Security & Artificial Intelligence Project

Authors :

- Alexandre Lemonnier
- Sarah Gutierez
- Victor Simonin

Project objectives

The goal of the project is to design, deploy and evaluate a data chain for the analysis of cybersecurity data. The data treatment will be performed as batch.

We chose Objective 1 :

- Anomaly detection for tracking attacks

Our dataset :

- The UGR'16 Dataset

Data date :

- June - Week 1, Date range: 01/06/2016 - 05/06/2016

In [ ]:

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import OneHotEncoder
import seaborn as sb
import numpy as np
import matplotlib.lines as mlines
from sklearn.ensemble import IsolationForest
from sklearn.neighbors import LocalOutlierFactor
```

Capture data

Here we import our train dataset from the first week of June.

In [ ]:

```
col_names = ["Date", "Duration", "Source_IP", "Destination_IP", "Source_Port",
             "Destination_Port", "Protocol", "Flag", "Forwarding_Status", "Service_Type", "Packets", "Bytes", "Label"]
```

In [ ]:

```
data_june = pd.read_csv('data/train/june.week1.csv', names=col_names, nrows=1000000)
data_june
```

Out[ ]:

	Date	Duration	Source_IP	Destination_IP	Source_Port	Destination_Port	Protocol	Flag	Forwarding_Status	Service_Type	Packets	Bytes	Label

	Date	Duration	Source_IP	Destination_IP	Source_Port	Destination_Port	Protocol	Flag	Forwarding_Status	Service_Type	Packets	Bytes	Label
0	2016-06-01 00:05:01	39.364	211.62.96.220	42.219.158.212	55107	64188	UDP	.A....	0	0	19	3958	background
1	2016-06-01 00:05:03	39.828	42.219.158.226	71.247.111.184	80	52475	TCP	.AP.S.	0	0	57	79635	background
2	2016-06-01 00:05:04	36.128	42.219.153.155	223.80.226.127	443	54691	TCP	.AP.S.	0	0	9	2791	background
3	2016-06-01 00:05:04	36.204	223.80.226.127	42.219.153.155	54691	443	TCP	.AP.S.	0	0	13	3896	background
4	2016-06-01 00:05:04	42.452	42.219.153.7	42.187.82.40	53	53	UDP	.A....	0	0	2	175	background
...	...	...	...	...	...	...	...	...	...	...	...	...	...
999995	2016-06-01 00:15:20	0.636	42.219.156.182	253.139.127.225	51944	25	TCP	.APRS.	0	0	6	355	background
999996	2016-06-01 00:15:20	0.636	42.219.156.182	253.139.127.225	58625	25	TCP	.APRS.	0	0	6	355	background
999997	2016-06-01 00:15:20	0.636	42.219.156.182	253.139.127.227	36784	25	TCP	.APRS.	0	0	6	355	background
999998	2016-06-01 00:15:20	0.636	42.219.156.182	253.139.127.227	39940	25	TCP	.APRS.	0	0	7	395	background
999999	2016-06-01 00:15:20	0.636	42.219.156.182	253.139.127.227	39953	25	TCP	.APRS.	0	0	6	355	background

1000000 rows × 13 columns

In [ ]:

```
genuine_repartition = 0.96
attack_repartition = 0.04
train_data_size = 10000
```

Let's build our train dataset with **96%** of genuine netflow and **4%** of attack.

In [ ]:

```
df_genuine = data_june[data_june['Label'] == 'background'].head(int(train_data_size * genuine_repartition)).copy()
df_attacks = data_june[data_june['Label'] != 'background'].head(int(train_data_size * attack_repartition)).copy()
train_df = pd.concat([df_genuine, df_attacks])
train_df
```

Out[ ]:

	Date	Duration	Source_IP	Destination_IP	Source_Port	Destination_Port	Protocol	Flag	Forwarding_Status	Service_Type	Packets	Bytes	Label
0	2016-06-01 00:05:01	39.364	211.62.96.220	42.219.158.212	55107	64188	UDP	.A....	0	0	19	3958	background
1	2016-06-01 00:05:03	39.828	42.219.158.226	71.247.111.184	80	52475	TCP	.AP.S.	0	0	57	79635	background
2	2016-06-01 00:05:04	36.128	42.219.153.155	223.80.226.127	443	54691	TCP	.AP.S.	0	0	9	2791	background
3	2016-06-01 00:05:04	36.204	223.80.226.127	42.219.153.155	54691	443	TCP	.AP.S.	0	0	13	3896	background
4	2016-06-01 00:05:04	42.452	42.219.153.7	42.187.82.40	53	53	UDP	.A....	0	0	2	175	background
...	...	...	...	...	...	...	...	...	...	...	...	...	...
236466	2016-06-01 00:07:38	0.000	42.219.158.188	143.72.8.137	25469	53	UDP	.A....	0	0	1	54	blacklist
236467	2016-06-01 00:07:38	0.000	42.219.158.188	143.72.8.137	33316	53	UDP	.A....	0	0	1	76	blacklist
236468	2016-06-01 00:07:38	0.000	42.219.158.188	143.72.8.137	36503	53	UDP	.A....	0	0	1	76	blacklist
236469	2016-06-01 00:07:38	0.000	42.219.158.188	143.72.8.137	37345	53	UDP	.A....	0	0	1	67	blacklist
236470	2016-06-01 00:07:38	0.000	42.219.158.188	143.72.8.137	37721	53	UDP	.A....	0	0	1	67	blacklist

10000 rows × 13 columns

Here we import our test dataset from the fifth week of July.

In [ ]:

```
data_july = pd.read_csv('data/test/july.week5.csv', names=col_names, nrows=1000000)
data_july
```

Out[ ]:

	Date	Duration	Source_IP	Destination_IP	Source_Port	Destination_Port	Protocol	Flag	Forwarding_Status	Service_Type	Packets	Bytes	Label
0	2016-07-27 13:43:21	48.380	187.96.221.207	42.219.153.7	53	53	UDP	.A....	0	0	2	209	background
1	2016-07-27 13:43:21	48.380	42.219.153.7	187.96.221.207	53	53	UDP	.A....	0	0	2	167	background
2	2016-07-27 13:43:25	50.632	42.219.153.191	62.205.150.146	80	1838	TCP	.AP...	0	0	9	2082	background
3	2016-07-27 13:43:25	51.052	62.205.150.146	42.219.153.191	1838	80	TCP	.AP...	0	0	9	7118	background
4	2016-07-27 13:43:27	46.996	92.225.28.133	42.219.155.111	443	59867	TCP	.AP...	0	0	4	674	background
...	...	...	...	...	...	...	...	...	...	...	...	...	...
999995	2016-07-27 13:52:59	16.928	42.219.153.89	104.235.226.45	52418	443	TCP	.AP.SF	0	0	6	816	background
999996	2016-07-27 13:52:59	16.932	42.219.153.89	104.235.226.45	52419	443	TCP	.AP.SF	0	0	6	816	background
999997	2016-07-27 13:52:59	1.704	42.219.154.121	192.22.7.102	38718	25	TCP	.AP.SF	0	0	3305	4711799	background
999998	2016-07-27 13:52:59	1.708	42.219.154.123	194.233.64.116	80	52656	TCP	.AP.S.	0	0	74	100883	background
999999	2016-07-27 13:52:59	1.748	133.18.60.180	42.219.159.82	51347	10021	TCP	.AP.SF	0	64	10	446	background

1000000 rows × 13 columns

In [ ]:

```
genuine_repartition_test = 0.90
attack_repartition_test = 0.10
test_data_size = 10000
df_genuine_test = data_july[data_july['Label'] == 'background'].head(int(test_data_size * genuine_repartition_test)).copy()
df_attacks_test = data_july[data_july['Label'] != 'background'].head(int(test_data_size * attack_repartition_test)).copy()
test_df = pd.concat([df_genuine_test, df_attacks_test])
test_df
```

Out[ ]:

	Date	Duration	Source_IP	Destination_IP	Source_Port	Destination_Port	Protocol	Flag	Forwarding_Status	Service_Type	Packets	Bytes	Label
0	2016-07-27 13:43:21	48.380	187.96.221.207	42.219.153.7	53	53	UDP	.A....	0	0	2	209	background
1	2016-07-27 13:43:21	48.380	42.219.153.7	187.96.221.207	53	53	UDP	.A....	0	0	2	167	background
2	2016-07-27 13:43:25	50.632	42.219.153.191	62.205.150.146	80	1838	TCP	.AP...	0	0	9	2082	background
3	2016-07-27 13:43:25	51.052	62.205.150.146	42.219.153.191	1838	80	TCP	.AP...	0	0	9	7118	background
4	2016-07-27 13:43:27	46.996	92.225.28.133	42.219.155.111	443	59867	TCP	.AP...	0	0	4	674	background
...	...	...	...	...	...	...	...	...	...	...	...	...	...
314398	2016-07-27 13:46:47	0.000	77.136.155.124	42.219.158.188	36115	80	TCP	.A.R..	0	0	1	40	blacklist
314399	2016-07-27 13:46:47	0.000	77.136.155.124	42.219.158.188	36116	80	TCP	.A.R..	0	0	1	40	blacklist

	Date	Duration	Source_IP	Destination_IP	Source_Port	Destination_Port	Protocol	Flag	Forwarding_Status	Service_Type	Packets	Bytes	Label
314400	2016-07-27 13:46:47	0.000	77.136.155.124	42.219.158.188	36162	80	TCP	.A.R..	0	0	1	40	blacklist
314401	2016-07-27 13:46:47	0.000	77.136.155.124	42.219.158.188	36163	80	TCP	.A.R..	0	0	1	40	blacklist
314402	2016-07-27 13:46:47	0.000	77.136.155.124	42.219.158.188	36164	80	TCP	.A.R..	0	0	1	40	blacklist

10000 rows × 13 columns

We stock the dataframes in the `data` list to apply the same cleaning:

In [ ]:
```
data = { "train" : train_df, "test": test_df}
```

General overview

In [ ]:
```
data["train"].info()
print('_'*40)
data["test"].info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 10000 entries, 0 to 236470
Data columns (total 13 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   Date               10000 non-null  object 
 1   Duration           10000 non-null  float64
 2   Source_IP          10000 non-null  object 
 3   Destination_IP     10000 non-null  object 
 4   Source_Port        10000 non-null  int64  
 5   Destination_Port   10000 non-null  int64  
 6   Protocol           10000 non-null  object 
 7   Flag               10000 non-null  object 
 8   Forwarding_Status  10000 non-null  int64  
 9   Service_Type       10000 non-null  int64  
 10  Packets            10000 non-null  int64  
 11  Bytes              10000 non-null  int64  
 12  Label              10000 non-null  object 
dtypes: float64(1), int64(6), object(6)
memory usage: 1.1+ MB
________________________________________
<class 'pandas.core.frame.DataFrame'>
Int64Index: 10000 entries, 0 to 314402
Data columns (total 13 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   Date               10000 non-null  object 
 1   Duration           10000 non-null  float64
 2   Source_IP          10000 non-null  object 
 3   Destination_IP     10000 non-null  object 
 4   Source_Port        10000 non-null  int64  
 5   Destination_Port   10000 non-null  int64  
 6   Protocol           10000 non-null  object 
 7   Flag               10000 non-null  object 
 8   Forwarding_Status  10000 non-null  int64  
```

