Machine Learning Applied on Predictive Maintenance of Aircrafts: An experimental study

Daniel Gareev
University of Luxembourg
Email: daniel.gareev.001@student.uni.lu

Jorge Augusto Meira
University of Luxembourg
Email: jorge.meira@uni.lu

Abstract

Maintenance is a constant for aircraft, and for this reason, it is well planned and scheduled with a proper timetable. The problem resides in unscheduled maintenance. Unscheduled maintenance occurs when a component of the aircraft fails, forcing the airline to fix it before the next take-off, often causing an AOG event (Aircraft on the ground). Such event cause delays and considerable costs. In this context, predictive maintenance tools to support aircraft engineers to identify potential failures are gaining the attention of the airlines in the last few years.

In this project, we describe an approach using machine learning to analyze historical data of the aircraft and predict potential failures in order to support aviation engineers to take appropriate actions to avoid unscheduled maintenance. We apply ARIMA model to time series forecasting to detect anomalies. The anomaly scores are then determined based on the residual errors.

1. Introduction

The main objective of this project is to review the state of the art on predictive maintenance for aircrafts, reproduce published results and develop a machine learning approach that combines techniques reviewed in the state of the art capable of giving insights and alerts to help engineers to decide the best moment to proceed with the maintenance on the aircraft. In the project, we aim to use aircraft sensor values to identify unusual behaviour on the aircraft hydraulic system that can lead to a potential failure, so that maintenance can be planned in advance. The measurements captured by the sensors can be seen as time series, which can be used to predict its future values.

We first develop a general autoregression model using an open dataset available online and then we extend it and apply anomaly detection algorithms to discover operationally significant anomalies in the aircraft dataset.

As shown in Figure 1, utilizing the ARINC 429 DITS data busses, the behavior of the aircraft captured by thousands of sensors is stored in the black-box and in the Quick Access Recorder(QAR). In this work, we use QAR parsed data.

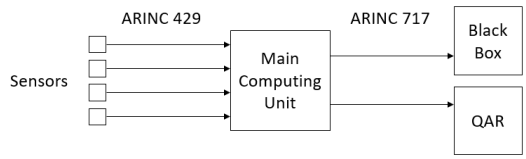

Fig. 1: Sensor data recording

2. Project description

2.1. Domains

The following domains are covered in the project:

- Machine learning
- Predictive maintenance
- Data preprocessing
- Anomaly detection
- Time series forecasting
- Python
- ARIMA / SARIMA models
- Autoregression

2.1.1. Scientific

The main scientific aspect covered by this project is machine learning applied to predictive maintenance. Specifically, machine learning applied to time series forecasting. Additionally, we review the state-of-the-art of anomaly detection methods and their application to the domain, as well as present specific work that have contributed to the domain.

2.1.2. Technical

The technical side of this project is related to developing a machine learning approach that is capable of giving insights and helping to determine when to proceed with the maintenance of the aircraft. We aim to develop a general time series autoregression model that could be then generalized to work with the time series data of the aircraft to predict when a failure of the aviation system might occur.

2.2. Targeted Deliverables

2.2.1. Scientific deliverables

The main scientific deliverable covered by this project is to analyze how machine learning can be applied to predictive maintenance. More specifically, we will study the following machine learning models, that are underlying for this project: AR (Auto-Regressive), ARMA (Auto Regressive Moving Average), ARIMA (Auto Regressive Integrated Moving Average). The objective is to deeply understand the assumptions that support such models to effectively apply them in the specific case. We will also present an introduction of time series as well as the difference between stationary and non-stationary time series.

Additionally, the state-of-the-art techniques related to machine learning applied to predictive maintenance will be reviewed in this project. For example, there are different techniques that can be applied, such as anomaly detection, classification, or regression.

2.2.2. Technical deliverables

The main technical deliverable of the project is to develop a framework for engineering, modelling and evaluating autoregressive machine learning algorithm for time series forecasting. In this project, a dataset of static historical data of the sensors is already available,

as well as several samples of the events (i.e., aircraft failures) [1]. The approach of this project stems from the hypothesis that anomalies should occur at some near data point before the failure event.

These are the technical steps that compose the framework: First, we load and preprocess (i.e., downsample, clean and normalize) the data. Next, we plot the time series and explore the data. Then, we apply the stationarity tests (such as ADF and KPSS) to the time series data. Additionally, the decomposition of the time series data can be applied to check the trend and seasonality of the time series data. Then, if the data is non-stationary, we will transform the time series into stationary data. Next, we will split the time series data on the test and train datasets. Next, we will apply ARIMA model to forecast the time series. The ARIMA model can be extended to the more sophisticated ones such as SARIMA to improve the accuracy and precision of the model based on the prediction performance and type of data. The objective is to develop an anomaly detection model based on time series forecasting that pinpoints the discrepancy (i.e., anomaly point) between predictions and real measurements at some data points close to the event.

3. Pre-requisites

3.1. Scientific pre-requisites

One of the scientific pre-requisites is basic understanding of the machine learning fundamentals. Specifically, knowledge in linear regression. In addition, a basic knowledge in Linear Algebra and Statistics is preferred, but not required.

3.2. Technical pre-requisites

The required technical prerequisites to start working on this project are prior programming experience in Python programming language, as well as experience with machine learning libraries such as *scikit-learn*, and tools such as *Jupyter Notebook*. Note that packages such as *Pandas*, *NumPy*, *matplotlib*, *statsmodels*, *sklearn*, *plotly* and *pyramid* should be installed on the machine.

4. Introduction to Time Series (Scientific Deliverable 1)

4.1. Requirements

The main objective of this deliverable is to build an understanding of time series as well as develop an overview of AR (Auto-Regressive), ARIMA (Auto-Regressive Integrated Moving Average) models. We will also focus on different characteristics of time series such as stationarity, seasonality and autocorrelation.

4.2. Design

The main objective is to follow several tutorials in order to build an understanding of the time series.

4.3. Production

4.3.1. Introduction to Time Series

A time series is a series or sequence of data points ordered in time. A time series can be represented by any data captured over time in sequential order. In a time series, time is the independent variable and the aim is to make a forecast for the future data points [5]. There are several important factors that we need to take into account when dealing with time series data, such as stationarity, seasonality and autocorrelation.

4.3.1.1 Stationarity of Time Series

A time series is stationary if its statistical properties do not change over time [5]. In other words, a time series is said to be stationary when it has constant mean and variance over time [5]. Rolling average and the rolling standard deviation of time series do not change over time. Therefore, time series with trends are not stationary since these characteristics will affect the statistical properties of time series at different times.

Usuallly, we would like to have a stationary time series for modelling, however, it is possible to make time series stationary through transforming it. We see that the time series example below is stationary (the mean and variance do not vary over time):

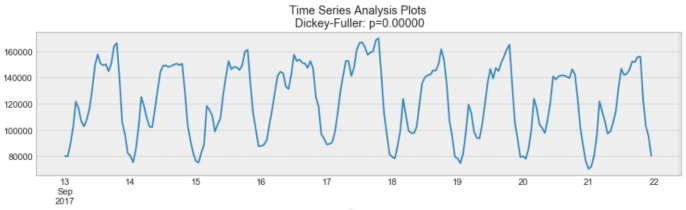

Fig. 2: Example of a stationary time series. Source: [5]

There are different statistical tests that can be done on time series to check whether they are stationary or not. One of them is the Dickey-Fuller test that tests the null hypothesis that a unit root is present [5]. *"If it is, then p>0, and a time series is not stationary, otherwise, p = 0, the null hypothesis is rejected, and the time series is considered to be stationary "*. [5]. This is assuming the significance level is 0, however, based on the problem, we can also have different significance level (e.g., 0.05). Therefore, If the p-value is less than some predefined significance level, we will reject the null hypothesis.

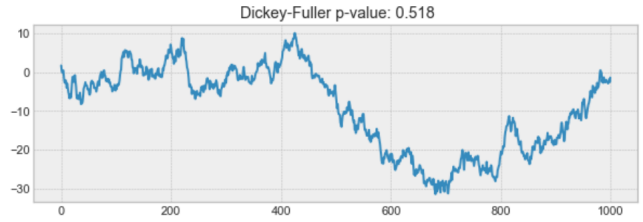

Fig. 3: Example of a non-stationary time series. Source: [5]

As an example, the time series above is not stationary as the mean is not constant over time.

4.3.2. Predicting Time Series

There are many ways to model a time series in order to make predictions, such as moving average, autoregression, ARIMA, Seasonal ARIMA, and even Recurrent Neural Network (LSTM). In this project, we are focusing on ARIMA model, which is a class of statistical models for analyzing and forecasting time series data. ARIMA, is one of the most widely used forecasting methods for time series data forecasting. Although the method can handle data with a trend, it

1. Note that due confidentiality reasons the data is anonymized, thus the values for the Hydraulic pressure from the engines do not reflect real values.

does not support time series with a seasonal component. As outlined in [7], the components of the model are:

- *AR: Autoregression. "A model that uses the dependent relationship between an observation and some number of lagged observations."*
- *I: Integrated. "The use of differencing of raw observations (e.g. subtracting an observation from an observation at the previous time step) in order to make the time series stationary."*
- *MA: Moving Average. "A model that uses the dependency between an observation and a residual error from a moving average model applied to lagged observations."*

Each of these can be specified in the model as a parameter. More specifically, the model is called ARIMA(p, d, q) where the parameters are given as integer values to quickly show the specific ARIMA model.

The parameters of the ARIMA model are defined as follows [8]:

- p: The number of lag observations included in the model, also called the lag order.
- d: The number of times that the raw observations are differenced, also called the degree of differencing.
- q: The size of the moving average window, also called the order of moving average.

A value of 0 can be used for a parameter, which indicates to not use that element of the model [8]. This way, the ARIMA model can be used as ARMA model (as in this case $d = 0$) , or simpler AR, I, or MA models [8].

These parameters are usually used when we need to remove trend or seasonality that affect the model [8]. If our model has a seasonal component, we can use a seasonal ARIMA model (SARIMA). In that case we have another set of parameters: P, D, and Q which describe the same associations as p,d, and q, but correspond with the seasonal components of the model [8].

We can automate the process of training and evaluating a large number of hyper-parameters for the ARIMA model by using a grid search procedure, such as auto ARIMA [8]. In machine learning this is called model tuning.

5. Related Work (Scientific Deliverable 2)

5.1. Requirements

The aim of this deliverable is to review the state-of-the-art of data-driven anomaly detection methods and their application to the domain as well as to discuss specific work that have contributed to domain.

5.2. Design

The deliverable has been produced by researching related work that technically relates to the proposed work. A short summary of work that proposes a different method to solve the problem, or uses the proposed method to solve a different problem is presented in this deliverable. It is worth noting that the deliverable does not directly or quantitatively compare the results of the project to prior work.

5.3. Production

5.3.1. Recent developments in anomaly detection methods

The problem of predicting failure events of systems has attracted an increasing attention of academy and industry due to its potential benefits. The research on the improvement of the safety and performance of flight operations and aviation systems is essential to the development of the industry. This section reviews the state-of-the-art of data-driven anomaly detection methods and their application to the domain.

Anomaly detection is an active area of ongoing research with a wide variety of methods and applications. There are multiple anomaly detection approaches that are applicable to predictive maintenance. Some of the methods include classical data-driven methods, such as statistical methods (e.g., Gaussian mixture models, regression model-based), distance-based methods (e.g, clustering-based methods, nearest neighbour-based methods), while others leverage recent advances in the area of neural networks, deep learning and temporal-logic based learning [12]. It is worth mentioning that unsupervised techniques applicable to time series data are specifically relevant to the aviation domain, where the lack of labeled data is the most usual case, and the nature of flights and sensor data is sequential [12].

Statistical anomaly detection methods are one of the most prevalent categories identified in paper [13] on anomaly detection techniques. It is also one of the groups mentioned in another paper on anomaly detection [14]. The regression model-based approach is one of the methods identified in [13] that is widely applicable to time series data. *"A two-step approach is used to detect anomalies: first, a model is fitted on the training data and then the model is used on test sequences to compute the residuals"* (i.e., the difference between the actual and predicted values) [13] and, subsequently, the anomaly scores. Some of the time series forecasting methods in this category are Autoregressive Moving Average (ARMA), Autoregressive Integrated Moving Average (ARIMA) [15] and Recurrent Neural Networks (RNNs) [16]. RNNs have been used as regression models, but they are not covered in this project. The time series forecasting methods are applicable to a variety of use cases. Many use cases like demand estimation, sales forecasting can be viewed as a time series forecasting task. For instance, Facebook uses different forecasting techniques such as ARIMA and exponential smoothing in their Prophet product [9].

5.3.2. Application of anomaly detection methods to the domain

This section explores the application of anomaly detection techniques to the aircraft industry and discusses relevant papers contributed to the improvement of the safety and performance of flight operations.

Despite its promising benefits, continuous real-time monitoring of assets, such as regression analysis, is still not widely used method to predict system failures for predictive maintenance, where only *11% o*f the respondents in the recent survey by PwC are using such methods to predict failure events [17].

