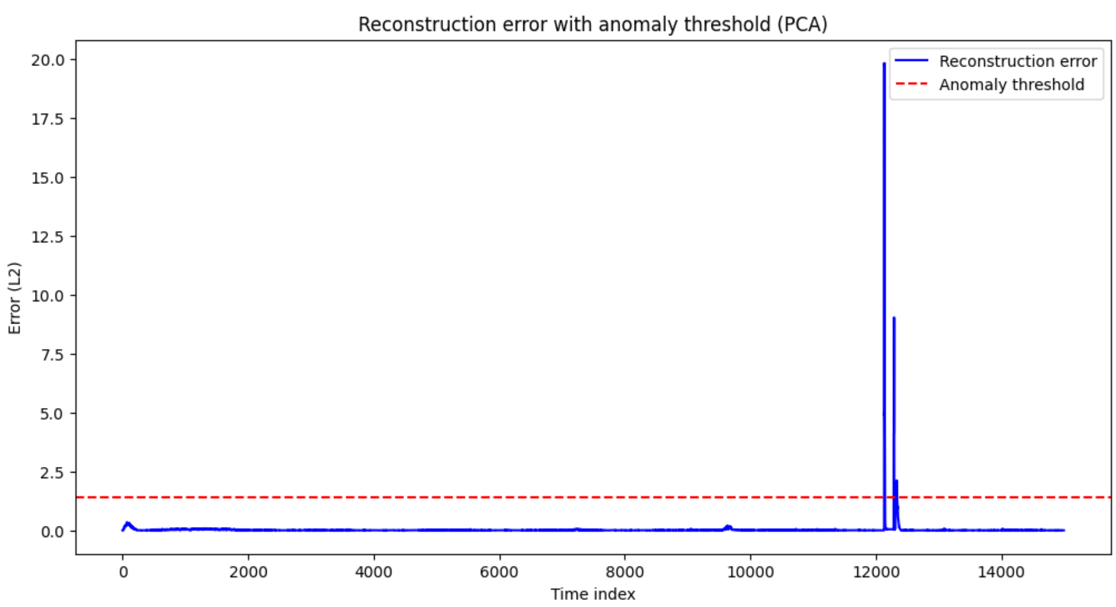

```
Total anomalies detected by PCA: 12
Total anomalies detected by Isolation Forest: 147
Total anomalies detected by Local Outlier Factor: 150


Total common anomalies detected by all methods: 0
Total common anomalies detected by PCA and Isolation Forest: 0
Total common anomalies detected by PCA and Local Outlier Factor: 12
Total common anomalies detected by Isolation Forest and Local Outlier Factor: 4
```

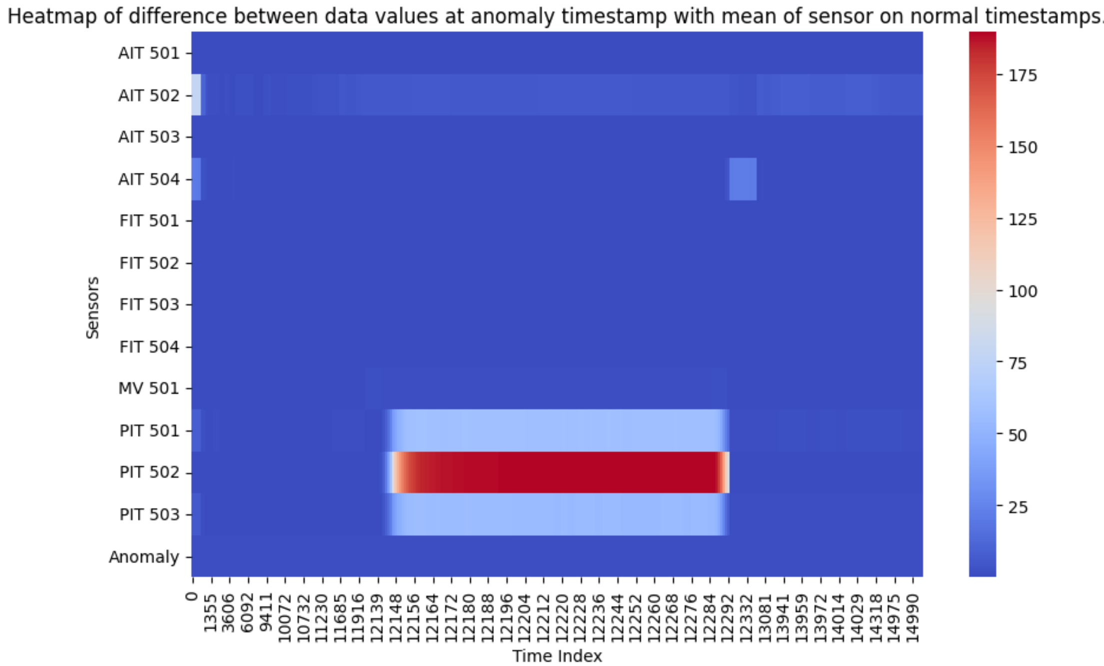
1.4.5 SSMP - P5

[54]:
```
p5 = prepare_dfs([p5_])
scaled_data = scale_data(p5)

pca_components, pca = pca_(scaled_data)
reconstruction_error, threshold_pca, anomalies_pca, indexes = metrics_pca(pca,
 ↪pca_components, scaled_data)

anomaly_scores_iso, anomalies_iforest = metrics_iso_forest(scaled_data)

anomaly_scores_lof, anomalies_lof = metrics_lof(scaled_data)

combined_anomalies_p5_or = anomalies_pca | anomalies_iforest | anomalies_lof
```

```python
print("\nConfusion matrix :\n")
print(confusion_matrix(labels, combined_anomalies_p5_or))

p5['Anomaly'] = combined_anomalies
p5['Reconstruction_Error'] = reconstruction_error
p5['Isolation_Score'] = anomaly_scores_iso
p5['Lof_Score'] = anomaly_scores_lof

plot_reconstruction_error(reconstruction_error, threshold_pca)
plot_heatmap(p5, (10, 6))

print(f"Total anomalies detected by PCA: {np.sum(anomalies_pca)}")
print(f"Total anomalies detected by Isolation Forest: {np.
 ↪sum(anomalies_iforest)}")
print(f"Total anomalies detected by Local Outlier Factor: {np.
 ↪sum(anomalies_lof)}\n\n")

check_common_anomalies_3methods(anomalies_pca, anomalies_iforest, anomalies_lof)
```

```
Scaled data shape: (14996, 12)
PCA components: (14996, 5)
Number of anomalies: 43

Confusion matrix :

[[12296   147]
 [ 2396   157]]
```





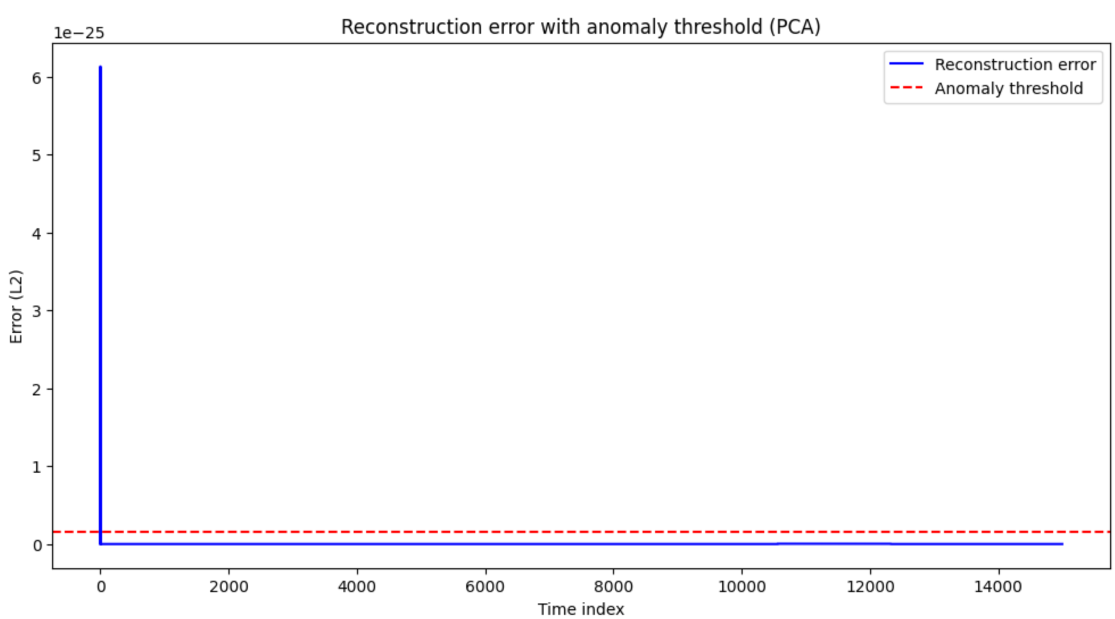

```
Total anomalies detected by PCA: 43
Total anomalies detected by Isolation Forest: 149
Total anomalies detected by Local Outlier Factor: 150


Total common anomalies detected by all methods: 5
Total common anomalies detected by PCA and Isolation Forest: 5
Total common anomalies detected by PCA and Local Outlier Factor: 20
Total common anomalies detected by Isolation Forest and Local Outlier Factor: 18
```