```
 9   Service_Type       10000 non-null  int64
 10  Packets            10000 non-null  int64
 11  Bytes              10000 non-null  int64
 12  Label              10000 non-null  object
dtypes: float64(1), int64(6), object(6)
memory usage: 1.1+ MB
```

Which features are categorical?

These values classify the samples into sets of similar samples.

- **Categorical**: Date, Source_IP, Destination_IP, Source_Port, Destination_Port, Protocol, Flag, Label.

Source_Port and *Destination_Port* are of **int64** type, so we will have to categorize them.

Which features are numerical?

These values change from sample to sample.

- **Numerical**: Duration, Forwarding Status, Type of Service, Packets, Bytes

In [ ]:

```python
for _, df in data.items():
    df['Source_Port'] = df['Source_Port'].astype(object)
    df['Destination_Port'] = df['Destination_Port'].astype(object)
```

In [ ]:

```python
cat_col = list(train_df.select_dtypes(include=object).columns.values)
cat_col
```

Out[ ]:

```
['Date',
 'Source_IP',
 'Destination_IP',
 'Source_Port',
 'Destination_Port',
 'Protocol',
 'Flag',
 'Label']
```

In [ ]:

```python
num_col = list(train_df._get_numeric_data().columns.values)
num_col
```

Out[ ]:

```
['Duration', 'Forwarding_Status', 'Service_Type', 'Packets', 'Bytes']
```

Which features may contain errors or typos?

We know that data come from a consistent netflow so they should be quite accurate.

Which features contain blank, null or empty values?

We can observe that dataframes do not contain any null value.

In [ ]:

```python
data["train"].isna().sum()
```

Out[ ]:

```
Date                    0
```

```
Duration            0
Source_IP           0
Destination_IP      0
Source_Port         0
Destination_Port    0
Protocol            0
Flag                0
Forwarding_Status   0
Service_Type        0
Packets             0
Bytes               0
Label               0
dtype: int64
```

Data analysis

Let's get statistics about the numerical data:

In [ ]:
```
data["train"].describe()
```

Out[ ]:

	Duration	Forwarding_Status	Service_Type	Packets	Bytes
count	10000.000000	10000.0	10000.000000	10000.000000	1.000000e+04
mean	1.267133	0.0	1.967500	17.802200	1.305524e+04
std	6.917004	0.0	12.731742	784.207307	9.818564e+05
min	0.000000	0.0	0.000000	1.000000	2.900000e+01
25%	0.000000	0.0	0.000000	1.000000	8.600000e+01
50%	0.148000	0.0	0.000000	3.000000	2.255000e+02
75%	0.789000	0.0	0.000000	7.000000	9.350000e+02
max	261.412000	0.0	208.000000	68650.000000	9.814513e+07

There is not numerical data with a negative value. That's a good point because all expected to be strictly positive.

However, we observe that `Forwarding_Status` column is full of zeroes. We decide to drop it.

In [ ]:
```
for _, df in data.items():
    df.drop(columns='Forwarding_Status', inplace=True)
data["train"].head()
```

Out[ ]:

	Date	Duration	Source_IP	Destination_IP	Source_Port	Destination_Port	Protocol	Flag	Service_Type	Packets	Bytes	Label
0	2016-06-01 00:05:01	39.364	211.62.96.220	42.219.158.212	55107	64188	UDP	.A....	0	19	3958	background
1	2016-06-01 00:05:03	39.828	42.219.158.226	71.247.111.184	80	52475	TCP	.AP.S.	0	57	79635	background
2	2016-06-01 00:05:04	36.128	42.219.153.155	223.80.226.127	443	54691	TCP	.AP.S.	0	9	2791	background
3	2016-06-01 00:05:04	36.204	223.80.226.127	42.219.153.155	54691	443	TCP	.AP.S.	0	13	3896	background
4	2016-06-01 00:05:04	42.452	42.219.153.7	42.187.82.40	53	53	UDP	.A....	0	2	175	background

Let's get statistics about the categorical data:

In [ ]:
```
train_df.describe(include=['O'])
```

Out[ ]:

	Date	Source_IP	Destination_IP	Source_Port	Destination_Port	Protocol	Flag	Label
count	10000	10000	10000	10000	10000	10000	10000	10000
unique	103	1704	2830	4435	4775	4	19	2
top	2016-06-01 00:05:16	42.219.156.211	42.219.156.211	53	53	TCP	.A....	background
freq	1182	1161	940	1947	1447	6415	3859	9600

We can see that Ips and Ports are not all unique and appears multiple times in the dataset.

Preprocessing data

In [ ]:
```
data["train"]['Label'].unique()
```

Out[ ]:
```
array(['background', 'blacklist'], dtype=object)
```

Let's numerize `Label` column with `0` value is the netflow is normal and `1` otherwise.

In [ ]:
```
for _, df in data.items():
    df['Label'] = np.where(df['Label'] == 'background', 0, 1)
print("Train dataset \'Label\' unique value:", data["train"]['Label'].unique())
print("Test dataset \'Label\' unique value:", data["test"]['Label'].unique())
```

```
Train dataset 'Label' unique value: [0 1]
Test dataset 'Label' unique value: [0 1]
```

Let's divide `Date` column in multiple attributes, to analyse which one is relevant with the `Label` value.

In [ ]:
```
for _, df in data.items():
    df['Date'] = pd.to_datetime(df['Date'], format="%Y-%m-%d %H:%M:%S")
    df['year'] = df['Date'].dt.year
    df['month'] = df['Date'].dt.month
    df['day'] = df['Date'].dt.day
    df['week'] = df['Date'].dt.isocalendar().week
    df['hour'] = df['Date'].dt.hour
    df['minute'] = df['Date'].dt.minute
    df['second'] = df['Date'].dt.second
    df['dayOfWeek'] = df['Date'].dt.dayofweek
    df.drop(columns='Date', inplace=True)
```

Dataframes should look like that now:

In [ ]:
```
data["train"].head()
```

Out[ ]:

	Duration	Source_IP	Destination_IP	Source_Port	Destination_Port	Protocol	Flag	Service_Type	Packets	Bytes	Label	year	month	day	week	hour	minute	second	dayOfWeek

	Duration	Source_IP	Destination_IP	Source_Port	Destination_Port	Protocol	Flag	Service_Type	Packets	Bytes	Label	year	month	day	week	hour	minute	second	dayOfWeek
0	39.364	211.62.96.220	42.219.158.212	55107	64188	UDP	.A....	0	19	3958	0	2016	6	1	22	0	5	1	2
1	39.828	42.219.158.226	71.247.111.184	80	52475	TCP	.AP.S.	0	57	79635	0	2016	6	1	22	0	5	3	2
2	36.128	42.219.153.155	223.80.226.127	443	54691	TCP	.AP.S.	0	9	2791	0	2016	6	1	22	0	5	4	2
3	36.204	223.80.226.127	42.219.153.155	54691	443	TCP	.AP.S.	0	13	3896	0	2016	6	1	22	0	5	4	2
4	42.452	42.219.153.7	42.187.82.40	53	53	UDP	.A....	0	2	175	0	2016	6	1	22	0	5	4	2

In [ ]:

```
corr = data["train"].corr()
corr.style.background_gradient(cmap='coolwarm')
```

