Environment Setup
We will be using JupyterLab within a Python virtual environment.

1. **Python Installation** First, you need to install Python on your machine. If you already have Python installed, please ensure it's the correct version (>=3.9). You can check your Python version by opening your terminal and typing python --version. If you do not have Python installed or your version is outdated, you can download Python 3.9 from the official Python website: https://www.python.org/downloads/ Follow the instructions provided by the installer. Make sure to check the box that says "Add Python 3.9 to PATH" before you click "Install Now".

2. **Setting up a Python Virtual Environment** Once Python is installed, you need to set up a virtual environment. This is a self-contained environment that allows you to keep the dependencies required by different projects separate. To create a virtual environment, open your terminal or powershell and navigate to the directory where you want to create the environment.

   Then, run the following command: **python3 -m venv myenv** (replace "myenv" with the name you want to give to your environment). Depending on your system, “python3” might not be recognized, in that case try with “python” or “py”. To activate the virtual environment, navigate into the newly created directory and run the following command:

   - On Windows: **myenv\Scripts\activate**
   - On MacOS/Linux: source **myenv/bin/activate**

You will know that your virtual environment is activated because its name will appear in parentheses at the start of your terminal line.

3. **Installing Required Libraries** With your virtual environment activated, you can now install the required libraries by running this command:

   pip install pandas==2.1.0 numpy==1.25.2 matplotlib==3.7.2 statsmodels==0.14.0 scikit-learn==1.3.0 jupyterlab==4.0.6 ipympl==0.9.3

Please wait for the installation process to complete. It might take a few minutes depending on your internet connection.

4. **Launching JupyterLab** Once all the installations are complete, you can launch JupyterLab by typing **jupyter lab** in your terminal. This will open a new tab in your default web browser with the JupyterLab interface. If you are new to JupyterLab, it's a web-based interactive development environment for Jupyter notebooks, code, and data. You can create and manage notebooks, which are documents that can contain both code (e.g., python or R) and rich text elements (paragraphs, equations, figures, links, etc.).

Please ensure that you complete these steps before the workshop. This will allow us to focus on the machine learning models for anomaly detection in network traffic scenarios, rather than on environment setup. If you encounter any issues during the setup, please do not hesitate to reach out for assistance.

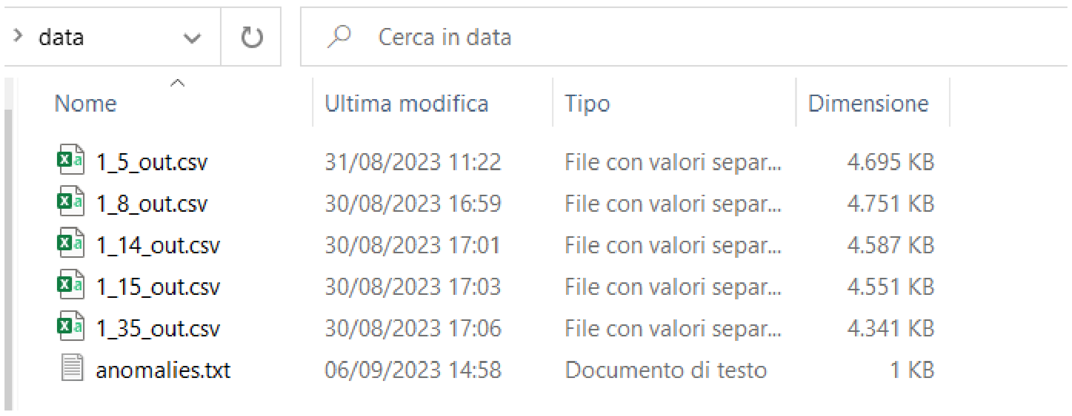
Target

Our target is to develop a procedure based on a machine learning model for anomaly detection in a real-life network traffic scenario.
On an average network we may have a few (1-10) devices and each one of them may have about one hundred ports. Each port is used for a different purpose, so its traffic may follow a different pattern, this means that we can't use the same ML model for everything. In fact, we need to train and maintain two different instances of model for each port, one for its inbound and one for its outbound traffic.
We will study the data coming from just a few ports, train and apply on them two or three ML models and discuss results. Once the raw data is preprocessed it's easy to apply a scikit-learn model, but there won't be a perfect model that outperforms all others in every scenario. So a key aspect of the workshop is being able to plot and understand how each model works and how to interpret its results.

Dataset

The data comes from one of our network monitoring software named Croono. It stores network traffic data. Since its size can get quite big and the data is mostly used for cool charts, it doesn't keep all of it. In fact, Croono has a housekeeping routine that reduces data resolution over time and stores it all on a Round Robin Database (RDD). A RDD is a fixed size database often used to store time series data, it relies on a circular buffer that constantly overwrites new data over old one.

This system is good for charts but far from perfect for ML purposes. Luckily, Croono provides an API which, given a port, returns the sum of inbound and outbound bytes exchanged in the previous 5 minutes. Since the goal is to detect anomalies in the range of 1-2 hours, five minutes is an acceptable precision.



Nome	Ultima modifica	Tipo	Dimensione
1_5_out.csv	31/08/2023 11:22	File con valori separ...	4.695 KB
1_8_out.csv	30/08/2023 16:59	File con valori separ...	4.751 KB
1_14_out.csv	30/08/2023 17:01	File con valori separ...	4.587 KB
1_15_out.csv	30/08/2023 17:03	File con valori separ...	4.551 KB
1_35_out.csv	30/08/2023 17:06	File con valori separ...	4.341 KB
anomalies.txt	06/09/2023 14:58	Documento di testo	1 KB

