Machine Learning for Predictive Maintenance

Author: Mustafa Nab*, Gazi University/Turkey

Supervisors:

Luca Gelisio

Danilo Enoque Ferreira de Lima

Arman Davtyan

Steffen Hauf

September 24, 2022

Abstract

Thousands of devices and sensors in European XFEL generate a large amount of data. For predictive maintenance reasons, these data essentially needed to be processed to find abnormal behavior. Therefore, time series data is segmented into small window sizes of 10 seconds, and the frequency magnitude for every window frame is estimated. The result is compared with the remaining, in order to figure out which time frame has high density magnitude. With the proposed method, it is possible to determine the threshold between abnormal and normal data.

*mustafa.nab@gazi.edu.tr

Contents

1 Introduction

Day after day, the volume of data produced by industry and scientific organizations is growing rapidly. The European XFEL is one of the research facilities that offers ultrashort X-ray pulses of outstanding spatial coherence and spectral brilliance [1]. At the European XFEL, X-ray experiments with up to 27000 photon pulses per second, arranged into 10 Hz trains of pulses at 4.5 MHz, can be performed. State-of-the-art technology, high-repetition-rate, and 2D image detector are capable of generating the images of scattered photons, produced by a single XFEL photon pulse [2].

Karabo is a distributed control system which also allows data acquisition [2]. It stores several properties (e.g. temperatures, voltages, ...) in a time-series Influx database. These are therefore available for future analysis.

The objective of this project at European XFEL [3] is to detect anomalies within the time series data. To do so, the time scale which has a high spectral density variance compared to other time scales is considered to be an abnormal. The project is in its early stage and aims at mitigating failure and downtime in facility by predicting failures. Hereby, in this project part of the task is to find an efficient mathematical method to collect and pre-process data from the Influx database, containing information from different devices.

After the goal of the project has been stated, this report aims at summarizing the work performed. In this work, only the conventional anomaly detection method in time series data is implemented and some other techniques are reviewed as they might be interesting in the next stages of the project. In the introduction section, there are a brief review and interpretation of time series data analysis and anomaly detection techniques. Afterward, in the experimental section, practical implementation and the result of the work are discussed more in detail. In the end, the deduction of the work and discussion about this work are explained.

1.1 Time Series Data Analysis

A time series is defined as set of observations that are ordered in the temporal axes, and it can be discrete or continuous. The analysis of time series data played a significant role in early natural science. For example, astronomers used time series of the relative stars and planets to predict astronomical events[4]. Moreover, the analysis of time series data can help to detect abnormalities, and irregularities, extract some patterns, or predict future values based on previously observed values. Typically, irregularity, inconsistency, abnormality, and time hiatus are natural outcomes of the data generation process that is often seen in scientific experiment data. So, time series analysis is usually used to detect changes in the evolution of a time series from before-to-after some intervention or unusual changes happening. It could be done through some statistical calculations or some state-of-the-art Machine learning or Deep Learning Models. Different techniques

of statistics and Machine Learning proposed in the literature are explained in a later section.

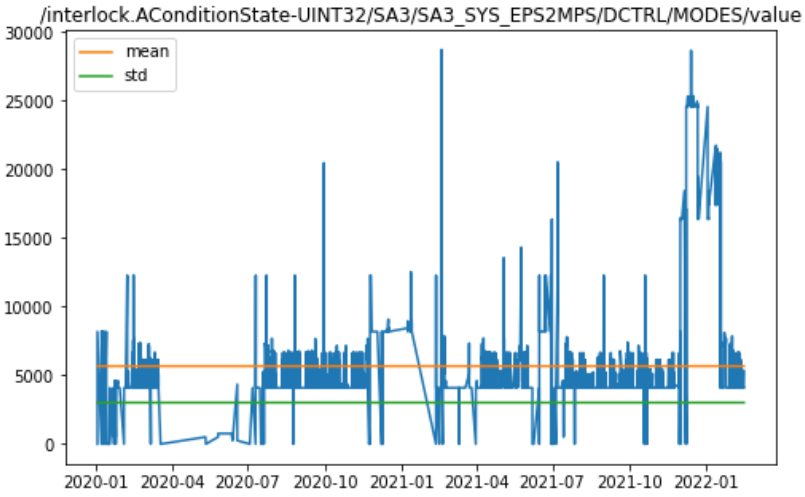

Figure 1: A time series data from a European XFEL device retrieved from the Influx database.