```
/usr/lib/python3.10/site-packages/pandas/io/formats/style.py:3554: RuntimeWarning: All-NaN slice encountered
  smin = np.nanmin(gmap) if vmin is None else vmin
/usr/lib/python3.10/site-packages/pandas/io/formats/style.py:3555: RuntimeWarning: All-NaN slice encountered
  smax = np.nanmax(gmap) if vmax is None else vmax
```

Out[ ]:

	Duration	Service_Type	Packets	Bytes	Label	year	month	day	week	hour	minute	second	dayOfWeek
Duration	1.000000	-0.018325	0.512220	0.383358	0.066698	nan	nan	nan	nan	nan	0.044027	0.000970	nan
Service_Type	-0.018325	1.000000	-0.002071	-0.001800	0.076680	nan	nan	nan	nan	nan	0.078836	0.036927	nan
Packets	0.512220	-0.002071	1.000000	0.883493	-0.002526	nan	nan	nan	nan	nan	-0.001886	-0.014319	nan
Bytes	0.383358	-0.001800	0.883493	1.000000	-0.001875	nan	nan	nan	nan	nan	-0.001505	-0.010885	nan
Label	0.066698	0.076680	-0.002526	-0.001875	1.000000	nan	nan	nan	nan	nan	0.827848	0.397177	nan
year	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan
month	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan
day	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan
week	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan
hour	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan
minute	0.044027	0.078836	-0.001886	-0.001505	0.827848	nan	nan	nan	nan	nan	1.000000	0.247712	nan
second	0.000970	0.036927	-0.014319	-0.010885	0.397177	nan	nan	nan	nan	nan	0.247712	1.000000	nan
dayOfWeek	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan

We can observe that we have a lot of NaN corresponding to the different attribute of the date.
Our data are based on the same `year` , `month` and `week` so we can drop these columns, they will not be relevant.
In our batch of data, `day` is not relevant at all because we are only focusing on one week. We drop `dayOfWeek` because here we are training on only one day.

In [ ]:

```
for _, df in data.items():
    df.drop(columns=['year', 'month', 'week', 'day', 'dayOfWeek'], inplace=True)
```

Feature engineering

Moreover, we decide to replace `hour` , `minute` , `second` column with a delta from the beginning of the day.

In [ ]:

```
for _, df in data.items():
    df['timeDelta'] = df['hour'] *3600 + df['minute'] * 60 + df['second']
    df.drop(columns=['hour', 'minute', 'second'], inplace=True)
data['train']
```

Out[ ]:

	Duration	Source_IP	Destination_IP	Source_Port	Destination_Port	Protocol	Flag	Service_Type	Packets	Bytes	Label	timeDelta
0	39.364	211.62.96.220	42.219.158.212	55107	64188	UDP	.A....	0	19	3958	0	301
1	39.828	42.219.158.226	71.247.111.184	80	52475	TCP	.AP.S.	0	57	79635	0	303
2	36.128	42.219.153.155	223.80.226.127	443	54691	TCP	.AP.S.	0	9	2791	0	304
3	36.204	223.80.226.127	42.219.153.155	54691	443	TCP	.AP.S.	0	13	3896	0	304
4	42.452	42.219.153.7	42.187.82.40	53	53	UDP	.A....	0	2	175	0	304
...	...	...	...	...	...	...	...	...	...	...	...	...
236466	0.000	42.219.158.188	143.72.8.137	25469	53	UDP	.A....	0	1	54	1	458
236467	0.000	42.219.158.188	143.72.8.137	33316	53	UDP	.A....	0	1	76	1	458
236468	0.000	42.219.158.188	143.72.8.137	36503	53	UDP	.A....	0	1	76	1	458
236469	0.000	42.219.158.188	143.72.8.137	37345	53	UDP	.A....	0	1	67	1	458
236470	0.000	42.219.158.188	143.72.8.137	37721	53	UDP	.A....	0	1	67	1	458

10000 rows × 12 columns

We decide to replace `Duration`, `Packets`, `Bytes` column with a ratio corresponding to the number of bytes by packets by minutes.

In [ ]:

```
for _, df in data.items():
    df['transfered_ratio'] = (df['Bytes'] / df['Packets']) / (df['Duration'] + 1)
    df.drop(columns=['Duration', 'Packets', 'Bytes'], inplace=True)
data["train"]
```

Out[ ]:

	Source_IP	Destination_IP	Source_Port	Destination_Port	Protocol	Flag	Service_Type	Label	timeDelta	transfered_ratio
0	211.62.96.220	42.219.158.212	55107	64188	UDP	.A....	0	0	301	5.160930
1	42.219.158.226	71.247.111.184	80	52475	TCP	.AP.S.	0	0	303	34.219292
2	42.219.153.155	223.80.226.127	443	54691	TCP	.AP.S.	0	0	304	8.352486
3	223.80.226.127	42.219.153.155	54691	443	TCP	.AP.S.	0	0	304	8.055379
4	42.219.153.7	42.187.82.40	53	53	UDP	.A....	0	0	304	2.013716
...	...	...	...	...	...	...	...	...	...	...
236466	42.219.158.188	143.72.8.137	25469	53	UDP	.A....	0	1	458	54.000000
236467	42.219.158.188	143.72.8.137	33316	53	UDP	.A....	0	1	458	76.000000
236468	42.219.158.188	143.72.8.137	36503	53	UDP	.A....	0	1	458	76.000000
236469	42.219.158.188	143.72.8.137	37345	53	UDP	.A....	0	1	458	67.000000
236470	42.219.158.188	143.72.8.137	37721	53	UDP	.A....	0	1	458	67.000000

10000 rows × 10 columns

In [ ]:

```
corr = data["train"].corr()
corr.style.background_gradient(cmap='coolwarm')
```

Out[ ]:

	Service_Type	Label	timeDelta	transfered_ratio
Service_Type	1.000000	0.076680	0.080787	-0.047649
Label	0.076680	1.000000	0.850875	0.030599
timeDelta	0.080787	0.850875	1.000000	0.012505
transfered_ratio	-0.047649	0.030599	0.012505	1.000000

In [ ]:

```
data["train"][["Source_IP", "Label"]].groupby(["Source_IP"],as_index=False).mean().sort_values(by='Label', ascending=False).head()
```

Out[ ]:

	Source_IP	Label
1292	50.33.218.194	1.0
839	222.11.231.252	1.0
695	213.30.31.179	1.0
836	221.17.125.37	1.0
1293	50.33.218.239	1.0

In [ ]:

```
data["train"][["Destination_IP", "Label"]].groupby(['Destination_IP'],as_index=False).mean().sort_values(by='Label', ascending=False).head()
```

Out[ ]:

	Destination_IP	Label
2103	51.210.95.1	1.0
835	204.97.46.104	1.0
1147	213.30.31.179	1.0
2523	71.161.145.125	1.0
2522	71.161.116.194	1.0

In [ ]:

```
data["train"][["Source_Port", "Label"]].groupby(['Source_Port'],as_index=False).mean().sort_values(by='Label', ascending=False).head()
```

Out[ ]:

	Source_Port	Label
3615	56920	1.0
1774	41416	1.0
2643	49678	1.0
1290	36503	1.0

	Source_Port	Label
2649	49747	1.0

In [ ]:

```
data["train"][["Destination_Port", "Label"]].groupby(['Destination_Port'],as_index=False).mean().sort_values(by='Label', ascending=False).head()
```

Out[ ]:

	Destination_Port	Label
973	31308	1.0
802	25469	1.0
1400	37563	1.0
1100	33892	1.0
788	25089	1.0

In [ ]:

```
data["train"][["Protocol", "Label"]].groupby(['Protocol'],as_index=False).mean().sort_values(by='Label', ascending=False)
```

Out[ ]:

	Protocol	Label
3	UDP	0.042237
2	TCP	0.039283
0	ESP	0.000000
1	ICMP	0.000000

In [ ]:

```
data["train"][["Flag", "Label"]].groupby(['Flag'],as_index=False).mean().sort_values(by='Label', ascending=False)
```

Out[ ]:

	Flag	Label
4	.A...F	0.257732
9	.A.RS.	0.181818
16	.APR.F	0.142857
7	.A.R..	0.120690
17	.APRS.	0.100000
13	.AP.S.	0.094456
11	.AP...	0.072727
6	.A..SF	0.065657
3	.A....	0.040425
2	...R..	0.030675
1	....S.	0.028281
15	.APR..	0.023256

	Flag	Label
14	.AP.SF	0.018586
18	.APRSF	0.015217
12	.AP..F	0.013333
5	.A..S.	0.009901
10	.A.RSF	0.000000
8	.A.R.F	0.000000
0	......	0.000000

In [ ]:

```
data["train"][["Service_Type", "Label"]].groupby(['Service_Type'],as_index=False).mean().sort_values(by='Label', ascending=False)
```

Out[ ]:

	Service_Type	Label
8	28	1.000000
13	72	0.397059
10	40	0.168539
1	2	0.117647
0	0	0.034931
2	8	0.024390
11	64	0.000000
16	200	0.000000
15	192	0.000000
14	184	0.000000
12	66	0.000000
9	32	0.000000
7	26	0.000000
6	24	0.000000
5	23	0.000000
4	20	0.000000
3	16	0.000000
17	208	0.000000

We can observe that for `Source_IP`, `Destination_IP`, `Source_Port` and `Destination_Port` column data is relevant for finding attacks because some IP and Port are only used for this type of netflow.

We will have to transform our categorical data in numerical data to use them in our classification model.
We will use a **One Hot Encoder** transformer to numerize our categorical data.

Graphical analysis

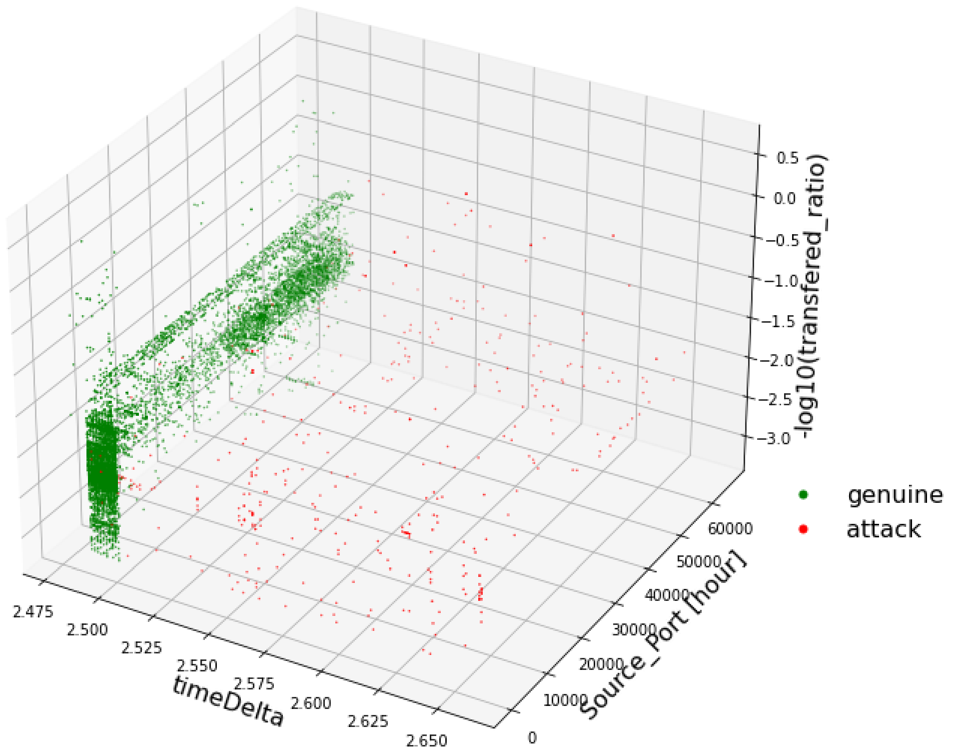
In [ ]:

```
def show3D_netflow_data(netflow_dataset, x_axis_name, y_axis_name, z_axis_name, only_error):
    X = netflow_dataset.drop(columns=['Label'])
    Y = netflow_dataset['Label']

    x = x_axis_name
    y = y_axis_name
    z = z_axis_name

    limit = len(X)

    sb.reset_orig()

    fig = plt.figure(figsize = (10, 12))

    ax = fig.add_subplot(111, projection='3d')
    if not only_error:
        ax.scatter(np.log10(X.loc[Y == 0, x][:limit]),
                   (X.loc[Y == 0, y][:limit] + 0.1),
                   -np.log10(X.loc[Y == 0, z][:limit] + 0.1),
                   c = 'g',
                   marker = '.',
                   s = 1,
                   label = 'genuine')

    ax.scatter(np.log10(X.loc[Y == 1, x][:limit]),
               (X.loc[Y == 1, y][:limit] + 0.1),
               -np.log10(X.loc[Y == 1, z][:limit] + 0.1),
               c = 'r',
               marker = '.',
               s = 1,
               label = 'attack')

    ax.set_xlabel(x, size = 16)
    ax.set_ylabel(y + ' [hour]', size = 16)
    ax.set_zlabel('-log10(' + z + ')', size = 16)
    ax.set_title('Error-based features separate out genuine and attack netflow', size = 20)

    plt.axis('tight')
    ax.grid(1)

    noFraudMarker = mlines.Line2D([],
                                  [],
                                  linewidth = 0,
                                  color = 'g',
                                  marker = '.',
                                  markersize = 10,
                                  label = 'genuine')

    fraudMarker = mlines.Line2D([],
                                [],
                                linewidth = 0,
                                color = 'r',
                                marker = '.',
                                markersize = 10,
                                label = 'attack')
```

```python
    plt.legend(handles = [noFraudMarker, fraudMarker],
               bbox_to_anchor = (1.20, 0.38),
               frameon = False,
               prop = {'size': 16})
```

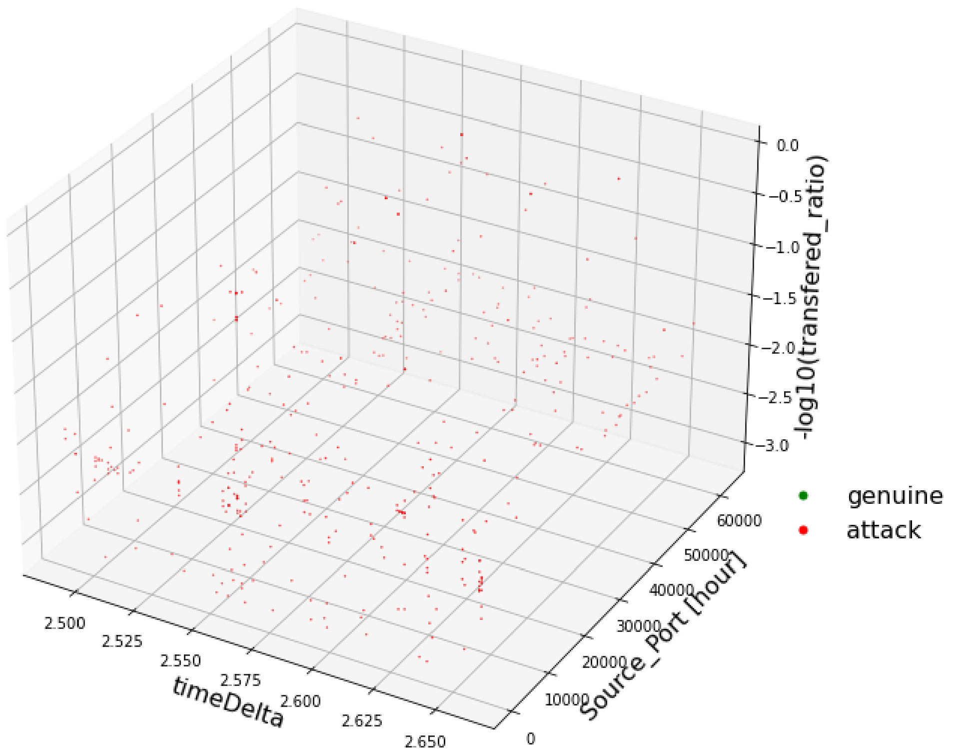
In [ ]:
```python
show3D_netflow_data(data["train"], 'timeDelta', 'Source_Port', 'transfered_ratio',  False)
```

Error-based features separate out genuine and attack netflow



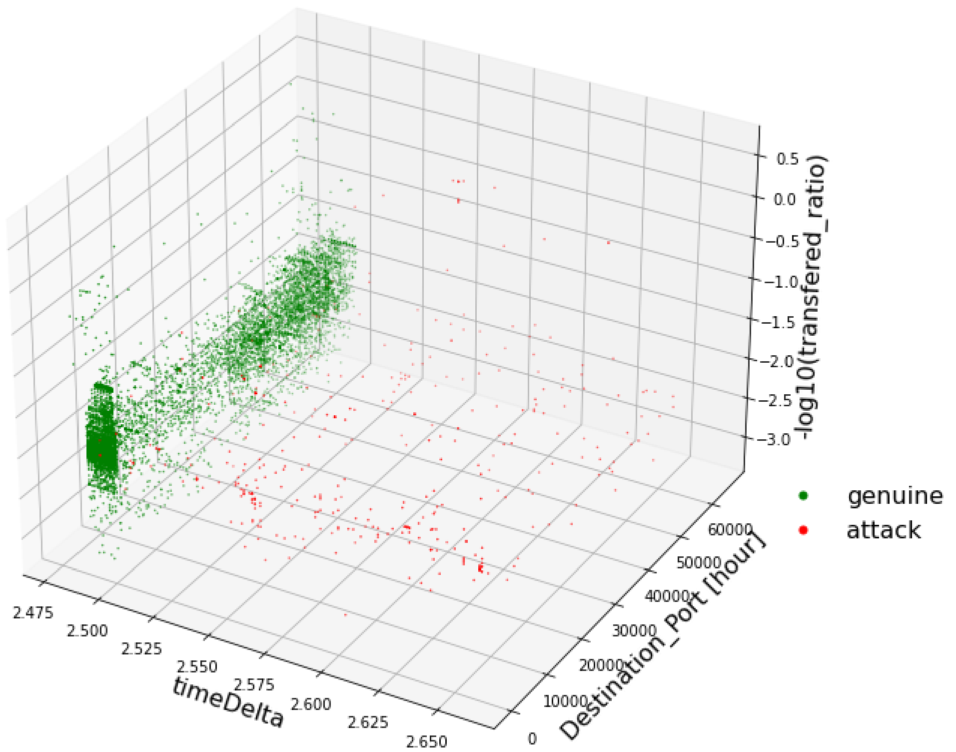
In [ ]:
```python
show3D_netflow_data(data["train"], 'timeDelta', 'Source_Port', 'transfered_ratio',  True)
```

Error-based features separate out genuine and attack netflow

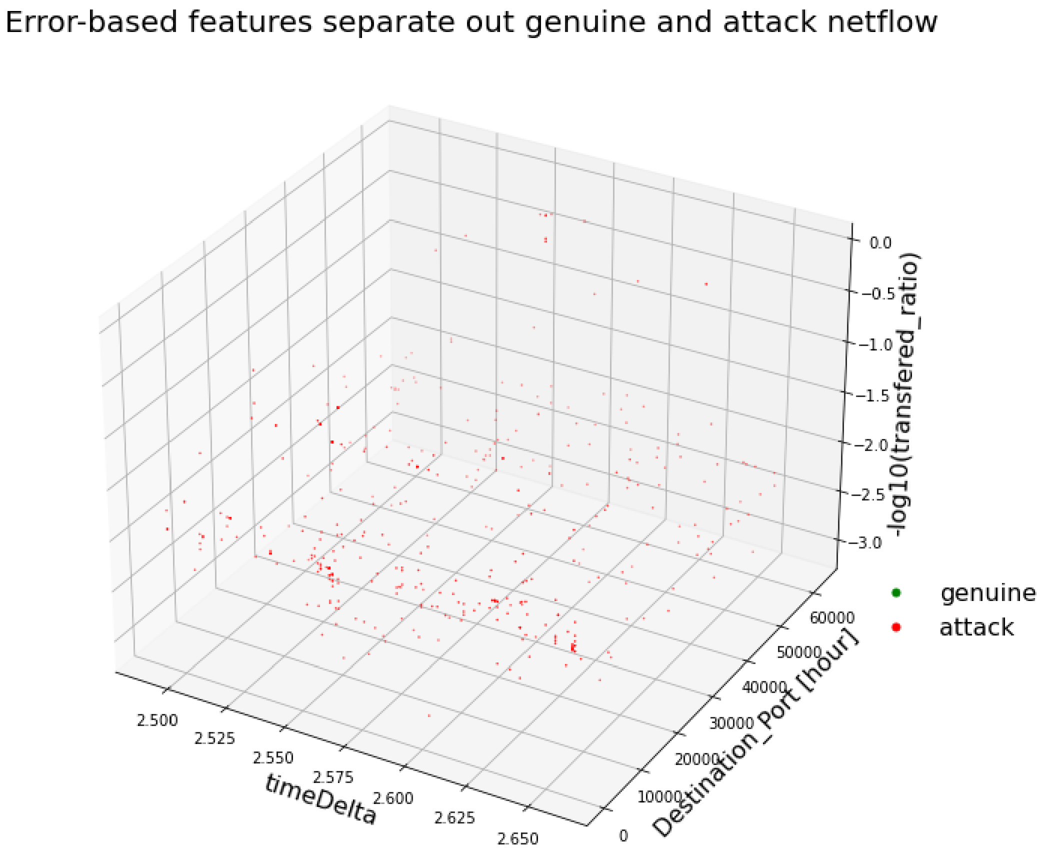