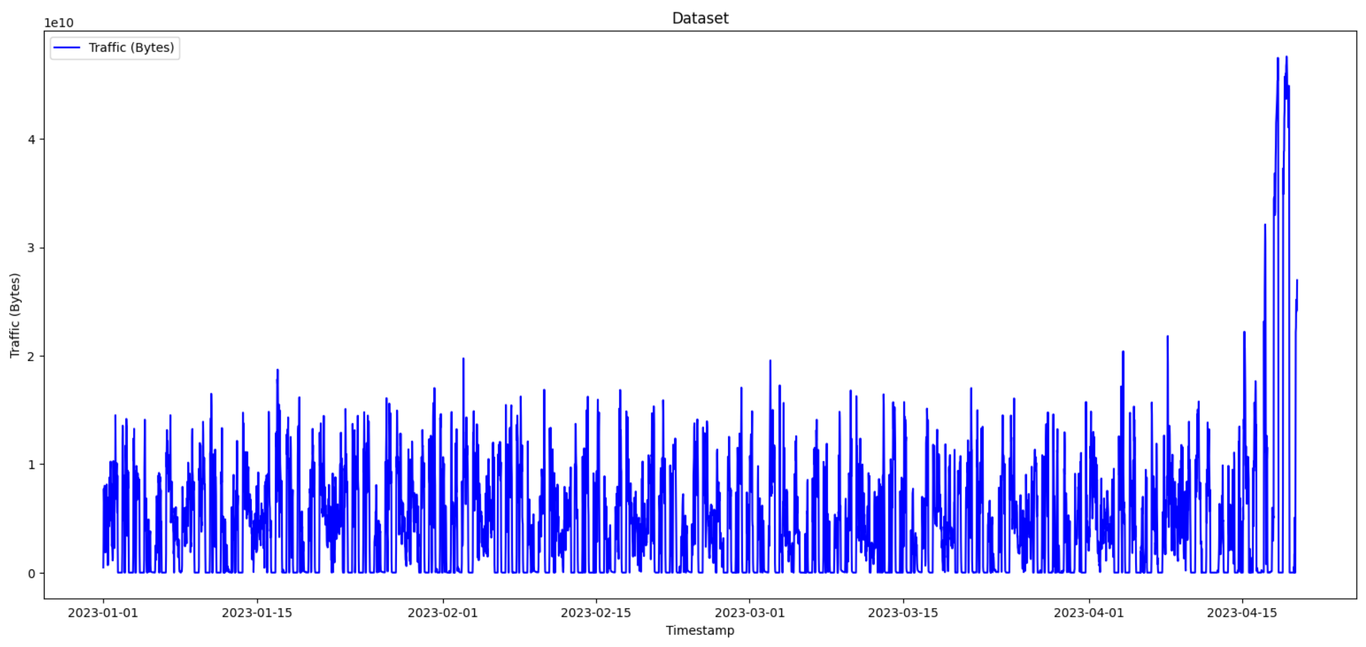
In this workshop we are going to study the outbound traffic coming from five ports of one of our internal network devices from 7/2022 to 8/2023. The dataset is composed by five csv files, one for each port, containing three columns: index, timestamp and network traffic represented in bytes. There is also a single text file named anomalies.txt which contains a list of dates on which anomalies occurred throughout the five ports.

Each csv can be loaded into a pandas dataframe using:

```
df = pd.read_csv("data/1_8_out.csv", index_col=0)
```

Since their size is about 4.5MB each, is better to work on small subsets while testing, for example:

```
df["ts"] = pd.to_datetime(df["ts"])
df = df[(df["ts"] >= "2023-01-01")&(df["ts"] <= "2023-04-19")]
df.plot(x="ts", y="traffic")
```



Preprocessing

The data is very noisy for two main reasons:

- There are many traffic spikes, which are perfectly normal in a real-life scenario but are annoying for anomaly detection purposes since they stand out as outliers.
- Measuring errors, exchanged bytes are measured by low level counters that often reset and output wrong or null values.

We proceed with these preprocessing steps:

1. Fill missing values with zeroes.

2. Resampling using a larger bucket: instead of having a point every 5 minutes is useful to have one every n minutes (where n>5) which contains the sum of the underlying point. This is done both to improve performance by reducing the number of points and to mitigate outliers.

3. Extreme outliers removal: a rolling window of about 2 hours is used to compute the z-score of each resulting point and those with a score greater than 2 are replaced with zeroes. This removes short spikes and most measuring error, rendering a nice continuous line.
   What is z-score?

$$z = \frac{x - \mu}{\sigma}$$

where:

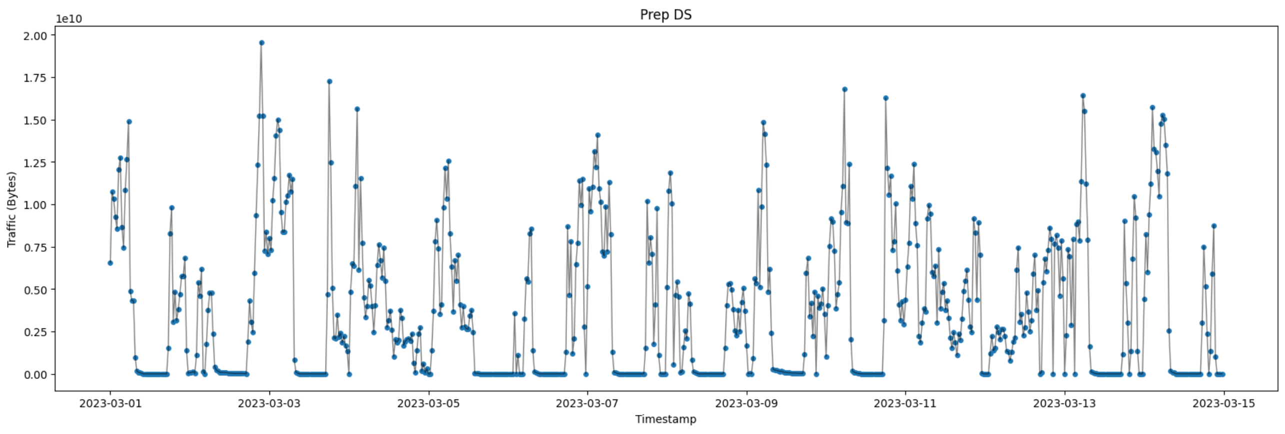
μ is the mean of the population,

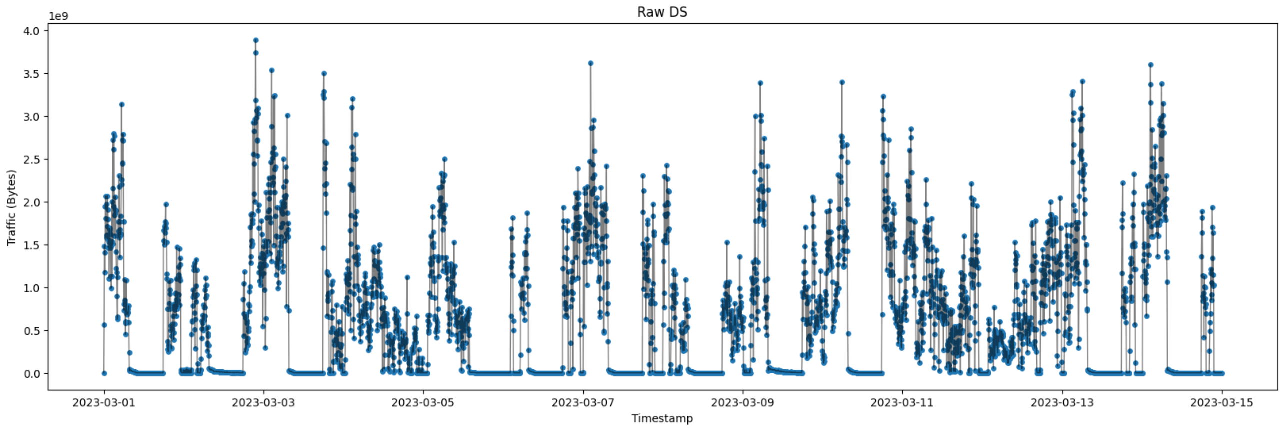
σ is the standard deviation of the population.

