Rule-Based Anomaly Detection

1) Problem Presentation
2) Demonstration
3) Opinion

Lisa Robert
Chirine Makhlouf
Alison Lopes-Sousa
Anissa Cedrati
Kevin Fonseca
Jordan Domingues

1) Problem presentation

The problem : **hackers trying to gain access to a website using a brute-force attack.**

The problem presented in this chapter is that we are facing the menace of hackers entering our website via a **brute-force attack**. They are using this technique which consists in using **many combinations of usernames and passwords until a combination works**. .

1) Problem presentation

To address this problem we will be using **anomaly detection** which is a process used to identify events, or observations that are different from what's considered normal or typical in a dataset. When a data point or event is identified as an anomaly it means that it **deviates from the usual pattern** or **behavior of the dataset**.
By noticing these anomalies we will be able to recognize a suspicious activity and stop it before the hackers get into the website.

To test our process we will be simulating attacks

2) Demonstration

Complete the following exercises to practice the concepts covered in this chapter:

1. Run the simulation for December 2018 into new log files without making the user base again. Be sure to run python3 simulate.py -h to review the command-line arguments. Set the seed to 27. This data will be used for the remaining exercises.

2. Find the number of unique usernames, attempts, successes, and failures, as well as the success/failure rates per IP address, using the data simulated from exercise 1.

3. Create two subplots with failures versus attempts on the left, and failure rate versus distinct usernames on the right. Draw decision boundaries for the resulting plots. Be sure to color each data point by whether or not it is a hacker IP address.

4. Build a rule-based criteria using the percentage difference from the median that flags an IP address if the failures and attempts are both five times their respective medians, or if the distinct usernames count is five times its median. Be sure to use a one-hour window. Remember to use the get_baselines() function to calculate the metrics needed for the baselines.

5. Calculate metrics to evaluate how well these rules performed using the evaluate() and classification_stats() functions from this chapter.

2) Demonstration

Lisa ROBERT

EXERCISE 1

Run the simulation for December 2018 into new log files without making the user base again. Be sure to run python3 simulate.py -h to review the command-line arguments. Set the seed to 27. This data will be used for the remaining exercises.

Entrée [6]:

```
!python3 simulate.py \
    -s 27 \
    -u user_data/user_base.txt \
    -i user_data/user_ips.json \
    -l dec_2018_log.csv \
    -hl dec_2018_attacks.csv \
    31 "2018-12-01"
```

```
[INFO] [ simulate.py ] Simulating 31.0 days...
[INFO] [ simulate.py ] Saving logs
[INFO] [ simulate.py ] All done!
```

Entrée [7]:

```
%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import seaborn as sns
```

Alison LOPES SOUSA

2) Demonstration

EXERCISE 2

Find the number of unique usernames, attempts, successes, and failures, as well as the success/failure rates per IP address, using the data simulated from exercise 1.

Entrée [8]:

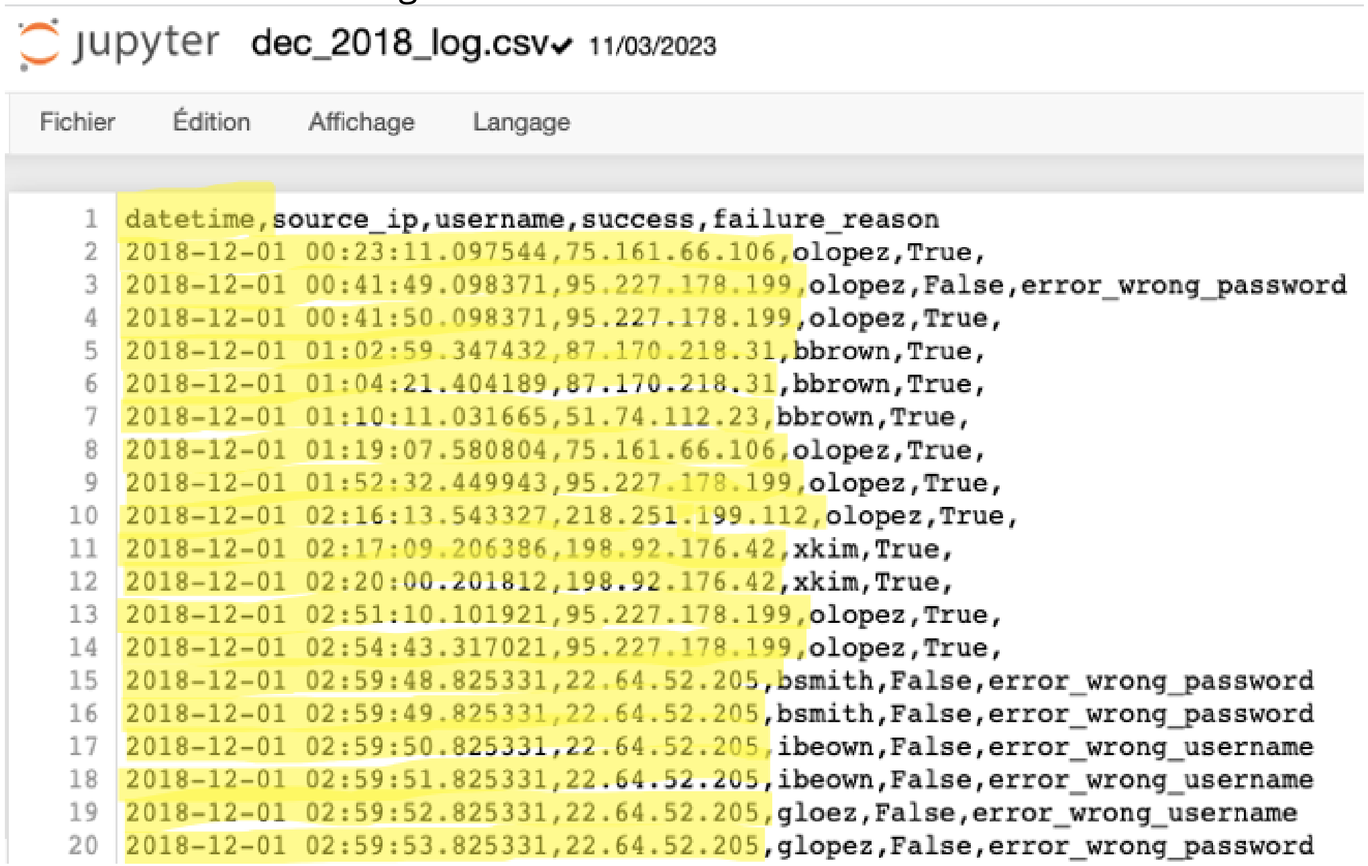
```
dec_log = pd.read_csv('dec_2018_log.csv', parse_dates=True, index_col='datetime')

log_aggs = dec_log.assign(
    failures=lambda x: np.invert(x.success)
).groupby('source_ip').agg(
    {'username': 'nunique', 'success': 'sum', 'failures': 'sum'}
).assign(
    attempts=lambda x: x.success + x.failures,
    success_rate=lambda x: x.success / x.attempts,
    failure_rate=lambda x: 1 - x.success_rate
).dropna().reset_index()

log_aggs.head()
```

Alison LOPES SOUSA

2) Demonstration

EXERCISE 2

Find the number of unique usernames, attempts, successes, and failures, as well as the success/failure rates per IP address, using the data simulated from exercise 1.