With continuous data collection and monitoring, statistical modelling methods help to determine the condition of in-service equipment in order to predict when maintenance should be performed. As aircraft systems are becoming more sophisticated, they are equipped with a large number of sensors that can deliver large amounts of data to be analyzed. In the case of aviation industry, the application of advanced methods for predictive maintenance can lead to lower maintenance and operational costs, as well as improved safety.

Multiple techniques have been studied and applied to predict potential aviation system failures (i.e., predictive maintenance). For example, in [1], the authors deal with the anomaly detection approach that is designed to acquire baseline of aircraft measurement data. The authors in another work [3], also perform anomaly detection approach, however, they use extreme learning machines (ELM) to

discover anomalies in aircraft data sets. Similarly, in [4], the authors have developed a neural network approach for performing anomaly detection, where the neural network anomaly detector 'learns' to recognize sets of multiple input sensor signal patterns from known nominal data. In [2], an approach using degradation messages to predict failures in a commercial aircraft is presented. As one of the technical limitations of analyzing aircraft data comes from the lack of data collected at a suitable rate, an alternative solution is proposed by sending in-flight messages concerning the health state of the aircraft systems and predicting based on on degradation messages using a particle filter (PF) framework [2].

The approach discussed in this project deals with anomaly detection based on time series forecasting. In the domain of aviation, the data collected from flights comes in the form of time series data where sequences of observations from various aircraft sensors are being collected during the flight. In this project, we apply SARIMA model to time series forecasting to detect anomalies. The anomaly scores are then determined based on the residual errors. To the best of our knowledge, some of the anomaly detection methods such as anomaly detection with time series forecasting have not yet found an application in the aviation domain. In the current state-of-the-art, we have identified some of the related work that deals with anomaly detection with time series forecasting, but in a different context [15].

5.4. Assessment

In this section, the state-of-the-art of anomaly detection methods and their application to the domain as well as the discussion of specific work that have contributed to domain have been presented.

6. Machine Learning Approach for Multi-Step Time Series Forecasting (Technical Deliverable)

6.1. Requirements

The requirements of the deliverable are as follows:

- Develop a general time series forecasting model, that can be generalized to various use case scenarios. The aim of the project is to experiment with different autoregressive models (such as ARIMA, SARIMA etc.,), model-specifc optimizations (hyper-parameter tuning), as well as pre-processing methods to obtain a relatively high predictive performance and acceptable training time. In this project, we will primarily focus on the ARIMA component, which is used to fit time-series data to forecast future data points.
- Next, we extend the general model described above to be able to detect anomalies with time series forecasting. The aim is to produce a robust data-driven anomaly detection model that will be able to find the discrepancy between predictions and real measurements over the time span of the specific event (i.e., aircraft failure).
- After implementing the models, we will discuss the results on how the models performed and what further improvements are possible. The predictive performance can be determined by computing the RMSE score (the square root of the variance of the residuals), fit of the model (BIC and AIC values), and training time.
- In case of anomaly detection, the performance is measured based on the precision and recall. A perfect anomaly detection technique should identify only real anomalies (i.e., anomalies related to the real events). However, anomaly detection algorithms can be affected by the false positives (i.e., false detections) and false negatives (i.e., missed detections).

6.2. Design

We use Python and libraries, such as *Pandas*, *NumPy*, *matplotlib*, *statsmodels*, *sklearn*, *plotly* and *pyramid* for the implementation of the deliverable.

The *statsmodels* package is especially used extensively in the project to implement the statistical models. It is a Python library that provides tools for the estimation of many different statistical models, statistical tests and data exploration [10].

6.3. Production

This section is split in two parts. In the first section, we develop a general autoregressive time series forecasting model. In the second section, we extend the time series forecasting model to be able detect anomalies. Therefore, the first section mainly focuses on the time series forecasting, while the second on the anomaly detection with time series forecasting.

6.3.1. Autoregression Forecast Model for Household Electricity Consumption

The objective of this section is to develop a general autoregression forecast model for the time series data. We will use an open dataset of the Household Electricity Consumption, as it is one of the popular datasets to practice time series problems. The dataset can be downloaded from the UCI Machine Learning repository [6], a popular online repository for machine learning datasets.

6.3.1.1 Data Preprocessing

The Household Power Consumption dataset is a multivariate time series dataset that consists of the electricity consumption data for one household. The data has been collected at a one-minute sampling rate (i.e., one data point per minute) over a period of almost 4 years. The dataset includes different measures about specific energy uses.

The description of the dataset variables is taken from the UCI website [6]:

- **Date:** Date (in format dd/mm/yyyy)
- **Time**: Time (in format hh:mm:ss)
- **global_active_power**: The total active power by the household (in kilowatt).
- **global_reactive_powe**r: The total reactive power by the household (in kilowatt).
- **voltage**: Average voltage (in volt).
- **global_intensity**: Average current intensity (in amp).
- **sub_metering_1**: Active energy for kitchen (in watt-hour of active energy).
- **sub_metering_2**: Active energy for laundry room (in watt-hour of active energy).
- **sub_metering_3**: Active energy for climate control systems (in watt-hours of active energy).

The active power refers to the real power consumed, and the reactive power is the unused power in the lines. The dataset includes the active power as well as some specific areas of the active power (e.g., kitchen area). We can see that the data consists of *2075259* samples with *7* different energy measurements features.

First, we can use the `read_csv()` function to load the dataset. We can combine the first two columns 'Date' (date in format*dd/mm/yyyy*) and 'Time' (time in format *hh:mm:ss*) into a single date-time column and specify it as an index (refer to Listing 1). We can see the overview of the dataset in Figure 4.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

# load all data
#We can use the read_csv() function to load the data
     and combine the first two columns into a single
     datetime column to use it as an index.
dataset = pd.read_csv('household_power_consumption.
    txt', sep=';', header=0, low_memory=False,
    infer_datetime_format=True, parse_dates={'
    datetime':[0,1]}, index_col=['datetime'])

dataset
```

Listing 1: Load and preprocess dataset

After loading the dataset, we can notice that about 1.25% of the data in the rows is missing (about *82 days*). We can mark all missing values indicated with a '?' character with a `NaN` value, which is a float. Next, we can use `astype()` function to convert the data into one array of floating point values. There exists many approach to missing-data problems and they usually depend on the problem and how the data behaves. As we have few instances with missing values, we use imputation to replace them with the observation from the same time the day before.

The pre-processing stage is summarised in Listing 2.

```
from numpy import isnan

# mark all missing values
dataset.replace('?', np.nan, inplace=True)

# make dataset numeric
dataset = dataset.astype('float32')

# fill missing values with a value at the same time
    one day ago
def fill_missing(values):
    one_day = 60 * 24
    for row in range(values.shape[0]):
        for col in range(values.shape[1]):
            if isnan(values[row, col]):
                values[row, col] = values[row -
                    one_day, col]

fill_missing(dataset.values)
```

Listing 2: Load and preprocess dataset

Next, we can downsample the the minute based observations to the daily totals. This is a good approach since we would get more computationally efficient dataset. We can use `resample()` function to transform our minute-based dataset into a daily dataset by calling resampling and specifying the preferred frequency 'D'. We can then call `sum()` as shown in Listing 3.

```
# resample data to daily
day_groups = dataset.resample('D')
dataset = day_groups.sum()
dataset
```

Listing 3: Resample the dataset

6.3.1.2 Exploratory Analysis

We can plot the data for global power usage ("Global active power"). The time series has a seasonal component that has a frequency of *12 months*, that can be observed in Figure 5.

6.3.1.3 Stationarity Tests

Stationarity is an important concept in time series analysis as many of the statistical tests and computations depend on it. One of the most basic methods to check the stationarity is to plot the data and determine visually whether the data has some known property of stationary (or non-stationary data). However, as this is usually not sufficient to check stationarity, we will not be making definite assumptions by only relying on this method.

Another more definite approach to detect stationarity in time series data is by using statistical tests developed to detect specific types of stationarity, such as The Dickey-Fuller Test and The KPSS Test. The `statsmodel` package provides a quick implementation of both tests as shown in Listings 4 and 5. Note that $'>>>'$ in the listings indicates the output of the code executed above.

```
import statsmodels
from statsmodels.tsa.stattools import adfuller

adf_test = adfuller(dataset['Global_active_power'])

adf_test

>>>(-3.69738466739032,
>>> 0.004150091885223323,
>>> 22,
>>> 1419,
>>> {'1%': -3.434966750462565,
>>>  '5%': -2.8635789736973725,
>>>  '10%': -2.5678555388041384},
>>> 20985.69197629365)
```

Listing 4: The implementation of the Dickey-Fuller Test

```
from statsmodels.tsa.stattools import kpss

kpss_test = kpss(dataset['Global_active_power'])

kpss_test

>>>(1.4954297343518077,
 >>>0.01,
>>> 20,
>>> {'10%': 0.347, '5%': 0.463, '2.5%': 0.574, '1%':
     0.739})
```

Listing 5: The implementation of the KPSS Test

After running the ADF test, we can see that the ADF value (the first value in the result) is $\approx$ 3.69 and the p-value (the second value) is $\approx$ 0.0041. The p value of 0.0041 suggests that we can

datetime	Global_active_power	Global_reactive_power	Voltage	Global_intensity	Sub_metering_1	Sub_metering_2	Sub_metering_3
2006-12-16 17:24:00	4.216	0.418	234.840	18.400	0.000	1.000	17.0
2006-12-16 17:25:00	5.360	0.436	233.630	23.000	0.000	1.000	16.0
2006-12-16 17:26:00	5.374	0.498	233.290	23.000	0.000	2.000	17.0
2006-12-16 17:27:00	5.388	0.502	233.740	23.000	0.000	1.000	17.0
2006-12-16 17:28:00	3.666	0.528	235.680	15.800	0.000	1.000	17.0
...	...	...	...	...	...	...	...
2010-11-26 20:58:00	0.946	0.000	240.430	4.000	0.000	0.000	0.0
2010-11-26 20:59:00	0.944	0.000	240.000	4.000	0.000	0.000	0.0
2010-11-26 21:00:00	0.938	0.000	239.820	3.800	0.000	0.000	0.0
2010-11-26 21:01:00	0.934	0.000	239.700	3.800	0.000	0.000	0.0
2010-11-26 21:02:00	0.932	0.000	239.550	3.800	0.000	0.000	0.0

Fig. 4: Household Electricity Consumption dataset

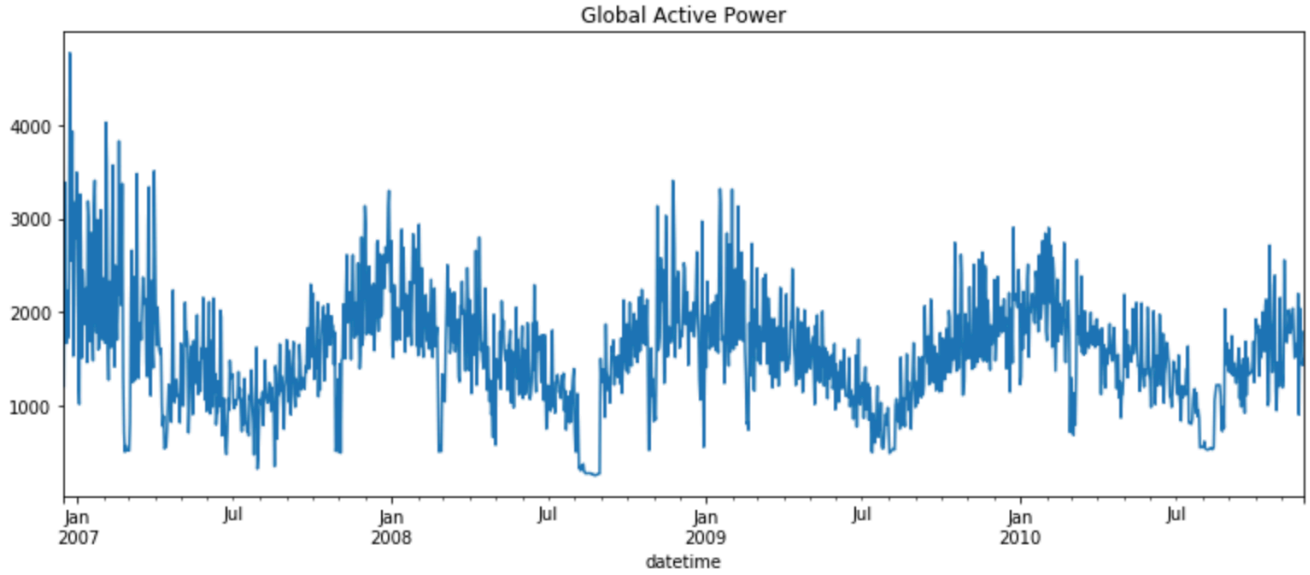

Fig. 5: The data for global power usage "Global active power". We can see that the time series has a seasonal component that has a frequency of *12 months*

reject the null hypothesis with a significance level of less than 1%. Therefore, rejecting the null hypothesis means that the process has no unit root, and that the time series is stationary. Moreover, after running another test for checking time-series stationarity, the KPSS test, we can confirm that we can reject the null hypothesis at the significance level of 1% as the p value is 0.01.

6.3.1.4 Data Modeling

The next step in our forecasting is training and testing the model. First, we extract the time-series values and apply *log transform* to normalize the data. Next, we split the data to train and test sets with entries of two years in the test data, and 1 year in the train data. The parameters of the model are `p,d,q` which are identified based on ACF and PACF plots, or by searching through a grid of hyper-parameters. We will use the latter method. The order of the model is specified as `model_order` (The `(p,d,q)` order of the model for the number of AR parameters, differences, and MA parameters.) and `model_seasonal_order` (The `(P,D,Q,s)` order of the seasonal component of the model for the AR parameters, differences, MA parameters, and periodicity). We use these parameters as we determined that they have the most minimal error in forecasting (as can be seen later in Model Optimization section 6.3.1.5). The process is summarized in Listing 6.