Figure 1 shows is thousands of samples generated by an experimental device in European XFEL. From the current plot, it's difficult to distinguish between regular and abnormal data, which justify, together with the requirement of automation, the need for filtering abnormal observations using statistical or Machine Learning techniques. It should be noted that data are typically not sampled on a regular grid, which complicates further analysis.

1.2 Anomaly Detection Techniques

1.2.1 ARMA Model

In recent years, there has been great interest in studying time series data from various scientific fields. The first common technique is ARMA Model which is a merger of the auto-regressive model and moving average model. The ARMA model is implemented on the irregular sample data[5] or evenly sampled data with a combination of Slotted nearest neighbor re-sampling. As the irregular observation creates a bias on the whole sample data, the multi-shift slotted technique is combined with ARMA. ARMA model is usually used in time series data decomposition and forecasting purposes and it works quite well while the time series data is stationary, which means that statistical properties, such as mean and variance don't change over time. Since most of the time series data is not stationary, it should be converted to stationary first. Then, can be used for further analysis. Here the mathematical equation of ARMA model presents as follows:

$$y_t = c + \phi_1 y_{t-1} + \epsilon_t \tag{1}$$

Where y_t is the value at time step, t, c is a constant, ϕ_1 is a coefficient, and ϵ_t is white noise term with ϵ_t $N(0, \sigma^2)$. The AR model however takes an order, p, which will dictates how many prior time steps to use in the regression. An AR(p) model can be expressed as:

$$y_t = c + \phi_1 y_{t-1} + \phi_2 y_{t-2} ... + \phi_p y_{t-p} + \epsilon_t \tag{2}$$

This is:

$$y_t = c + \sum_{i=1}^{p} \phi_i y_{t-i} \tag{3}$$

Where ϕ_i is the corresponding coefficient for each respective prior time step y_{t-i} therefor:

$$\phi = (\phi_1, \phi_2, \phi_i) \tag{4}$$

The MA Model express as follow:

$$y_t = c + \theta_1 \epsilon_{t_1} \tag{5}$$

In comparison to AR Model, an MA model is a linear regression of the current value of the series against previously observed white noise error terms. and similarly, to AR, MA also takes an order term(q), which imposes how many prior errors to be considered. The MA(q) model expresses as follows:

$$y_t = c + \theta_1 y_{t-1} + \theta_2 y_{t-2} ... + \theta_p y_{t-p} + \epsilon_t \tag{6}$$

Where θ_j is the corresponding coefficient for each respective prior error ϵ_{t_j} therefor:

$$\theta = (\theta_1, \theta_2, \theta_j) \tag{7}$$

The combination of AR and MA can be presented as ARMA:

$$y_t = c + \sum_{i=1}^{p} \theta_i y_{t-i} + \sum_{j=1}^{q} \theta_j \epsilon_{t-j} + \epsilon_t \tag{8}$$

1.2.2 Fourier Transform

The next method is Fourier Transform, which is widely used in different scientific fields. The Fourier Transform has more practical implementation in data analysis[6] compare to theoretical physics. The Fourier Transform performs well when we are exploring the frequency content of a signal or where each frequency is dominant in that content. The DFT reveals the whole frequency component of a signal time series and the formula is as follows:

$$x_k = \sum_{n=0}^{N-1} x_n . e^{-i 2\pi k n / N} = \sum_{n=0}^{N-1} x_n [cos(2\pi k n / N) - i.sin(2\pi k n / N)] \tag{9}$$

Where N is the number of samples, n is the current sample, k indicate the current frequency, where $k \in [0, N-1]$, x_n is the sine value at sample n and X_k the matrix which includes both amplitude, and phase. The last expression is derived from Euler's formula, which links trigonometric functions to the complex exponential function $e^{i.x} = cosx + i.sinx$.

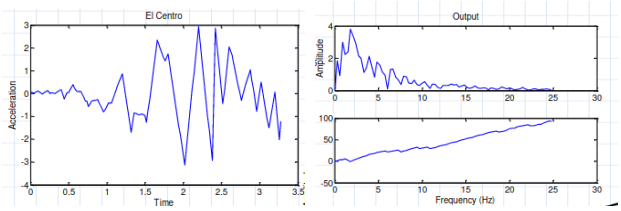

Figure 2: Signal Decomposition by Fourier Transform[7]

In Figure 2 the Fourier Transform is applied to an example dataset. The phase component indicates the frequency changes after transformation.