jupyter dec_2018_log.csv 11/03/2023

Fichier Édition Affichage Langage

```
1  datetime,source_ip,username,success,failure_reason
2  2018-12-01 00:23:11.097544,75.161.66.106,olopez,True,
3  2018-12-01 00:41:49.098371,95.227.178.199,olopez,False,error_wrong_password
4  2018-12-01 00:41:50.098371,95.227.178.199,olopez,True,
5  2018-12-01 01:02:59.347432,87.170.218.31,bbrown,True,
6  2018-12-01 01:04:21.404189,87.170.218.31,bbrown,True,
7  2018-12-01 01:10:11.031665,51.74.112.23,bbrown,True,
8  2018-12-01 01:19:07.580804,75.161.66.106,olopez,True,
9  2018-12-01 01:52:32.449943,95.227.178.199,olopez,True,
10 2018-12-01 02:16:13.543327,218.251.199.112,olopez,True,
11 2018-12-01 02:17:09.206386,198.92.176.42,xkim,True,
12 2018-12-01 02:20:00.201812,198.92.176.42,xkim,True,
13 2018-12-01 02:51:10.101921,95.227.178.199,olopez,True,
14 2018-12-01 02:54:43.317021,95.227.178.199,olopez,True,
15 2018-12-01 02:59:48.825331,22.64.52.205,bsmith,False,error_wrong_password
16 2018-12-01 02:59:49.825331,22.64.52.205,bsmith,False,error_wrong_password
17 2018-12-01 02:59:50.825331,22.64.52.205,ibeown,False,error_wrong_username
18 2018-12-01 02:59:51.825331,22.64.52.205,ibeown,False,error_wrong_username
19 2018-12-01 02:59:52.825331,22.64.52.205,gloez,False,error_wrong_username
20 2018-12-01 02:59:53.825331,22.64.52.205,glopez,False,error_wrong_password
```

Alison LOPES SOUSA

2) Demonstration

EXERCISE 2

Find the number of unique usernames, attempts, successes, and failures, as well as the success/failure rates per IP address, using the data simulated from exercise 1.

Out[8]:

	source_ip	username	success	failures	attempts	success_rate	failure_rate
0	1.138.149.116	1	14	0	14	1.000000	0.000000
1	1.54.45.23	49	17	68	85	0.200000	0.800000
2	100.43.18.36	1	7	0	7	1.000000	0.000000
3	101.113.31.197	1	10	1	11	0.909091	0.090909
4	101.154.143.93	1	30	5	35	0.857143	0.142857

2) Demonstration

EXERCISE 3

Create two subplots with failures versus attempts on the left, and failure rate versus distinct usernames on the right. Draw decision boundaries for the resulting plots. Be sure to color each data point by whether or not it is a hacker IP address.

Entrée [9]:

```
is_attack_ip = log_aggs.source_ip.isin(
    pd.read_csv('dec_2018_attacks.csv').source_ip
)

fig, axes = plt.subplots(1, 2, figsize=(10, 3))

for ax, (x, y) in zip(axes, (('attempts', 'failures'), ('username', 'failure_rate'))):
    ax = sns.scatterplot(
        x=log_aggs[x],
        y=log_aggs[y],
        hue=is_attack_ip,
        ax=ax
    )
    ax.set_title(f'{y.title()} vs. {x.title()}')

# boundaries
axes[0].plot([0, 80], [80, 0], 'r--')
axes[1].axhline(0.5, color='red', linestyle='--')
```