```
import math
import statsmodels.api as sm
import statsmodels.tsa.api as smt
from sklearn.metrics import mean_squared_error
from matplotlib import pyplot
import matplotlib.pyplot as plt
import plotly.plotly as py
import plotly.tools as tls

train = dataset.loc["2007-07-01":"2009-06-30"].
    Global_active_power.values
test = dataset.loc["2009-07-01":"2010-06-30"].
    Global_active_power.values

train_log, test_log = np.log10(train), np.log10(test
    )
model_order = (1, 1, 1)
model_seasonal_order = (0, 1, 1, 12)
```

Listing 6: Normalize the data and split the test and train data

One of the methods available in Python to model and predict time series data is known as SARIMAX, which stands for Seasonal Auto Regressive Integrated Moving Averages with eXogenous regressors [18]. We apply SARIMAX model to predict future data points as our data has the periodicity of *12* for monthly data. We loop through train set to predict the next data point and to determine the next data point after that prediction for further forecasting. Therefore, previous data points are used to predict the next data points at any given time. Finally, we transform the predicted data back to scale by the power of *10* and plot the results. A SARIMAX model can be created using the *statsmodels* Python library as follows:

1) Define the model by calling `SARIMAX()` and passing in the p, d, and q parameters, as well as P, D, Q, s parameters.
2) The model is prepared on the training data by calling the `fit()` function.
3) Predictions can be made by calling the `predict()` function and specifying the index of the time or times to be predicted.

This is summarized in Listing 7.

```
history = [x for x in train_log]
predictions = list()
predict_log=list()
for t in range(len(test_log)):
    model = sm.tsa.SARIMAX(history, order=
        model_order, seasonal_order=
        model_seasonal_order,
    enforce_stationarity=False,enforce_invertibility
        =False)
    model_fit = model.fit(disp=0)
    output = model_fit.forecast()
    predict_log.append(output[0])
    yhat = 10**output[0]
    predictions.append(yhat)
    obs = test_log[t]
    history.append(obs)
    print('predicted=%f, expected=%f' % (output[0],
        obs))
error = math.sqrt(mean_squared_error(test_log,
    predict_log))
print('Test rmse: %.3f' % error)
# plot
figsize=(12, 7)
plt.figure(figsize=figsize)
pyplot.plot(test,label='Actuals')
pyplot.plot(predictions, color='red',label='
    Predicted')
pyplot.legend(loc='upper right')
pyplot.show()
```

Listing 7: Training the model and predicting the values

The result of running the code above gives us predicted and expected values for each data point (e.g., predicted=3.119937, expected=3.096196), prints out the RMSE score and then plots a graph that consists of the actual and predicted data points as shown in Figure 6. The obtained RMSE score is *0.100*.

6.3.1.5 Model Optimization

To tune the parameters we use auto_arima function from the *pyramid* Python library (the equivalent of R's `auto.arima`) which returns the optimal set of parameters for the model in the specified range. To find the best model, the auto_arima optimizes for a given information criterion (one of AIC, BIC etc.,) on the fitted model, and returns the ARIMA which minimizes the value. When we run the function, we can observe that the model ARIMA `(1, 1, 1)` performs better than others. Note that some of the output has been hidden in Listing 8.

```
from pyramid.arima import auto_arima
stepwise_model = auto_arima(train_log, start_p=0,
    start_q=0,
                            max_p=10, max_q=10, m=12,
                            start_P=0, seasonal=True,
                            d=1, D=1, trace=True,
                            error_action='ignore',
                            suppress_warnings=True,
                            stepwise=True)

>>>Fit ARIMA: order=(0, 1, 1) seasonal_order=(0, 1,
    2, 12); AIC=-872.016, BIC=-849.134, Fit time
    =10.308 seconds
>>>Fit ARIMA: order=(0, 1, 1) seasonal_order=(1, 1,
    2, 12); AIC=-868.762, BIC=-841.303, Fit time
    =16.350 seconds
>>>Fit ARIMA: order=(1, 1, 1) seasonal_order=(0, 1,
    1, 12); AIC=-881.309, BIC=-858.426, Fit time
    =5.032 seconds
```

Listing 8: Auto Arima for parameters selection

Another way to search through the best parameters is to use the grid search method where all of the values of the model will be iterated in the specific range, as opposed to auto-arima, where the algorithm is selective. This can be done as shown in Listing 9 (note that some of the output has been hidden).

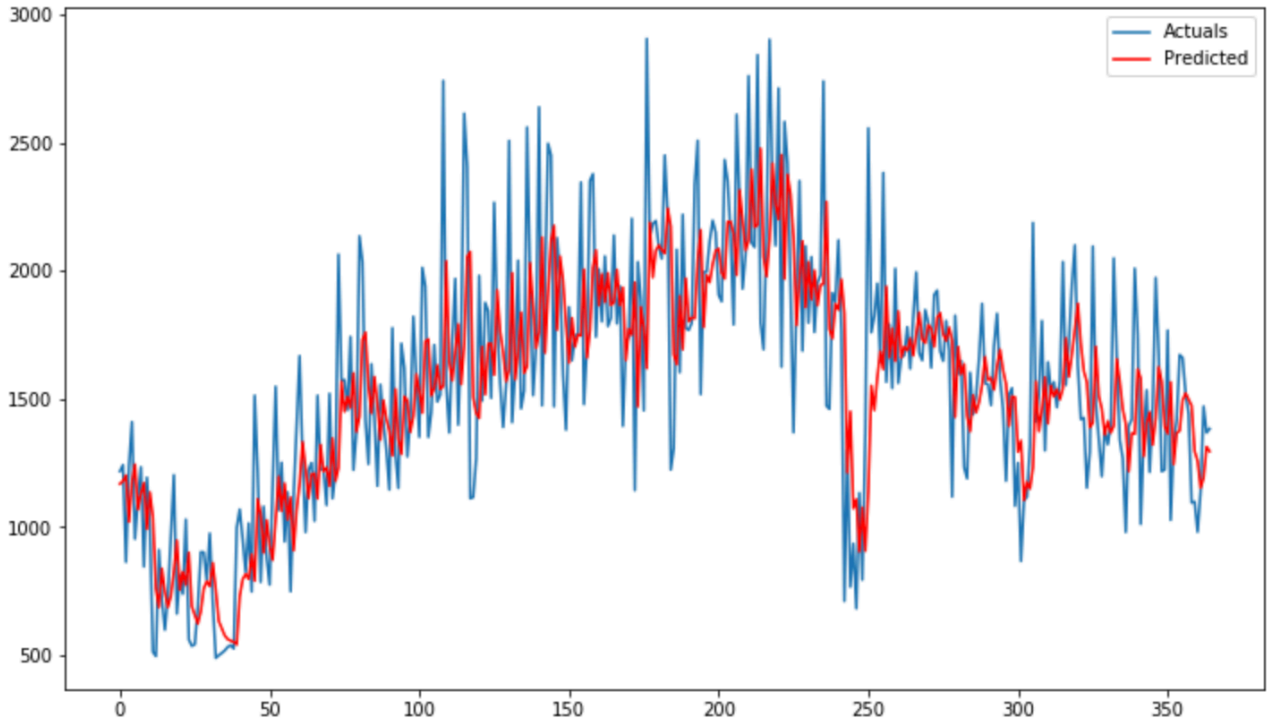

Fig. 6: We loop through train set to predict the next data point and to determine the next data point after that prediction for further forecasting. Predicted and actual values for each data point for the global power usage ("Global active power") are shown in this graph.

```
import itertools

p = d = q = range(0, 2)
pdq = list(itertools.product(p, d, q))
seasonal_pdq = [(x[0], x[1], x[2], 12) for x in list
    (itertools.product(p, d, q))]

for param in pdq:
    for param_seasonal in seasonal_pdq:
        try:
            mod = sm.tsa.statespace.SARIMAX(history,
                order=param,seasonal_order=
                param_seasonal,enforce_stationarity=
                False,enforce_invertibility=False)
            results = mod.fit()
            print('ARIMA{}x{}12 - AIC:{}'.format(
                param,param_seasonal,results.aic))
        except:
            continue

>>>ARIMA(1, 1, 1)x(0, 1, 0, 12)12 - AIC
    :-404.675971317
>>>ARIMA(1, 1, 1)x(0, 1, 1, 12)12 - AIC
    :-870.133858881
```

Listing 9: Grid search for parameters selection. Source of the code: [18]

The grid search returns the same set of optimal parameters `(1,1,1)` with the lowest AIC and BIC scores.

6.3.2. Autoregression Forecast Model for the Aircraft Engines Hydraulic Pressure

This section describes the framework of the approach for detecting anomalies based on time series forecasting applied to the aircraft domain. The case study presented in this section is concerned with the prediction of hydraulic pressure sensors data points. The objective of this section is to take the Aircraft Engines Hydraulic Pressure dataset and use it to develop a rigorous model that can be able to spot anomaly points based on the residual values.

By our hypothesis, we want to investigate the anomalies *30 days* before and after the event, *60 days* before the event only, and *30 days* before the event only. This is due to the characteristics of the aircraft operation, as we want to see the behavioural trend before the event and after the event, when the maintenance has already occurred.

6.3.2.1 Data Preprocessing

The Aircraft Engines Hydraulic Pressure is the dataset that describes the measures of hydraulic pressure sensors of the aircraft over almost *3* years. At a given day, each data point in the dataset is related to one flight. The dataset is not available in the open access and has been provided by the academic tutor of the project (Jorge Augusto Meira) who is currently working on the similar types of projects at the Interdisciplinary Centre for Security, Reliability and Trust (SnT) in Luxembourg [19]. Note that due confidentiality reasons the data is anonymized, thus the values for the Hydraulic pressure from the engines do not reflect real values. The dataset also contains *4* dates of the real aircraft system failures: *(4/17/2017, 2/28/2018, 5/14/2018 and 6/30/2018)*. The sample of the dataset can be observed in Figure 7.

Datetime	hp1
2017-01-01 00:00:00	1361.619196
2017-01-01 07:24:19	1458.532174
2017-01-01 16:55:11	1422.418272
2017-01-02 00:00:00	1356.589000
2017-01-02 00:25:34	1452.569763

Fig. 7: The Aircraft Engines Hydraulic Pressure dataset without preprocessing applied

The preprocessing starts with importing the necessary Python libraries and loading the dataset. To make sure the values are ordered by the datetime in the chronological order, we will use `sort_values()` method. We can also drop the first column that contains the row numbers (see Listing 10).

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

# load all data
#We can use the read_csv() function to load the data
dataset = pd.read_csv('press1.csv',
    infer_datetime_format=True, index_col=['Datetime
    '])
dataset = dataset.sort_values(by="Datetime")

dataset = dataset.drop(dataset.columns[0],axis = 1)

# summarize
print(dataset.shape)
print(dataset.head())
dataset
```

Listing 10: Load and preprocess dataset

We can then downsample the dataset to the daily mean values, as we have inconsistent number of data points in a single day and our aim is to make prediction for the days. We can use `resample()` function to transform our dataset into a daily dataset by calling resampling and specifying the frequency 'D'. We can then call `mean()` to compute the mean values (Listing 11). The overview of the resulting dataset can be observed in Figure 8.

```
# resample data to daily
dataset.index = pd.to_datetime(dataset.index)

day_groups = dataset.resample('D')
dataset = day_groups.mean()

# make dataset numeric
dataset = dataset.astype('float32')

dataset
```

Listing 11: Load and preprocess dataset

Datetime	hp1
2017-01-01	1414.189941
2017-01-02	1421.627808
2017-01-03	1501.075195
2017-01-04	1365.320312
2017-01-05	1455.590088

Fig. 8: The Aircraft Engines Hydraulic Pressure dataset with preprocessing applied

6.3.2.2 Explaratory Analysis

We can plot the values as follows to see the time series data (Listing 12). We cannot observe neither the visible seasonality in the resulting Figure 9, nor the clear trends, that might give a hint that the time series is stationary.

```
from statsmodels.graphics.tsaplots import plot_acf

ax = plt.gca()

dataset['hp1'].plot(figsize=(13, 5), title = 'HP_1',
     ax = ax)

plt.show()
```

Listing 12: Plotting the graph of the dataset

6.3.2.3 Stationarity Tests

In this section, we will check stationarity by carrying out The Dickey-Fuller Test and The KPSS Tests (Listing.

```
import statsmodels
from statsmodels.tsa.stattools import adfuller

adf_test = adfuller(dataset['hp1'])

adf_test

>>>(-5.8122076756018215,
>>> 4.37478135592795e-07,
>>> 15,
>>> 1029,
>>> {'1%': -3.436720930946538,
>>>  '10%': -2.56826769663245,
>>>  '5%': -2.8643528789984187},
>>> 11713.184117225961)
```

Listing 13: The implementation of the Dickey-Fuller Test

The p-value is very less than the significance level of *0.05* and hence we can reject the null hypothesis and take that the time series is stationary (Listing 13).