4. Scaling: each port's data is scaled using its own scaler, which needs to be stored somewhere. This is done to make any ML algorithm results comparable, even across different ports with different scales.

5. Adding features: depending on the complexity of the model this can be useful or not. For example we can add an integer in [0, 23] representing the hour or an integer in [0, 6] representing the week day.

Here’s an example of preprocessing on two weeks of data, using a resampling bucket of 30m:





This is a good moment for splitting the data into training and test sets.

ML Model

As stated before, in real life two different instance of ML model must be trained for each port, one for its inbound and one for its outbound traffic. This means maintaining something like a few hundred models. Today we will train just a few but it's good to keep performance in mind.

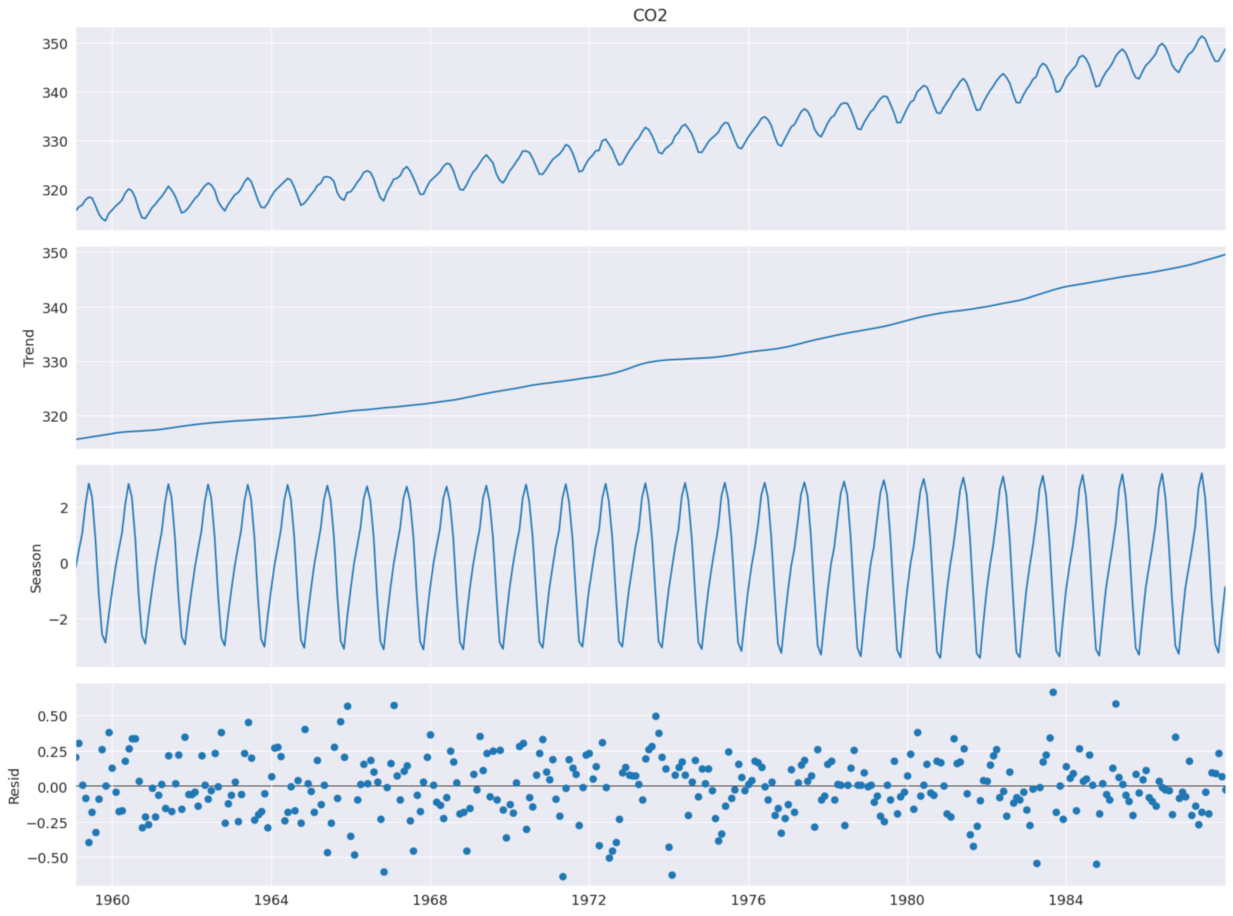
It makes sense to run a training job every 24h in order to digest new data and forget the old one. In fact, depending on the model, it may not be a good idea to feed the model data which is too old because traffic patterns change over time and performance could degenerate.

There will be another job which run much more frequently, for example every hour, which will apply the models on the most recent data and look for anomalies. So even if today we will apply each model to a big test set, in production the model is applied continuously to the most recent few data points. Especially in the case of forecasting algorithms, this does make a difference.

Finding a good ML algorithm to use is a matter of trial and error. In this case we can see two main approaches:

- Forecasting: we use the algorithm to predict the most recent portion of traffic and then compare its predictions with the real data, points that differs too much will be marked as anomalies.

    - STL (Seasonal and Trend decomposition using LOESS) is a decomposition-based technique that separates time series data into its underlying components, including seasonality, trend, and residual. By isolating these components, STL facilitates more accurate forecasting and trend analysis, making it valuable for time series data with complex patterns. It's limited to univariate time series.