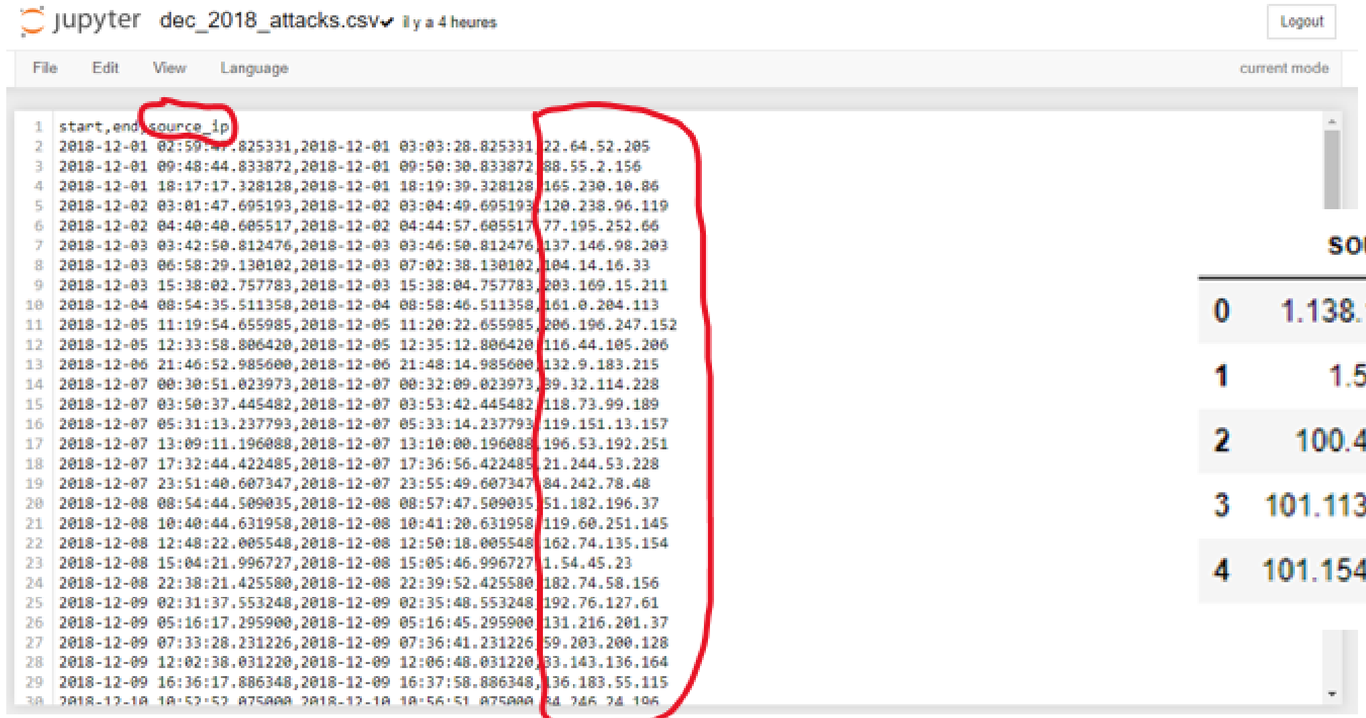
Out[9]: <matplotlib.lines.Line2D at 0x7f949d3c0370>

Chirine MAKHLOUF

2) Demonstration

EXERCISE 3

Create two subplots with failures versus attempts on the left, and failure rate versus distinct usernames on the right. Draw decision boundaries for the resulting plots. Be sure to color each data point by whether or not it is a hacker IP address.




```
start,end,source_ip
2018-12-01 02:50:47.825331,2018-12-01 03:03:28.825331,22.64.52.205
2018-12-01 09:48:44.833872,2018-12-01 09:50:30.833872,88.55.2.156
2018-12-01 18:17:17.328128,2018-12-01 18:19:39.328128,165.230.10.86
2018-12-02 03:01:47.695193,2018-12-02 03:04:49.695193,120.238.96.119
2018-12-02 04:40:40.605517,2018-12-02 04:44:57.605517,77.195.252.66
2018-12-03 03:42:50.812476,2018-12-03 03:46:50.812476,137.146.98.203
2018-12-03 06:58:29.130102,2018-12-03 07:02:38.130102,104.14.16.33
2018-12-03 15:38:02.757783,2018-12-03 15:38:04.757783,203.169.15.211
2018-12-04 08:54:35.511358,2018-12-04 08:58:46.511358,161.0.204.113
2018-12-05 11:19:54.655985,2018-12-05 11:20:22.655985,206.196.247.152
2018-12-05 12:33:58.806420,2018-12-05 12:35:12.806420,116.44.105.206
2018-12-06 21:46:52.985600,2018-12-06 21:48:14.985600,132.9.183.215
2018-12-07 00:30:51.023973,2018-12-07 00:32:09.023973,89.32.114.228
2018-12-07 03:50:37.445482,2018-12-07 03:53:42.445482,118.73.99.189
2018-12-07 05:31:13.237793,2018-12-07 05:33:14.237793,119.151.13.157
2018-12-07 13:09:11.196088,2018-12-07 13:10:00.196088,196.53.192.251
2018-12-07 17:32:44.422485,2018-12-07 17:36:56.422485,21.244.53.228
2018-12-07 23:51:40.607347,2018-12-07 23:55:49.607347,84.242.78.48
2018-12-08 08:54:44.509035,2018-12-08 08:57:47.509035,51.182.196.37
2018-12-08 10:40:44.631958,2018-12-08 10:41:20.631958,119.60.251.145
2018-12-08 12:48:22.005548,2018-12-08 12:50:18.005548,162.74.135.154
2018-12-08 15:04:21.996727,2018-12-08 15:05:46.996727,1.54.45.23
2018-12-08 22:38:21.425580,2018-12-08 22:39:52.425580,182.74.58.156
2018-12-09 02:31:37.553248,2018-12-09 02:35:48.553248,192.76.127.61
2018-12-09 05:16:17.295900,2018-12-09 05:16:45.295900,131.216.201.37
2018-12-09 07:33:28.231226,2018-12-09 07:36:41.231226,59.203.200.128
2018-12-09 12:02:38.031220,2018-12-09 12:06:48.031220,33.143.136.164
2018-12-09 16:36:17.886348,2018-12-09 16:37:58.886348,136.183.55.115
2018-12-10 10:52:52.075000,2018-12-10 10:56:51.075000,84.246.24.196
```