1.4.6 SSMP - P6

[55]:
```
p6 = prepare_dfs([p6_])
scaled_data = scale_data(p6)

pca_components, pca = pca_(scaled_data)
reconstruction_error, threshold_pca, anomalies_pca, indexes = metrics_pca(pca,
    pca_components, scaled_data)

anomaly_scores_iso, anomalies_iforest = metrics_iso_forest(scaled_data)

anomaly_scores_lof, anomalies_lof = metrics_lof(scaled_data)

combined_anomalies_p6_or = anomalies_pca | anomalies_iforest | anomalies_lof
```

```
print("\nConfusion matrix :\n")
print(confusion_matrix(labels, combined_anomalies_p6_or))

p6['Anomaly'] = combined_anomalies
p6['Reconstruction_Error'] = reconstruction_error
p6['Isolation_Score'] = anomaly_scores_iso
p6['Lof_Score'] = anomaly_scores_lof

plot_reconstruction_error(reconstruction_error, threshold_pca)
plot_heatmap(p6, (10, 6))

print(f"Total anomalies detected by PCA: {np.sum(anomalies_pca)}")
print(f"Total anomalies detected by Isolation Forest: {np.
↪sum(anomalies_iforest)}")
print(f"Total anomalies detected by Local Outlier Factor: {np.
↪sum(anomalies_lof)}\n\n")

check_common_anomalies_3methods(anomalies_pca, anomalies_iforest, anomalies_lof)
```

```
Scaled data shape: (14996, 3)
PCA components: (14996, 3)
Number of anomalies: 1

Confusion matrix :

[[12385    58]
 [ 2533    20]]
```

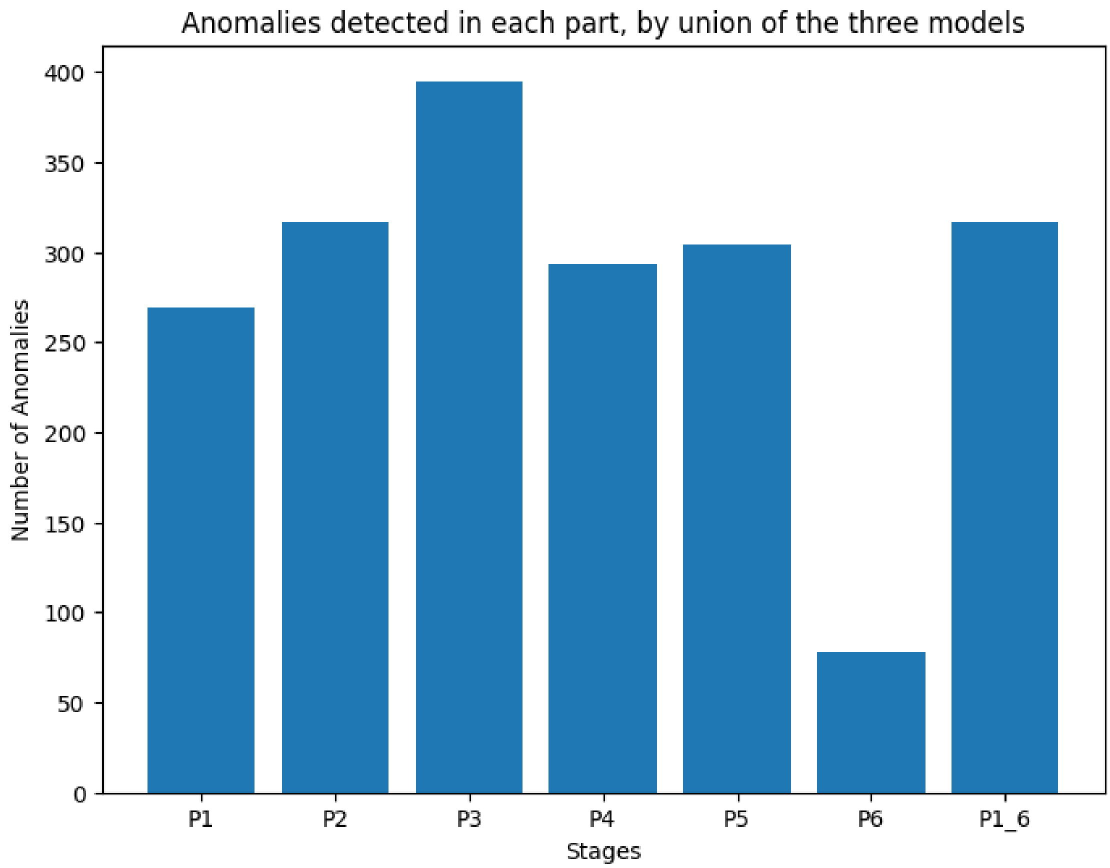



```
Total anomalies detected by PCA: 1
Total anomalies detected by Isolation Forest: 43
Total anomalies detected by Local Outlier Factor: 45


Total common anomalies detected by all methods: 0
Total common anomalies detected by PCA and Isolation Forest: 0
Total common anomalies detected by PCA and Local Outlier Factor: 0
Total common anomalies detected by Isolation Forest and Local Outlier Factor: 11
```

[56]:

```
anomalies_count = {
    "P1": np.sum(combined_anomalies_p1_or),
    "P2": np.sum(combined_anomalies_p2_or),
    "P3": np.sum(combined_anomalies_p3_or),
    "P4": np.sum(combined_anomalies_p4_or),
    "P5": np.sum(combined_anomalies_p5_or),
    "P6": np.sum(combined_anomalies_p6_or),
    "P1_6": np.sum(combined_anomalies)
}

plt.figure(figsize=(8, 6))
plt.bar(anomalies_count.keys(), anomalies_count.values())
plt.title('Anomalies detected in each part, by union of the three models')
plt.ylabel('Number of Anomalies')
plt.xlabel('Stages')
```

```
plt.show()
```



[57]:
```
combined_anomalies = (
    combined_anomalies_p1_or | combined_anomalies_p2_or |␣
 ↪combined_anomalies_p3_or | combined_anomalies_p4_or |␣
 ↪combined_anomalies_p5_or | combined_anomalies_p6_or
)
total_anomalies = combined_anomalies
```

[58]:
```
print("Total counted anomalies combined ", total_anomalies.sum())
```

```
Total counted anomalies combined  1415
```

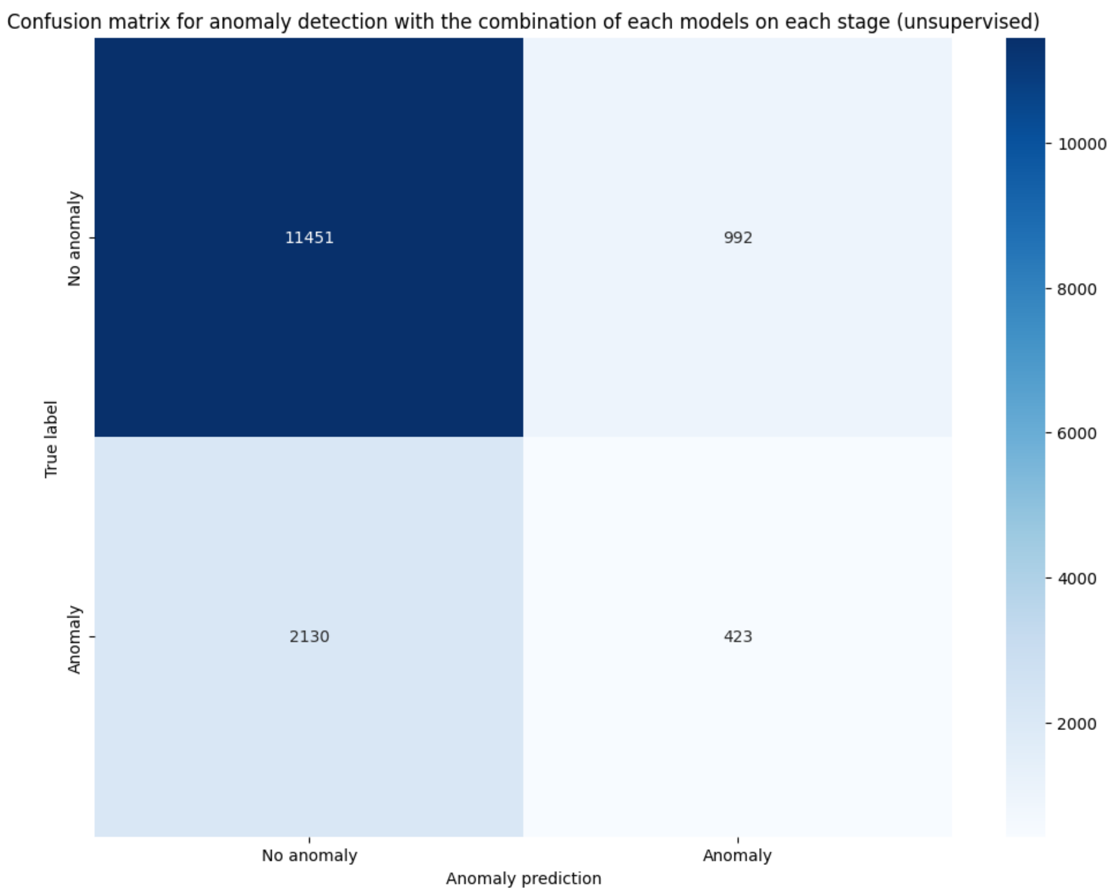
[59]:
```
print("Classification Report:")
print(classification_report(true_labels, total_anomalies))

cm_combined = confusion_matrix(true_labels, total_anomalies)
```

```
Classification Report:
              precision    recall  f1-score   support
```

```
              0       0.84      0.92      0.88     12443
              1       0.30      0.17      0.21      2553

       accuracy                           0.79     14996
      macro avg       0.57      0.54      0.55     14996
   weighted avg       0.75      0.79      0.77     14996
```

[60]:

```python
plt.figure(figsize=(12, 9))
sns.heatmap(cm_combined, annot=True, fmt='d', cmap='Blues', xticklabels=['No␣
 ↪anomaly', 'Anomaly'], yticklabels=['No anomaly', 'Anomaly'])
plt.title('Confusion matrix for anomaly detection with the combination of each␣
 ↪models on each stage (unsupervised)')
plt.xlabel('Anomaly prediction')
plt.ylabel('True label')
plt.show()
```



1.4.7 Attack types

The provided paper tells us this:

0. Total number of attacks: 36.
1. ***SSSP***: 26
2. ***SSMP***: 4
3. ***MSSP***: 2
4. ***MSMP***: 4

2 Deep learning approaches (Supervised)

Let's prepare the data to train a supervised model to predict anomalies.