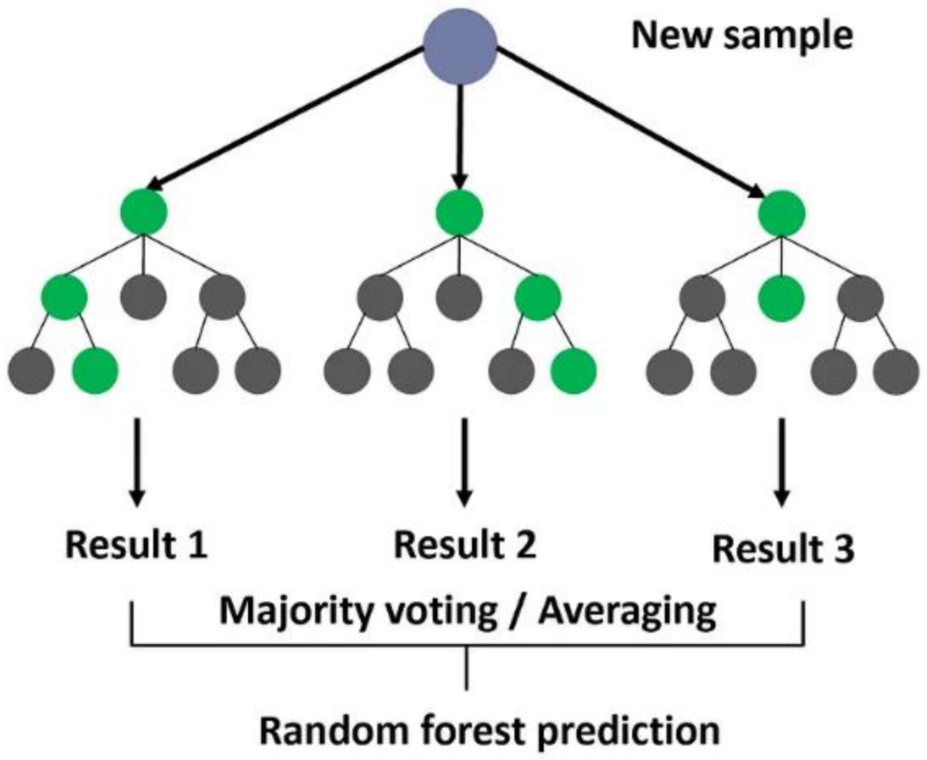


- SARIMAX (Seasonal Autoregressive Integrated Moving Average with Exogenous Variables) can capture complex time series patterns, including seasonality and trends. It can incorporate exogenous variables but requires extensive preprocessing and finetuning and may struggle with highly irregular data.

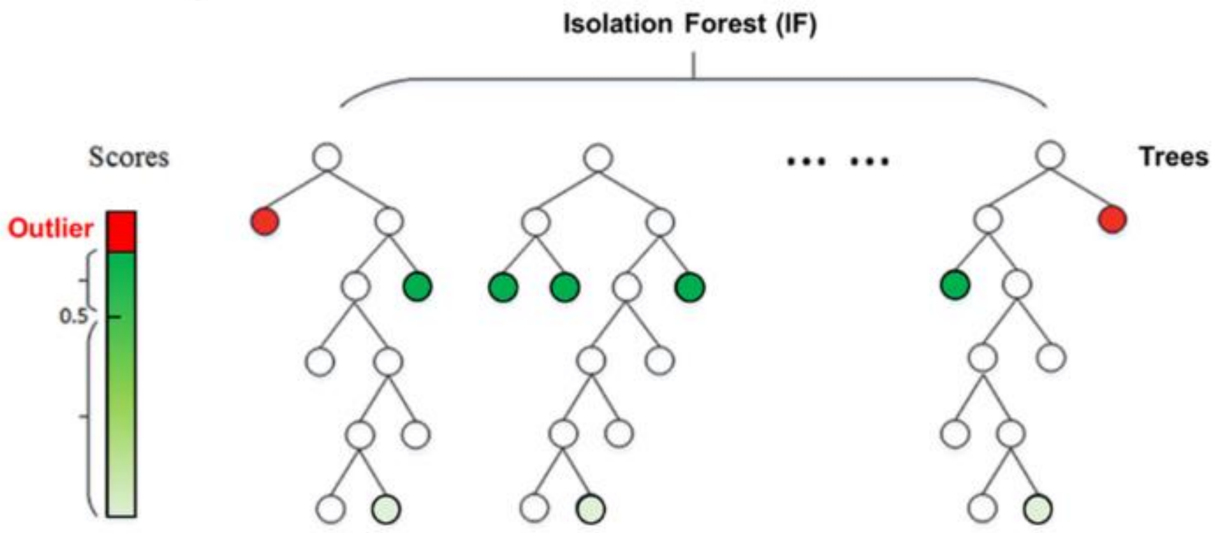
$$\phi_p(L)\tilde{\phi}_P(L^s)\Delta^d\Delta_s^D y_t = A(t) + \theta_q(L)\tilde{\theta}_Q(L^s)\epsilon_t$$

- $\phi_p(L)$ is the non-seasonal autoregressive lag polynomial
- $\tilde{\phi}_P(L^s)$ is the seasonal autoregressive lag polynomial
- $\Delta^d\Delta_s^D y_t$ is the time series, differenced d times, and seasonally differenced D times.
- $A(t)$ is the trend polynomial (including the intercept)
- $\theta_q(L)$ is the non-seasonal moving average lag polynomial
- $\tilde{\theta}_Q(L^s)$ is the seasonal moving average lag polynomial

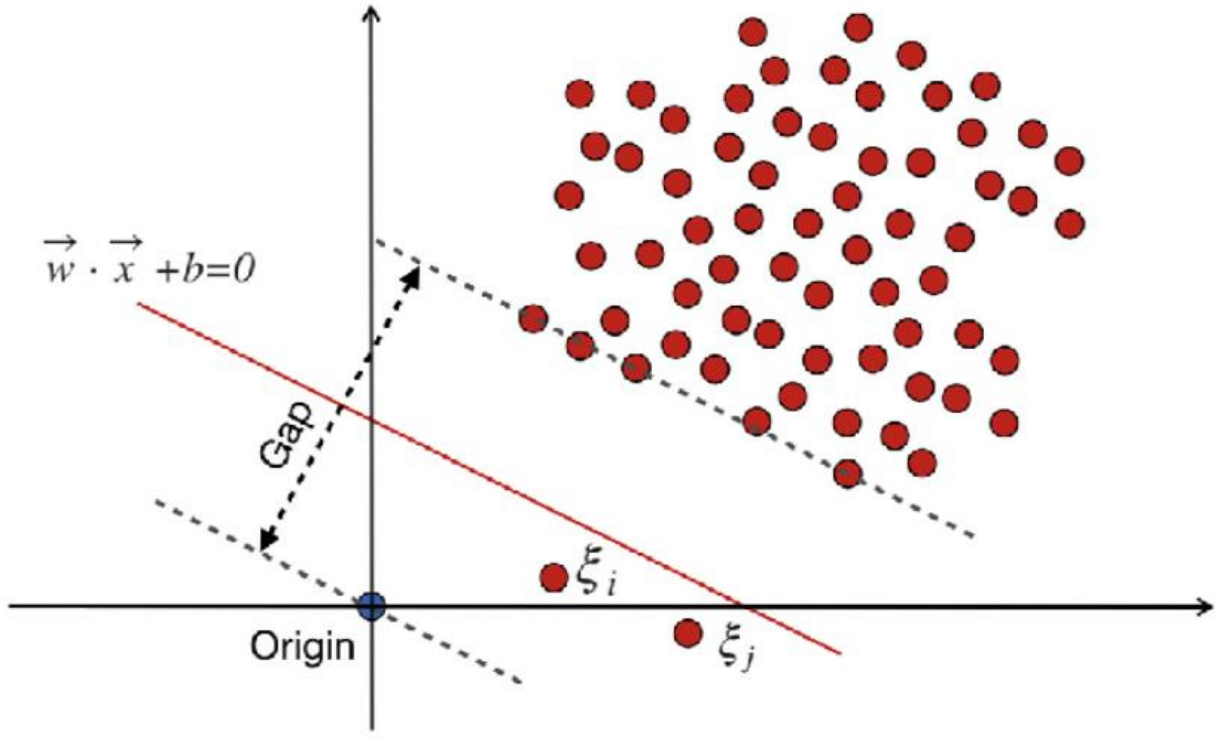
- Random Forest is an ensemble based algorithm that excels in both classification and regression tasks, including forecasting. It works by constructing multiple decision trees and aggregating their predictions to provide robust predictions, making it a versatile choice for a wide range of forecasting problems across various domains. Lacks interpretability of results and may perform badly with short time series.