```
In [ ]: show3D_netflow_data(data["train"], 'timeDelta', 'Destination_Port', 'transfered_ratio',  False)
```

Error-based features separate out genuine and attack netflow




```
In [ ]: show3D_netflow_data(data["train"], 'timeDelta', 'Destination_Port', 'transfered_ratio', True)
```



We can observe that `Source_Port` and `Destination_Port` are relevant to find if a netflow is genuine or not.

Supervised machine learning model benchmarking

In [ ]:

```
import xgboost as xgb
from sklearn.compose import make_column_transformer
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC, LinearSVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import Perceptron
from sklearn.linear_model import SGDClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import f1_score, precision_recall_curve, confusion_matrix, mean_squared_error, average_precision_score, plot_confusion_matrix
```

```
/home/leme/.local/lib/python3.10/site-packages/pkg_resources/__init__.py:122: PkgResourcesDeprecationWarning: 2.0.5-build-libtorrent-rasterbar-src-libtorrent-rasterbar-2.0.5-bindi
ngs-python is an invalid version and will not be supported in a future release
```

```
  warnings.warn(
```

In [ ]:
```
cat_col = ['Source_IP', 'Destination_IP', 'Source_Port', 'Destination_Port', 'Protocol', 'Flag', 'Service_Type']
cat_col
```

Out[ ]:
```
['Source_IP',
 'Destination_IP',
 'Source_Port',
 'Destination_Port',
 'Protocol',
 'Flag',
 'Service_Type']
```

In [ ]:
```
num_col = ['Duration', 'Packets', 'Bytes', 'timeDelta']
num_col
```

Out[ ]:
```
['Duration', 'Packets', 'Bytes', 'timeDelta']
```

In [ ]:
```
transformer = make_column_transformer(
    (OneHotEncoder(handle_unknown='ignore'), cat_col))
```

In [ ]:
```
X_train = data["train"].drop(columns=['Label'])
Y_train = data["train"]['Label'].copy()
```

In [ ]:
```
X_test = data["test"].drop(columns=["Label"])
Y_test = data["test"]['Label'].copy()
```

Let's benchmark some model with their default parameters

In [ ]:
```
classifier_score = {}
```

In [ ]:
```
def classifier_testing(classifier):
    print(f"Testing {classifier[0]}...")
    pipeline = Pipeline([
        ('transformer', transformer),
        ('classifier', classifier[1]),
    ])
    print("Fitting...")
    pipeline.fit(X_train, Y_train)
    print("Predicting...")
    y_pred = pipeline.predict(X_test)
    print("Scoring...")
    score = f1_score(Y_test, y_pred)
    print(f"{classifier[0]} score : {score}")
    classifier_score[classifier[0]] = score
```

In [ ]:
```
classifiers = {
    "SVC": SVC(),
    "LinearSVC": LinearSVC(),
```

```python
    "LogisticRegression": LogisticRegression(),
    "RandomForest": RandomForestClassifier(),
    "Perceptron" : Perceptron(),
    "SGD" : SGDClassifier(),
    "DecisionTree" : DecisionTreeClassifier()
}
```

In [ ]:

```python
for key, value in classifiers.items():
    classifier_testing((key, value))
```

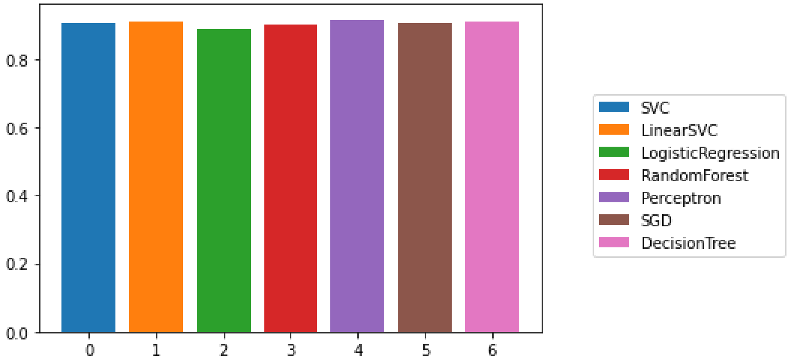
```
Testing SVC...
Fitting...
Predicting...
Scoring...
SVC score : 0.9055161114145276
Testing LinearSVC...
Fitting...
Predicting...
Scoring...
LinearSVC score : 0.9074074074074073
Testing LogisticRegression...
Fitting...
Predicting...
Scoring...
LogisticRegression score : 0.8872848417545809
Testing RandomForest...
Fitting...
Predicting...
Scoring...
RandomForest score : 0.9010989010989011
Testing Perceptron...
Fitting...
Predicting...
Scoring...
Perceptron score : 0.9150599270453361
Testing SGD...
Fitting...
Predicting...
Scoring...
SGD score : 0.9059271343121261
Testing DecisionTree...
Fitting...
Predicting...
Scoring...
DecisionTree score : 0.9109552077711818
```

In [ ]:

```python
for i, test in enumerate(classifier_score.items()):
    plt.bar(i, test[1], label=test[0])
plt.legend(bbox_to_anchor=(1, 0.25, 0.5, 0.5))
```

Out[ ]:

```
<matplotlib.legend.Legend at 0x7fec38f0d930>
```



In [ ]:

```
trainX, testX, trainY, testY = train_test_split(data["train"].drop(columns='Label').copy(), data["train"]['Label'].copy(), test_size=0.2, random_state=42)

weights = (trainY == 0).sum() / (1.0 * (trainY == 1).sum())
xgb_model = xgb.XGBClassifier(max_depth=3, scale_pos_weight=weights, n_jobs=4, random_state=42)
pipeline = Pipeline([
        ('transformer', transformer),
        ('classifier', xgb_model),
    ])

proba = pipeline.fit(trainX, trainY).predict_proba(testX)
y_pred = pipeline.predict(testX)

auprc = average_precision_score(testY, proba[:, 1])
print(f"Xgboost score: {auprc}")
confusion_matrix(testY, y_pred)
```

```
Xgboost score: 0.9601196179638286
```

Out[ ]:

```
array([[1930,    0],
       [   3,   67]])
```

In [ ]:

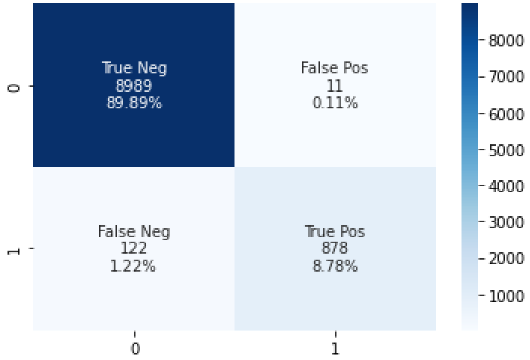
```
def print_confusion_matrix(cf_matrix):
    group_names = ['True Neg','False Pos','False Neg','True Pos']
    group_counts = ['{0:0.0f}'.format(value) for value in
                cf_matrix.flatten()]
    group_percentages = ['{0:.2%}'.format(value) for value in
                     cf_matrix.flatten()/np.sum(cf_matrix)]
    labels = [f"{v1}\n{v2}\n{v3}" for v1, v2, v3 in
          zip(group_names,group_counts,group_percentages)]
    labels = np.asarray(labels).reshape(2,2)
    sb.heatmap(cf_matrix, annot=labels, fmt='', cmap='Blues')
```

In [ ]:

```
weights = (Y_train == 0).sum() / (1.0 * (Y_train == 1).sum())
xgb_model = xgb.XGBClassifier( scale_pos_weight=weights, n_jobs=4, random_state=42)
pipeline = Pipeline([
        ('transformer', transformer),
        ('classifier', xgb_model),
    ])

proba = pipeline.fit(X_train, Y_train).predict_proba(X_test)
```

```
y_pred = pipeline.predict(X_test)
auprc = average_precision_score(Y_test, proba[:, 1])
print(f"Xgboost score: {auprc}")
print_confusion_matrix(confusion_matrix(Y_test, y_pred))
```

```
Xgboost score: 0.9468087123751252
```

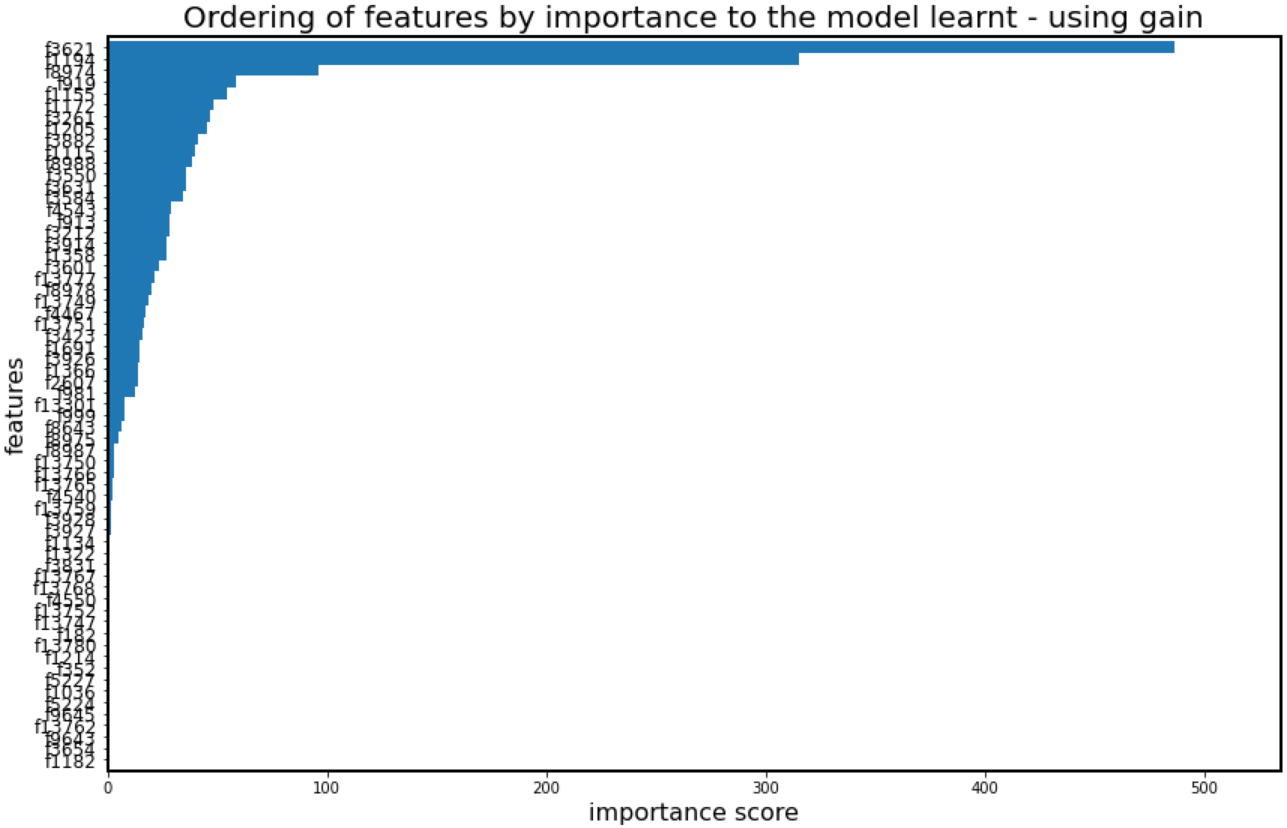


In [ ]:

```
fig = plt.figure(figsize=(14, 9))
ax = fig.add_subplot(111)


ax = xgb.plot_importance(xgb_model, height= 1,  grid=False,
                     show_values = False, importance_type = 'gain', ax =ax)

for axis in ['top', 'bottom', 'left', 'right']:
    ax.spines[axis].set_linewidth(2)

ax.set_xlabel('importance score', size=16)
ax.set_ylabel('features', size=16)
ax.set_yticklabels(ax.get_yticklabels(), size=12)
ax.set_title('Ordering of features by importance to the model learnt - using gain', size=20)
```

Out[ ]:

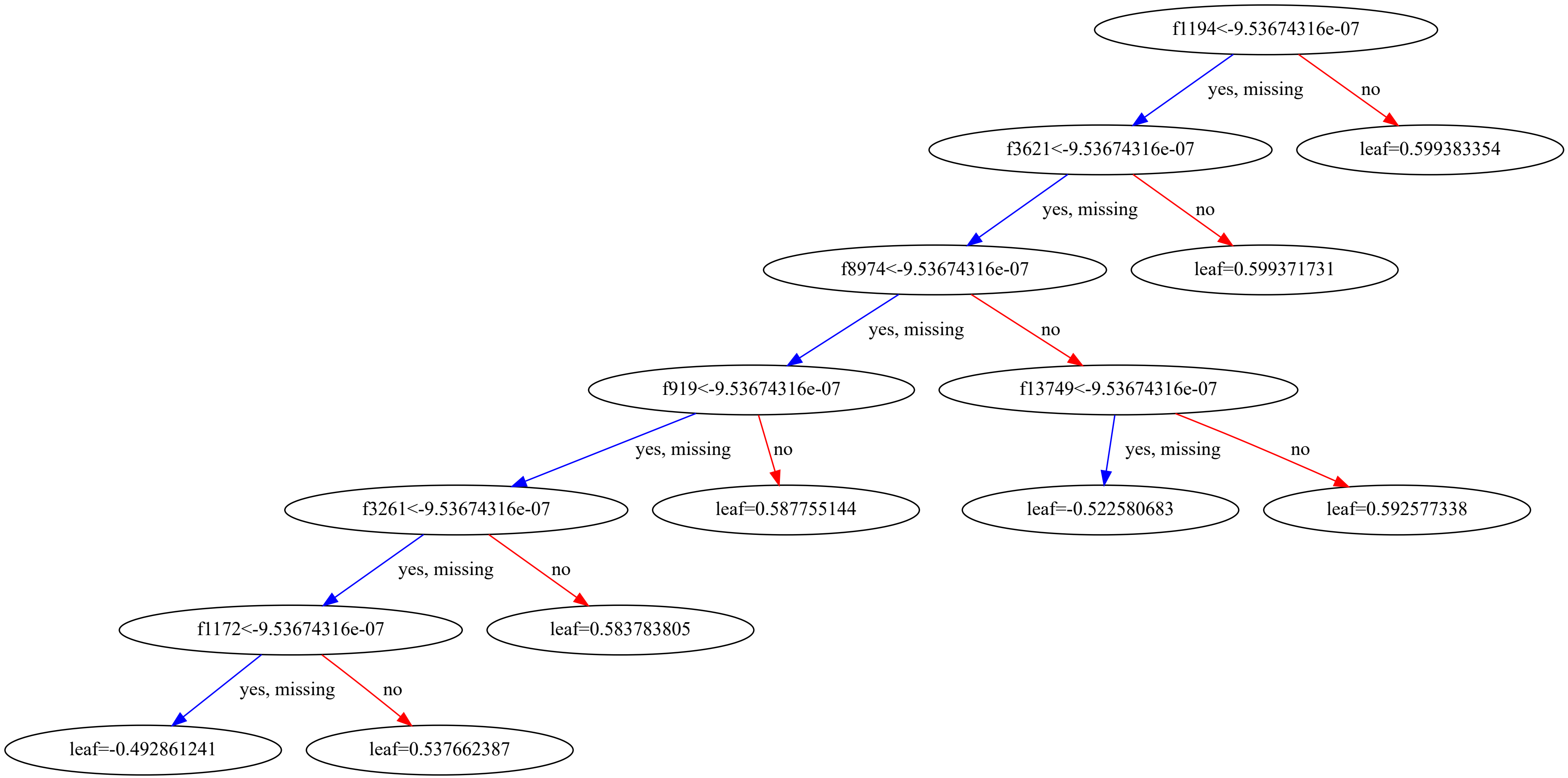
```
Text(0.5, 1.0, 'Ordering of features by importance to the model learnt - using gain')
```



In [ ]:

```
xgb.to_graphviz(xgb_model)
```

Out[ ]:



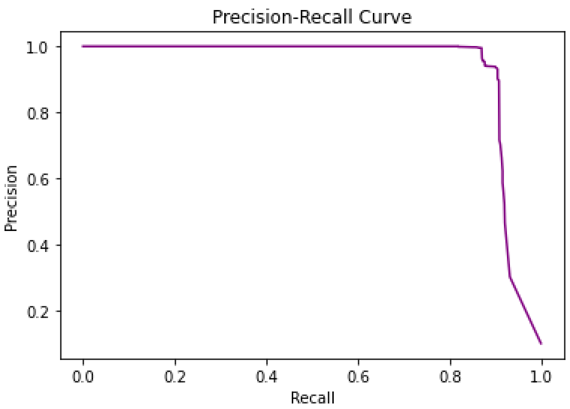
In [ ]:

```
pipeline = Pipeline([
        ('transformer', transformer),
        ('classifier', RandomForestClassifier()),
    ])
print("Fitting...")
pipeline.fit(X_train, Y_train)
print("Predicting...")
y_pred = pipeline.predict(X_test)
print("Scoring...")
score = f1_score(Y_test, y_pred)
print(f"DecisionTree score : {score}")
y_score = pipeline.predict_proba(X_test)[:, 1]
```

```
Fitting...
Predicting...
Scoring...
DecisionTree score : 0.8968560397131825
```

In [ ]:
```
dT_results = X_test.copy()
dT_results["is_attack"] = y_pred == -1
```

In [ ]:
```
precision, recall, thresholds = precision_recall_curve(Y_test, y_score)
#create precision recall curve
fig, ax = plt.subplots()
ax.plot(recall, precision, color='purple')

#add axis labels to plot
ax.set_title('Precision-Recall Curve')
ax.set_ylabel('Precision')
ax.set_xlabel('Recall')

#display plot
plt.show()
```



Unsurpervised machine learning model

Isolation Forest

In [ ]:
```
from sklearn.ensemble import IsolationForest
pipeline = Pipeline(
    [
        ('transformer', transformer),
        ('classifier', IsolationForest(contamination=0.04, random_state=42))
    ]
)
```

In [ ]:
```
%time pipeline.fit(data["train"])
```

```
CPU times: user 832 ms, sys: 1.82 ms, total: 834 ms
Wall time: 863 ms
```

Out[ ]:
```
Pipeline(steps=[('transformer',
                 ColumnTransformer(transformers=[('onehotencoder',
```

```
                                          OneHotEncoder(handle_unknown='ignore'),
                                          ['Source_IP',
                                           'Destination_IP',
                                           'Source_Port',
                                           'Destination_Port',
                                           'Protocol', 'Flag',
                                           'Service_Type'])])),
                    ('classifier',
                     IsolationForest(contamination=0.04, random_state=42))])
```

In [ ]:

```python
if_attack = pipeline.predict(data["train"])
```

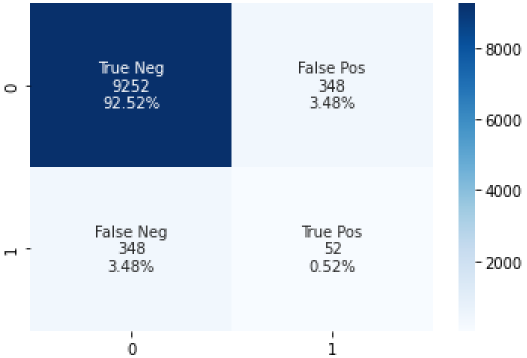
In [ ]:

```python
if_results = data["train"].copy()
if_results["if_attack"] = if_attack == -1
```

In [ ]:

```python
nb_attacks = len(data["train"][data["train"].Label == 1])
nb_genuine = len(data["train"][data["train"].Label == 0])
true_positive = len(if_results[(if_results["Label"] == 1) & (if_results["if_attack"] == True)])
false_positive = len(if_results[(if_results["Label"] == 0) & (if_results["if_attack"] == True)])
true_negative = len(if_results[(if_results["Label"] == 0) & (if_results["if_attack"] == False)])
false_negative = len(if_results[(if_results["Label"] == 1) & (if_results["if_attack"] == False)])
print("True positive : ", len(if_results[(if_results["Label"] == 1) & (if_results["if_attack"] == True)]), "/", nb_attacks)
print("False positive :" , len(if_results[(if_results["Label"] == 0) & (if_results["if_attack"] == True)]), "/", nb_genuine)
print("True negative :" , len(if_results[(if_results["Label"] == 0) & (if_results["if_attack"] == False)]), "/", nb_genuine)
print("False negative :" , len(if_results[(if_results["Label"] == 1) & (if_results["if_attack"] == False)]), "/", nb_attacks)

print_confusion_matrix(np.array([[true_negative, false_positive],[false_negative, true_positive]]))
```

```
True positive :  52 / 400
False positive : 348 / 9600
True negative : 9252 / 9600
False negative : 348 / 400
```



In [ ]:

```python
if_results.groupby(by=['if_attack', 'Label']).count()
```

Out[ ]:

		Source_IP	Destination_IP	Source_Port	Destination_Port	Protocol	Flag	Service_Type	timeDelta	transfered_ratio
if_attack	**Label**									

		Source_IP	Destination_IP	Source_Port	Destination_Port	Protocol	Flag	Service_Type	timeDelta	transfered_ratio
if_attack	**Label**									
False	**0**	9252	9252	9252	9252	9252	9252	9252	9252	9252
	1	348	348	348	348	348	348	348	348	348
True	**0**	348	348	348	348	348	348	348	348	348
	1	52	52	52	52	52	52	52	52	52

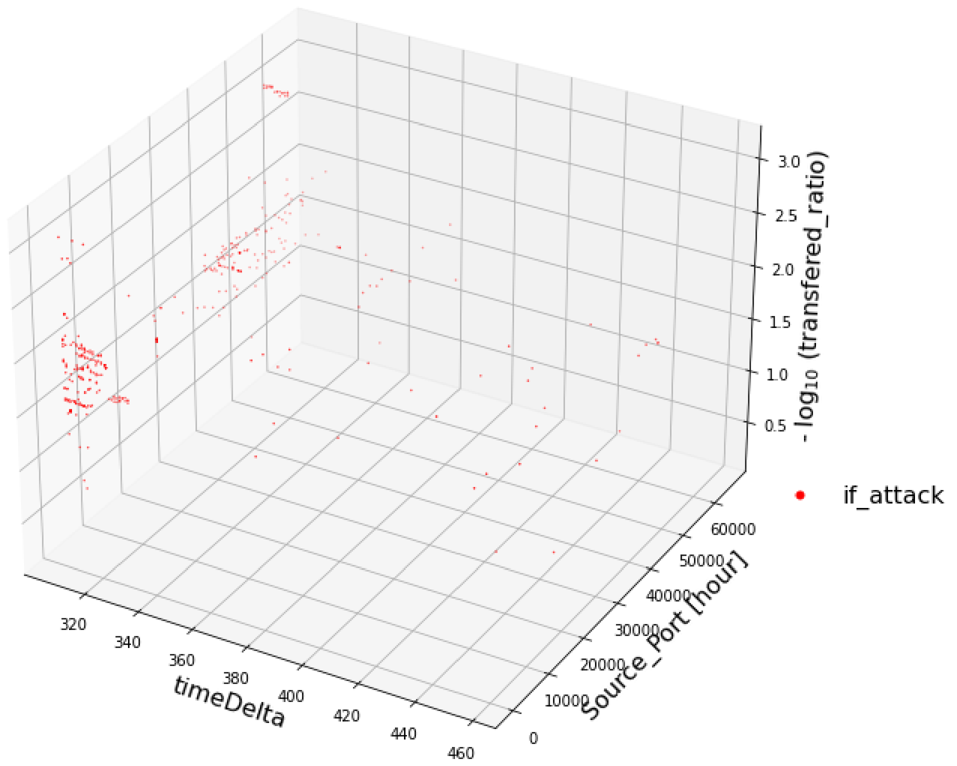
In [ ]:

```
def show3D_netflow_if_attack_only(transac_dataset, x_axis_name, y_axis_name, z_axis_name):
    X = transac_dataset.drop(columns=['if_attack'])
    Y = transac_dataset['if_attack']
    x = x_axis_name
    y = y_axis_name
    z = z_axis_name
    limit = len(X)
    sb.reset_orig()
    fig = plt.figure(figsize = (10, 12))
    ax = fig.add_subplot(111, projection='3d')

    ax.scatter(X.loc[Y == 1, x][:limit],
               X.loc[Y == 1, y][:limit],
               np.log10(X.loc[Y == 1, z][:limit]),
               c = 'r',
               marker = '.',
               s = 1,
               label = 'if_attack')

    ax.set_xlabel(x, size = 16)
    ax.set_ylabel(y + ' [hour]', size = 16)
    ax.set_zlabel('- log$_{10}$ (' + z + ')', size = 16)
    ax.set_title('Features separate for attacks netflow', size = 20)

    plt.axis('tight')
    ax.grid(1)

    fraudMarker = mlines.Line2D([], [], linewidth = 0, color = 'r', marker = '.', markersize = 10, label = 'if_attack')