log_aggs:

	source_ip	username	success	failures	attempts	success_rate	failure_rate
0	1.138.149.116	1	14	0	14	1.000000	0.000000
1	1.54.45.23	49	17	68	85	0.200000	0.800000
2	100.43.18.36	1	7	0	7	1.000000	0.000000
3	101.113.31.197	1	10	1	11	0.909091	0.090909
4	101.154.143.93	1	30	5	35	0.857143	0.142857

2) Demonstration

EXERCISE 3

Create two subplots with failures versus attempts on the left, and failure rate versus distinct usernames on the right. Draw decision boundaries for the resulting plots. Be sure to color each data point by whether or not it is a hacker IP address.

Entrée [9]:

```
is_attack_ip = log_aggs.source_ip.isin(
    pd.read_csv('dec_2018_attacks.csv').source_ip
)

fig, axes = plt.subplots(1, 2, figsize=(10, 3))

for ax, (x, y) in zip(axes, (('attempts', 'failures'), ('username', 'failure_rate'))):
    ax = sns.scatterplot(
        x=log_aggs[x],
        y=log_aggs[y],
        hue=is_attack_ip,
        ax=ax
    )
    ax.set_title(f'{y.title()} vs. {x.title()}')

# boundaries
axes[0].plot([0, 80], [80, 0], 'r--')
axes[1].axhline(0.5, color='red', linestyle='--')
```

Out[9]: <matplotlib.lines.Line2D at 0x7f949d3c0370>

2) Demonstration

EXERCISE 3

Create two subplots with failures versus attempts on the left, and failure rate versus distinct usernames on the right. Draw decision boundaries for the resulting plots. Be sure to color each data point by whether or not it is a hacker IP address.

	source_ip	username	success	failures	attempts	success_rate	failure_rate
0	1.138.149.116	1	14	0	14	1.000000	0.000000
1	1.54.45.23	49	17	68	85	0.200000	0.800000
2	100.43.18.36	1	7	0	7	1.000000	0.000000
3	101.113.31.197	1	10	1	11	0.909091	0.090909
4	101.154.143.93	1	30	5	35	0.857143	0.142857

2) Demonstration

EXERCISE 3

Create two subplots with failures versus attempts on the left, and failure rate versus distinct usernames on the right. Draw decision boundaries for the resulting plots. Be sure to color each data point by whether or not it is a hacker IP address.