- Outlier detection: we feed the data points as a set to the algorithm and it will compute an anomaly score for each one, then we apply a threshold to mark points as anomalies.
    - Isolation Forest constructs a binary tree by randomly selecting features and splitting data points until anomalies are isolated more quickly than normal data, making it well-suited for high-dimensional datasets. Tends to perform well when anomalies are few and distinct from normal data. It may need a bit of fine tuning and its output provides very low interpretability.



    - One-Class SVM learns a boundary that encapsulates normal data points while minimizing the inclusion of anomalies. Can handle non-linear relationships through kernel tricks but may be slow to train and requires quite a bit of fine tuning.



I would choose two or three of these and compare them.

Here’s a brief recap of the algorithms:

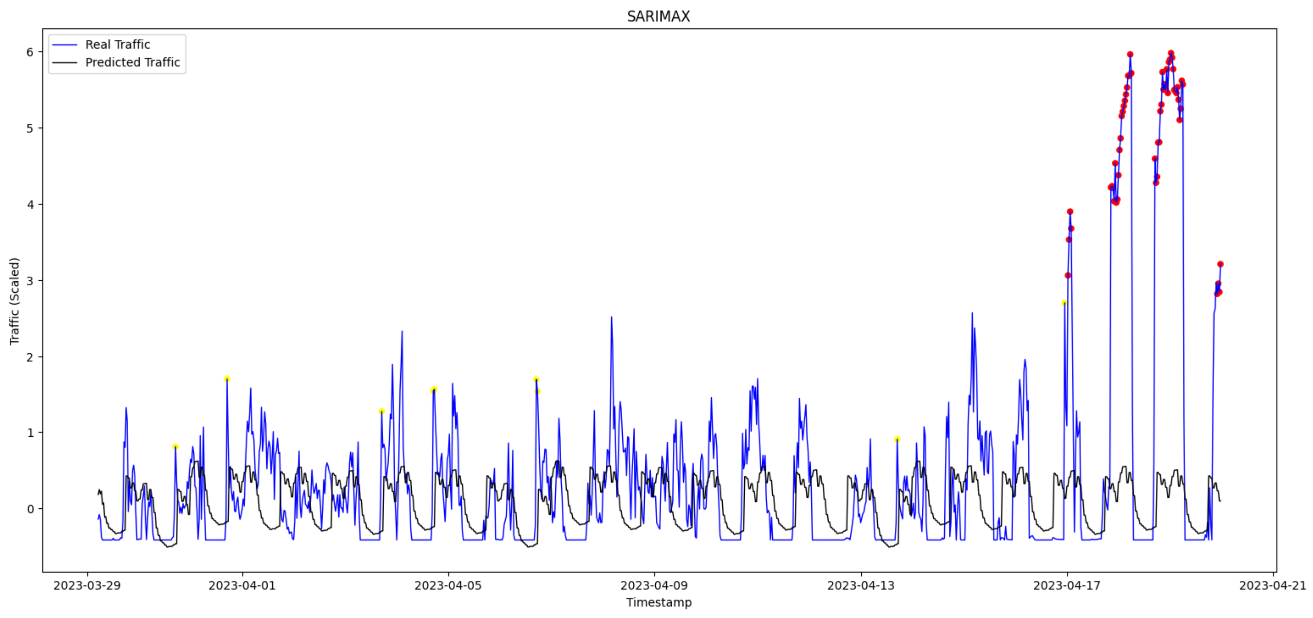
- Forecasting:
    - STL
    - Sarimax
    - Random Forest
- Outlier Detection:
    - Isolation Forest
    - One-Class SVM

Result
It is difficult to objectively measure performance on these algorithms' anomaly detection capabilities without having a big, labelled dataset available. Here is where we branch off from Academia: in Business our target is to develop a tool which should work on all the potential customers' infrastructures, without the luxury of too many preconditions or a good training dataset. So the path is collecting production data, make observations, conduct small test, develop a sub-optimal solution to be improved over through new data and the customer's feedback.

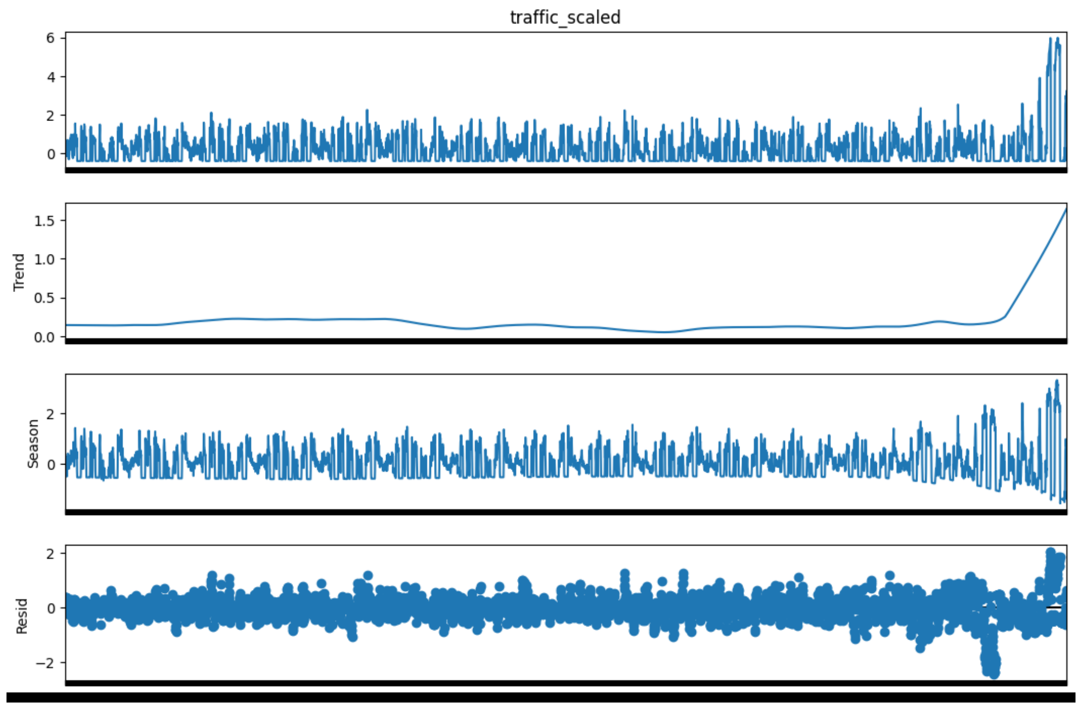
We will briefly test the previously mentioned ML models one by one. The following experiments were conducted on the dataset 1_8_out.csv, taking only data from 2023-01-01 to 2023-04-20 and performing an 80/20 split for training and test sets. Then we'll look into execution times.