- Features (X): all sensor data from P1-P6 stages
- Target (y): the binary labels for anomalies (0 for normal, 1 for anomaly)
- Train-test split: 70-30 ratio

[61]:
```
df_for_mlp = prepare_dfs([labels, p1_, p2_, p3_, p4_, p5_, p6_])
```

[62]:
```
df_for_mlp.head()
```

[62]:
```
   Label  FIT 101   LIT 101  MV 101  P1_STATE  P101 Status     AIT 201  \
0      0      0.0  729.8658       1         3            2  142.527557
1      0      0.0  729.4340       1         3            2  142.527557
2      0      0.0  729.1200       1         3            2  142.527557
3      0      0.0  728.6882       1         3            2  142.527557
4      0      0.0  727.7069       1         3            2  142.527557

    AIT 202     AIT 203   FIT 201  MV201  P203 Status  P205 Status   AIT 301  \
0  9.293002  198.077423  2.335437      2            2            2  8.522921
1  9.293002  198.385025  2.335437      2            2            2  8.522921
2  9.293002  198.436300  2.335437      2            2            2  8.522921
3  9.289157  198.667000  2.335437      2            2            2  8.522921
4  9.289157  198.897720  2.335437      2            2            2  8.522921

      AIT 302     AIT 303  DPIT 301   FIT 301     LIT 301  MV 301  MV 302  \
0  256.431274  143.158966  1.190857  0.000512  730.702100       1       1
1  256.431274  143.158966  1.190857  0.000512  730.902344       1       1
2  256.431274  143.158966  1.190857  0.000512  732.344300       1       1
3  256.431274  143.158966  1.190857  0.000512  732.704800       1       1
4  256.431274  143.158966  1.190857  0.000512  732.744800       1       1

   MV 303  MV 304  P3_STATE  P301 Status    AIT 402   FIT 401     LIT 401  \
0       1       1        99            1  87.951805  0.781740  1000.62805
1       1       1        99            1  87.823630  0.782380  1000.55115
2       1       1        99            1  87.798004  0.783021  1000.28200
3       1       1        99            1  87.695465  0.783021  1000.74341
4       1       1        99            1  87.618560  0.781228  1000.39734
```

```
   P401 Status  UV401   AIT 501     AIT 502     AIT 503    AIT 504   FIT 501  \
0            2      2  7.489618  147.398100  1016.27789  46.065113  0.781594
1            2      2  7.489618  147.398100  1016.27789  45.757500  0.782235
2            2      2  7.489618  147.398100  1016.27789  45.603690  0.782235
3            2      2  7.489618  147.167389  1016.27789  45.603690  0.783133
4            2      2  7.489618  147.090485  1016.27789  45.219173  0.783773

    FIT 502   FIT 503   FIT 504  MV 501     PIT 501   PIT 502     PIT 503  \
0  0.310362  0.623628  0.213432       2  167.601257  2.963509  119.921173
1  0.315102  0.623628  0.212984       2  167.601257  2.963509  119.921173
2  0.317023  0.623628  0.212984       2  167.601257  2.963509  119.921173
3  0.308057  0.623628  0.212792       2  167.601257  2.963509  119.921173
4  0.303446  0.623628  0.214009       2  167.601257  2.963509  119.921173

   FIT 601  LSH 601  P601 Status
0  0.00032        1            1
1  0.00032        1            1
2  0.00032        1            1
3  0.00032        1            1
4  0.00032        1            1
```

[63]:
```
X = df_for_mlp.drop(['Label'], axis=1)
y = df_for_mlp['Label'].astype(int)
```

[64]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,␣
 ↪random_state=42)
```

2.1 Base binary classification model

The first deep learning approach implemented a binary classifier to detect anomalies vs normal behavior. This architecture was chosen for its simplicity and effectiveness.

[65]:
```
model = Sequential([
    Dense(64, activation='relu', input_dim=X_train.shape[1]),
    Dense(32, activation='relu'),
    Dense(1, activation='sigmoid')
])

model.compile(optimizer='adam', loss='binary_crossentropy',␣
 ↪metrics=['accuracy'])

model.fit(X_train, y_train, epochs=30, batch_size=32, validation_split=0.2)
```

```
c:\Users\rokra\AppData\Local\Programs\Python\Python312\Lib\site-
packages\keras\src\layers\core\dense.py:87: UserWarning: Do not pass an
`input_shape`/`input_dim` argument to a layer. When using Sequential models,
```

```
prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)

Epoch 1/30
263/263            2s 2ms/step -
accuracy: 0.7051 - loss: 28.2794 - val_accuracy: 0.8262 - val_loss: 0.3954
Epoch 2/30
263/263            1s 2ms/step -
accuracy: 0.8708 - loss: 0.4454 - val_accuracy: 0.9267 - val_loss: 0.2315
Epoch 3/30
263/263            1s 2ms/step -
accuracy: 0.8832 - loss: 0.3291 - val_accuracy: 0.9038 - val_loss: 0.2398
Epoch 4/30
263/263            0s 1ms/step -
accuracy: 0.8845 - loss: 0.3186 - val_accuracy: 0.9005 - val_loss: 0.2517
Epoch 5/30
263/263            0s 1ms/step -
accuracy: 0.8917 - loss: 0.2956 - val_accuracy: 0.9071 - val_loss: 0.2064
Epoch 6/30
263/263            0s 1ms/step -
accuracy: 0.8822 - loss: 0.3769 - val_accuracy: 0.8871 - val_loss: 0.4596
Epoch 7/30
263/263            0s 1ms/step -
accuracy: 0.8750 - loss: 0.4713 - val_accuracy: 0.8895 - val_loss: 0.3169
Epoch 8/30
263/263            0s 1ms/step -
accuracy: 0.9043 - loss: 0.2627 - val_accuracy: 0.9152 - val_loss: 0.2588
Epoch 9/30
263/263            0s 1ms/step -
accuracy: 0.8960 - loss: 0.3097 - val_accuracy: 0.8795 - val_loss: 0.3410
Epoch 10/30
263/263            0s 1ms/step -
accuracy: 0.8942 - loss: 0.3391 - val_accuracy: 0.8571 - val_loss: 0.4435
Epoch 11/30
263/263            0s 2ms/step -
accuracy: 0.9090 - loss: 0.2471 - val_accuracy: 0.8881 - val_loss: 0.2221
Epoch 12/30
263/263            0s 1ms/step -
accuracy: 0.8755 - loss: 0.4334 - val_accuracy: 0.9167 - val_loss: 0.2864
Epoch 13/30
263/263            0s 1ms/step -
accuracy: 0.9034 - loss: 0.2588 - val_accuracy: 0.7686 - val_loss: 1.0070
Epoch 14/30
263/263            0s 1ms/step -
accuracy: 0.8961 - loss: 0.3168 - val_accuracy: 0.8933 - val_loss: 0.2094
Epoch 15/30
263/263            0s 1ms/step -
accuracy: 0.9002 - loss: 0.3286 - val_accuracy: 0.9310 - val_loss: 0.1418
```

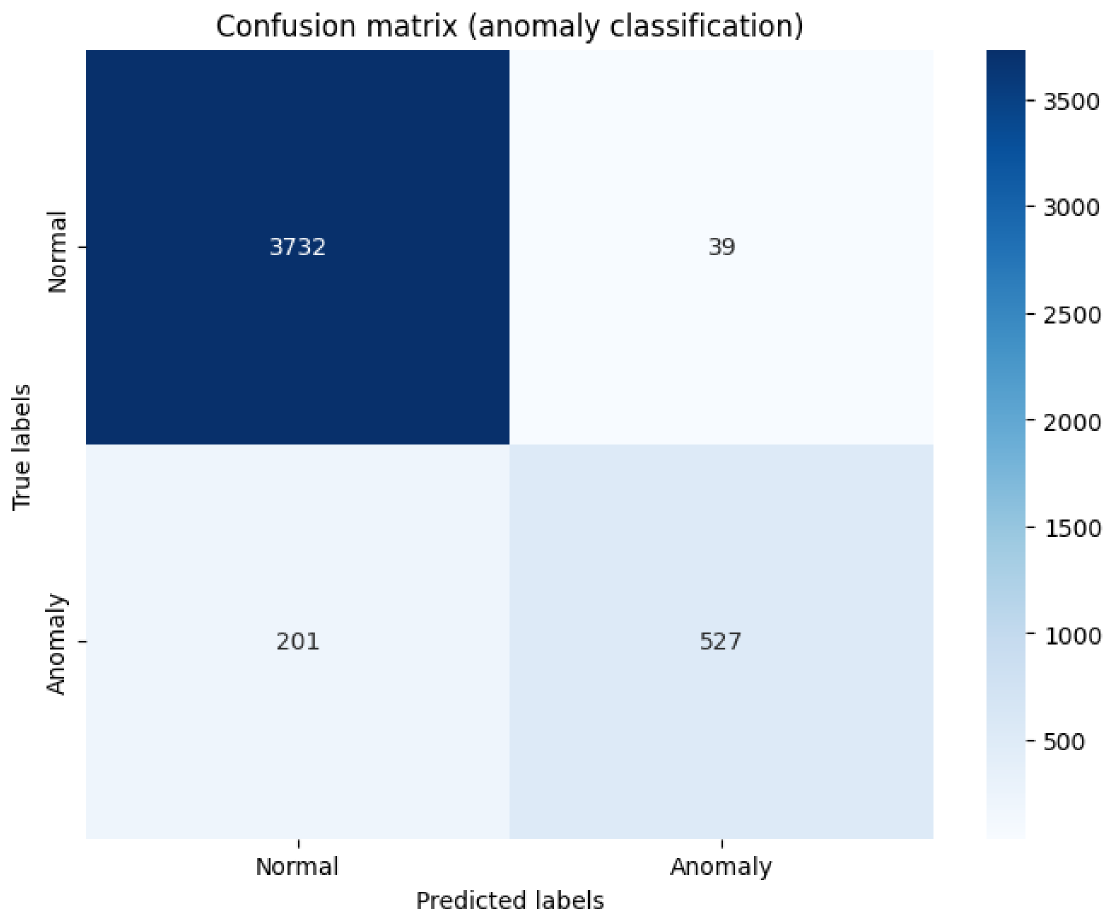
```
Epoch 16/30
263/263            0s 1ms/step -
accuracy: 0.9283 - loss: 0.1765 - val_accuracy: 0.8990 - val_loss: 0.1826
Epoch 17/30
263/263            0s 1ms/step -
accuracy: 0.9087 - loss: 0.2307 - val_accuracy: 0.9262 - val_loss: 0.1544
Epoch 18/30
263/263            0s 1ms/step -
accuracy: 0.9040 - loss: 0.3059 - val_accuracy: 0.8748 - val_loss: 0.2960
Epoch 19/30
263/263            0s 1ms/step -
accuracy: 0.9215 - loss: 0.1814 - val_accuracy: 0.9110 - val_loss: 0.1935
Epoch 20/30
263/263            0s 1ms/step -
accuracy: 0.9102 - loss: 0.2656 - val_accuracy: 0.9343 - val_loss: 0.1228
Epoch 21/30
263/263            0s 1ms/step -
accuracy: 0.9294 - loss: 0.1864 - val_accuracy: 0.9486 - val_loss: 0.1457
Epoch 22/30
263/263            0s 1ms/step -
accuracy: 0.9118 - loss: 0.2857 - val_accuracy: 0.8876 - val_loss: 0.3171
Epoch 23/30
263/263            0s 1ms/step -
accuracy: 0.9267 - loss: 0.2039 - val_accuracy: 0.9510 - val_loss: 0.1012
Epoch 24/30
263/263            0s 1ms/step -
accuracy: 0.9295 - loss: 0.1845 - val_accuracy: 0.9186 - val_loss: 0.1695
Epoch 25/30
263/263            0s 1ms/step -
accuracy: 0.9441 - loss: 0.1485 - val_accuracy: 0.9624 - val_loss: 0.0869
Epoch 26/30
263/263            0s 1ms/step -
accuracy: 0.9480 - loss: 0.1449 - val_accuracy: 0.9781 - val_loss: 0.0835
Epoch 27/30
263/263            0s 1ms/step -
accuracy: 0.9466 - loss: 0.1421 - val_accuracy: 0.8343 - val_loss: 0.7129
Epoch 28/30
263/263            0s 1ms/step -
accuracy: 0.9292 - loss: 0.2301 - val_accuracy: 0.9229 - val_loss: 0.1907
Epoch 29/30
263/263            0s 1ms/step -
accuracy: 0.9464 - loss: 0.1421 - val_accuracy: 0.9171 - val_loss: 0.1878
Epoch 30/30
263/263            0s 1ms/step -
accuracy: 0.9355 - loss: 0.1808 - val_accuracy: 0.9500 - val_loss: 0.1258
```