    plt.legend(handles= [fraudMarker], bbox_to_anchor = (1.20, 0.38), frameon = False, prop = {'size': 16})
```

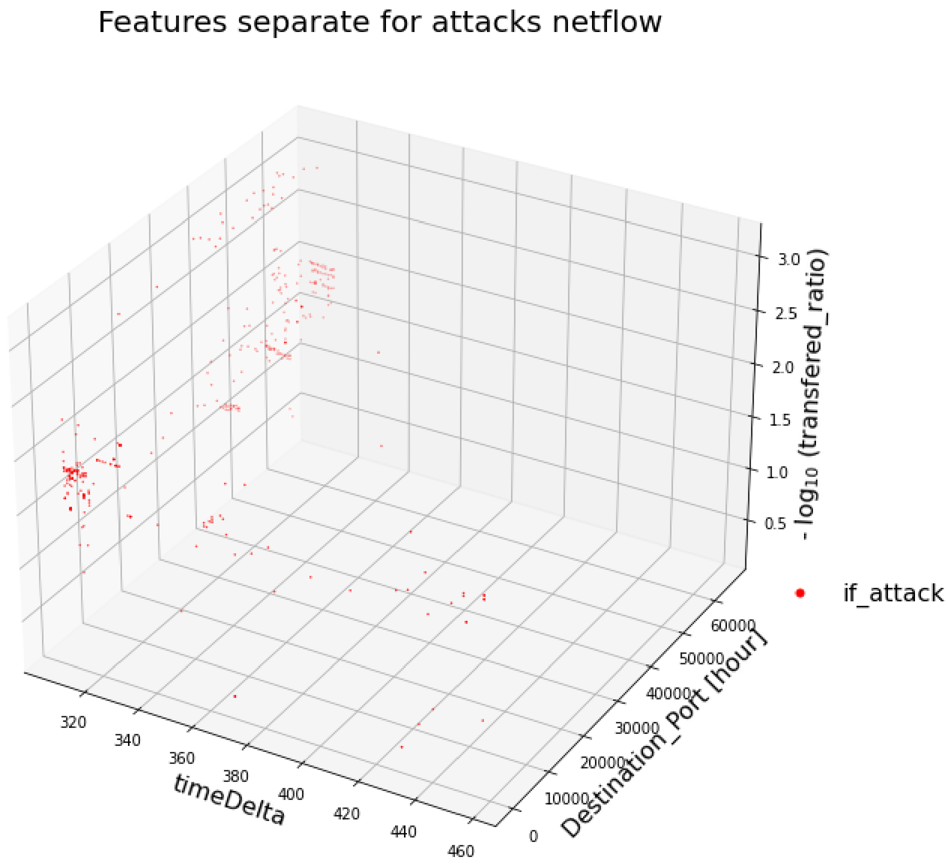
In [ ]:

```
show3D_netflow_if_attack_only(if_results, 'timeDelta', 'Source_Port', 'transfered_ratio')
```

Features separate for attacks netflow



In [ ]:
```
show3D_netflow_if_attack_only(if_results, 'timeDelta', 'Destination_Port', 'transfered_ratio')
```



Local Outlier Factor

In [ ]:

```
from sklearn.neighbors import LocalOutlierFactor
pipeline = Pipeline(
    [
        ('transformer', transformer),
        ('classifier', LocalOutlierFactor(contamination=0.04))
    ]
)
lof_attack = pipeline.fit_predict(data["train"])
```

In [ ]:

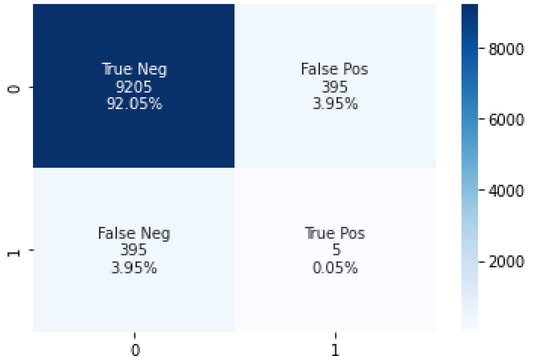
```
lof_results = data["train"].copy()
lof_results["lof_attack"] = lof_attack == -1
```

In [ ]:

```
nb_attacks = len(data["train"][data["train"].Label == 1])
nb_genuine = len(data["train"][data["train"].Label == 0])
true_positive = len(lof_results[(lof_results["Label"] == 1) & (lof_results["lof_attack"] == True)])
```

```
false_positive = len(lof_results[(lof_results["Label"] == 0) & (lof_results["lof_attack"] == True)])
true_negative = len(lof_results[(lof_results["Label"] == 0) & (lof_results["lof_attack"] == False)])
false_negative = len(lof_results[(lof_results["Label"] == 1) & (lof_results["lof_attack"] == False)])
print("True positive : ", len(lof_results[(lof_results["Label"] == 1) & (lof_results["lof_attack"] == True)]), "/", nb_attacks)
print("False positive :" , len(lof_results[(lof_results["Label"] == 0) & (lof_results["lof_attack"] == True)]), "/", nb_genuine)
print("True negative :" , len(lof_results[(lof_results["Label"] == 0) & (lof_results["lof_attack"] == False)]), "/", nb_genuine)
print("False negative :" , len(lof_results[(lof_results["Label"] == 1) & (lof_results["lof_attack"] == False)]), "/", nb_attacks)
print_confusion_matrix(np.array([[true_negative, false_positive],[false_negative, true_positive]]))
```

```
True positive :  5 / 400
False positive : 395 / 9600
True negative : 9205 / 9600
False negative : 395 / 400
```



In [ ]:

```
lof_results.groupby(by=['lof_attack', 'Label']).count()
```

Out[ ]:

lof_attack	Label	Source_IP	Destination_IP	Source_Port	Destination_Port	Protocol	Flag	Service_Type	timeDelta	transfered_ratio
False	**0**	9205	9205	9205	9205	9205	9205	9205	9205	9205
	1	395	395	395	395	395	395	395	395	395
True	**0**	395	395	395	395	395	395	395	395	395
	1	5	5	5	5	5	5	5	5	5

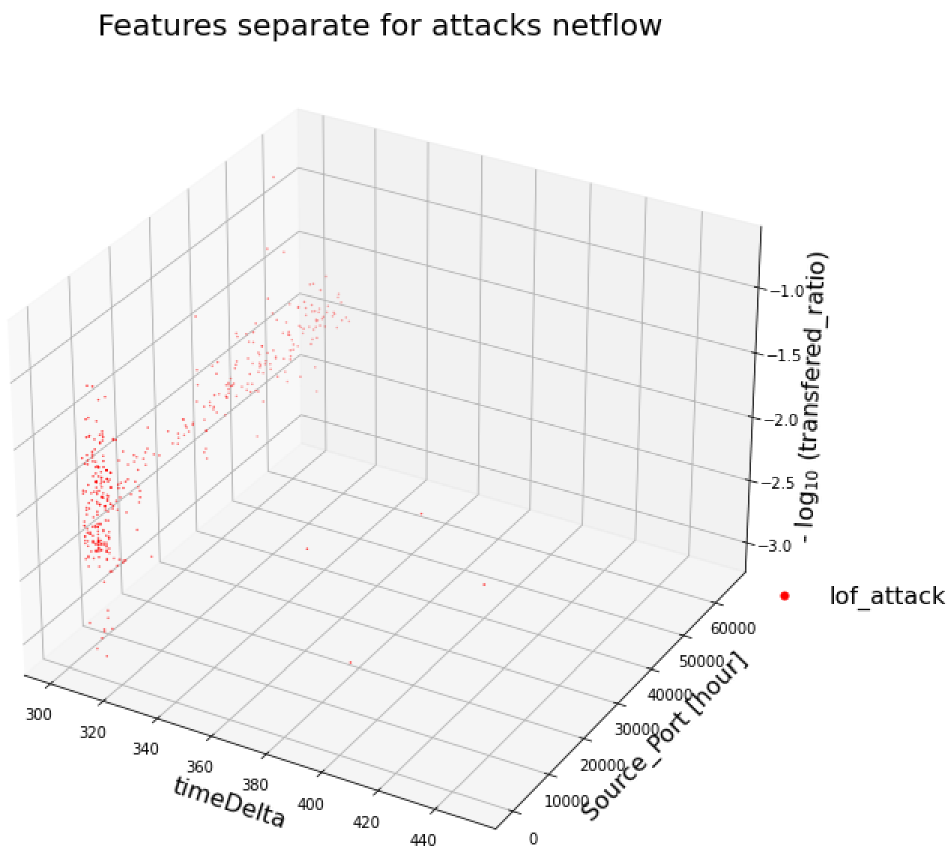
In [ ]:

```
def show3D_netflow_lof_attack_only(transac_dataset, x_axis_name, y_axis_name, z_axis_name):
    X = transac_dataset.drop(columns=['lof_attack'])
    Y = transac_dataset['lof_attack']
    x = x_axis_name
    y = y_axis_name
    z = z_axis_name
    limit = len(X)
    sb.reset_orig()
    fig = plt.figure(figsize = (10, 12))
    ax = fig.add_subplot(111, projection='3d')

    ax.scatter(X.loc[Y == 1, x][:limit],
               X.loc[Y == 1, y][:limit],
```

```python
            -np.log10(X.loc[Y == 1, z][:limit]),
            c = 'r',
            marker = '.',
            s = 1,
            label = 'lof_attack')

    ax.set_xlabel(x, size = 16)
    ax.set_ylabel(y + ' [hour]', size = 16)
    ax.set_zlabel('- log$_{10}$ (' + z + ')', size = 16)
    ax.set_title('Features separate for attacks netflow', size = 20)

    plt.axis('tight')
    ax.grid(1)

    fraudMarker = mlines.Line2D([], [], linewidth = 0, color = 'r', marker = '.', markersize = 10, label = 'lof_attack')

    plt.legend(handles= [fraudMarker], bbox_to_anchor = (1.20, 0.38), frameon = False, prop = {'size': 16})
```

In [ ]:

```python
show3D_netflow_lof_attack_only(lof_results, 'timeDelta', 'Source_Port', 'transfered_ratio')
```



In [ ]:

```
show3D_netflow_lof_attack_only(lof_results, 'timeDelta', 'Destination_Port', 'transfered_ratio')
```