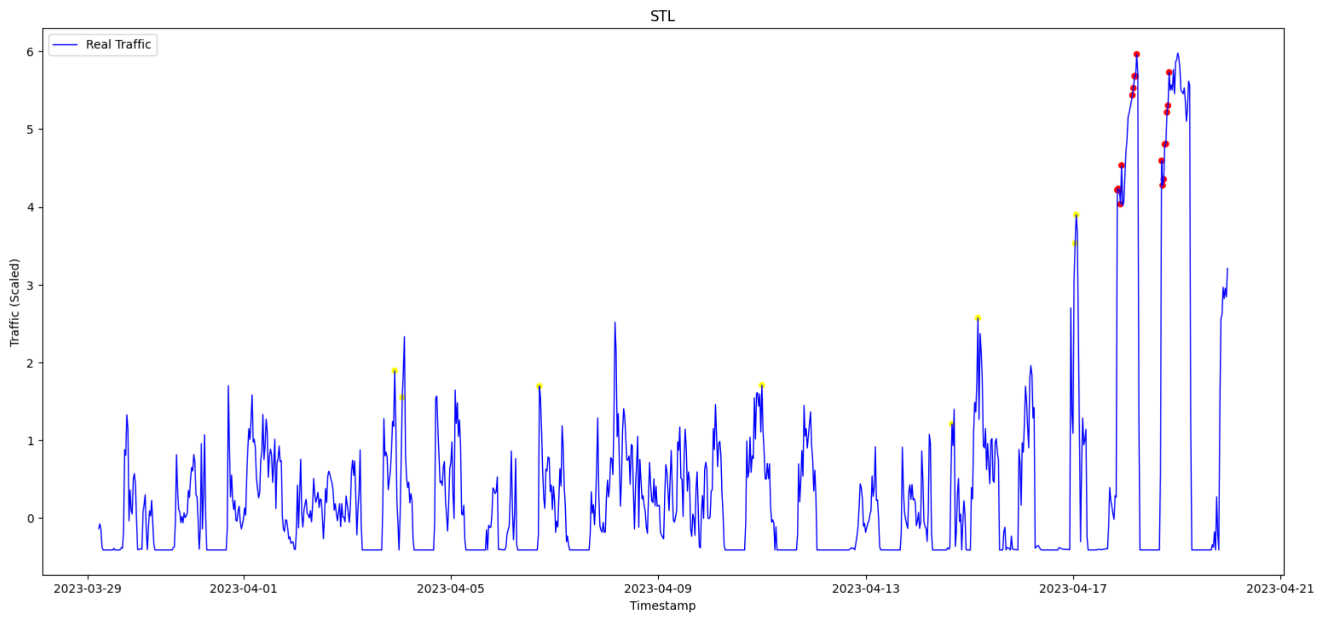
Within the following graphs, detected anomalies are represented with yellow or red dots. A dot is painted red when it belongs to segments where at least 4 consecutive datapoints have been detected as anomalies. In real life, customers are alerted of anomalies only on red dots.

- SARIMAX
  - In the chart below we can see the real traffic data in blue and sarimax's forecast in black. The model managed to learn the day/night pattern but not the weekly one.
  - By applying an appropriate threshold we can still detect big anomalies, but the same could be done with simple statistics.
  - One could finetune this model's parameters to gain way better results to the cost of performance.

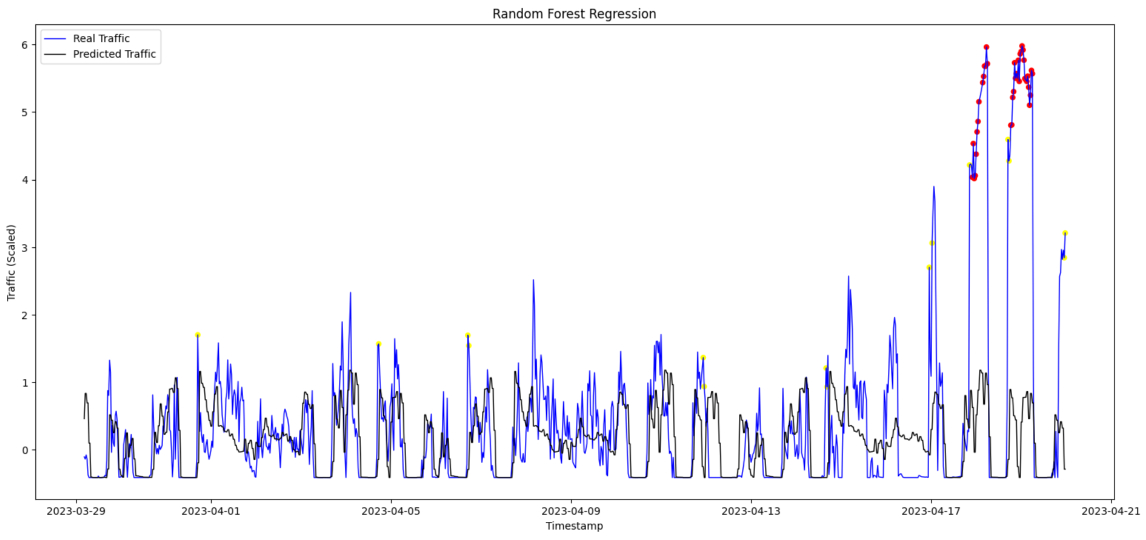


- Seasonal Trend decomposition using LOESS (STL)
    - Decomposes our signal into Trend, Season and Residual. The last one is what remain once we subtract the first two components.
    - The more a signal is coherent and doesn't contain anomalies, the more its residual should be close to zero.
    - In order to detect anomalies, we can set a threshold on residual, which grows with the difference between what we have and what we expect. We can see that there isn't a lot of margin between normal behaviour and the anomaly of 18/04/2023.