```
[65]: <keras.src.callbacks.history.History at 0x212ee322840>
```

[66]:
```python
y_pred = model.predict(X_test)
y_pred_labels = (y_pred > 0.5).astype(int)

print(classification_report(y_test, y_pred_labels))
```

```
141/141              0s 824us/step
              precision    recall  f1-score   support

           0       0.95      0.99      0.97      3771
           1       0.93      0.72      0.81       728

    accuracy                           0.95      4499
   macro avg       0.94      0.86      0.89      4499
weighted avg       0.95      0.95      0.94      4499
```

[67]:
```python
cm = confusion_matrix(y_test, y_pred_labels)

plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['Normal',␣
↪'Anomaly'], yticklabels=['Normal', 'Anomaly'])
plt.title('Confusion matrix (anomaly classification)')
plt.xlabel('Predicted labels')
plt.ylabel('True labels')
plt.show()
```



Better performance than the unsupervised methods discussed earlier.

The results showed strong performance beacause we have: - High precision and recall for normal cases (class 0) - Good performance on anomaly detection (class 1)

Despite those good metrics, there is much more false negative than false positive which can be problematic in such cybersecurity problems.

2.2 Attacks types without labels

Now let's prepare the data and create a model to classify the data among five categories: - Benign - Spoofing - Switch_ON - Switch_close - Switch_off

The training data is composed of: - Features (X): all sensor data from P1-P6 stages - Target (y): the class of the attack - Train-test split: 70-30 ratio

[68]:
```
df_for_mlp_types = pd.concat([attacks, prepare_dfs([p1_, p2_, p3_, p4_, p5_,␣
↪p6_])], axis=1).copy()
```

[69]:
```
df_for_mlp_types['Attack'].unique()
```

[69]:
```
array(['benign', 'Spoofing', 'Switch_ON', 'Switch_close', 'Switch_off'],
      dtype=object)
```

[70]:
```
df_for_mlp_types.head()
```

[70]:
```
   Attack  FIT 101   LIT 101  MV 101  P1_STATE  P101 Status     AIT 201  \
0  benign      0.0  729.8658       1         3            2  142.527557
1  benign      0.0  729.4340       1         3            2  142.527557
2  benign      0.0  729.1200       1         3            2  142.527557
3  benign      0.0  728.6882       1         3            2  142.527557
4  benign      0.0  727.7069       1         3            2  142.527557

    AIT 202     AIT 203   FIT 201  MV201  P203 Status  P205 Status   AIT 301  \
0  9.293002  198.077423  2.335437      2            2            2  8.522921
1  9.293002  198.385025  2.335437      2            2            2  8.522921
2  9.293002  198.436300  2.335437      2            2            2  8.522921
3  9.289157  198.667000  2.335437      2            2            2  8.522921
4  9.289157  198.897720  2.335437      2            2            2  8.522921

      AIT 302     AIT 303  DPIT 301   FIT 301     LIT 301  MV 301  MV 302  \
0  256.431274  143.158966  1.190857  0.000512  730.702100       1       1
1  256.431274  143.158966  1.190857  0.000512  730.902344       1       1
2  256.431274  143.158966  1.190857  0.000512  732.344300       1       1
3  256.431274  143.158966  1.190857  0.000512  732.704800       1       1
4  256.431274  143.158966  1.190857  0.000512  732.744800       1       1

   MV 303  MV 304  P3_STATE  P301 Status    AIT 402   FIT 401     LIT 401  \
0       1       1        99            1  87.951805  0.781740  1000.62805
1       1       1        99            1  87.823630  0.782380  1000.55115
2       1       1        99            1  87.798004  0.783021  1000.28200
3       1       1        99            1  87.695465  0.783021  1000.74341
4       1       1        99            1  87.618560  0.781228  1000.39734

   P401 Status  UV401   AIT 501     AIT 502     AIT 503    AIT 504   FIT 501  \
0            2      2  7.489618  147.398100  1016.27789  46.065113  0.781594
1            2      2  7.489618  147.398100  1016.27789  45.757500  0.782235
2            2      2  7.489618  147.398100  1016.27789  45.603690  0.782235
3            2      2  7.489618  147.167389  1016.27789  45.603690  0.783133
4            2      2  7.489618  147.090485  1016.27789  45.219173  0.783773

    FIT 502   FIT 503   FIT 504  MV 501     PIT 501   PIT 502     PIT 503  \
0  0.310362  0.623628  0.213432       2  167.601257  2.963509  119.921173
1  0.315102  0.623628  0.212984       2  167.601257  2.963509  119.921173
2  0.317023  0.623628  0.212984       2  167.601257  2.963509  119.921173
3  0.308057  0.623628  0.212792       2  167.601257  2.963509  119.921173
4  0.303446  0.623628  0.214009       2  167.601257  2.963509  119.921173
```

```
   FIT 601  LSH 601  P601 Status
0  0.00032        1            1
1  0.00032        1            1
2  0.00032        1            1
3  0.00032        1            1
4  0.00032        1            1
```

[71]:
```
X_types = df_for_mlp_types.drop(['Attack'], axis=1)
y_types = df_for_mlp_types['Attack']
# caca
```

[72]:
```
y_types_encoded = pd.get_dummies(y_types).values