Entrée [9]:

```
is_attack_ip = log_aggs.source_ip.isin(
    pd.read_csv('dec_2018_attacks.csv').source_ip
)

fig, axes = plt.subplots(1, 2, figsize=(10, 3))

for ax, (x, y) in zip(axes, (('attempts', 'failures'), ('username', 'failure_rate'))):
    ax = sns.scatterplot(
        x=log_aggs[x],
        y=log_aggs[y],
        hue=is_attack_ip,
        ax=ax
    )
    ax.set_title(f'{y.title()} vs. {x.title()}')

# boundaries
axes[0].plot([0, 80], [80, 0], 'r--')
axes[1].axhline(0.5, color='red', linestyle='--')
```

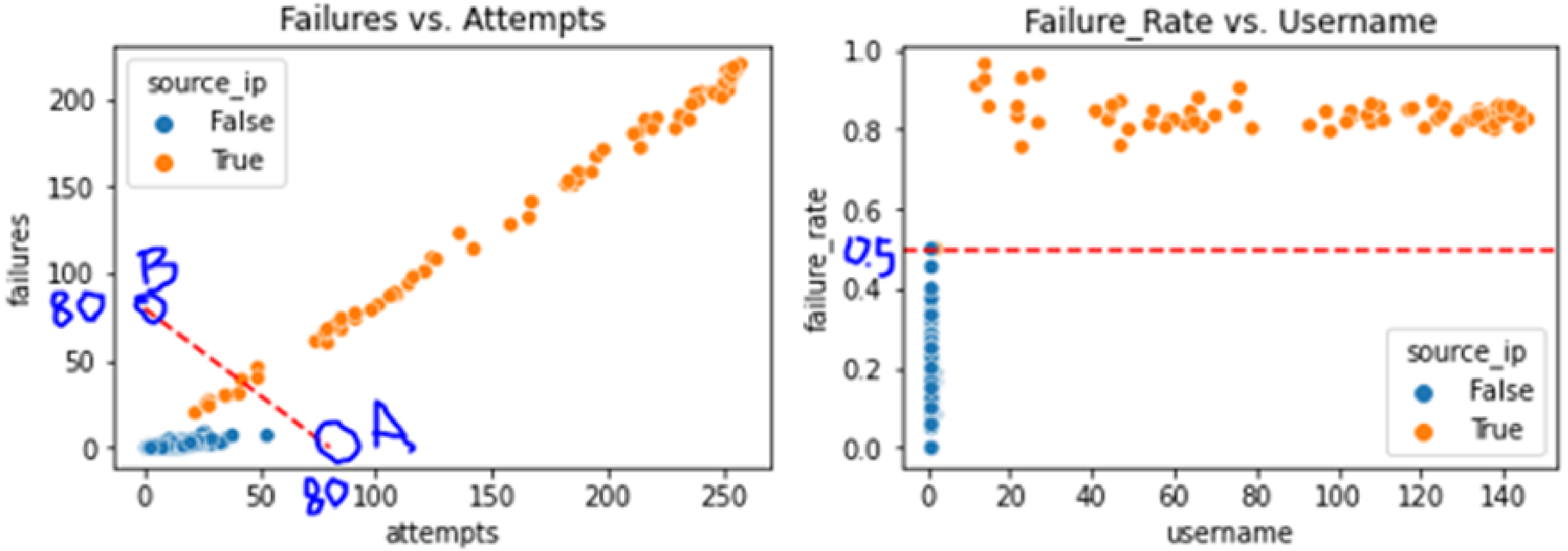
Out[9]: <matplotlib.lines.Line2D at 0x7f949d3c0370>

Chirine MAKHLOUF

2) Demonstration

EXERCISE 3

Create two subplots with failures versus attempts on the left, and failure rate versus distinct usernames on the right. Draw decision boundaries for the resulting plots. Be sure to color each data point by whether or not it is a hacker IP address.



2) Demonstration

EXERCISE 4

Build a rule-based criteria using the percentage difference from the median that flags an IP address if the failures and attempts are both five times their respective medians, or if the distinct usernames count is five times its median. Be sure to use a one-hour window. Remember to use the get_baselines() function to calculate the metrics needed for the baselines.