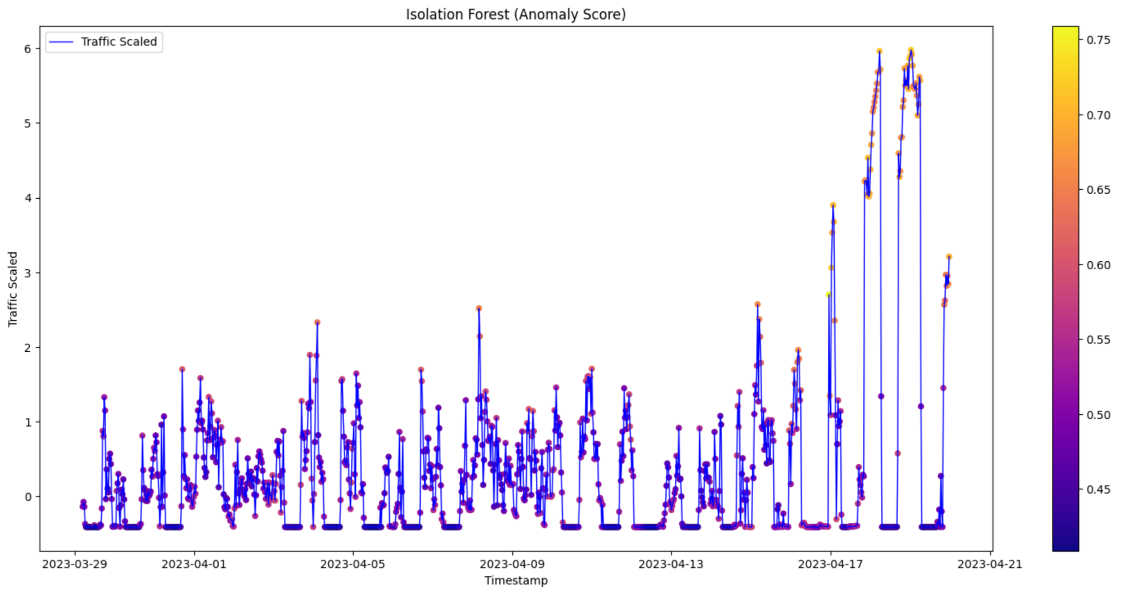
- Random Forest
  - It managed to learn both daily and weekly patterns, its predictions are interesting because they look less regular than sarimax' and their scale is more similar to the real data.

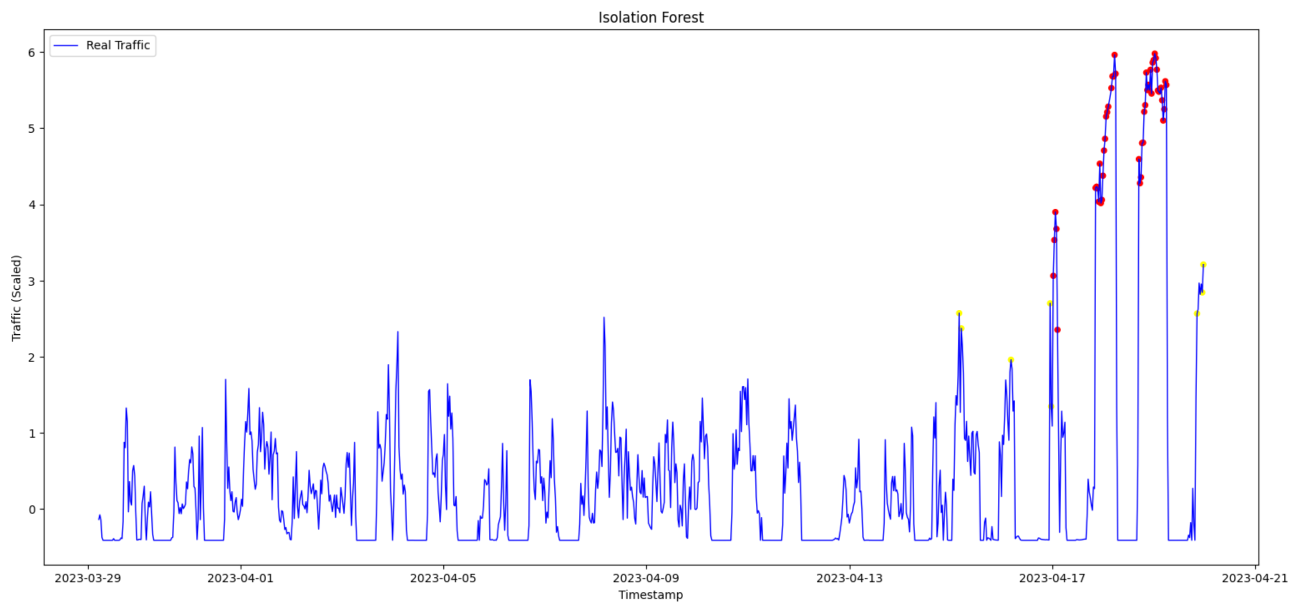


- Isolation Forest
  - This algorithm doesn't provide predictions, so in order to get some interpretation of results and to aid with choosing a good threshold it is better to plot the anomaly score assigned to each point.
  - In the chart below points are painted with a colour gradient that matches their anomaly score, going from blue to yellow. This can be achieved by passing the scores to pyplot as colour parameter and selecting a colormap:

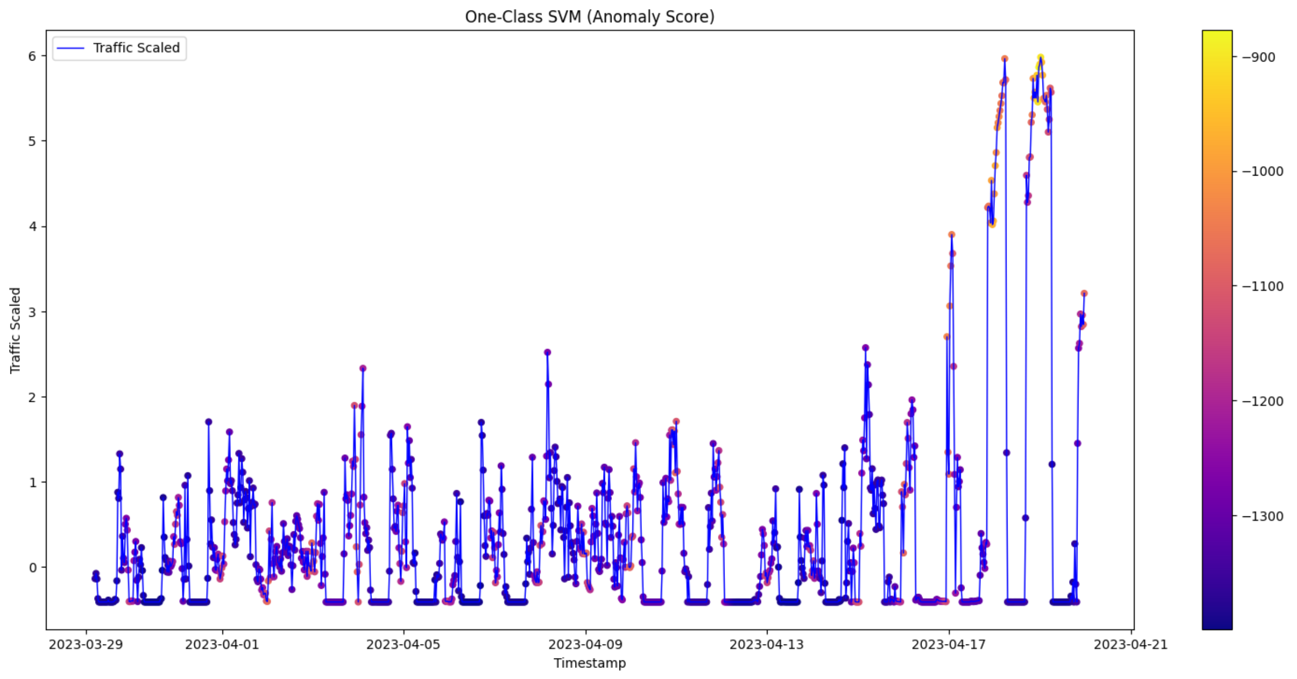
```
plt.scatter(x, y, s=20, c=df["anomaly_score"], cmap="plasma")
plt.colorbar()
```