X_train_types, X_test_types, y_train_types, y_test_types =␣
↪train_test_split(X_types, y_types_encoded, test_size=0.3, random_state=42)
```

The architecture is deeper than the binary classifier to handle the more complex multi-class task.

[73]:
```
model_types = Sequential([
    Dense(64, activation='relu', input_dim=X_train_types.shape[1]),
    Dense(64, activation='relu'),
    Dense(32, activation='relu'),
    Dense(5, activation='softmax')
])

model_types.compile(optimizer='adam', loss='categorical_crossentropy',␣
↪metrics=['accuracy'])

model_types.fit(X_train_types, y_train_types, epochs=30, batch_size=32,␣
↪validation_split=0.2)
```

```
Epoch 1/30
```

```
c:\Users\rokra\AppData\Local\Programs\Python\Python312\Lib\site-
packages\keras\src\layers\core\dense.py:87: UserWarning: Do not pass an
`input_shape`/`input_dim` argument to a layer. When using Sequential models,
prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

```
263/263              2s 2ms/step -
accuracy: 0.7298 - loss: 16.6702 - val_accuracy: 0.9205 - val_loss: 0.2707
Epoch 2/30
263/263              0s 1ms/step -
accuracy: 0.8908 - loss: 0.3392 - val_accuracy: 0.8948 - val_loss: 0.2339
Epoch 3/30
263/263              0s 1ms/step -
accuracy: 0.9063 - loss: 0.2897 - val_accuracy: 0.9119 - val_loss: 0.2393
Epoch 4/30
263/263              0s 1ms/step -
```

```
accuracy: 0.8807 - loss: 0.4152 - val_accuracy: 0.9448 - val_loss: 0.1470
Epoch 5/30
263/263            0s 1ms/step -
accuracy: 0.9340 - loss: 0.1809 - val_accuracy: 0.9495 - val_loss: 0.1221
Epoch 6/30
263/263            0s 1ms/step -
accuracy: 0.9318 - loss: 0.1765 - val_accuracy: 0.9586 - val_loss: 0.1134
Epoch 7/30
263/263            0s 1ms/step -
accuracy: 0.9285 - loss: 0.2143 - val_accuracy: 0.9652 - val_loss: 0.1050
Epoch 8/30
263/263            0s 1ms/step -
accuracy: 0.9336 - loss: 0.1938 - val_accuracy: 0.9610 - val_loss: 0.1195
Epoch 9/30
263/263            0s 1ms/step -
accuracy: 0.9352 - loss: 0.1825 - val_accuracy: 0.9433 - val_loss: 0.1551
Epoch 10/30
263/263            0s 1ms/step -
accuracy: 0.9409 - loss: 0.1504 - val_accuracy: 0.9438 - val_loss: 0.1864
Epoch 11/30
263/263            0s 1ms/step -
accuracy: 0.9475 - loss: 0.1418 - val_accuracy: 0.9633 - val_loss: 0.1010
Epoch 12/30
263/263            0s 1ms/step -
accuracy: 0.9504 - loss: 0.1346 - val_accuracy: 0.9290 - val_loss: 0.1458
Epoch 13/30
263/263            0s 1ms/step -
accuracy: 0.9282 - loss: 0.2240 - val_accuracy: 0.9490 - val_loss: 0.1049
Epoch 14/30
263/263            0s 2ms/step -
accuracy: 0.9087 - loss: 0.2933 - val_accuracy: 0.9505 - val_loss: 0.1044
Epoch 15/30
263/263            0s 1ms/step -
accuracy: 0.9450 - loss: 0.1307 - val_accuracy: 0.9505 - val_loss: 0.1045
Epoch 16/30
263/263            0s 2ms/step -
accuracy: 0.9567 - loss: 0.1141 - val_accuracy: 0.9638 - val_loss: 0.1101
Epoch 17/30
263/263            0s 2ms/step -
accuracy: 0.9673 - loss: 0.0857 - val_accuracy: 0.9743 - val_loss: 0.0721
Epoch 18/30
263/263            0s 1ms/step -
accuracy: 0.9620 - loss: 0.0985 - val_accuracy: 0.9605 - val_loss: 0.0785
Epoch 19/30
263/263            0s 1ms/step -
accuracy: 0.9580 - loss: 0.1099 - val_accuracy: 0.9624 - val_loss: 0.0944
Epoch 20/30
263/263            0s 1ms/step -
```

```
accuracy: 0.9694 - loss: 0.0833 - val_accuracy: 0.9686 - val_loss: 0.0805
Epoch 21/30
263/263              0s 1ms/step -
accuracy: 0.9620 - loss: 0.0963 - val_accuracy: 0.9729 - val_loss: 0.0887
Epoch 22/30
263/263              0s 2ms/step -
accuracy: 0.9745 - loss: 0.0692 - val_accuracy: 0.9833 - val_loss: 0.0648
Epoch 23/30
263/263              0s 1ms/step -
accuracy: 0.9513 - loss: 0.1365 - val_accuracy: 0.9667 - val_loss: 0.0853
Epoch 24/30
263/263              0s 1ms/step -
accuracy: 0.9643 - loss: 0.0886 - val_accuracy: 0.9414 - val_loss: 0.1317
Epoch 25/30
263/263              0s 1ms/step -
accuracy: 0.9684 - loss: 0.0789 - val_accuracy: 0.9738 - val_loss: 0.0734
Epoch 26/30
263/263              0s 1ms/step -
accuracy: 0.9725 - loss: 0.0688 - val_accuracy: 0.9810 - val_loss: 0.0587
Epoch 27/30
263/263              0s 1ms/step -
accuracy: 0.9771 - loss: 0.0643 - val_accuracy: 0.9667 - val_loss: 0.0881
Epoch 28/30
263/263              0s 1ms/step -
accuracy: 0.9756 - loss: 0.0757 - val_accuracy: 0.9819 - val_loss: 0.0430
Epoch 29/30
263/263              0s 1ms/step -
accuracy: 0.9722 - loss: 0.0692 - val_accuracy: 0.9848 - val_loss: 0.0569
Epoch 30/30
263/263              0s 1ms/step -
accuracy: 0.9726 - loss: 0.0703 - val_accuracy: 0.9762 - val_loss: 0.0564
```

[73]:
```
<keras.src.callbacks.history.History at 0x212ecdb2ff0>
```

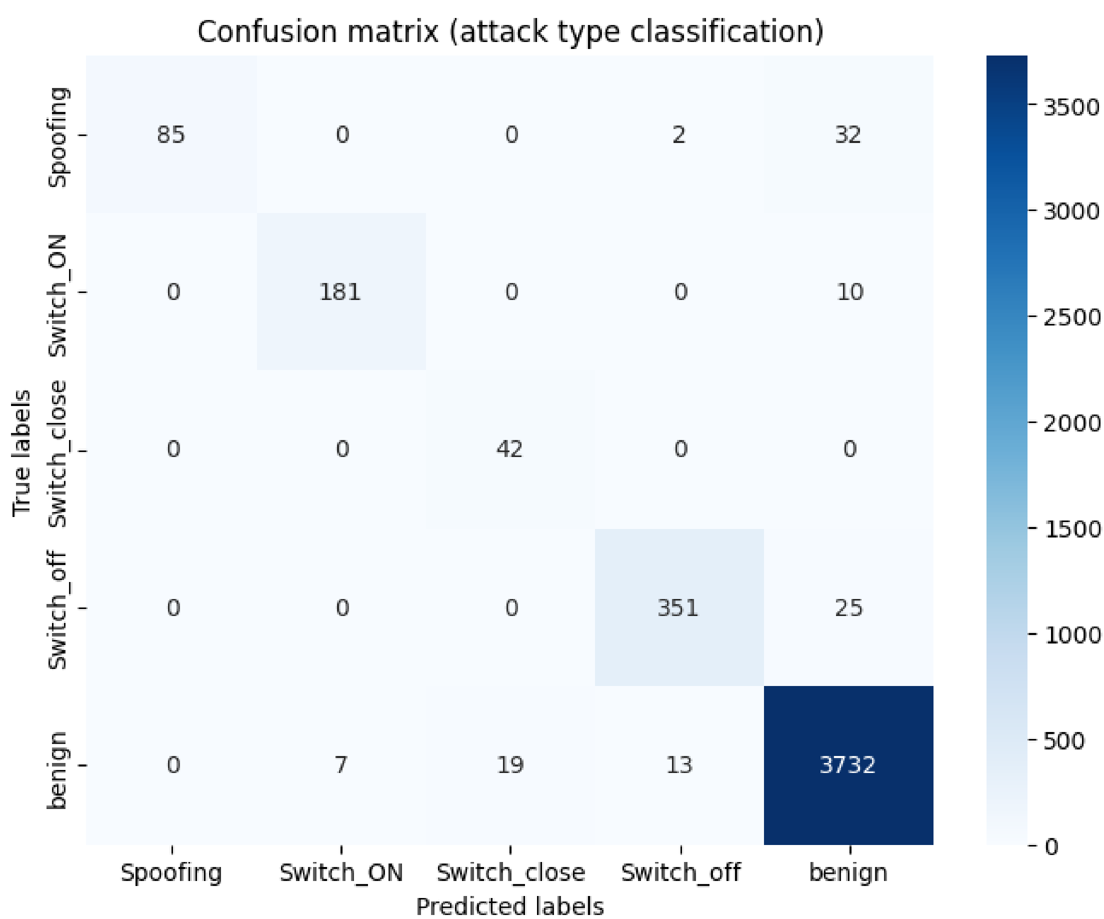
[74]:
```python
y_pred_types = model_types.predict(X_test_types)
y_pred_labels_types = np.argmax(y_pred_types, axis=1)
y_test_labels_types = np.argmax(y_test_types, axis=1)

print(classification_report(y_test_labels_types, y_pred_labels_types,␣
 ↪target_names=['Spoofing', 'Switch_ON', 'Switch_close', 'Switch_off',␣
 ↪'benign']))
```

```
141/141              0s 977us/step
              precision    recall  f1-score   support

    Spoofing       1.00      0.71      0.83       119
   Switch_ON       0.96      0.95      0.96       191
Switch_close       0.69      1.00      0.82        42
```

```
   Switch_off       0.96      0.93      0.95       376
       benign       0.98      0.99      0.99      3771

     accuracy                           0.98      4499
    macro avg       0.92      0.92      0.91      4499
 weighted avg       0.98      0.98      0.98      4499
```

[75]:

```python
conf_matrix = confusion_matrix(y_test_labels_types, y_pred_labels_types)

plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues',␣
 ↪xticklabels=['Spoofing', 'Switch_ON', 'Switch_close', 'Switch_off',␣
 ↪'benign'],
            yticklabels=['Spoofing', 'Switch_ON', 'Switch_close', 'Switch_off',␣
 ↪'benign'])
plt.title('Confusion matrix (attack type classification)')
plt.xlabel('Predicted labels')
plt.ylabel('True labels')
plt.show()
```



We can see that our model is very good a detecting the attack type (here on the test dataset) with a slightly more errors on the `Switch of` attack predictions

2.2.1 Attack types with labels

The same architecture was used but included the binary anomaly label information as an additional feature. This provided slightly better performance due to the additional contextual information.