Entrée [10]:

```
hourly_ip_logs = dec_log.assign(
    failures=lambda x: np.invert(x.success)
).groupby('source_ip').resample('1H').agg(
    {'username': 'nunique', 'success': 'sum', 'failures': 'sum'}
).assign(
    attempts=lambda x: x.success + x.failures,
    success_rate=lambda x: x.success / x.attempts,
    failure_rate=lambda x: 1 - x.success_rate
).dropna().reset_index()

hourly_ip_logs.head()
```

Out[10]:

	source_ip	datetime	username	success	failures	attempts	success_rate	failure_rate
0	1.138.149.116	2018-12-11 13:00:00	1	3	0	3	1.0	0.0
1	1.138.149.116	2018-12-11 14:00:00	1	4	0	4	1.0	0.0
2	1.138.149.116	2018-12-15 20:00:00	1	2	0	2	1.0	0.0
3	1.138.149.116	2018-12-17 23:00:00	1	2	0	2	1.0	0.0
4	1.138.149.116	2018-12-18 00:00:00	1	1	0	1	1.0	0.0

2) Demonstration

EXERCISE 4

Build a rule-based criteria using the percentage difference from the median that flags an IP address if the failures and attempts are both five times their respective medians, or if the distinct usernames count is five times its median. Be sure to use a one-hour window. Remember to use the get_baselines() function to calculate the metrics needed for the baselines.

Entrée [11]:

```python
def get_baselines(hourly_ip_logs, func, *args, **kwargs):
    """
    Calculate hourly bootstrapped statistic per column.

    Parameters:
        - hourly_ip_logs: Data to sample from.
        - func: Statistic to calculate.
        - args: Additional positional arguments for `func`
        - kwargs: Additional keyword arguments for `func`

    Returns:
        `pandas.DataFrame` of hourly bootstrapped statistics
    """
    if isinstance(func, str):
        func = getattr(pd.DataFrame, func)

    return hourly_ip_logs.assign(
        hour=lambda x: x.datetime.dt.hour
    ).groupby('hour').apply(
        lambda x: x.sample(10, random_state=0, replace=True).pipe(func, *args, **kwargs, numeric_only=True)
    )
```

2) Demonstration

EXERCISE 4

Build a rule-based criteria using the percentage difference from the median that flags an IP address if the failures and attempts are both five times their respective medians, or if the distinct usernames count is five times its median. Be sure to use a one-hour window. Remember to use the get_baselines() function to calculate the metrics needed for the baselines.

Entrée [12]:

```
medians = get_baselines(hourly_ip_logs, 'median')
```

Entrée [13]:

```
flagged_ips = hourly_ip_logs.assign(
    hour=lambda x: x.datetime.dt.hour
).join(
    medians, on='hour', rsuffix='_median'
).assign(
    flag_median=lambda x: np.logical_or(
        np.logical_and(
            x.failures_median * 5 <= x.failures,
            x.attempts_median * 5 <= x.attempts
        ), x.username_median * 5 <= x.username
    )
).query('flag_median').source_ip.drop_duplicates()
```

2) Demonstration

Lisa ROBERT

EXERCISE 5

Calculate metrics to evaluate how well these rules performed using the evaluate() and classification_stats() functions from this chapter.

Entrée [14]:

```python
def evaluate(alerted_ips, attack_ips, log_ips):
    """
    Calculate true positives (TP), false positives (FP),
    true negatives (TN), and false negatives (FN) for
    IP addresses flagged as suspicious.

    Parameters:
        - alerted_ips: `pandas.Series` of flagged IP addresses
        - attack_ips: `pandas.Series` of attacker IP addresses
        - log_ips: `pandas.Series` of all IP addresses seen

    Returns:
        Tuple of form (TP, FP, TN, FN)
    """
    tp = alerted_ips.isin(attack_ips).sum()
    tn = np.invert(np.isin(log_ips[~log_ips.isin(alerted_ips)].unique(), attack_ips)).sum()
    fp = np.invert(
        np.isin(log_ips[log_ips.isin(alerted_ips)].unique(), attack_ips)
    ).sum()
    fn = np.invert(
        np.isin(log_ips[log_ips.isin(attack_ips)].unique(), alerted_ips)
    ).sum()
    return tp, fp, tn, fn
```

2) Demonstration

Lisa ROBERT

EXERCISE 5

Calculate metrics to evaluate how well these rules performed using the evaluate() and classification_stats() functions from this chapter.