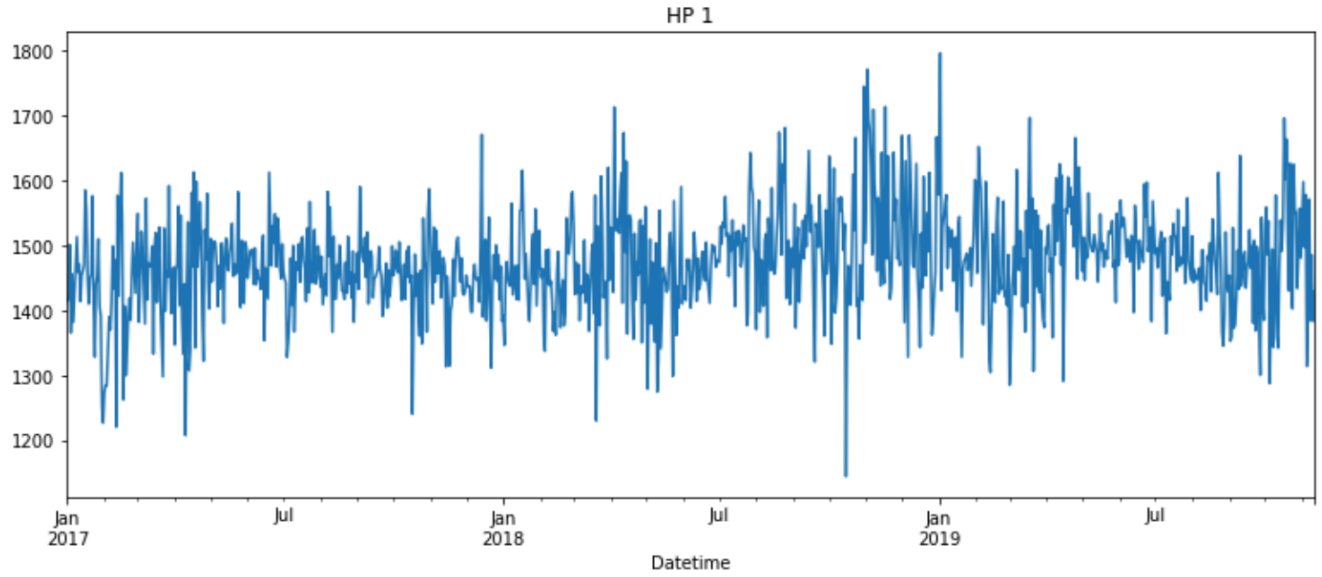

Fig. 9: Plot of the time series data for Hydraulic Pressure 1 dataset. We cannot observe neither the visible seasonality, nor the clear trends, that might give a hint that the time series is stationary

```
from statsmodels.tsa.stattools import kpss

kpss_test = kpss(dataset['hp1'])

kpss_test

>>>(1.548371353793523,
>>> 0.01,
>>> 22,
>>> {'1%': 0.739, '10%': 0.347, '2.5%': 0.574, '5%':
     0.463})
```

Listing 14: The implementation of the KPSS Test

Similarly, the p-value determined in the KPSS test is 0.01 (Listing 14). Assuming significance level is *0.01*, we reject the null hypothesis, and classify these time series as stationary.

6.3.2.4 Data Modeling

The next step is training and testing the model. We extract the time-series values and apply log transform to normalize the data. Then, we split the data to train and test sets. The train dataset consists of the year *2019*, where the behaviour of the aircraft was normal and no event occurred. As our approach stems from the hypothesis that some of the anomalies should occur before or after the event, our test dataset will consist of the following samples: *30 days* before and after the event, *60 days* before the event only, and *30 days* before the event only.

To create the test dataset for the specific event, we need to set the `d` value to the one of the events, for example, 4/17/2017. Then, change the offset value of days in the `td` variable, for example, if we want to get first *30 days* before and after the event, use `days=30`. The `start` is the first date in the test dataset, `end` is the last day. Therefore, to get the *30 days* before and after the event we use `end = date + td`. However, to get the first *30 days* before the event without the *30 days* after, we use `end = date` instead of `end = date + td`. This will set the last date in the dataset to be the same as of the event. That way, if we want to get *60 days* before the event only, we need to set the offset value of `days=60` and `end = date`. Note that the `rm_window` is used for the rolling mean calculation that will be used for the anomaly classification later on. All of the described above is shown in Listing 15.

```
import math
import statsmodels.api as sm
import statsmodels.tsa.api as smt
from sklearn.metrics import mean_squared_error
from matplotlib import pyplot
import matplotlib.pyplot as plt
import plotly.plotly as py
import plotly.tools as tls
import warnings

warnings.filterwarnings('ignore')

# The dates of the anomaly events:
# 4/17/2017 13:05
# 2/28/2018 16:20
# 5/14/2018 12:00
# 6/30/2018 20:45

d = "4/17/2017"
date = pd.to_datetime(d)

#offset value of days
td = pd.offsets.DateOffset(days=30)

#offset value for the rolling mean calculation
rm_window = pd.offsets.DateOffset(days=7)

#use n days before and after the event
start = date - td - rm_window
end = date + td

# use only the first n days days before the event
# end = date

test = dataset.loc[start:end].hp1.values
train = dataset.loc['2019-01-01':].hp1.values

train_log, test_log = np.log10(train), np.log10(test
    )
```

Listing 15: Normalize the data and split the test and train data

We can check the current shape of the test and train arrays by calling `.shape` property. We have *68* entries in the test dataset and *315* entries in the train dataset (note that the dataset contains the data till 2019/11/11, therefore, we get *315* days for test dataset).

```
test.shape
train.shape
>>>(68,)
>>>(315,)
```

Listing 16: Checking the shape of the arrays

Next, we can specify the order of the model:

```
model_order = (1, 1, 1)
model_seasonal_order = (0, 0, 0, 1)
```

Listing 17: Specify the order of the model

Similarly to the model for the Household Electricity Consumption, we loop through train set to predict the next data point and to determine the next data point after that prediction for further forecasting.

```
history = [x for x in train_log]
predictions = list()
predict_log=list()
for t in range(len(test_log)):
    model = sm.tsa.SARIMAX(history, order=
        model_order, seasonal_order=
        model_seasonal_order,
    enforce_stationarity=False,enforce_invertibility
        =False)
    model_fit = model.fit(disp=0)
    output = model_fit.forecast()
    predict_log.append(output[0])
    yhat = 10**output[0]
    predictions.append(yhat)
    obs = test_log[t]
    history.append(obs)
    print('predicted=%f, expected=%f' % (output[0],
        obs))
error = math.sqrt(mean_squared_error(test_log,
    predict_log))
print('Test rmse: %.3f' % error)

figsize=(12, 7)
plt.figure(figsize=figsize)
pyplot.plot(test,label='Actuals')
pyplot.plot(predictions, color='red',label='
    Predicted')
pyplot.legend(loc='upper right')
pyplot.show()
```

Listing 18: Training the model and predicting the values

The result of running the code above gives us predicted and expected values for each data point and then plots a graph that consists of these data points as shown in Figure 10. The RMSE test score for the current test dataset is: *0.025*.

6.3.2.5 Model Optimization

In a similar manner, we use `auto_arima` function and grid search to find the best parameters for the model. By running the function, we can see that the best parameters are `(1,1,1)` (note that some of the output is hidden).

```
from pyramid.arima import auto_arima
stepwise_model = auto_arima(train_log, start_p=0,
    start_q=0,
                            max_p=10, max_q=10, m=1,
                            start_P=0, seasonal=True,
                            d=1, D=1, trace=True,
                            error_action='ignore',
                            suppress_warnings=True,
                            stepwise=True)

>>>Fit ARIMA: order=(1, 1, 0) seasonal_order=(0, 0,
    0, 1); AIC=-1427.186, BIC=-1415.938, Fit time
    =0.083 seconds
>>>Fit ARIMA: order=(0, 1, 1) seasonal_order=(0, 0,
    0, 1); AIC=-1508.673, BIC=-1497.425, Fit time
    =0.107 seconds
>>>Fit ARIMA: order=(1, 1, 1) seasonal_order=(0, 0,
    0, 1); AIC=-1510.230, BIC=-1495.232, Fit time
    =0.261 seconds
```

Listing 19: Auto Arima implementation

By running the grid search function, we can also observe that the best parameters are `(1,1,1)`.

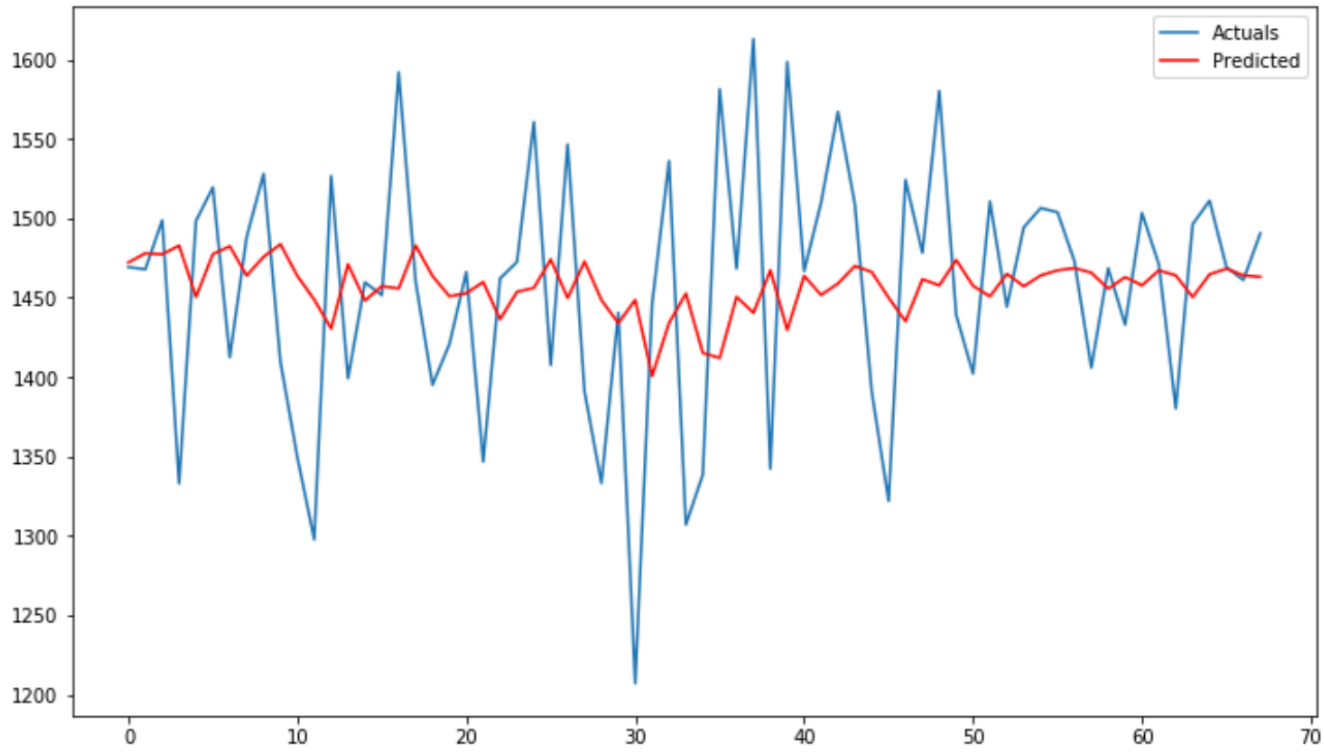

Fig. 10: We loop through train set to predict the next data point and to determine the next data point after that prediction for further forecasting. The graph shows predicted and actual values for the Hydraulic Pressure 1 dataset.

6.3.2.6 Anomaly Detection with Time Series Forecasting

In this section, we have taken the same approach that was followed in one of the tutorials [11], where the author aims to detect anomalies with time series forecasting. We try to experiment with different models, parameters of the model, and pre-processing methods to reproduce the results.

First, we create a dataframe `predicted_df`, with the actual and predicted values. This dataframe can be visualized and used later on when we try to detect anomalies in our data. We create 3 columns: 'datetime' (date), 'actuals' (the actual values) and 'predicted' (the predicted values). Note that we take `start:end` to access the selected rows by the date labels in the same way the test dataset has been created.

```
predicted_df=pd.DataFrame()
predicted_df['datetime']=dataset.loc[start:end].
    index
predicted_df['actuals']=test
predicted_df['predicted']=predictions
predicted_df.reset_index(inplace=True)
del predicted_df['index']

predicted_df.head()
predicted_df
```

The dataset that we get is shown in Figure 11.

	datetime	actuals	predicted
0	2017-03-11	1469.318481	1472.189148
1	2017-03-12	1467.940552	1478.021555
2	2017-03-13	1498.744019	1477.337224
3	2017-03-14	1333.132202	1482.854198
4	2017-03-15	1498.457642	1450.575173
5	2017-03-16	1519.501587	1477.479002

Fig. 11: predicted_df dataset

Next, the author takes the following steps to detect anomalies [11]:

1) Compute the error term (the value of 'actual' - 'predicted').
2) Compute the rolling mean and rolling standard deviation (we take 7 days as a window).
3) Classify the data with an error of 1.5, 1.75 and 2 standard deviations as limits for low, medium and high anomalies respectively.

The code to calculate the variables is shown in Listing 20.

```
import numpy as np
def detect_classify_anomalies(df,window):
    df.replace([np.inf, -np.inf], np.NaN, inplace=
        True)
    df.fillna(0,inplace=True)
    df['error']=df['actuals']-df['predicted']
    df['percentage_change'] = ((df['actuals'] - df['
        predicted']) / df['actuals']) * 100
    df['meanval'] = df['error'].rolling(window=
        window).mean()
    df['deviation'] = df['error'].rolling(window=
        window).std()
    df['-3s'] = df['meanval'] - (2 * df['deviation'
        ])
    df['3s'] = df['meanval'] + (2 * df['deviation'])
    df['-2s'] = df['meanval'] - (1.75 * df['
        deviation'])
    df['2s'] = df['meanval'] + (1.75 * df['deviation
        '])
    df['-1s'] = df['meanval'] - (1.5 * df['deviation
        '])
    df['1s'] = df['meanval'] + (1.5 * df['deviation'
        ])
    cut_list = df[['error', '-3s', '-2s', '-1s', '
        meanval', '1s', '2s', '3s']]
    cut_values = cut_list.values
    cut_sort = np.sort(cut_values)
    df['impact'] = [(lambda x: np.where(cut_sort ==
        df['error'][x])[1][0])(x) for x in
                                range(len(df['error'
                                    ]))]
    severity = {0: 3, 1: 2, 2: 1, 3: 0, 4: 0, 5: 1,
        6: 2, 7: 3}
    region = {0: "NEGATIVE", 1: "NEGATIVE", 2: "
        NEGATIVE", 3: "NEGATIVE", 4: "POSITIVE", 5:
        "POSITIVE", 6: "POSITIVE",
              7: "POSITIVE"}
    df['color'] =  df['impact'].map(severity)
    df['region'] = df['impact'].map(region)
    df['anomaly_points'] = np.where(df['color'] ==
        3, df['error'], np.nan)
    df = df.sort_values(by='datetime', ascending=
        False)
    df.datetime = pd.to_datetime(df['datetime'].
        astype(str), format="%Y-%m-%d")

    return df
```

Listing 20: Classifying anomalies in the data. Source of the code: [11]

Now, we create `classify_df` dataframe by calling `detect_classify_anomalies()` function with the dataset that contains the date, and actual/predicted values. We also specify a window of *7 days* for the rolling mean calculation. The resulting dataframe consists of date, actual and predicted values, error, error change in percent, rolling mean and rolling standard deviation, classified data with an error of 1.5,1.75 and 2 standard deviations and the impact (see Figure 12).

```
classify_df=detect_classify_anomalies(predicted_df
    ,7)
classify_df.reset_index(inplace=True)
del classify_df['index']

classify_df.head()
classify_df
```

Listing 21: Visualize the results (Part 3)

Finally, we can call `plot_anomaly()` function to visualize the results. Note that the code to visualize the results is shown in the Appendix section 8 due to its large size .

```
plot_anomaly(classify_df.iloc[:-7,:],"Pressure_1")
```

Listing 22: Calling plot_anomaly() function

The first graph represents represents the error term (the red line), the moving average (the green line), and the upper and lower limit boundary (dotted lines) (Figure 13). The second graph shows actual values (the blue line), and predicted values (the orange line) (Figure 13).

6.4. Assessment

6.4.1. Discussion

In this deliverable, we have developed a general time series forecasting model that we later generalized to a specific use case. We have experimented with different models such as ARIMA, SARIMA, model-specifc parameters, as well as pre-processing methods to obtain a relatively good predictive performance and acceptable training time. We have produced a step-by-step approach on how to develop a robust data-driven anomaly detection model based on time series forecasting that pinpoints the discrepancy between predictions and real measurements over the time span of the specific failure event.

The same methodology described above can be applied for various time spans (e.g., *30 days* before and after the event, *60 days* before the event, and *30 days* before the event). We will focus on *60 days* before the event, as it provides the most accurate results based on our problem. Moreover, we extend this to four different hydraulic pressure sensors, as we have data from three more sensors. The resulting graphs of this time span are presented in the Appendix section 8 of the paper. The data is shown for each event over four sensors. We can see that our approach has been able to detect the majority of the real anomalies (three out of four events has been associated with at least one anomaly point).

6.4.2. Performance of the model

Precision and recall are two key metrics used to identify the performance of the anomaly detection algorithm [20].

- **Precision**: This metric measures the extent (percentage) of how well the algorithm identifies the real anomalies from the set $S(t)$ for some threshold t [20].
- **Recall**: This metric measures the extent (percentage) of how well the algorithm identifies all anomalies in a given dataset [20]. The percentage of the anomalies captured in $S(t)$ out of the full set of anomalies, is called recall [20].

Here, we give short formal definitions:

$$precision = \frac{true\ positive}{true\ positive\ +\ false\ positive}$$

$$recall = \frac{true\ positive}{true\ positive\ +\ false\ negative}$$

Note that we will not use precision-recall curves to optimize the thresholds, algorithms, or parameters of our models due to lack of time, rather, we use the metrics to measure the predictive performance of the algorithm at the default threshold and model-specific parameters.

Hence, to asses the performance of the model, we should be able to analyze the anomaly detection results on the rest of the dataset. This can be done by splitting the selected time period on test and train dataset (Listing 23) and taking the first *730* dates from the dataset,

	datetime	actuals	predicted	error	percentage_change	meanval	deviation	-3s	3s	-2s	2s	-1s	1s	impact	color	region	anomaly_points
0	2017-05-17	1490.520996	1463.132902	27.388094	1.837485	5.354851	44.438141	-83.521430	94.231132	-72.411895	83.121597	-61.302359	72.012062	4	0	POSITIVE	NaN
1	2017-05-16	1461.131226	1464.108874	-2.977648	-0.203791	7.947242	46.422719	-84.898197	100.792681	-73.292517	89.187001	-61.686837	77.581321	3	0	NEGATIVE	NaN
2	2017-05-15	1469.230347	1468.319024	0.911323	0.062027	4.102960	48.543944	-92.984928	101.190849	-80.848942	89.054863	-68.712956	76.918877	3	0	NEGATIVE	NaN
3	2017-05-14	1511.080933	1464.739093	46.341840	3.066801	5.811501	48.623292	-91.435084	103.058085	-79.279261	90.902262	-67.123438	78.746439	4	0	POSITIVE	NaN
4	2017-05-13	1496.342773	1450.422788	45.919985	3.068815	-9.350709	50.394152	-110.139013	91.437595	-97.540475	78.839057	-84.941937	66.240519	4	0	POSITIVE	NaN
5	2017-05-12	1380.288086	1464.071297	-83.783211	-6.069980	-15.305574	44.942458	-105.190491	74.579343	-93.954876	63.343728	-82.719262	52.108114	2	1	NEGATIVE	NaN
6	2017-05-11	1470.915405	1467.231829	3.683576	0.250427	1.891710	36.636713	-71.381716	75.165135	-62.222538	66.005957	-53.063360	56.846779	4	0	POSITIVE	NaN
7	2017-05-10	1503.362183	1457.827354	45.534828	3.028866	7.426589	39.747131	-72.067673	86.920851	-62.130890	76.984068	-52.194107	67.047285	4	0	POSITIVE	NaN
8	2017-05-09	1433.080322	1462.967940	-29.887618	-2.085551	6.218886	38.505275	-70.791664	83.229436	-61.165345	73.603117	-51.539027	63.976798	3	0	NEGATIVE	NaN
9	2017-05-08	1468.537842	1455.666738	12.871104	0.876457	7.539224	37.197079	-66.854934	81.933383	-57.555665	72.634113	-48.256395	63.334843	4	0	POSITIVE	NaN
10	2017-05-07	1405.980835	1465.774461	-59.793626	-4.252805	14.248751	42.216354	-70.183956	98.681458	-59.629868	88.127370	-49.075780	77.573281	1	2	NEGATIVE	NaN

Fig. 12: classify_df dataset

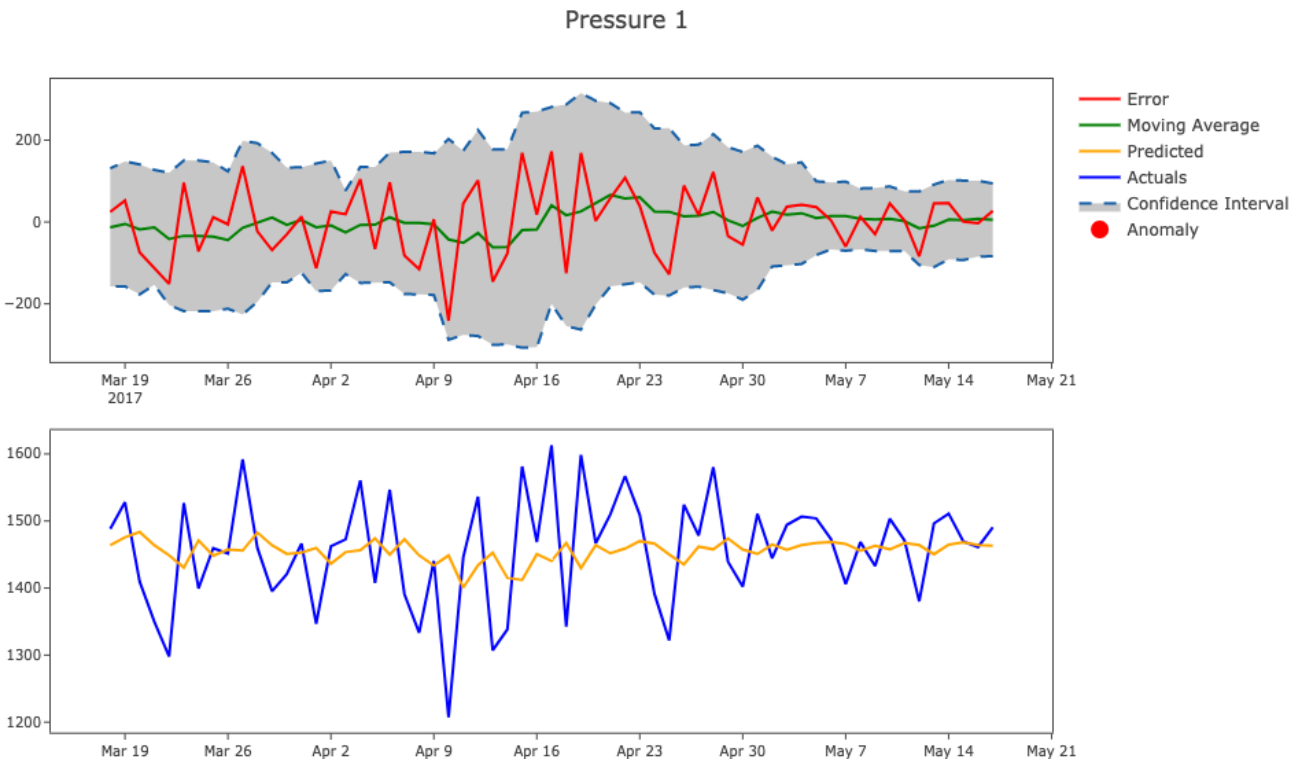

Fig. 13: The sample of the anomaly detection report for the 30 days before and after the event of 4/17/2017. The first graph represents represents the error term (the red line), the moving average (the green line), and the upper and lower limit boundary (dotted lines). The second graph shows actual values (the blue line), and predicted values (the orange line).

as we have a total of *730* data points in the test dataset. The rest of the methodology is still the same. Note that the RMSE score is *0.022* for the selected time period.

```
train = dataset.loc["2019-01-01":].hp1.values
test = dataset.loc["2017-01-01":"2018-12-31"].hp1.
    values

test.shape
train.shape

>>>(730,)
>>>(315,)
```

Listing 23: Splitting train and test datasets

```
predicted_df=pd.DataFrame()
predicted_df['datetime']=dataset['datetime'][:730]
predicted_df['actuals']=test
predicted_df['predicted']=predictions
predicted_df.reset_index(inplace=True)
del predicted_df['index']

predicted_df.head()
predicted_df
```

Listing 24: predicted_df

After plotting the graph (Figure 14), we can observe that a number of the false positives is present (*9 out of 13* anomaly points are the actual false positives), the algorithm was also able to catch at least one anomaly for three out of four of the events as evidenced in the Appendix section 8.

Hence, at the default threshold, in a given time span, we can automatically detect 75% of the real events based on the anomalies. Despite the low precision rate of 30%, the rate in which false negatives occur is 25%. Also, note that due to a high rate of false positives, additional checks can still be necessary.

We expect that further improvements with different models, specific-model parameters, thresholds are possible and can be optimized based on the precision-recall curves.

7. Conclusion

In this project, we have reviewed the state of the art on predictive maintenance for aircrafts, reproduced published results and developed a machine learning approach that combines techniques reviewed in the state of the art capable of giving insights and alerts to help engineers to decide the best moment to proceed with the maintenance on the aircraft. We used aircraft sensor values to identify unusual behaviour on the aircraft hydraulic system that can lead to a potential failure, so that maintenance can be planned in advance.

We first developed a general autoregression model using an open dataset available online and then we extended it and applied anomaly detection algorithms to discover operationally significant anomalies in the aircraft dataset.

References

[1] Sun, H., & Guo, Y. Q. (2017, July). A data-driven anomaly detection approach for acquiring baseline of aircraft engine measurement data. In 2017 36th Chinese Control Conference (CCC) (pp. 7225-7229). IEEE.

[2] Gomes, J. P. P., Rodrigues, L. R., Leão, B. P., Galvão, R. K. H., & Yoneyama, T. (2016). Using degradation messages to predict hydraulic system failures in a commercial aircraft. IEEE Transactions on Automation Science and Engineering, 15(1), 214-224.

[3] Janakiraman, V. M., & Nielsen, D. (2016, July). Anomaly detection in aviation data using extreme learning machines. In 2016 International Joint Conference on Neural Networks (IJCNN) (pp. 1993-2000). IEEE.

[4] Brotherton, T., & Johnson, T. (2001, March). Anomaly detection for advanced military aircraft using neural networks. In 2001 IEEE Aerospace Conference Proceedings (Cat. No. 01TH8542) (Vol. 6, pp. 3113-3123). IEEE.

[5] Peixeiro, M The Complete Guide to Time Series Analysis and Forecasting. https://towardsdatascience.com/the-complete-guide-to-time-series-analysis-and-forecasting-70d476bfe775

[6] Individual household electric power consumption Data Set. https://archive.ics.uci.edu/ml/datasets/individual+household+electric+power+consumption

[7] Time Series: ARIMA Model. https://medium.com/@kangeugine/time-series-arima-model-11140bc08c6

[8] How to Create an ARIMA Model for Time Series Forecasting in Python. https://machinelearningmastery.com/arima-for-time-series-forecasting-with-python/

[9] Prophet: forecasting at scale. https://research.fb.com/blog/2017/02/prophet-forecasting-at-scale/

[10] Seabold, Skipper, and Josef Perktold. "Statsmodels: Econometric and statistical modeling with python." Proceedings of the 9th Python in Science Conference. 2010.

[11] Adithya Krishnan. Anomaly Detection with Time Series Forecasting https://towardsdatascience.com/anomaly-detection-with-time-series-forecasting-c34c6d04b24a

[12] Basora, Luis, Xavier Olive, and Thomas Dubot. "Recent Advances in Anomaly Detection Methods Applied to Aviation." Aerospace 6.11 (2019): 117.

[13] Chandola, Varun, Arindam Banerjee, and Vipin Kumar. "Anomaly detection: A survey." ACM computing surveys (CSUR) 41.3 (2009): 15.

[14] Pimentel, Marco AF, et al. "A review of novelty detection." Signal processing 99 (2014): 215-249.

[15] Bianco, Ana Maria, et al. "Outlier detection in regression models with arima errors using robust estimates." Journal of Forecasting 20.8 (2001): 565-579.

[16] Nanduri, Anvardh, and Lance Sherry. "Anomaly detection in aircraft data using Recurrent Neural Networks (RNN)." 2016 Integrated Communications Navigation and Surveillance (ICNS). IEEE, 2016.

[17] Predictive Maintenance 4.0 - Predict the Unpredictable — PwC Belgium

[18] Vincent A Guide to Time Series Forecasting with ARIMA in Python 3 https://www.digitalocean.com/community/tutorials/a-guide-to-time-series-forecasting-with-arima-in-python-3

[19] SnT - Interdisciplinary Centre for Security, Reliability and Trust https://wwwen.uni.lu/snt

[20] Wheeler Evaluating anomaly detection algorithms with precision-recall curves https://medium.com/wwblog/evaluating-anomaly-detection-algorithms-with-precision-recall-curves-f3eb5b679476

8. Appendix

8.1. The snippet of code used to visualize the results.

```
import plotly.graph_objs as go
from plotly.offline import plot, iplot

def plot_anomaly(df,metric_name):
    dates = df.datetime
    bool_array = (abs(df['anomaly_points']) > 0)

    actuals = df["actuals"][-len(bool_array):]
    anomaly_points = bool_array * actuals
    anomaly_points[anomaly_points == 0] = np.nan
    color_map= {0: 'rgb(228, 222, 249)', 1: "yellow", 2: "orange", 3: "red"}
    table = go.Table(
    domain=dict(x=[0, 1],
                y=[0, 0.3]),
    columnwidth=[1, 2 ],
    header = dict(height = 20,
                  values = [['<b>Date</b>'],['<b>Actual Values </b>'],
                            ['<b>Predicted</b>'], ['<b>% Difference</b>'],['<b>Severity (0-3)</b>']],
                  font = dict(color=['rgb(45, 45, 45)'] * 5, size=14),
                  fill = dict(color='#d562be')),
    cells = dict(values = [df.round(3)[k].tolist() for k in ['datetime', 'actuals', 'predicted',
                                                             'percentage_change','color']],
                  line = dict(color='#506784'),
                  align = ['center'] * 5,
                  font = dict(color=['rgb(40, 40, 40)'] * 5, size=12),

                  suffix=[None] + [''] + [''] + ['%'] + [''],
                  height = 27,
                  fill=dict(color=
                       [df['color'].map(color_map)],
                       )
    ))

    anomalies = go.Scatter(name="Anomaly",
                       x=dates,
                       xaxis='x1',
                       yaxis='y1',
                       y=df['anomaly_points'],
                       mode='markers',
                       marker = dict(color ='red',
                       size = 11,line = dict(
                                        color = "red",
                                        width = 2)))

    upper_bound = go.Scatter(hoverinfo="skip",
                         x=dates,
                         showlegend =False,
                         xaxis='x1',
                         yaxis='y1',
                         y=df['3s'],
                         marker=dict(color="#444"),
                         line=dict(
                             color=('rgb(23, 96, 167)'),
                             width=2,
                             dash='dash'),
                         fillcolor='rgba(68, 68, 68, 0.3)',
                         fill='tonexty')

    lower_bound = go.Scatter(name='Confidence Interval',
                          x=dates,
                          xaxis='x1',
                          yaxis='y1',
                          y=df['-3s'],
                          marker=dict(color="#444"),
                          line=dict(
                              color=('rgb(23, 96, 167)'),
                              width=2,
                              dash='dash'),
                          fillcolor='rgba(68, 68, 68, 0.3)',
                          fill='tonexty')
```

Listing 25: Visualize the results. Source of the code: [11]

```
Actuals = go.Scatter(name= 'Actuals',
                 x= dates,
                 y= df['actuals'],
                 xaxis='x2', yaxis='y2',
                 mode='line',
                 marker=dict(size=12,
                             line=dict(width=1),
                             color="blue"))

Predicted = go.Scatter(name= 'Predicted',
                 x= dates,
                 y= df['predicted'],
                 xaxis='x2', yaxis='y2',
                 mode='line',
                 marker=dict(size=12,
                             line=dict(width=1),
                             color="orange"))

Error = go.Scatter(name="Error",
               x=dates, y=df['error'],
               xaxis='x1',
               yaxis='y1',
               mode='line',
               marker=dict(size=12,
                           line=dict(width=1),
                           color="red"),
               text="Error")

anomalies_map = go.Scatter(name = "anomaly_actual",
                           showlegend=False,
                           x=dates,
                           y=anomaly_points,
                           mode='markers',
                           xaxis='x2',
                           yaxis='y2',
                           marker = dict(color ="red",
                           size = 11,
                           line = dict(
                             color = "red",
                             width = 2)))

Mvingavrg = go.Scatter(name="Moving_Average",
                       x=dates,
                       y=df['meanval'],
                       mode='line',
                       xaxis='x1',
                       yaxis='y1',
                       marker=dict(size=12,
                                   line=dict(width=1),
                                   color="green"),
                       text="Moving_average")

axis=dict(
showline=True,
zeroline=False,
showgrid=True,
mirror=True,
ticklen=4,
gridcolor='#ffffff',
tickfont=dict(size=10))

layout = dict(
width=1000,
height=865,
autosize=False,
title= metric_name,
margin = dict(t=75),
showlegend=True,
xaxis1=dict(axis, **dict(domain=[0, 1], anchor='y1', showticklabels=True)),
xaxis2=dict(axis, **dict(domain=[0, 1], anchor='y2', showticklabels=True)),
yaxis1=dict(axis, **dict(domain=[2 * 0.21 + 0.20 + 0.09, 1], anchor='x1', hoverformat='.2f')),
yaxis2=dict(axis, **dict(domain=[0.21 + 0.12, 2 * 0.31 + 0.02], anchor='x2', hoverformat='.2f')))

fig = go.Figure(data = [table,anomalies,anomalies_map,
                    upper_bound,lower_bound,Actuals,Predicted,
                    Mvingavrg,Error], layout = layout)

plot(fig)
pyplot.show()
```

Listing 26: Visualize the results. Source of the code: [11]

8.2. Data for the first two years:

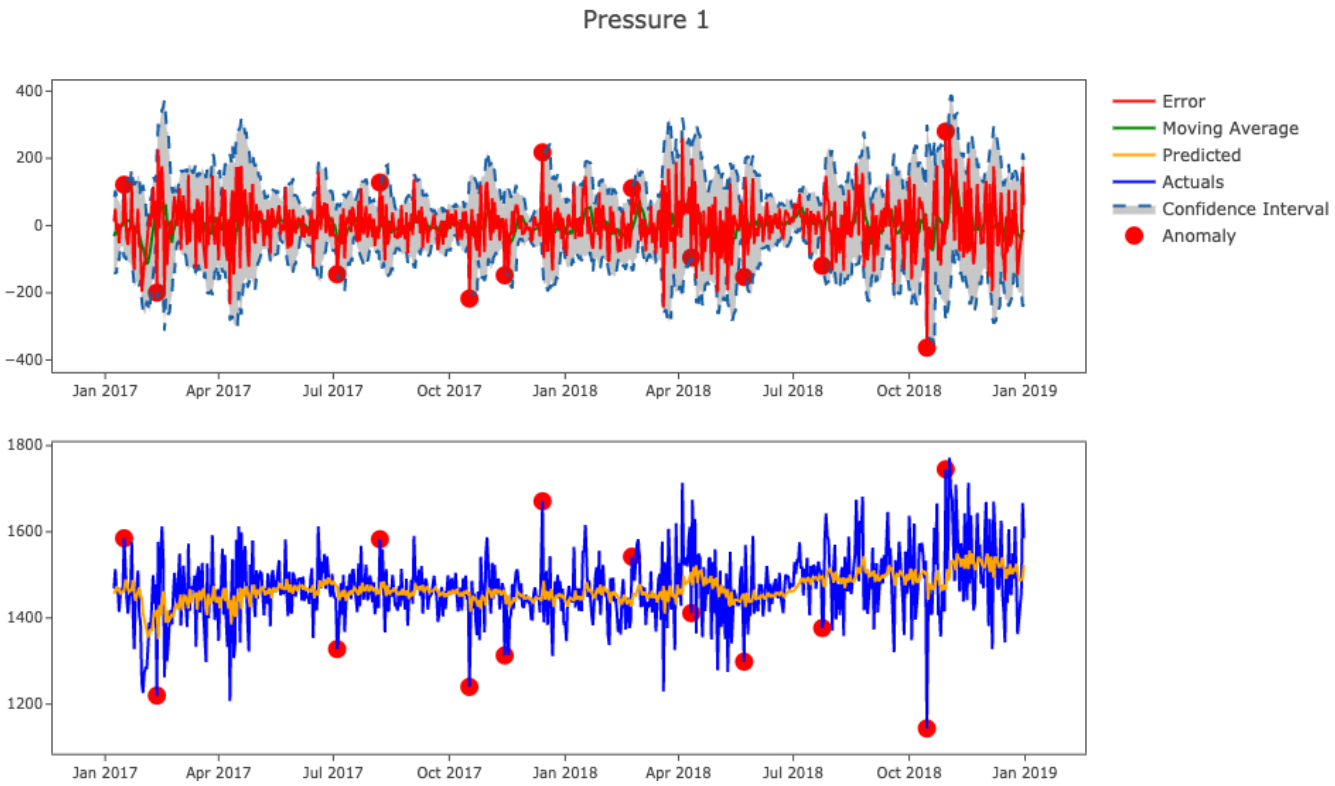

Fig. 14: The data from Hydraulic Pressure 1 for the first two years (*730* days). We split the selected time period on test and train dataset and take the first *730* dates from the dataset, as we have a total of *730* data points in the test dataset.

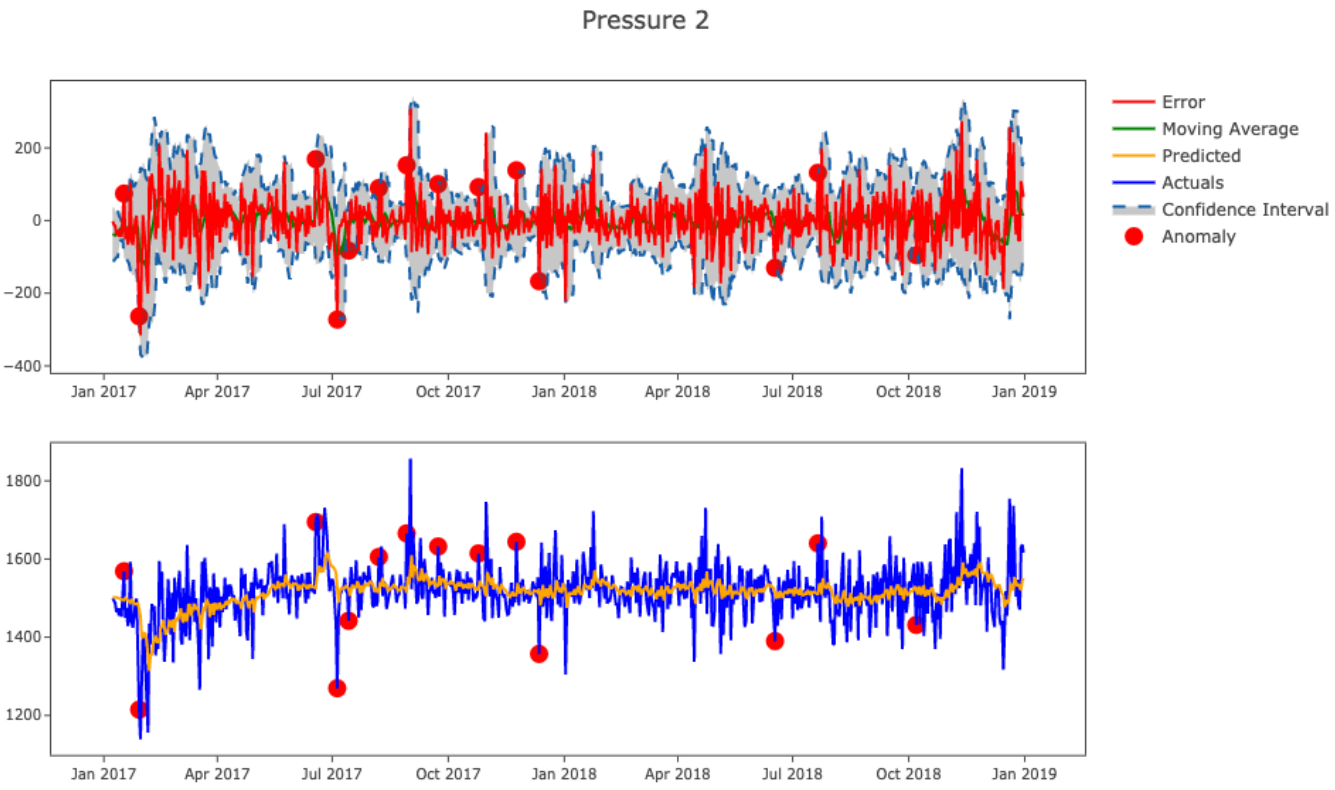

Fig. 15: The data from Hydraulic Pressure 2 for the first two years (*730* days). We split the selected time period on test and train dataset and take the first *730* dates from the dataset, as we have a total of *730* data points in the test dataset.

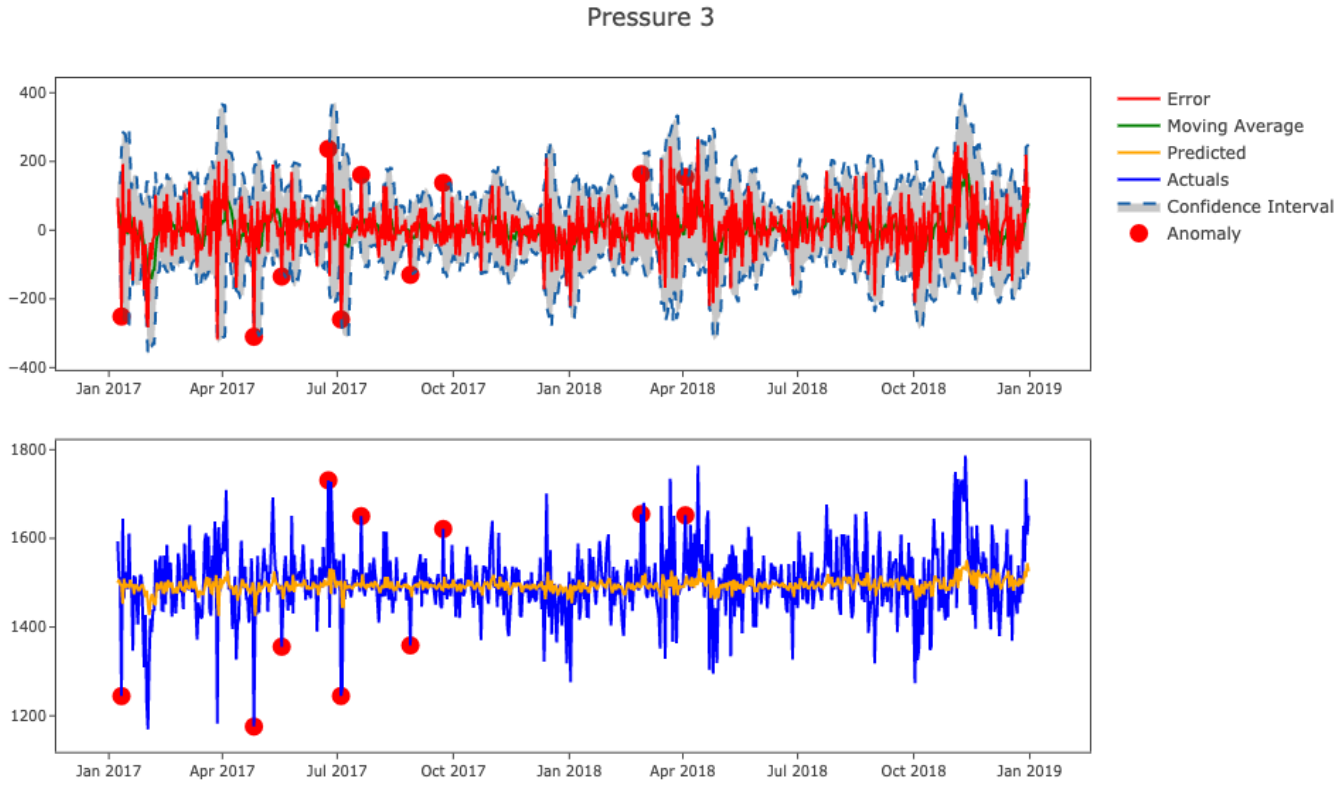

Fig. 16: The data from Hydraulic Pressure 3 for the first two years (*730* days). We split the selected time period on test and train dataset and take the first *730* dates from the dataset, as we have a total of *730* data points in the test dataset.

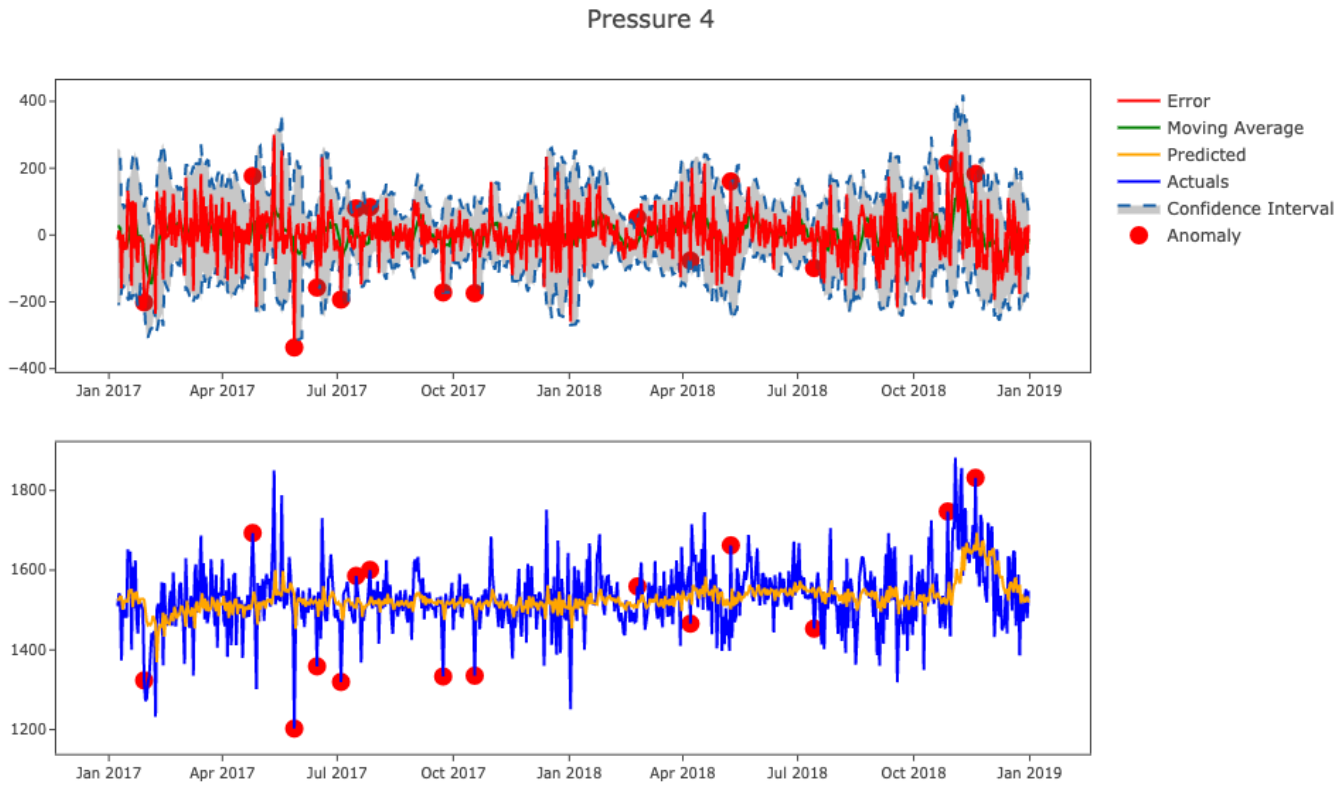

Fig. 17: The data from Hydraulic Pressure 4 for the first two years (*730* days). We split the selected time period on test and train dataset and take the first *730* dates from the dataset, as we have a total of *730* data points in the test dataset.

8.3. The event of 4/17/2017:

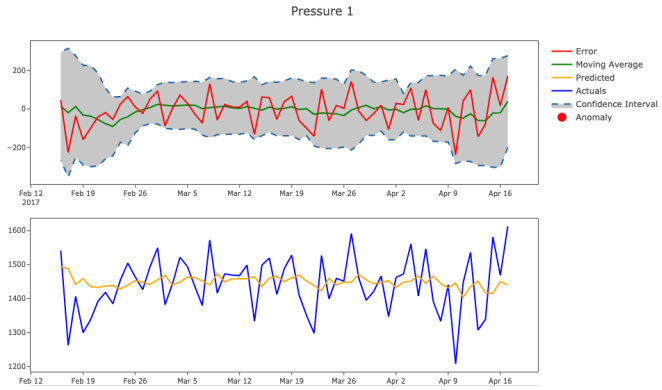

Fig. 18: The data from Hydraulic Pressure 1. The data is shown for 60 days before the event of 4/17/2017.

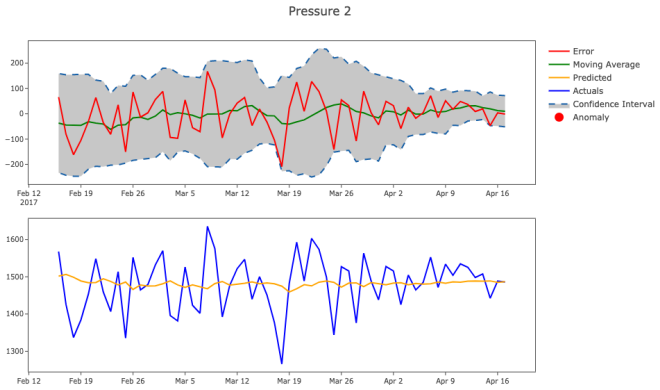

Fig. 19: The data from Hydraulic Pressure 2. The data is shown for 60 days before the event of 4/17/2017.

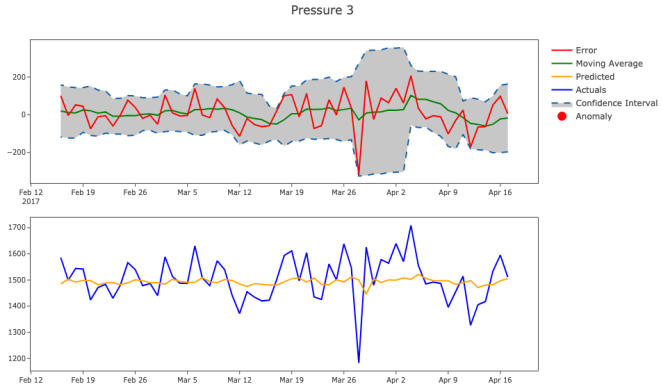

Fig. 20: The data from Hydraulic Pressure 3. The data is shown for 60 days before the event of 4/17/2017.

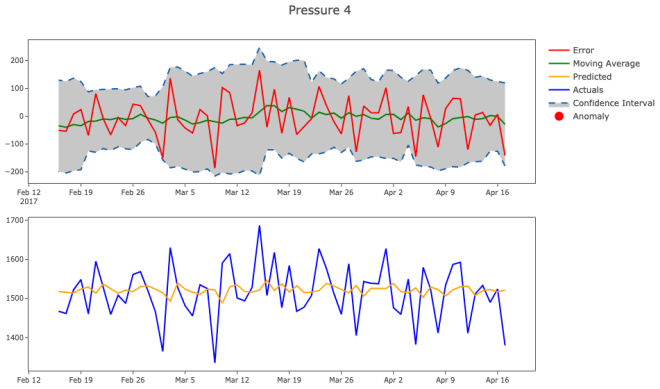

Fig. 21: The data from Hydraulic Pressure 4. The data is shown for 60 days before the event of 4/17/2017.

8.4. The event of 2/28/2018:

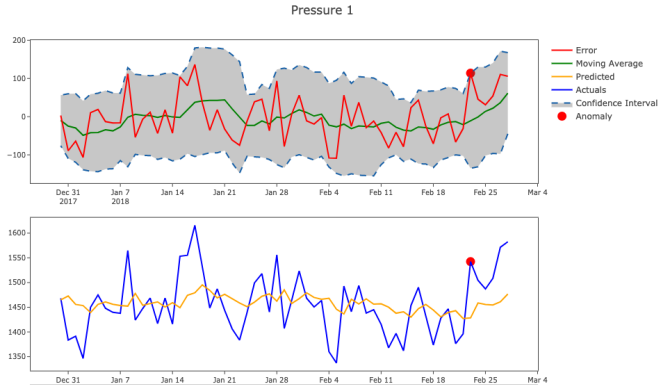

Fig. 22: The data from Hydraulic Pressure 1. The data is shown for 60 days before the event of 2/28/2018.

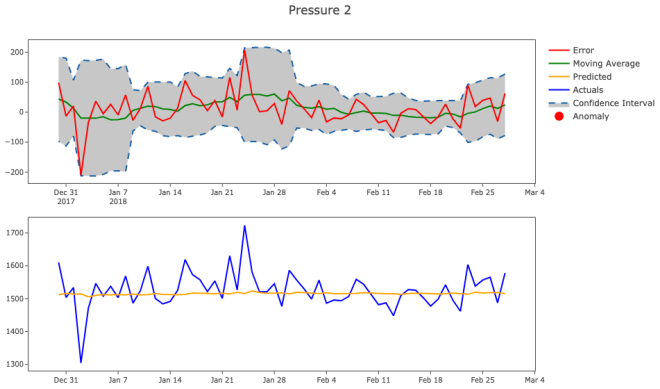

Fig. 23: The data from Hydraulic Pressure 2. The data is shown for 60 days before the event of 2/28/2018.

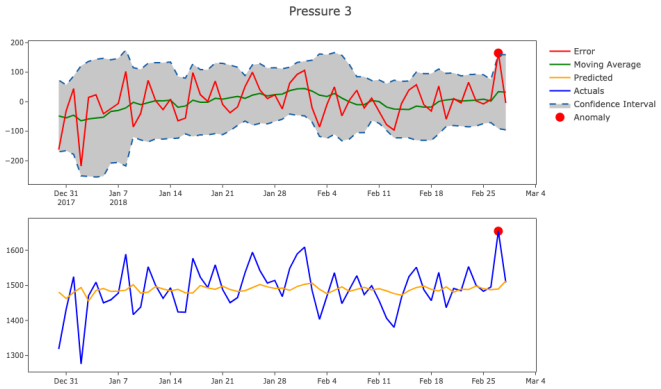

Fig. 24: The data from Hydraulic Pressure 3. The data is shown for 60 days before the event of 2/28/2018.

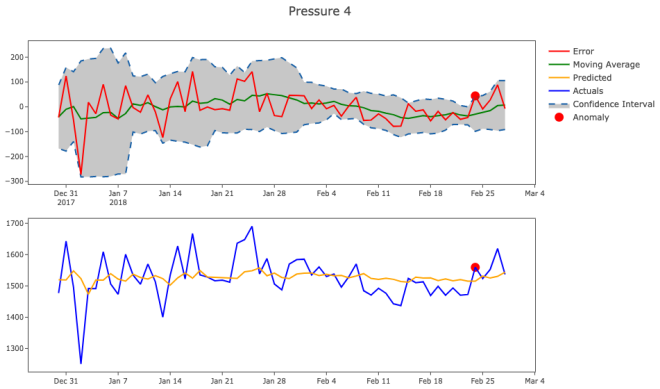

Fig. 25: The data from Hydraulic Pressure 4. The data is shown for 60 days before the event of 2/28/2018.

Fig. 26: 30 days before the event only

8.5. The event of 5/14/2018:

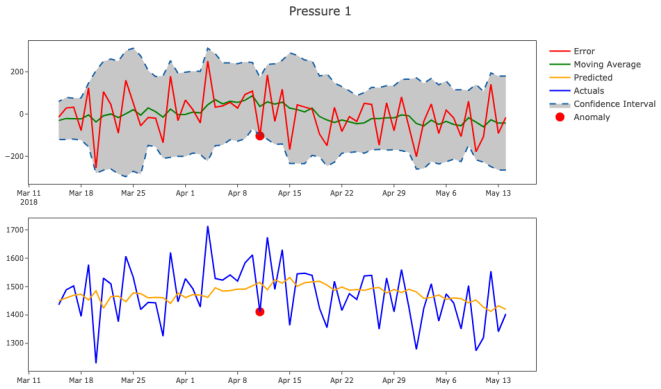

Fig. 27: The data from Hydraulic Pressure 1. The data is shown for 60 days before the event of 5/14/2018.

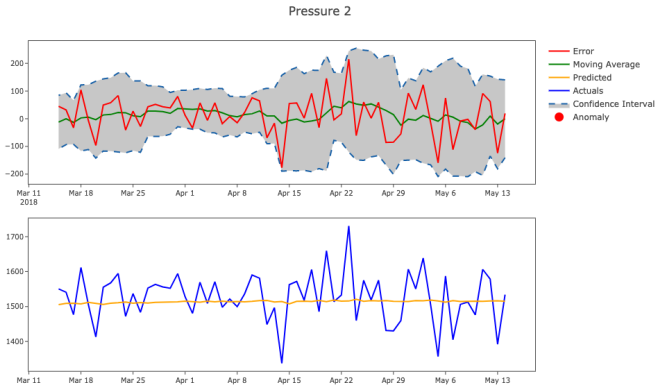

Fig. 28: The data from Hydraulic Pressure 1. The data is shown for 60 days before the event of 5/14/2018.

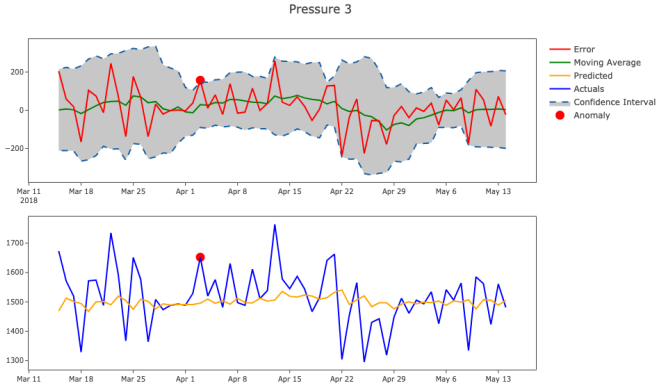

Fig. 29: The data from Hydraulic Pressure 3. The data is shown for 60 days before the event of 5/14/2018.

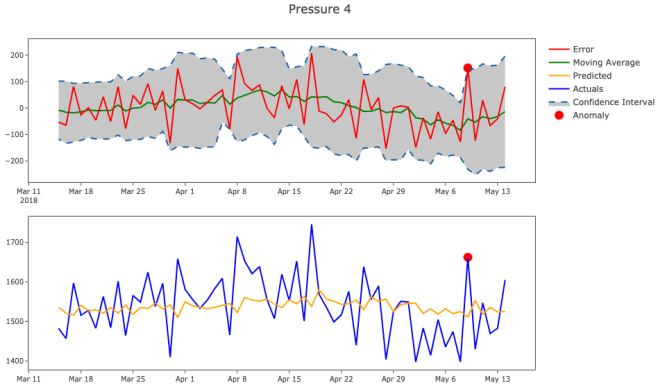

Fig. 30: The data from Hydraulic Pressure 4. The data is shown for 60 days before the event of 5/14/2018.

Fig. 31: 30 days before the event only

8.6. The event of 6/30/2018:

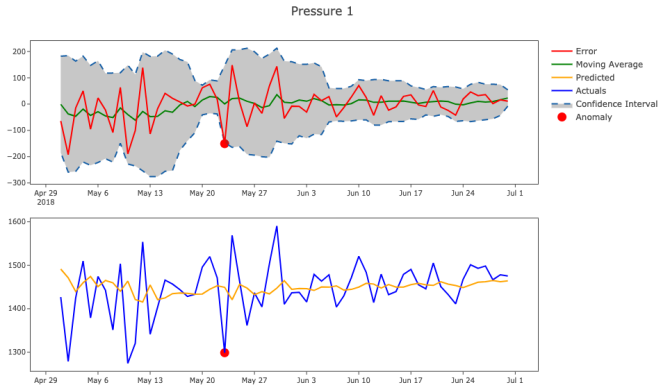

Fig. 32: The data from Hydraulic Pressure 1. The data is shown for 60 days before the event of 6/30/2018.

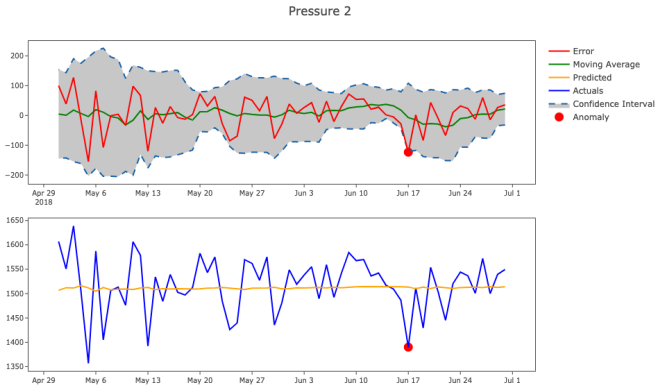

Fig. 33: The data from Hydraulic Pressure 2. The data is shown for 60 days before the event of 6/30/2018.

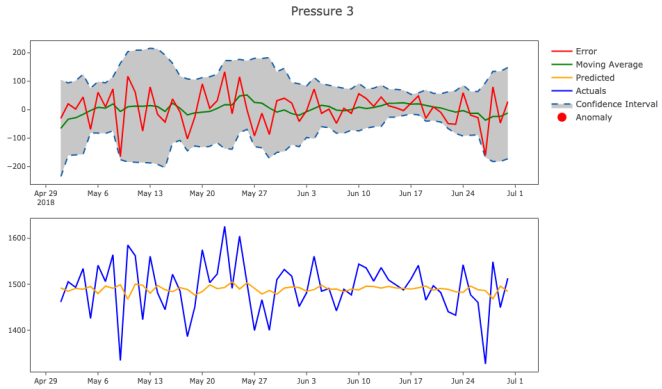

Fig. 34: The data from Hydraulic Pressure 3. The data is shown for 60 days before the event of 6/30/2018.

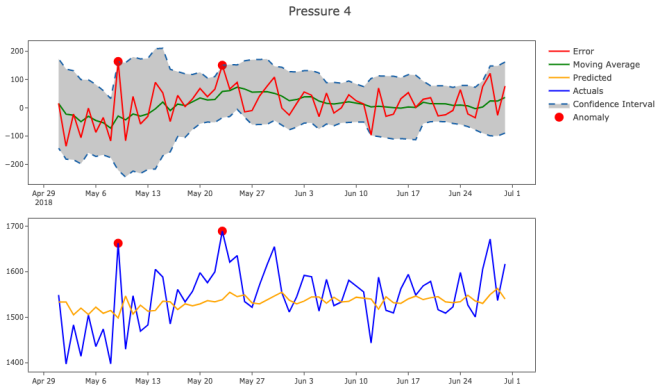

Fig. 35: The data from Hydraulic Pressure 4. The data is shown for 60 days before the event of 6/30/2018.

Fig. 36: 30 days before the event only