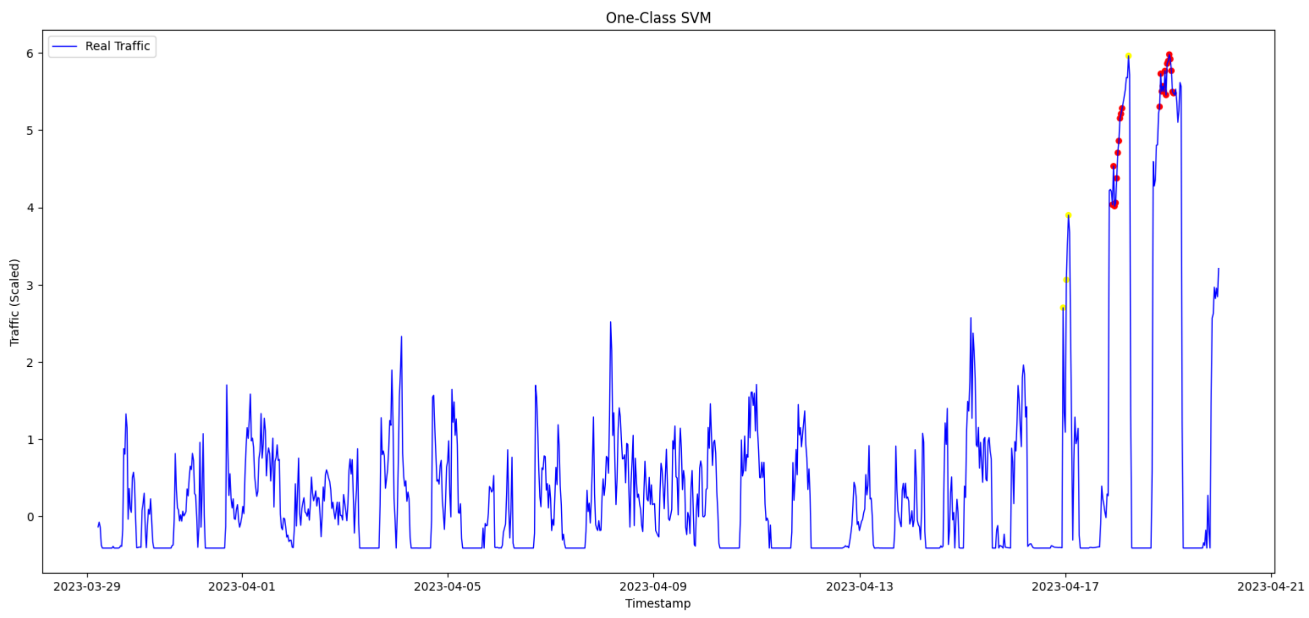
  - In general, scores are coherent so choosing a proper threshold gives good results.





- One-Class SVM
  - Again, we plot the anomaly scores before proceeding in detecting anomalies.
  - The model gives a fine result even without too much finetuning, but we can see that quite a few scores are quite high even for low values, which makes choosing a threshold problematic.





Execution times

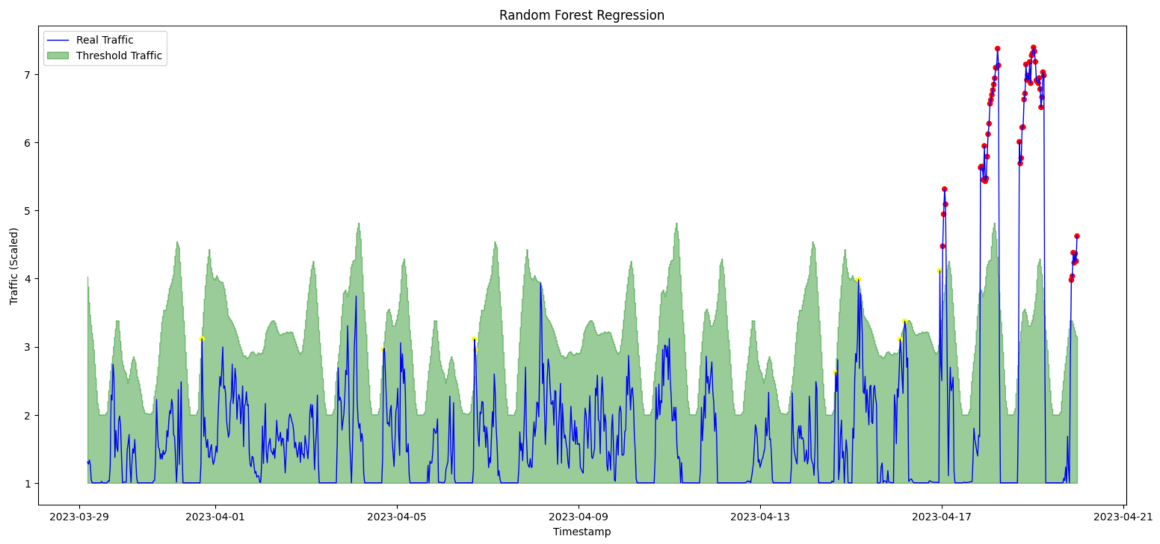
This table shows the execution times for each model on the same machine during the simple experiment above. It's not an objective performance metric for the algorithms themselves but, knowing that in production we should train hundreds of models on months of data, it can give an idea about which models we should drop and which we should continue to work on.

STL	Random Forest	SVM	Isolation Forest	SARIMAX
0.292s	0.494s	1.279s	1.859s	7.927s

What's Next

Now that we have a working anomaly detection pipeline, we should start to finetune the preprocessing parameters and the ML model. Also it is a good moment to start working on more data, by feeding the model a bigger chunk of csv file, or to experiment with different ports. Plus with forecast algorithms we fed the test set all at once, while in production we would execute an anomaly detection job every hour or so on just the last data points, so one could check how using an iterative approach would changes the results.

In the end, it's interesting to add a little more interpretability to the model results and the possible alerts that the software would send to the customer. As with charts, instead of the raw prediction or scores, we could display a green band behind the real traffic line showing what the accepted range of traffic is. It's clearer with an example:


Random Forest Regression
Real Traffic
Threshold Traffic
Traffic (Scaled)
Timestamp