[76]:
```
df_for_mlp_types_labels = pd.concat([attacks, labels, prepare_dfs([p1_, p2_,␣
↪p3_, p4_, p5_, p6_])], axis=1).copy()
```

[77]:
```
X_types_labels = df_for_mlp_types_labels.drop(['Attack'], axis=1)
y_types_labels = df_for_mlp_types_labels['Attack']
```

[78]:
```
y_types_encoded_labels = pd.get_dummies(y_types_labels).values

X_train_types_labels, X_test_types_labels, y_train_types_labels,␣
↪y_test_types_labels = train_test_split(X_types_labels,␣
↪y_types_encoded_labels, test_size=0.3, random_state=42)
```

[79]:
```
model_types_labels = Sequential([
    Dense(64, activation='relu', input_dim=X_train_types_labels.shape[1]),
    Dense(64, activation='relu'),
    Dense(32, activation='relu'),
    Dense(5, activation='softmax')
])

model_types_labels.compile(optimizer='adam', loss='categorical_crossentropy',␣
↪metrics=['accuracy'])

model_types_labels.fit(X_train_types_labels, y_train_types_labels, epochs=30,␣
↪batch_size=32, validation_split=0.2)
```

```
Epoch 1/30
```

```
c:\Users\rokra\AppData\Local\Programs\Python\Python312\Lib\site-
packages\keras\src\layers\core\dense.py:87: UserWarning: Do not pass an
`input_shape`/`input_dim` argument to a layer. When using Sequential models,
prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

```
263/263              3s 3ms/step -
accuracy: 0.7607 - loss: 10.8224 - val_accuracy: 0.6990 - val_loss: 0.6636
Epoch 2/30
263/263              1s 3ms/step -
accuracy: 0.8633 - loss: 0.5205 - val_accuracy: 0.8881 - val_loss: 0.3864
Epoch 3/30
```

```
263/263            1s 2ms/step -
accuracy: 0.8997 - loss: 0.3367 - val_accuracy: 0.9062 - val_loss: 0.4006
Epoch 4/30
263/263            1s 2ms/step -
accuracy: 0.9233 - loss: 0.3505 - val_accuracy: 0.9414 - val_loss: 0.1368
Epoch 5/30
263/263            1s 2ms/step -
accuracy: 0.9307 - loss: 0.2391 - val_accuracy: 0.9210 - val_loss: 0.5193
Epoch 6/30
263/263            0s 1ms/step -
accuracy: 0.9028 - loss: 0.4309 - val_accuracy: 0.8090 - val_loss: 0.5867
Epoch 7/30
263/263            0s 1ms/step -
accuracy: 0.9292 - loss: 0.2717 - val_accuracy: 0.9457 - val_loss: 0.1786
Epoch 8/30
263/263            1s 2ms/step -
accuracy: 0.9539 - loss: 0.1403 - val_accuracy: 0.9795 - val_loss: 0.0686
Epoch 9/30
263/263            0s 1ms/step -
accuracy: 0.9671 - loss: 0.0997 - val_accuracy: 0.9610 - val_loss: 0.1190
Epoch 10/30
263/263            0s 1ms/step -
accuracy: 0.9582 - loss: 0.1385 - val_accuracy: 0.9376 - val_loss: 0.2712
Epoch 11/30
263/263            0s 2ms/step -
accuracy: 0.9771 - loss: 0.0713 - val_accuracy: 0.9743 - val_loss: 0.1010
Epoch 12/30
263/263            0s 2ms/step -
accuracy: 0.9795 - loss: 0.0626 - val_accuracy: 0.9962 - val_loss: 0.0279
Epoch 13/30
263/263            0s 1ms/step -
accuracy: 0.9654 - loss: 0.1156 - val_accuracy: 0.9781 - val_loss: 0.0781
Epoch 14/30
263/263            1s 2ms/step -
accuracy: 0.9748 - loss: 0.0862 - val_accuracy: 0.9538 - val_loss: 0.2051
Epoch 15/30
263/263            0s 1ms/step -
accuracy: 0.9638 - loss: 0.1240 - val_accuracy: 0.9538 - val_loss: 0.1182
Epoch 16/30
263/263            0s 1ms/step -
accuracy: 0.9831 - loss: 0.0532 - val_accuracy: 0.9971 - val_loss: 0.0269
Epoch 17/30
263/263            0s 2ms/step -
accuracy: 0.9717 - loss: 0.0912 - val_accuracy: 0.9814 - val_loss: 0.0604
Epoch 18/30
263/263            1s 2ms/step -
accuracy: 0.9788 - loss: 0.0619 - val_accuracy: 0.9776 - val_loss: 0.0733
Epoch 19/30
```

```
263/263            1s 2ms/step -
accuracy: 0.9894 - loss: 0.0343 - val_accuracy: 0.9981 - val_loss: 0.0204
Epoch 20/30
263/263            1s 3ms/step -
accuracy: 0.9619 - loss: 0.1537 - val_accuracy: 0.9824 - val_loss: 0.0781
Epoch 21/30
263/263            1s 2ms/step -
accuracy: 0.9853 - loss: 0.0396 - val_accuracy: 0.9990 - val_loss: 0.0135
Epoch 22/30
263/263            0s 2ms/step -
accuracy: 0.9814 - loss: 0.0662 - val_accuracy: 0.9890 - val_loss: 0.0328
Epoch 23/30
263/263            0s 1ms/step -
accuracy: 0.9917 - loss: 0.0281 - val_accuracy: 0.9976 - val_loss: 0.0111
Epoch 24/30
263/263            0s 1ms/step -
accuracy: 0.9822 - loss: 0.0541 - val_accuracy: 0.9710 - val_loss: 0.0799
Epoch 25/30
263/263            1s 2ms/step -
accuracy: 0.9811 - loss: 0.0514 - val_accuracy: 0.9895 - val_loss: 0.0669
Epoch 26/30
263/263            0s 1ms/step -
accuracy: 0.9829 - loss: 0.0531 - val_accuracy: 0.9805 - val_loss: 0.0552
Epoch 27/30
263/263            0s 1ms/step -
accuracy: 0.9694 - loss: 0.1094 - val_accuracy: 0.9981 - val_loss: 0.0134
Epoch 28/30
263/263            0s 2ms/step -
accuracy: 0.9941 - loss: 0.0204 - val_accuracy: 0.9929 - val_loss: 0.0354
Epoch 29/30
263/263            1s 2ms/step -
accuracy: 0.9890 - loss: 0.0443 - val_accuracy: 0.9781 - val_loss: 0.0460
Epoch 30/30
263/263            0s 2ms/step -
accuracy: 0.9935 - loss: 0.0207 - val_accuracy: 0.9814 - val_loss: 0.0299
```

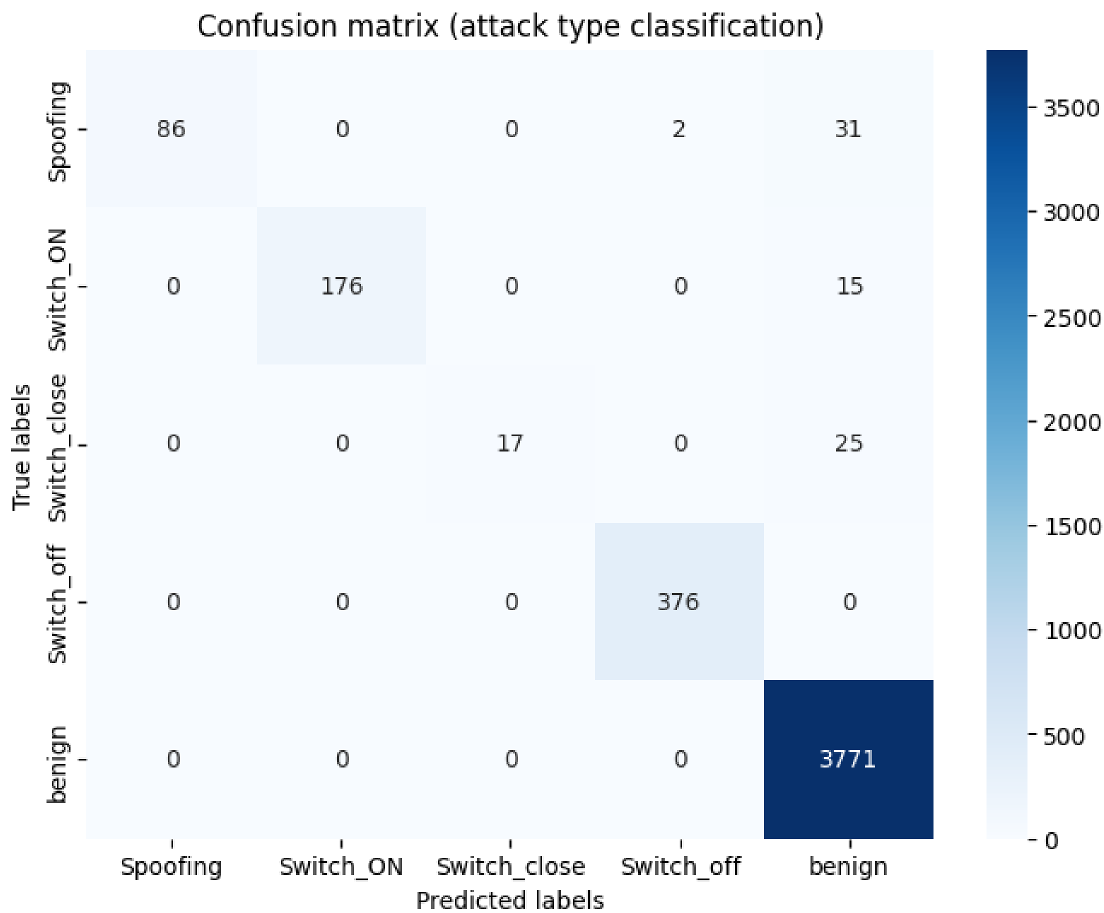
[79]:
```
<keras.src.callbacks.history.History at 0x212ef7fa9f0>
```

[80]:
```
y_pred_types_labels = model_types_labels.predict(X_test_types_labels)
y_pred_labels_types_labels = np.argmax(y_pred_types_labels, axis=1)
y_test_labels_types_labels = np.argmax(y_test_types_labels, axis=1)