The basic concept behind STFT is that STFT breaks up the signal in the time domain into a number of signals of shorter duration, then transforms each signal into a frequency domain. The formulation is presented as follows:

$$STFT\{x(T)\}(\tau, \omega)\Xi = \int_{-\infty}^{\infty} x(t)w(t-\tau)e^{iwt}dt \tag{10}$$

In continuous time STFT $x(t)$ is the time-domain signal to be transformed, τ is slow time; small frame of t, ω is the frequency, $w(t)$ refers to window functions such Hann window or Gaussian window bell centered around zero and $X(\tau, \omega)$ is a complex function that represents the phase and magnitude of the signal over time and frequency; this is the Fourier Transform of $x(t)w(t-\tau)$.

$$STFT\{x_n\}(m, \omega)\Xi x(m, \omega) = \sum_{n=-\infty}^{\infty} x_n w_{n-m} e^{-i\omega t_n} \tag{11}$$

Where x_n is a sequence of discretized time-domain signals to be transformed, m is the time index, ω is the frequency, w_n is the sequence of the discretized window function, and $X(m, \omega)$ is the short Fourier Transform of time domain sequence. In this formula, time is discretized but the frequency is continuous, if Fast Fourier Transform is being used, both will be discrete.

1.2.3 Lomb–Scargle Algorithm

Another approach is based on the Lomb–Scargle algorithm. This algorithm shows robust performance in detecting periodic patterns in unevenly spaced data in time series[8].

This is a well-known algorithm for detecting and characterizing periodicity in unevenly sampled time series, and has seen particularly wide use within the astronomy community. Recently has been also applied to biological experiment data[9]. In this case, the biological experiment resulted in periodic patterns in time series and since the Fourier Transform (FT) does not perform well when values are placed evenly and there is no missing value, the Lomb–Scargle has been applied. For a time series comprising N_t measurements $X_j \Xi X(t_j)$ sampled at times $t_j (j = 1, ..., N_t)$, assumed throughout to have been scaled and shifted such that its mean is zero and its variance is unity, the normalized L-S periodogram at frequency is:

$$P_n(f) = \frac{1}{2}\{\frac{[\sum_j x_j cos\omega(t_j - \tau)]^2}{\sum_n cos^2(t_n - \tau)} + \frac{[\sum_j x_j sin\omega(t_j - \tau)]^2}{\sum_n sin^2(t_n - \tau)}\} \tag{12}$$

Here $\omega \Xi 2\pi f$ is the angular frequency and all summations run from $j = 1$ to $j = N_t$. The frequency-dependent time offset τ is evaluated at each ω frequency via:

$$tan2\omega\tau = \frac{\sum_j sin2\omega t_j}{\sum_j cos2\omega t_j} \tag{13}$$

1.2.4 Standard deviation

Another, simpler, approach to identify anomalies is based on the estimation of the standard deviation within the specific window frame of time series data. The standard deviation is the scatter of data points relative to its mean. The formula for population standard deviation is like the following:

$$\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \mu)^2} \tag{14}$$

Where σ is the population standard deviation and μ is the assumed mean. The formula for sample standard deviation is:

$$s = \sqrt{\frac{1}{n-1}\sum_{i=1}^{N}(x_i - \bar{x})^2} \tag{15}$$

Where s is the sample standard deviation and $\bar{x}$ is arithmetic mean of the observations.

1.2.5 STL Decomposition

When the time series is stationary and seasonal, the STL decomposition (Seasonal and Trend decomposition using Loess) is applicable to distinguish outliers. It basically separates a signal series into seasonal, trend, and residual[10]. The residuals are identified as

outliers. To measure the strength of trend and seasonality in time series can be expressed as follows:

$$y_t = T_t + S_t + R_t, \tag{16}$$

Where T_t is the smoothed trend component, S_t is the seasonal component and R_t is a remainder component.