Entrée [15]:

```
# make this easier to call
from functools import partial
scores = partial(
    evaluate,
    attack_ips=pd.read_csv('dec_2018_attacks.csv').source_ip,
    log_ips=dec_log.source_ip.drop_duplicates()
)
```

Entrée [16]:

```
def classification_stats(tp, fp, tn, fn):
    """Calculate metrics"""
    return {
        'FPR': fp / (fp + tn),
        'FDR': fp / (fp + tp),
        'FNR': fn / (fn + tp),
        'FOR': fn / (fn + tn)
    }
```

Entrée [17]:

```
classification_stats(*scores(flagged_ips))
```

Out[17]:

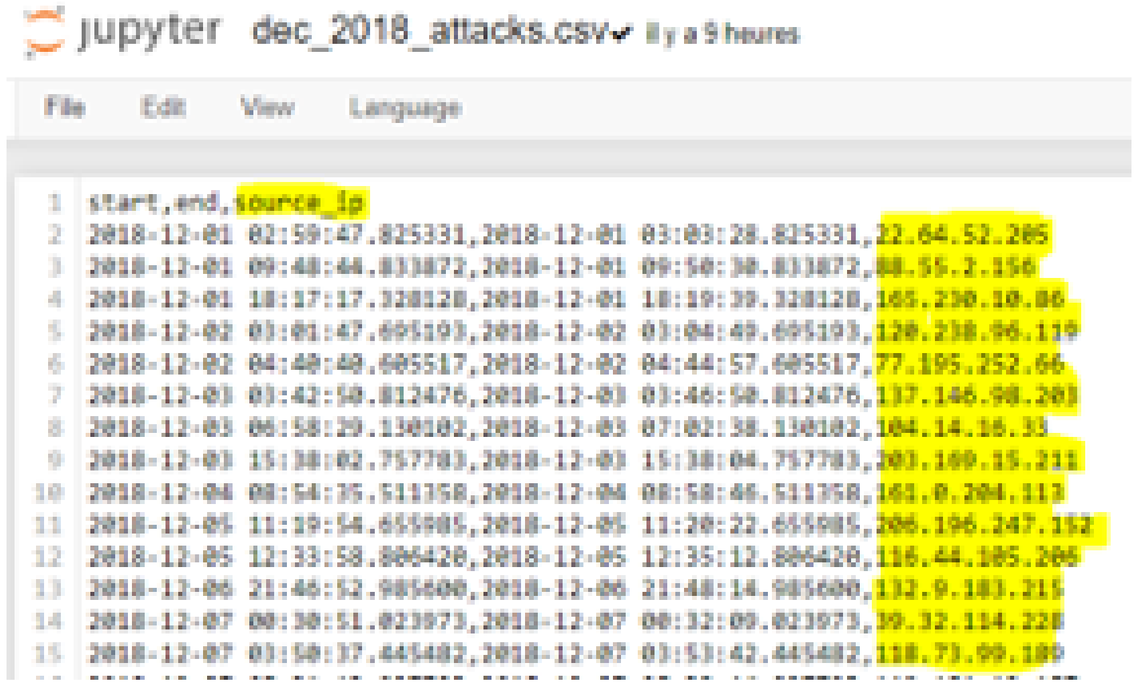
```
{'FPR': 0.07003891050583658,
 'FDR': 0.18947368421052632,
 'FNR': 0.01282051282051282,
 'FOR': 0.004166666666666667}
```

2) Demonstration

Lisa ROBERT

EXERCISE 5

Calculate metrics to evaluate how well these rules performed using the evaluate() and classification_stats() functions from this chapter.



3) Opinion

Jordan DOMINGUES
Kévin FONSECA

Intellectual aspect

- Difficulties understanding the exercice.
- Confusing instructions & difficult to follow.
- Programming & coding are unkown areas for us.
- No initiation& training in these fields.
- Poor knowledge.
- Connection with our training / Master ?

Technical aspect

- Problems of exploitation.
- Different operating systems.
- Not the same ease of use.
- Exercice should have been shared directly and simultaneously.
- Problems ordering & recording on the program.

THANK YOU FOR YOUR ATTENTION