print(classification_report(y_test_labels_types, y_pred_labels_types,␣
 ↪target_names=['Spoofing', 'Switch_ON', 'Switch_close', 'Switch_off',␣
 ↪'benign']))
```

```
141/141            0s 2ms/step
              precision    recall  f1-score   support
```

```
    Spoofing       1.00      0.71      0.83       119
   Switch_ON       0.96      0.95      0.96       191
Switch_close       0.69      1.00      0.82        42
  Switch_off       0.96      0.93      0.95       376
      benign       0.98      0.99      0.99      3771

    accuracy                           0.98      4499
   macro avg       0.92      0.92      0.91      4499
weighted avg       0.98      0.98      0.98      4499
```

[81]:
```
conf_matrix = confusion_matrix(y_test_labels_types_labels,␣
 ↪y_pred_labels_types_labels)

plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues',␣
 ↪xticklabels=['Spoofing', 'Switch_ON', 'Switch_close', 'Switch_off',␣
 ↪'benign'],
            yticklabels=['Spoofing', 'Switch_ON', 'Switch_close', 'Switch_off',␣
 ↪'benign'])
plt.title('Confusion matrix (attack type classification)')
plt.xlabel('Predicted labels')
plt.ylabel('True labels')
plt.show()
```



Now the model is close to the best possible accuracy.

2.3 Without Spoofing extrapolate ?

We decided to implement a new approach to test the model's ability to detect spoofing attacks to see if our model is able to generalize on unseen attacks.

1. Spoofing attacks are excluded from the training data
2. Testing include all attack types
3. We keep the same classification architecture as before

[82]:
```
X = df_for_mlp_types_labels.drop(['Attack', 'Label'], axis=1)
y = df_for_mlp_types_labels['Label'].astype(int)

X_train, X_test, y_train, y_test, train_attacks, test_attacks =␣
↪train_test_split(
    X, y, df_for_mlp_types_labels['Attack'], test_size=0.3, random_state=42
)
```

We remove "Spoofing" from the training set

[83]:
```
train_df = pd.concat([X_train, train_attacks, y_train], axis=1)
train_df = train_df[train_df['Attack'] != 'Spoofing']

X_train_filtered = train_df.drop(['Attack', 'Label'], axis=1)  # features␣
↪without Attack and Label
y_train_filtered = train_df['Label'].astype(int)
```

[84]:
```
model_filtered = Sequential([
    Dense(64, activation='relu', input_dim=X_train_filtered.shape[1]),
    Dense(32, activation='relu'),
    Dense(1, activation='sigmoid')
])

model_filtered.compile(optimizer='adam', loss='binary_crossentropy',␣
↪metrics=['accuracy'])

# Train the model
model_filtered.fit(X_train_filtered, y_train_filtered, epochs=30,␣
↪batch_size=32, validation_split=0.2)
```

```
Epoch 1/30

c:\Users\rokra\AppData\Local\Programs\Python\Python312\Lib\site-
packages\keras\src\layers\core\dense.py:87: UserWarning: Do not pass an
`input_shape`/`input_dim` argument to a layer. When using Sequential models,
prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)

257/257             3s 2ms/step -
accuracy: 0.7619 - loss: 22.0663 - val_accuracy: 0.9117 - val_loss: 0.3650
Epoch 2/30
257/257             0s 1ms/step -
accuracy: 0.8923 - loss: 0.4233 - val_accuracy: 0.9234 - val_loss: 0.1799
Epoch 3/30
257/257             0s 2ms/step -
accuracy: 0.9081 - loss: 0.2934 - val_accuracy: 0.9137 - val_loss: 0.2083
Epoch 4/30
257/257             1s 2ms/step -
accuracy: 0.9192 - loss: 0.2821 - val_accuracy: 0.9459 - val_loss: 0.1921
Epoch 5/30
257/257             0s 2ms/step -
accuracy: 0.9164 - loss: 0.2835 - val_accuracy: 0.8771 - val_loss: 0.6341
Epoch 6/30
257/257             1s 3ms/step -
accuracy: 0.9210 - loss: 0.2622 - val_accuracy: 0.9424 - val_loss: 0.1851
Epoch 7/30
257/257             1s 2ms/step -
accuracy: 0.9481 - loss: 0.1548 - val_accuracy: 0.9405 - val_loss: 0.1563
```

```
Epoch 8/30
257/257            1s 2ms/step -
accuracy: 0.9259 - loss: 0.2751 - val_accuracy: 0.9229 - val_loss: 0.2347
Epoch 9/30
257/257            1s 2ms/step -
accuracy: 0.9110 - loss: 0.3925 - val_accuracy: 0.9561 - val_loss: 0.1245
Epoch 10/30
257/257            0s 1ms/step -
accuracy: 0.9521 - loss: 0.1417 - val_accuracy: 0.8532 - val_loss: 0.5039
Epoch 11/30
257/257            0s 1ms/step -
accuracy: 0.9219 - loss: 0.2909 - val_accuracy: 0.9307 - val_loss: 0.2206
Epoch 12/30
257/257            1s 3ms/step -
accuracy: 0.9581 - loss: 0.1465 - val_accuracy: 0.9376 - val_loss: 0.1796
Epoch 13/30
257/257            1s 3ms/step -
accuracy: 0.9589 - loss: 0.1368 - val_accuracy: 0.9688 - val_loss: 0.1078
Epoch 14/30
257/257            1s 2ms/step -
accuracy: 0.9522 - loss: 0.1487 - val_accuracy: 0.8459 - val_loss: 0.6835
Epoch 15/30
257/257            1s 2ms/step -
accuracy: 0.9321 - loss: 0.3173 - val_accuracy: 0.9380 - val_loss: 0.1672
Epoch 16/30
257/257            1s 2ms/step -
accuracy: 0.9656 - loss: 0.0966 - val_accuracy: 0.9454 - val_loss: 0.2257
Epoch 17/30
257/257            0s 2ms/step -
accuracy: 0.9528 - loss: 0.1664 - val_accuracy: 0.9624 - val_loss: 0.0956
Epoch 18/30
257/257            0s 1ms/step -
accuracy: 0.9384 - loss: 0.2567 - val_accuracy: 0.9346 - val_loss: 0.2156
Epoch 19/30
257/257            0s 2ms/step -
accuracy: 0.9492 - loss: 0.2018 - val_accuracy: 0.9737 - val_loss: 0.0815
Epoch 20/30
257/257            1s 2ms/step -
accuracy: 0.9733 - loss: 0.0831 - val_accuracy: 0.8995 - val_loss: 0.5926
Epoch 21/30
257/257            1s 2ms/step -
accuracy: 0.9594 - loss: 0.1746 - val_accuracy: 0.9873 - val_loss: 0.0413
Epoch 22/30
257/257            1s 3ms/step -
accuracy: 0.9548 - loss: 0.1948 - val_accuracy: 0.8780 - val_loss: 0.4413
Epoch 23/30
257/257            1s 3ms/step -
accuracy: 0.9269 - loss: 0.3323 - val_accuracy: 0.9800 - val_loss: 0.0801
```