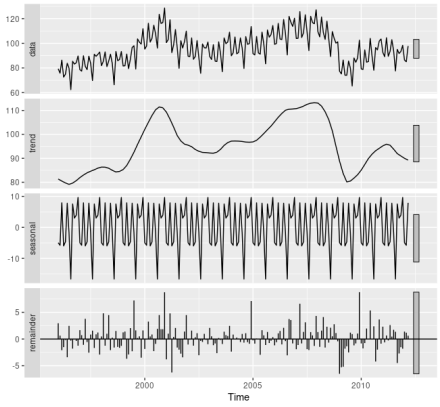

Figure 3: Signal Decomposition by STL, example data[10]

In Figure3 Three additive components obtained from a robust STL decomposition with flexible trend-cycle, fixed seasonality and residual are shown.

2 Experiment and Result

In data Analysis in order to understand a problem, it is better to get acquainted with the data and the problem domain. Thus, In the first stage, the existing implementation, which is written in python language, is being reviewed. Furthermore, The data has been retrieved from the Influx database for further exploratory analysis. The exploratory analysis is performed in Jupyter notebook and python libraries such as pandas, matplolib, h5py, numpy are used.

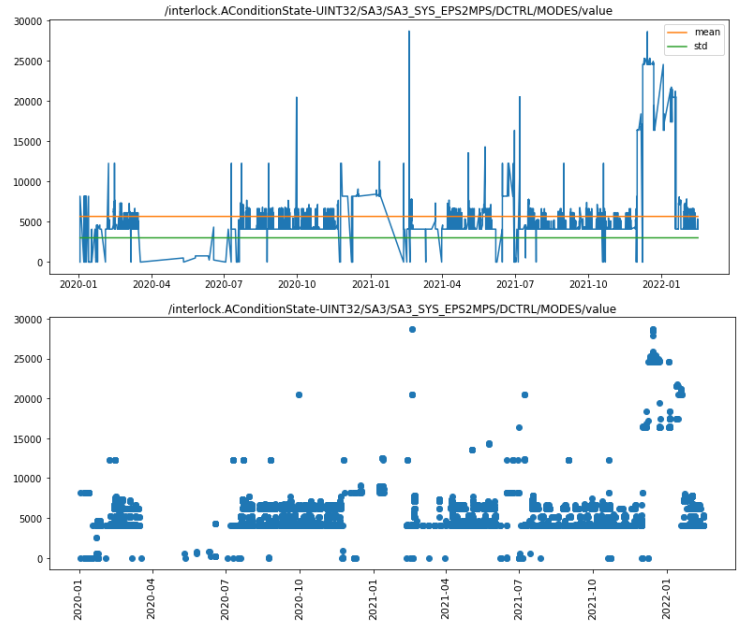

Figure 4: Time series data from one of one device

In figure 4, data points are more scattered from the mean for some time intervals.

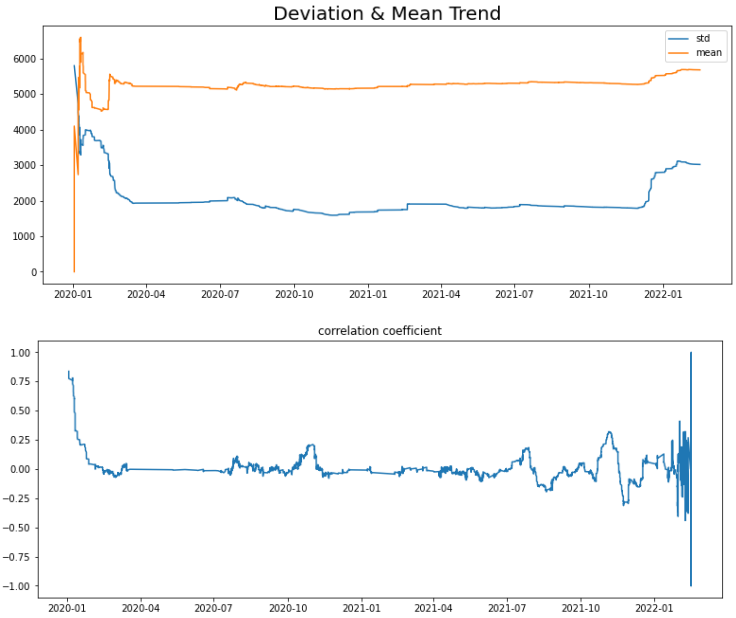

Figure 5: Standard deviation and correlation trend

In figure 5 The standard deviation change similarly to the mean.

Since the oscillation in every time frame is not resemble till the end. The standard deviation technique discussed in the introduction is not applicable to this problem. As The ARMA is an iterative model to calculate the squared error for every observation. Therefor, the time and space complexity in this operation is tremendously high. The other approach like Fourier Transform and Lomb-scargle is a good option for this problem, if it is being merged with Machine Learning or Deep Learning Model; Models such as Convolutional Neural networks, XGBoost or Random Forest.

After understanding the data and problem domain, anomaly detection techniques are available in the literature, being studied. From a different perspective, the windowing function seems to be applicable to this data, which consists in separating the time series into small chunks. In particular, the windowing function was applied and the time series was divided into chunks of 10 seconds. By counting the number of times that an event occurs within a specific time period, and then dividing the count by the length of the time period or window frame, the comparison with other frames is possible, and this gives the frequency rate per frame.

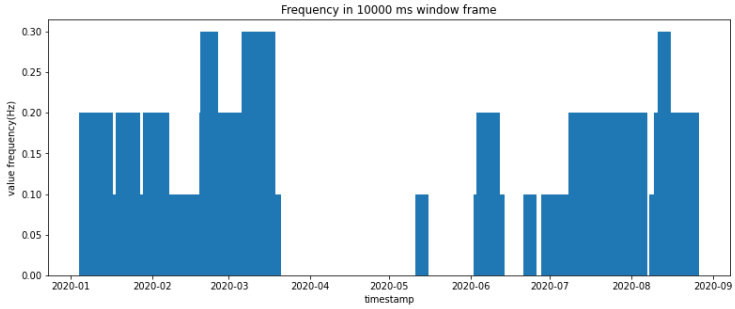

Figure 6: Frequency rate in 10 second time frame

In figure 6 the timestamps are comparable with each other as a result of a new feature called frequency. This technique is applied to two more devices' data to make a comparison between them and to see at which timestamp there is more unstable stream behavior.

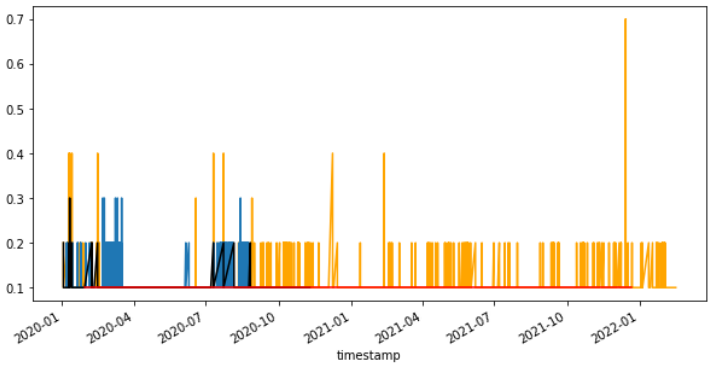

Figure 7: Comparison of devices data in terms of frequency

Figure 7 shows that the trend of frequencies is between 0.1 and 0.2 Hz, and as a result those timestamps which have a frequency over 0.2 are potentially to be considered abnormal.

3 Discussion and Conclusions

The method studied above is simple and fast. One may use it to identify unusually large variations in the data, by triggering an alarm if the calculated counts raise above a given threshold. Another approach to be researched is to use such information in complex Machine Learning models, on which the self-consistency of the preprocessed information is checked against past events.

4 Acknowledgement

I would like to express my sincere thanks to my supervisors Luca Gelisio, Danilo Enoque Ferreira de Lima, Arman Davtyan, and Steffen Hauf, who guided me in doing this project and motivated me to go through during the whole project time enthusiastically. Especially, Danilo Enoque Ferreira de Lima, who offered plenty of his time and provided me invaluable advice, which ended up in the successful accomplishment of my project. At last, I would like to thank DESY and XFEL for providing this opportunity, where, it became possible to work on time series data analysis project and obtain enormous experience.

References

[1] Mission and vision. online: https://www.xfel.eu/organization/mission/indexeng.html. accessed: July 19, 2022 - september 8, 2022.

[2] Steffen Hauf, Burkhard Heisen, Steve Aplin, Marijan Beg, Martin Bergemann, Valerii Bondar, Djelloul Boukhelef, Cyril Danilevsky, Wajid Ehsan, Sergey Essenov, Riccardo Fabbri, Gero Flucke, Daniel Fulla Marsa, Dennis Göries, Gabriele Giovanetti, David Hickin, Tobiasz Jarosiewicz, Ebad Kamil, Dmitry Khakhulin, Anna Klimovskaia, Thomas Kluyver, Yury Kirienko, Manuela Kuhn, Luis Maia, Denys Mamchyk, Valerio Mariani, Leonce Mekinda, Thomas Michelat, Astrid Münnich, Anna Padee, Andrea Parenti, Hugo Santos, Alessandro Silenzi, Martin Teichmann, Kerstin Weger, John Wiggins, Krzysztof Wrona, Chen Xu, Christopher Youngman, Jun Zhu, Hans Fangohr, and Sandor Brockhauser. The karabo distributed control system. *Journal of Synchrotron Radiation*, 26:1448–1461, 9 2019.

[3] Machine learning for predictive maintenance. desy-konferenzverwaltung (indico). online: https://indico.desy.de/event/33121/contributions/121195/. accessed: July 19, 2022 - september 8, 2022.

[4] Gebhard Kirchgässner and Jürgen Wolters. *Introduction to Modern Time Series Analysis*. Springer-Verlag Berlin Heidelberg, Berlin, Heidelberg, 2007.

[5] Piet M.T. Broersen. Time series analysis for irregularly sampled data. *IFAC Proceedings Volumes*, 38(1):154–159, 2005. 16th IFAC World Congress.

[6] Gerhard Heinzel, A. O. Rüdiger, and Roland Schilling. Spectrum and spectral density estimation by the discrete fourier transform (dft), including a comprehensive list of window functions and some new at-top windows. 2002.

[7] Slide 1 —an introduction to short-time fourier transform (stft) m ahmadizadeh. online: http://sharif.edu/ ahmadizadeh/courses/advstdyn/short-time 20fourier 20transform.pdf. accessed: September 2, 2022.

[8] Jacob T. VanderPlas. Understanding the lomb–scargle periodogram. *The Astrophysical Journal Supplement Series*, 236(1):16, may 2018.

[9] Earl F. Glynn, Jie Chen, and Arcady R. Mushegian. Detecting periodic patterns in unevenly spaced gene expression time series using lomb-scargle periodograms. *Bioinformatics*, 22:310–316, 2 2006.

[10] Robin John Hyndman and George Athanasopoulos. *Forecasting: Principles and Practice*. OTexts, Australia, 2nd edition, 2018.

APPENDIX

[121]:
```
import h5py
import numpy as np
import pandas as pd
```

[122]:
```
filename = "/gpfs/exfel/data/user/danilo/influx/new_scraped.h5"

def h5py_dataset_iterator(g, prefix=''):
    for key, item in g.items():
        path = '{}/{}'.format(prefix, key)
        if isinstance(item, h5py.Dataset):
            yield (path, item)
        elif isinstance(item, h5py.Group):
            yield from h5py_dataset_iterator(item, path)
```

[123]:
```
with h5py.File(filename, 'r') as f:

    directory = []
    dataset = []

    df = pd.DataFrame()

    for (path, dset) in h5py_dataset_iterator(f):
        directory.append(path)
        dataset.append(np.array(dset[()]))

timestamp, value = dataset[::2], dataset[1::2]
timestamp_dir, value_dir =  directory[::2], directory[1::2]
```

[124]:
```
print(len(timestamp), len(value), len(timestamp_dir), len(value_dir))
```

```
1458 1458 1458 1458
```

[125]:
```
dictionary = {'timestamp_dir': [], 'value_dir': [], 'timestamp':[], 'value': []}

for i in range(len(timestamp)):
    dictionary['timestamp_dir'].append(timestamp_dir[i])
    dictionary['value_dir'].append(value_dir[i])
    dictionary['timestamp'].append(timestamp[i])
```

APPENDIX

```
    dictionary['value'].append(value[i])
```

[126]:
```
df= pd.DataFrame.from_dict(dictionary)
```

[127]:
```
import matplotlib.pyplot as plt
import seaborn as sns
import datetime
```

[128]:
```
from datetime import datetime
from datetime import timedelta
import seaborn as sns


def iter_data(df, start, end):
    for index, row in df[start :end].iterrows():

        ts = row['timestamp']
        ts = (ts*1e9).astype('datetime64[ns]')


        plt.figure(figsize=(13,5))
        plt.title(row.value_dir)
        plt.plot(ts, row.value)
        plt.plot(ts, [np.mean(row['value'])] * len(row['value']), label='mean')
        plt.plot(ts, [np.std(row['value'])] * len(row['value']), label='std')
        plt.legend()
        plt.show()



        plt.figure(figsize=(11,5))
        plt.title(row.value_dir)
        plt.scatter(ts, row['value'], marker='o')
        plt.tick_params(axis='x',labelsize=11,rotation=90)
        plt.tight_layout()


        data = pd.DataFrame({'timestamp': ts, 'value': row['value']})
        data.set_index('timestamp', inplace=True)
        return data
```

[129]:
```
def get_corr(data):

    value_temp = np.array(data['value'])
    autocorrelation =[]
```

APPENDIX

```
    for shift in range(1,len(value_temp)):

        correlation = np.corrcoef(value_temp[:-shift], value_temp[shift:])[0, 1]
        autocorrelation.append(correlation)

    time_temp = np.array(data.index)

    plt.figure(figsize=(13,5))
    plt.title('correlation coefficient')
    plt.plot(time_temp[0:-1],autocorrelation)
    plt.show()
```

[130]:
```
def get_std_mean(data):

    time_index = pd.Series(dtype=float, index = data.index)
    time_index_mean = pd.Series(dtype=float, index = data.index)


    for timestamp in time_index.index:
        window= data.loc[:timestamp]
        time_index.at[timestamp]= window.std()

    for timestamp in time_index_mean.index:
        window=data.loc[:timestamp]
        time_index_mean.at[timestamp]= window.mean()


    plt.figure(figsize=(13,5))
    plt.plot(time_index, label='std')
    plt.plot(time_index_mean, label='mean')


    plt.title('Deviation & Mean Trend', fontsize=20)
    plt.legend()
    plt.show()
```

[255]:
```
from datetime import timedelta

def window_func(data):

    window_dt = pd.Timedelta(seconds=10)

    data["timestamp_window"] = data.index  + window_dt

    time_unique_endlist = np.unique(data.timestamp_window)
```

APPENDIX

```
    time_unique_endlist = time_unique_endlist[time_unique_endlist <=␣
↪max(data["timestamp_window"])]


    df_merged = pd.DataFrame()
    frequency = []
    for time_i in time_unique_endlist:
        start_time = time_i - window_dt
        rolling_sample = data[(data.index >= start_time) & (data.index <=␣
↪time_i)]
        freq = rolling_sample['value'].nunique()/10
        frequency.append(freq)
        rolling_sample.insert(0, 'frequency', freq)
        df_merged = df_merged.append(rolling_sample)


    plt.figure(figsize=(13,5))
    plt.bar(df_merged.index, df_merged['frequency'], width=5)
    plt.title('Frequency in 10000 ms window frame')
    plt.xlabel('timestamp')
    plt.ylabel('value frequency(Hz)')
    plt.show()



    plt.figure(figsize=(13,5))
    plt.bar(df_merged.loc[df_merged.frequency > 0.2].index ,df_merged.
↪loc[df_merged.frequency > 0.2].frequency, width=10)
    plt.title('Frequency > 0.2 in window frame')
    plt.xlabel('timestamp')
    plt.ylabel('value frequency(Hz)')
    plt.show()

    return df_merged
```

[ ]:
```
first_data = iter_data(df, 41, 42)
```

[ ]:
```
get_std_mean(first_data)
```

[ ]:
```
get_corr(first_data)
```

[ ]:
```
first_data_freq = window_func(first_data)
```

[ ]:
```
second_data = iter_data(df, 37, 38)
```

[ ]:
```
get_std_mean(second_data)
```

APPENDIX

[ ]:
```
second_data_freq=window_func(second_data)
```

[ ]:
```
thirth_data = iter_data(df, 45, 46)
```

[ ]:
```
get_std_mean(thirth_data)
```

[ ]:
```
thirth_data_freq= window_func(thirth_data)
```

[ ]:
```
fourth_data = iter_data(df, 89, 90)
```

[ ]:
```
get_std_mean(fourth_data)
```

[ ]:
```
fourth_data_freq =window_func(fourth_data)
```

[ ]:
```
first_data_freq['frequency'].plot(color='orange', figsize=(10, 5))
second_data_freq['frequency'].plot(figsize=(10, 5), figsize=(10, 5))
thirth_data_freq['frequency'].plot(color='black', figsize=(10, 5))
fourth_data_freq['frequency'].plot(color='red', figsize=(10, 5))
```