```
Epoch 24/30
257/257            0s 1ms/step -
accuracy: 0.9759 - loss: 0.0837 - val_accuracy: 0.9444 - val_loss: 0.2489
Epoch 25/30
257/257            0s 1ms/step -
accuracy: 0.9701 - loss: 0.1420 - val_accuracy: 0.9561 - val_loss: 0.1299
Epoch 26/30
257/257            0s 1ms/step -
accuracy: 0.9654 - loss: 0.1102 - val_accuracy: 0.9844 - val_loss: 0.0464
Epoch 27/30
257/257            0s 1ms/step -
accuracy: 0.9741 - loss: 0.0923 - val_accuracy: 0.9829 - val_loss: 0.0524
Epoch 28/30
257/257            0s 1ms/step -
accuracy: 0.9771 - loss: 0.0765 - val_accuracy: 0.9766 - val_loss: 0.0716
Epoch 29/30
257/257            0s 1ms/step -
accuracy: 0.9779 - loss: 0.0709 - val_accuracy: 0.9585 - val_loss: 0.0901
Epoch 30/30
257/257            0s 2ms/step -
accuracy: 0.9731 - loss: 0.0857 - val_accuracy: 0.9722 - val_loss: 0.0830
```

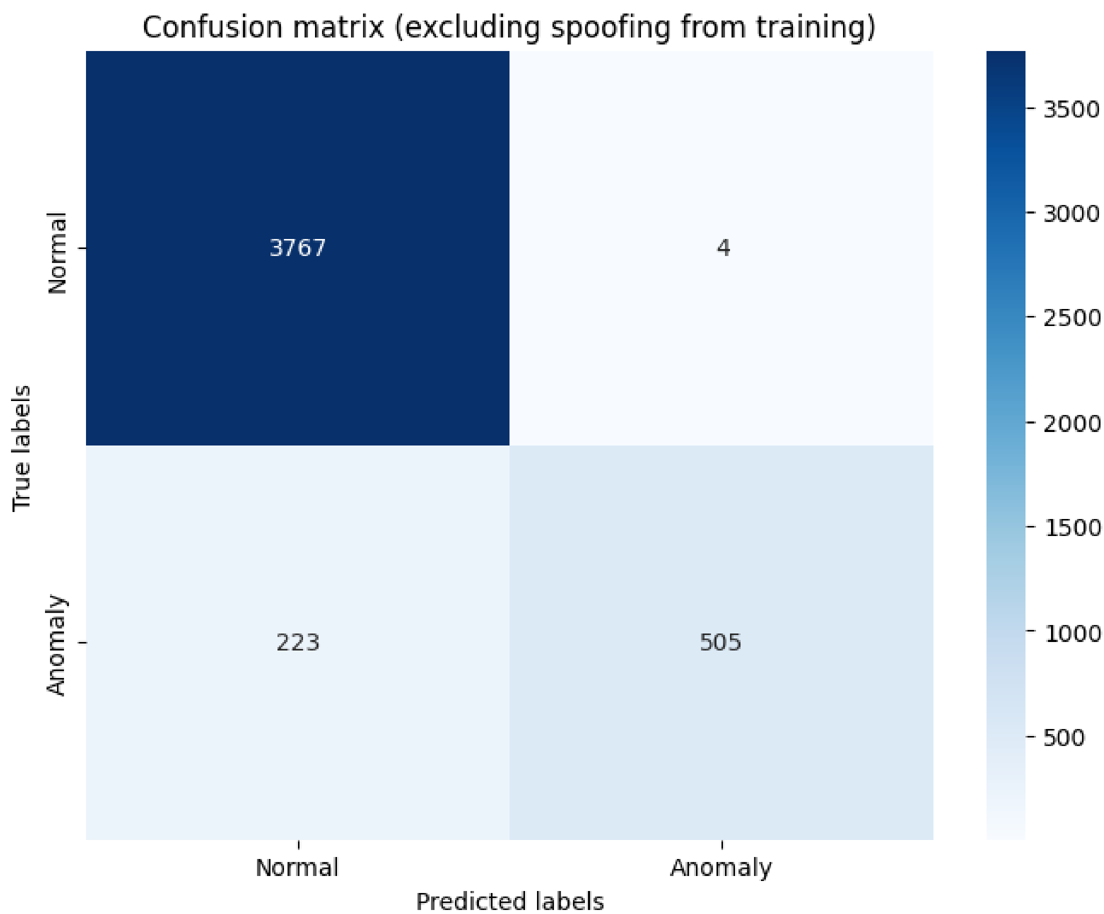
[84]:
```
<keras.src.callbacks.history.History at 0x21286692030>
```

[85]:
```python
y_pred_filtered = model_filtered.predict(X_test)
y_pred_labels_filtered = (y_pred_filtered > 0.5).astype(int)

print(classification_report(y_test, y_pred_labels_filtered))
```

```
141/141            0s 1ms/step
              precision    recall  f1-score   support

           0       0.94      1.00      0.97      3771
           1       0.99      0.69      0.82       728

    accuracy                           0.95      4499
   macro avg       0.97      0.85      0.89      4499
weighted avg       0.95      0.95      0.95      4499
```

[86]:
```python
conf_matrix_filtered = confusion_matrix(y_test, y_pred_labels_filtered)
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix_filtered, annot=True, fmt='d', cmap='Blues',
            xticklabels=['Normal', 'Anomaly'], yticklabels=['Normal',␣
↪'Anomaly'])
plt.title('Confusion matrix (excluding spoofing from training)')
plt.xlabel('Predicted labels')
```

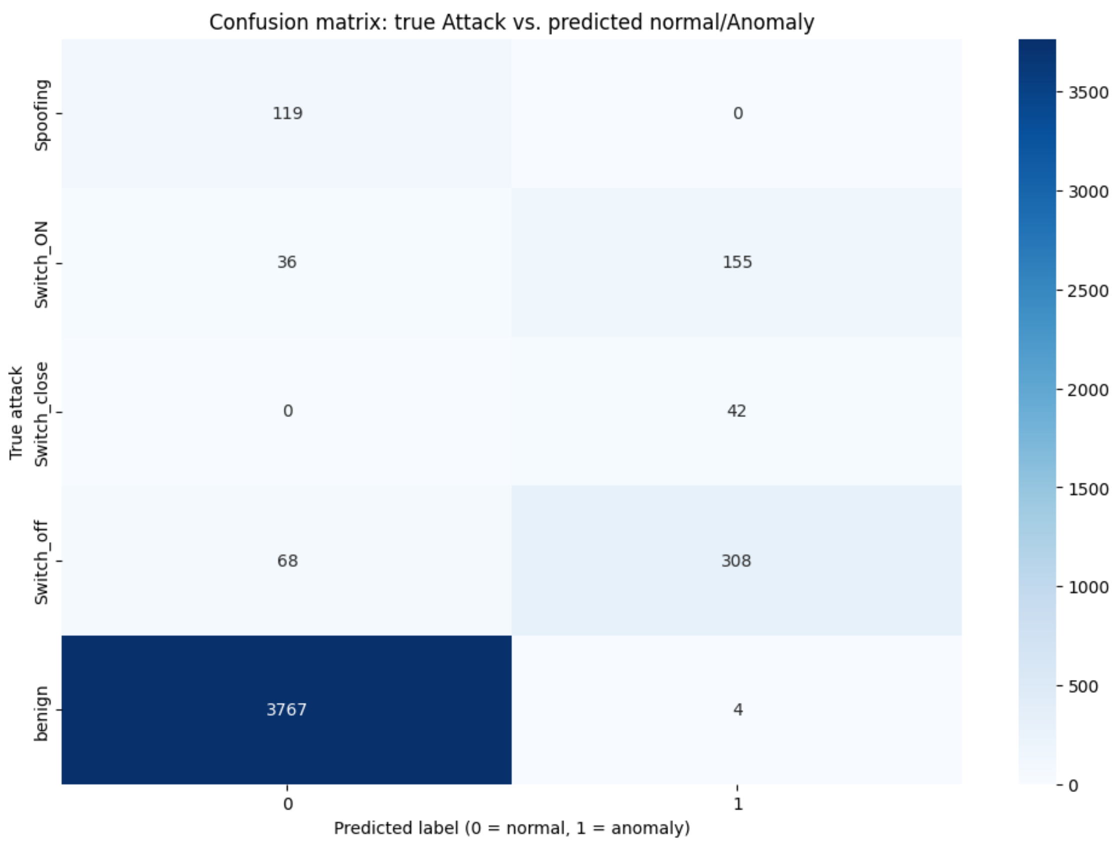
```
plt.ylabel('True labels')
plt.show()
```



[87]:
```
test_attacks = test_attacks.reset_index(drop=True)
y_test = y_test.reset_index(drop=True)
y_pred_labels_filtered = pd.Series(y_pred_labels_filtered.flatten(),␣
↪index=y_test.index)

test_results = pd.DataFrame({
    'True Attack': test_attacks,
    'True Label': y_test,
    'Predicted Label': y_pred_labels_filtered
})
```

[88]:
```
# Generate the confusion matrix: True Attack vs. Predicted Labels
conf_matrix_attack = pd.crosstab(
    test_results['True Attack'],
    test_results['Predicted Label'],
```

```
    rownames=['True Attack'],
    colnames=['Predicted Label'],
    dropna=False
)

plt.figure(figsize=(12, 8))
sns.heatmap(conf_matrix_attack, annot=True, fmt='d', cmap='Blues')
plt.title('Confusion matrix: true Attack vs. predicted normal/Anomaly')
plt.xlabel('Predicted label (0 = normal, 1 = anomaly)')
plt.ylabel('True attack')
plt.show()
```



"Spoofing" attacks were frequently misclassified as normal behavior. It shows the incapability from this model to detect novel attack types. That's why supervised deep learning alone might not be sufficient for comprehensive attack detection.

-> Combining supervised and unsupervised approaches might be more effective.

[ ]: