Freie und Hansestadt Hamburg
Landesbetrieb Straßen, Brücken und Gewässer

DaQuAI – Anomaly Detection

@amartya-saikia

1. Introduction

As urban areas around the world strive to promote sustainable transportation modes, the monitoring and management of bicycle traffic have become increasingly vital. The Free and Hanseatic City of Hamburg, known for its commitment to eco-friendly mobility solutions, has deployed a network of sensors to collect comprehensive data on bicycle counts. This study presents a novel approach to enhance the quality of this data by detecting anomalies and uncovering underlying patterns utilizing machine learning techniques.
The research leverages a rich dataset comprising time-series information from various sensors strategically placed across Hamburg's bicycle infrastructure. To effectively identify anomalous data points, we employ state-of-the-art deep learning models.
By harnessing the power of machine learning, our methodology captures intricate temporal and spatial dependencies within the sensor data, enabling the detection of subtle deviations from expected patterns.
This study contributes to the field of urban mobility by addressing the following key objectives:

Anomaly Detection: We develop a robust anomaly detection system capable of identifying irregularities in bicycle count data, which may result from events such as accidents, road closures, or unusual weather conditions.

Pattern Discovery: Through deep learning-based analysis, we unveil hidden patterns within the sensor data, shedding light on the factors influencing bicycle traffic in Hamburg. These insights can inform urban planning and traffic management strategies.

Data Quality Enhancement: By filtering out anomalous data, we improve the overall quality and reliability of bicycle count statistics. This is vital for accurate decision-making and resource allocation in the city's transportation planning.

The results of our study demonstrate the effectiveness of machine learning techniques in uncovering anomalous data patterns in bicycle count sensor data. This research not only contributes to the advancement of anomaly detection methods but also supports Hamburg's commitment to sustainable urban mobility. The findings have the potential to enhance the city's transportation planning and infrastructure management, ultimately promoting eco-friendly modes of transportation and improving the quality of life for its residents.

2. Description

Time series sensor data, generated from a wide array of applications such as industrial processes, financial markets, environmental monitoring, and healthcare, often contain valuable insights buried within complex temporal patterns. However, they are also susceptible to anomalies or deviations from these patterns that may signify critical events, errors, or opportunities. This abstract explores the fundamental principles of anomaly detection techniques applied to time series sensor data.

Anomaly detection in time series data involves the identification of data points or subsequences that significantly differ from the expected or normal behavior. This process is critical for various real-world applications, including fault detection, fraud prevention, and predictive maintenance. Understanding how these techniques work is essential for unlocking the full potential of sensor data.

This abstract delves into the key aspects of anomaly detection in time series sensor data:

Data Preprocessing: The journey begins with data preprocessing, which includes tasks such as cleaning, imputation, and feature engineering. Transforming raw sensor data into a suitable format is crucial for effective anomaly detection.

Feature Extraction: Extracting informative features from time series data is a pivotal step. Techniques like Fourier analysis, wavelet transforms, and statistical measures help capture relevant characteristics for anomaly detection.

Model Selection: Anomaly detection methods can be categorized into statistical, machine learning, and deep learning-based approaches. Selecting an appropriate model depends on the data characteristics and the nature of anomalies being targeted.

Training and Evaluation: Supervised, unsupervised, and semi-supervised techniques are employed for model training. Evaluation metrics like precision, recall, F1-score, and ROC curves are used to assess a model's performance.

Thresholding and Alerting: Anomalies are detected by comparing model predictions to predefined thresholds or confidence intervals. False positives and false negatives must be balanced to ensure a robust alerting system.

Dynamic Adaptation: In many applications, the underlying data distribution may change over time. Adaptive anomaly detection techniques continuously update models to stay attuned to evolving patterns.

Interpretation and Action: Once anomalies are identified, their interpretation is essential. Domain knowledge plays a crucial role in deciding whether detected anomalies require action or further investigation.

Understanding how anomaly detection techniques work on time series sensor data is vital for leveraging these methods in practical scenarios. By effectively identifying anomalies, organizations can mitigate risks, enhance operational efficiency, and harness the hidden insights within their temporal data. This abstract serves as a foundation for exploring the intricacies and applications of anomaly detection in the realm of time series sensor data analysis.

3. Literature

Efficiently detecting anomalies in time series sensor data is pivotal for various applications, including urban traffic management. In this study, we explore the effectiveness of four distinct anomaly detection techniques - K-Means clustering, DBSCAN (Density-Based Spatial Clustering of Applications with Noise), Facebook's Prophet model, and statistical modeling - in the context of hourly vehicle count data collected from sensor networks.

+ **K-Means** clustering is employed to segment the time series data into clusters based on the similarity of vehicle count patterns. Anomalies are detected by identifying data points that deviate significantly from their cluster centroids. This approach captures anomalies that exhibit unique temporal characteristics.

+ **DBSCAN**, a density-based clustering method, offers a different perspective on anomaly detection. It groups together data points that are densely connected, isolating outliers as anomalies. This technique is particularly effective in detecting isolated incidents or sudden changes in vehicle counts.

+ **Facebook's Prophet** model introduces a powerful tool for time series forecasting. In our study, we adapt the Prophet to predict the expected vehicle counts. Anomalies are identified by comparing the actual counts to the predicted values, enabling the detection of unusual fluctuations or trends.

+ **Statistical modeling** leverages traditional statistical methods, such as ARIMA or Exponential Smoothing, to model and forecast time series data. Anomalies are detected by analyzing the residuals, highlighting deviations from the expected statistical patterns.

Through comprehensive experimentation and evaluation, this study aims to provide insights into the strengths and limitations of each anomaly detection technique. Furthermore, it explores the possibility of combining these methods to create an ensemble approach, harnessing the complementary aspects of each technique for improved accuracy and robustness.

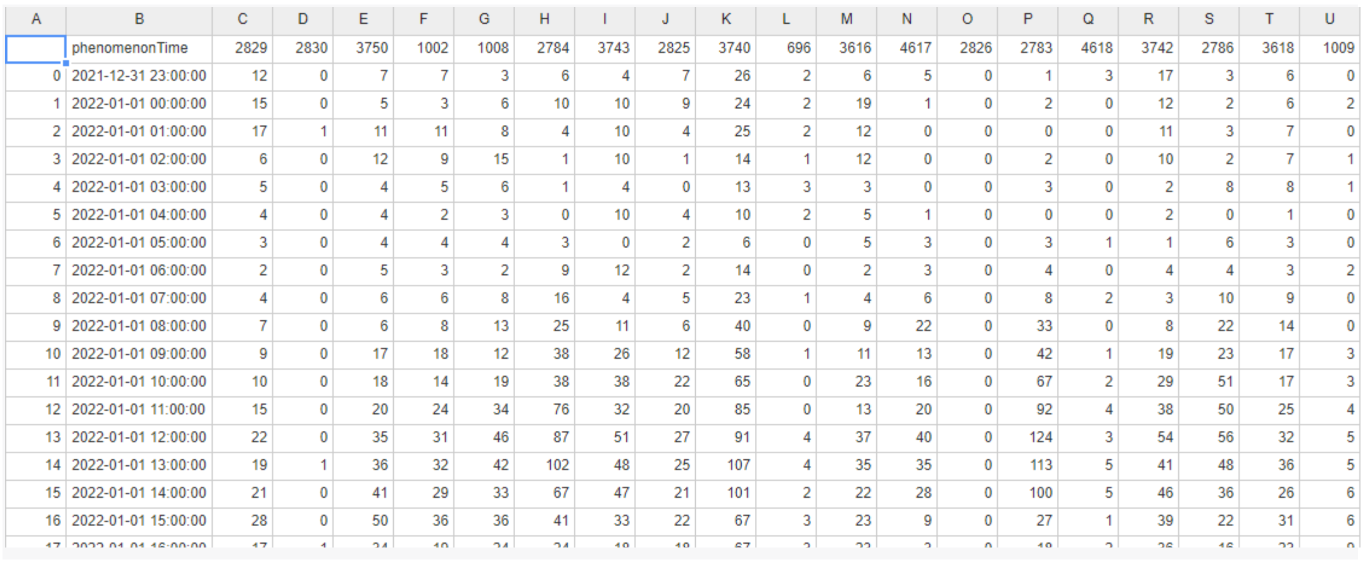
By applying these diverse methods to hourly vehicle count data, this research contributes to the development of effective anomaly detection strategies for urban traffic management. The findings have the potential to enhance the resilience and efficiency of traffic control systems, improving safety and mobility in urban environments.

4. Technical Details:

This research project presents a comprehensive workflow for the acquisition, processing, analysis, and visualization of real-time sensor data sourced from the SensorThings API. The primary objective is to detect anomalies within this data, utilizing machine learning models while considering external factors such as public holidays, school holidays, and weather conditions. The study showcases a holistic data science approach to harness the full potential of sensor data for informed decision-making.
The initial phase of the project involves the retrieval of real-time raw sensor data from the SensorThings API. This API serves as a gateway to various sensor networks, offering a standardized interface for data access. The acquired data is subsequently stored efficiently in a MongoDB database, ensuring scalability and flexibility for handling large datasets.
Python, a versatile programming language for data science and machine learning, is employed for the subsequent data processing steps. The project encompasses essential data science practices, including data cleaning, feature engineering, and exploratory data analysis. This phase aims to prepare the data for machine learning modeling effectively.
Machine learning models are at the core of anomaly detection in this study. Various algorithms are explored, with a focus on identifying the two best-performing models. These models are trained on historical sensor data to recognize anomalies in real-time observations accurately.
To enrich the anomaly detection process, external factors are integrated into the analysis. Public holidays, school holidays, and weather conditions are considered as potential influencers on sensor data patterns. This integration enhances the models' capacity to distinguish genuine anomalies from contextually appropriate variations.
In the final phase of the project, the detected anomalies from the two best-performing models are visualized in a user-friendly format. The visualization includes temporal trends and anomalies, providing stakeholders with actionable insights into the sensor data. This integration of external factors into the visualization allows for a more holistic understanding of the anomalies' context and potential implications.
The presented workflow serves as a blueprint for harnessing real-time sensor data for anomaly detection while considering relevant external factors. The study's findings have practical implications in various domains, including environmental monitoring, infrastructure management, and urban planning,

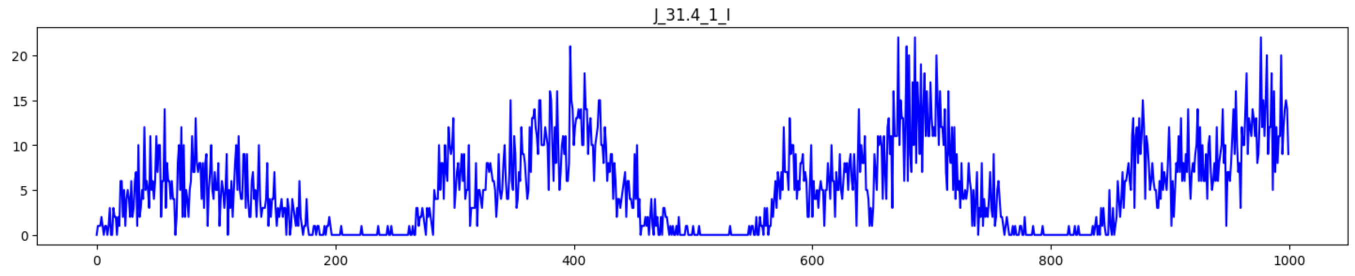
where the timely identification of anomalies can lead to more efficient resource allocation and decision-making.

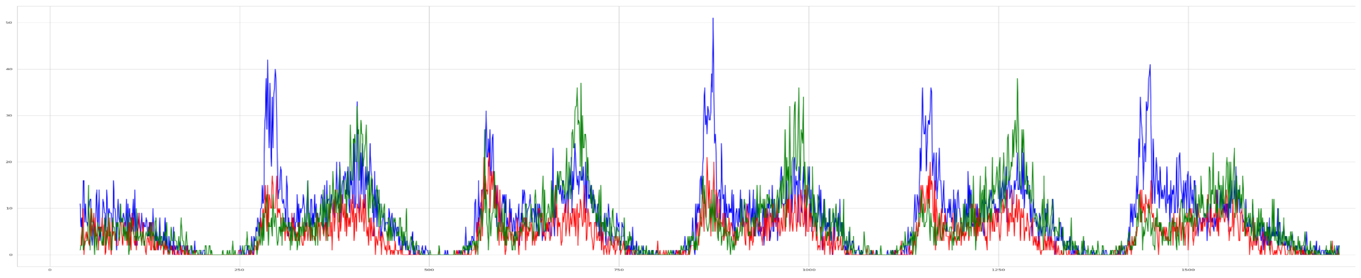
Data: This research delves into the analysis of real-time sensor data, encompassing both univariate and multivariate time series datasets. The data comprises hourly timestamped records, with each row representing a discrete time interval and columns capturing bicycle counts at distinct sensor locations. This abstract outlines the significance and potential applications of such comprehensive sensor data.
Univariate time series data, focusing on the bicycle count at individual sensor locations, offers insights into localized trends and patterns. Analyzing these univariate time series helps identify specific hotspots of bicycle activity, track temporal variations, and comprehend the influence of external factors on each location's traffic.
On the other hand, multivariate time series data takes a holistic approach by considering the interdependencies between sensor locations. This allows for the exploration of broader trends, such as network-wide bicycle flow dynamics, spatial correlations, and congestion patterns. Multivariate analysis unveils intricate relationships among sensor locations, offering a more comprehensive view of urban bicycle traffic.
The hourly time stamped records enable the detection of temporal patterns, seasonal fluctuations, and anomalies within the data. By applying advanced time series analysis techniques, such as seasonality decomposition, autoregressive modeling, and spectral analysis, this research aims to uncover hidden insights that can inform urban planning, transportation management, and infrastructure optimization.
Furthermore, the ability to work with real-time data provides decision-makers with timely information critical for effective resource allocation and response to changing conditions. The research also explores the potential for predictive modeling, allowing for the anticipation of future bicycle traffic based on historical data and external factors, ultimately contributing to enhanced traffic management strategies.



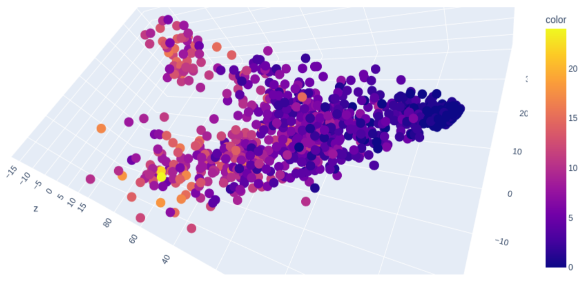
A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
	phenomenonTime	2829	2830	3750	1002	1008	2784	3743	2825	3740	696	3616	4617	2826	2783	4618	3742	2786	3618	1009
0	2021-12-31 23:00:00	12	0	7	7	3	6	4	7	26	2	6	5	0	1	3	17	3	6	0
1	2022-01-01 00:00:00	15	0	5	3	6	10	10	9	24	2	19	1	0	2	0	12	2	6	2
2	2022-01-01 01:00:00	17	1	11	11	8	4	10	4	25	2	12	0	0	0	0	11	3	7	0
3	2022-01-01 02:00:00	6	0	12	9	15	1	10	1	14	1	12	0	0	2	0	10	2	7	1
4	2022-01-01 03:00:00	5	0	4	5	6	1	4	0	13	3	3	0	0	3	0	2	8	8	1
5	2022-01-01 04:00:00	4	0	4	2	3	0	10	4	10	2	5	1	0	0	0	2	0	1	0
6	2022-01-01 05:00:00	3	0	4	4	4	3	0	2	6	0	5	3	0	3	1	1	6	3	0
7	2022-01-01 06:00:00	2	0	5	3	2	9	12	2	14	0	2	3	0	4	0	4	4	3	2
8	2022-01-01 07:00:00	4	0	6	6	8	16	4	5	23	1	4	6	0	8	2	3	10	9	0
9	2022-01-01 08:00:00	7	0	6	8	13	25	11	6	40	0	9	22	0	33	0	8	22	14	0
10	2022-01-01 09:00:00	9	0	17	18	12	38	26	12	58	1	11	13	0	42	1	19	23	17	3
11	2022-01-01 10:00:00	10	0	18	14	19	38	38	22	65	0	23	16	0	67	2	29	51	17	3
12	2022-01-01 11:00:00	15	0	20	24	34	76	32	20	85	0	13	20	0	92	4	38	50	25	4
13	2022-01-01 12:00:00	22	0	35	31	46	87	51	27	91	4	37	40	0	124	3	54	56	32	5
14	2022-01-01 13:00:00	19	1	36	32	42	102	48	25	107	4	35	35	0	113	5	41	48	36	5
15	2022-01-01 14:00:00	21	0	41	29	33	67	47	21	101	2	22	28	0	100	5	46	36	26	6
16	2022-01-01 15:00:00	28	0	50	36	36	41	33	22	67	3	23	9	0	27	1	39	22	31	6
17	2022-01-01 16:00:00	17	1	24	19	24	24	18	18	67	3	22	3	0	18	2	26	16	22	0

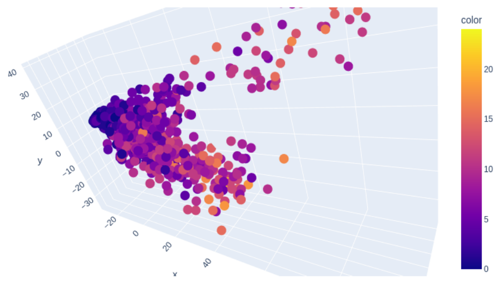
The rows have hourly time stamp information and the columns have sensor data for bicycle count per sensor id. A quick visualization of univariate and multivariate data respectively with Python:





PCA: Principal Component Analysis (PCA), a powerful dimensionality reduction technique, is applied to time series sensor data in this study. The research investigates the utility of PCA in distilling meaningful information from high-dimensional time series datasets. By capturing the most significant variations in the data, PCA simplifies analysis, enhances visualization, and offers insights into the underlying structures and patterns. This abstract highlights the potential of PCA as a valuable tool for feature extraction and data compression in time series sensor data analysis, fostering more efficient data-driven decision-making across diverse applications. Here are PCA plots of the data:





Machine Learning Models:

Machine learning models play a pivotal role in anomaly detection for time series sensor data due to their ability to provide accurate, efficient, and scalable solutions for identifying irregular patterns and deviations. Here are key reasons highlighting their importance in this context:

+ **Complex Pattern Recognition**: Time series sensor data often exhibit intricate temporal dependencies and patterns that may be challenging to detect with traditional statistical methods. Machine learning models, such as recurrent neural networks (RNNs) and convolutional neural networks (CNNs), excel at capturing these complex relationships, enabling the detection of subtle anomalies that might otherwise go unnoticed.

+ **Adaptability**: Machine learning models can adapt to changing data distributions and evolving anomalies. As time series data can exhibit seasonality, trends, or shifts over time, machine learning models can continuously update their understanding of "normal" behavior, making them well-suited for dynamic environments.

+ **Multivariate Analysis**: Many time series sensor datasets involve multiple variables or sensors that interact with each other. Machine learning models can perform multivariate analysis, considering these interdependencies to detect anomalies that may manifest as unusual combinations of variables or correlated deviations across sensors.

+ **Scalability**: With the increasing volume and velocity of sensor data, scalability is crucial. Machine learning models can be parallelized and optimized for large-scale data processing, allowing for real-time or near-real-time anomaly detection in high-throughput environments.

+ **Feature Extraction**: Machine learning models can automatically extract relevant features from time series data, reducing the need for manual feature engineering. This capability is particularly valuable when dealing with high-dimensional sensor data.

+ **Classification and Prediction**: Machine learning models can go beyond anomaly detection by classifying anomalies into different categories or predicting when and where anomalies are likely to occur. This information can be instrumental in proactive maintenance and resource allocation.

+ **False Positive Reduction**: Machine learning models can be fine-tuned to minimize false positives, ensuring that detected anomalies are more likely to be genuinely unusual events rather than noise or minor fluctuations.

+ **Interpretability**: Some machine learning models offer interpretability features, allowing stakeholders to understand why a particular data point is flagged as an anomaly. This aids in root cause analysis and decision-making.

+ **Continuous Monitoring**: Machine learning models enable continuous, automated monitoring of time series data. This is invaluable for applications like industrial process control, where prompt anomaly detection can prevent equipment failures and production disruptions.

+ **Enhanced Decision Support**: By providing actionable insights based on anomaly detection, machine learning models empower organizations to make data-driven decisions, optimize operations, and allocate resources more efficiently, ultimately saving time and resources.

In conclusion, machine learning models are indispensable tools for anomaly detection in time series sensor data. They offer a potent combination of pattern recognition, adaptability, scalability, and automation, making them essential for extracting valuable insights, improving operational efficiency, and enhancing decision-making across a wide range of industries and applications.

1. kMeans

This study explores the application of K-Means clustering as an innovative approach to detect anomalies within time series sensor data. K-Means, a widely used unsupervised machine learning algorithm, is adapted to this context to identify unusual patterns in temporal sensor data. The following abstract provides a high-level overview of this methodology and includes pseudocode to illustrate the key steps involved in using K-Means for anomaly detection in time series sensor data.

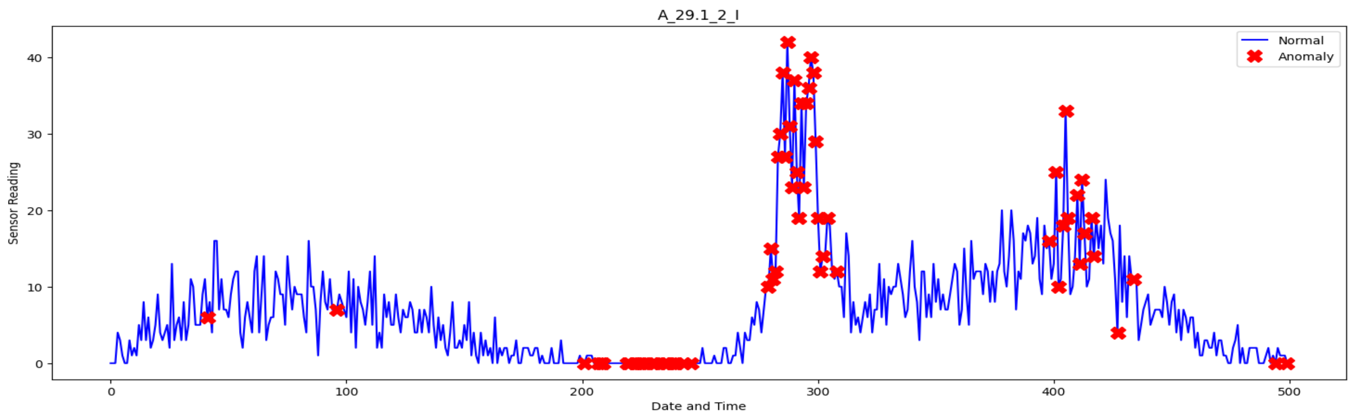


Fig: Anomaly detection with kMeans

```
# Import required libraries
import numpy as np
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score

# Load the time series sensor data
sensor_data = load_sensor_data()

# Specify the number of clusters (K) for K-Means
num_clusters = determine_optimal_k(sensor_data)

# Fit K-Means model to the sensor data
kmeans = KMeans(n_clusters=num_clusters, random_state=42)
kmeans.fit(sensor_data)

# Predict cluster labels for each data point
cluster_labels = kmeans.predict(sensor_data)

# Calculate the silhouette score to evaluate clustering quality
silhouette_avg = silhouette_score(sensor_data, cluster_labels)

# Identify anomalies based on silhouette score
anomaly_threshold = determine_anomaly_threshold(silhouette_avg)

# Detect anomalies
anomalies = []
for i, silhouette_score in enumerate(silhouette_avg):
    if silhouette_score < anomaly_threshold:
        anomalies.append(i)

# Visualize anomalies in the sensor data
visualize_anomalies(sensor_data, anomalies)

# Output detected anomalies
print("Detected Anomalies: ", anomalies)
```

In this research, we leverage the K-Means clustering algorithm to uncover anomalies in time series sensor data. K-Means' adaptability to clustering temporal patterns enables it to identify deviations from expected behavior effectively. Through the use of pseudocode, we illustrate the core steps involved in this anomaly detection approach, including determining the optimal number of clusters, fitting the K-Means model, and calculating silhouette scores to detect anomalies. This methodology promises to enhance anomaly detection capabilities in various applications, including industrial process monitoring, environmental sensing, and urban infrastructure management.

2. DBSCAN

This research investigates the application of DBSCAN (Density-Based Spatial Clustering of Applications with Noise) as a robust method for detecting anomalies in time series sensor data. DBSCAN, a density-based clustering algorithm, offers a novel approach to identifying unusual patterns and deviations within temporal sensor data. The abstract below outlines the significance of using DBSCAN for anomaly detection in time series data and includes pseudocode to illustrate the core steps of the methodology.

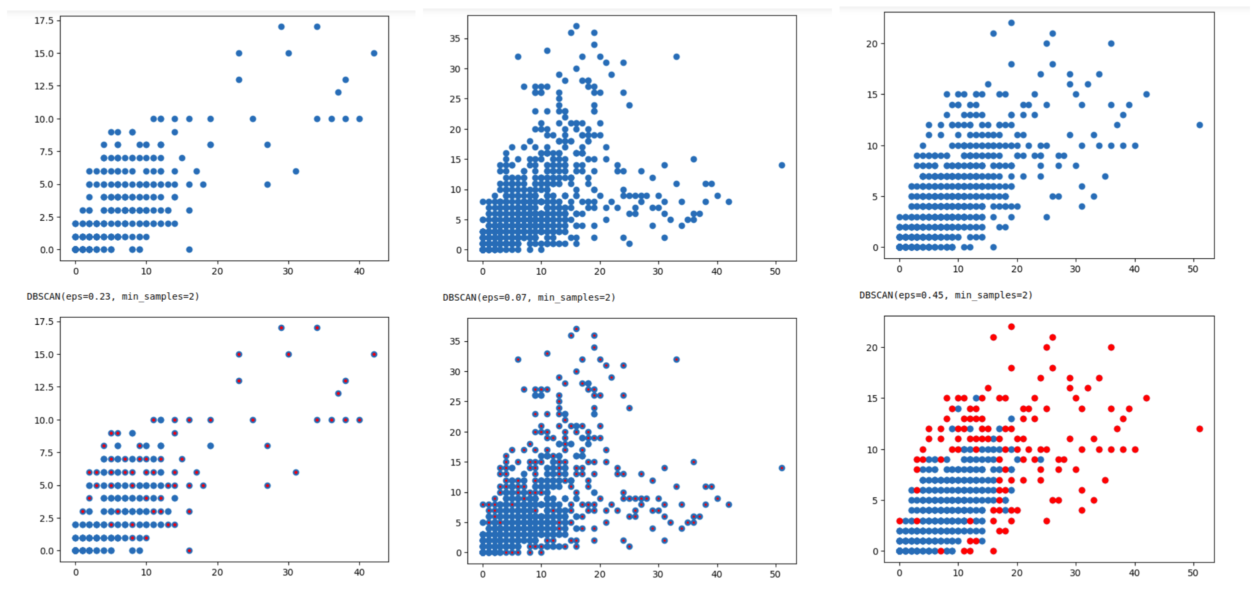


Fig: Anomaly detection with DBSCAN with ε & min_sample pair

```
# Import required libraries
from sklearn.cluster import DBSCAN
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import silhouette_score

# Load the time series sensor data
sensor_data = load_sensor_data()

# Standardize the data to ensure consistent scales
scaler = StandardScaler()
sensor_data_std = scaler.fit_transform(sensor_data)

# Determine optimal parameters for DBSCAN
eps, min_samples = determine_dbscan_parameters(sensor_data_std)

# Create a DBSCAN model with determined parameters
dbscan_model = DBSCAN(eps=eps, min_samples=min_samples)

# Fit DBSCAN model to the standardized data
clusters = dbscan_model.fit_predict(sensor_data_std)

# Calculate the silhouette score to evaluate clustering quality
silhouette_avg = silhouette_score(sensor_data_std, clusters)

# Determine anomalies based on silhouette score
anomaly_threshold = determine_anomaly_threshold(silhouette_avg)
```

```
# Detect anomalies
anomalies = []
for i, silhouette_score in enumerate(silhouette_avg):
    if silhouette_score < anomaly_threshold:
        anomalies.append(i)

# Visualize anomalies in the sensor data
visualize_anomalies(sensor_data, anomalies)

# Output detected anomalies
print("Detected Anomalies: ", anomalies)
```

In this study, we explore the utility of DBSCAN as an innovative technique for detecting anomalies in time series sensor data. DBSCAN's ability to uncover spatially dense regions in high-dimensional data makes it a promising choice for identifying anomalies within temporal patterns. The provided pseudocode outlines the essential steps of this methodology, from data standardization to determining optimal DBSCAN parameters, clustering, and ultimately detecting anomalies based on silhouette scores.

3. Facebook's Prophet

This research explores the application of Facebook's Prophet model as a powerful tool for detecting anomalies in time series sensor data. Prophet, designed for forecasting with time series data, offers a robust framework for capturing patterns, seasonality, and trends. The abstract below highlights the significance of employing Prophet for anomaly detection in time series data and includes pseudocode to illustrate the essential steps of this methodology.

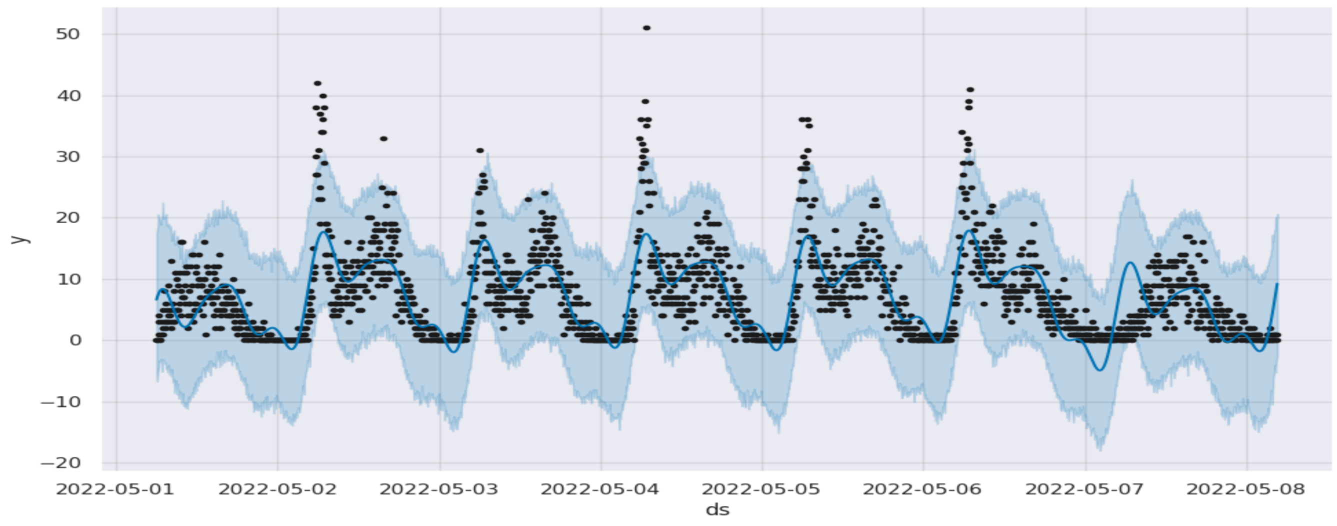


Fig: Anomaly detection with fb prophet

```
# Import required libraries
from fbprophet import Prophet
import pandas as pd

# Load the time series sensor data
sensor_data = load_sensor_data()

# Prepare the data in a Prophet-compatible format
data = pd.DataFrame({'ds': sensor_data.index, 'y': sensor_data['count']})

# Initialize and fit the Prophet model
model = Prophet()
model.fit(data)

# Make predictions on future time intervals
future = model.make_future_dataframe(periods=24, freq='H')  # Extend for 24 hours (adjust as needed)
forecast = model.predict(future)

# Calculate residuals (differences between actual and predicted values)
residuals = sensor_data['count'] - forecast['yhat']

# Determine anomalies based on residuals
anomaly_threshold = determine_anomaly_threshold(residuals)

# Detect anomalies
anomalies = []
for i, residual in enumerate(residuals):
    if abs(residual) > anomaly_threshold:
        anomalies.append(sensor_data.index[i])

# Visualize anomalies in the sensor data
visualize_anomalies(sensor_data, anomalies)

# Output detected anomalies
print("Detected Anomalies: ", anomalies)
```

In this study, we investigate the utility of Facebook's Prophet model for anomaly detection within time series sensor data. Prophet's capabilities in capturing time-dependent patterns and seasonality make it an invaluable resource for identifying unusual deviations within temporal sensor data. The provided pseudocode outlines the core steps of this methodology, including data preparation, model initialization, prediction, and the subsequent detection of anomalies based on residuals.

4. Statistical Modeling

This research presents a custom statistical model designed for anomaly detection in time series sensor data with a unique approach. The approach involves clustering the time series data based on multiple factors, including seasons (Spring, Summer, Autumn, Winter), days of the week (Monday to Sunday), and time zones (t1 to t6). Each cluster is represented by variables in the format of "time_daysofw_seasonsNUM," where NUM corresponds to the month of the season. The report details the steps to calculate mean and standard deviation for each cluster and then establish upper and lower thresholds for anomaly detection using confidence intervals.

Data Clustering:

Seasonal Clustering:
The first step in the process involves clustering the time series data according to seasons. Seasons are categorized as follows:

Spring: March, April, May
Summer: June, July, August
Autumn: September, October, November
Winter: December, January, February

For each season, the data is further divided into months (e.g., Spring1, Spring2, Spring3 for March, April, and May of the Spring season, respectively).

Day of the Week Clustering:
Next, the data is clustered based on days of the week, with each day representing a cluster:

Monday
Tuesday
Wednesday
Thursday
Friday
Saturday
Sunday

Time Zone Clustering:
Within each day, the data is divided into six time zones (t1 to t6), corresponding to different hours of the day:

t1: 6 a.m. to 9 a.m.
t2: 10 a.m. to 12 p.m.
t3: 1 p.m. to 3 p.m.
t4: 4 p.m. to 6 p.m.
t5: 7 p.m. to 10 p.m.
t6: 11 p.m. to 5 a.m.

Variable Naming Convention:
To represent each cluster, variables are created in the format of "time_daysofw_seasonsNUM," where:

'**time**' represents one of the six time zones (t1 to t6).
'**daysofw**' represents the day of the week.
'**seasons**' represents the season (Spring, Summer, Autumn, or Winter).
'**NUM**' represents the month within the season.

For example, "t1_mon_spring3" represents the cluster of data points for every Monday morning from 6 a.m. to 9 a.m. in May (Spring3).

Statistical Measures:

Mean (μ) and Standard Deviation (σ)

For each cluster, the mean (μ) and standard deviation (σ) of the time series data are calculated. These measures provide insights into the central tendency and the spread of data within each cluster.

Threshold Calculation:

To identify anomalies, upper threshold (UT) and lower threshold (LT) values are computed using confidence intervals. The threshold calculation is as follows:

Upper Threshold (**UT**) = μ + 0.2 * σ
Lower Threshold (**LT**) = μ - 0.2 * σ

These thresholds define the bounds for identifying anomalous data points.

Creating the Anomaly Detection Dataset:

A dataset is generated for each cluster, containing the following columns:

min: The minimum value of the data within the cluster.
max: The maximum value of the data within the cluster.
upper threshold: The calculated upper threshold value.
mean: The mean value calculated for the cluster.
lower threshold: The calculated lower threshold value.

We calculated the bounds information for hourly data points from each sensor in the form of `timeperiod_dayofweek_seasonNUM["sensorID"]`.

By clustering the data based on seasons, days of the week, and time zones, and by calculating mean and standard deviation measures, we establish upper and lower thresholds for anomaly detection. The results are stored in a CSV file named '**bound_name_finale.csv**'.

The mean, upper threshold and lower threshold are plotted in green with the top point being the upper threshold, the lower green point being the lower threshold and the mid green point being the mean.

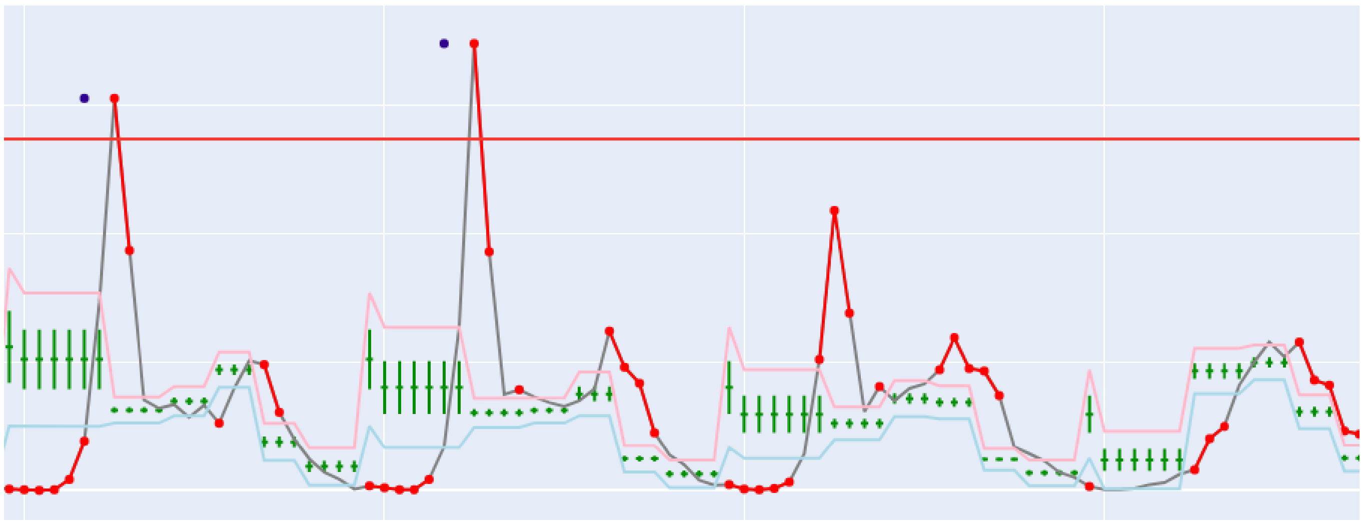


Fig: Anomaly detection with statistical model

We calculated the bounds information for hourly data points from each sensor in the form of `timeperiod_dayofweek_seasonNUM["sensorID"]. This work is done in "init_bounds.ipynb".`

In the '**fetch.ipynb**' notebook, a data processing pipeline was implemented to compile essential information for each 'Zählfelde' (counting field) and associate it with a unique 'Zählfelde ID.' This process involved integrating anomaly detection data from '**anomalies.csv**,' which was generated using Facebook's Prophet model in the '**anom_prophet.ipynb**' notebook.

The final dataset includes crucial attributes such as 'Zählfelde Name,' 'Zählfelde ID,' 'Anomaly,' 'Anomaly Value,' and bounds data retrieved from 'bounds_name_finale.csv.' Each 'Zählfelde' data was fetched from 'MQ' (measurement quantity), e.g., MQ27.1, MQ27.2, directories which spanned the years 2022 and 2023.

The resulting dataset was saved in each respective 'MQ' data folder, with each file named as 'filename+_ab.csv.'

In the realm of anomaly detection for time series sensor data, this research conducted a comprehensive evaluation of various models. Among the array of models assessed, Facebook Prophet and a custom statistical model emerged as top performers. This abstract highlights the exceptional performance of these models in detecting anomalies within time series sensor data, signifying their efficacy in providing accurate and reliable results. Such findings hold significant implications for enhancing anomaly detection capabilities across diverse applications, offering valuable insights into the critical selection of models for data-driven decision-making.

Code: The code was written in Python and submitted as Jupyter Notebooks with inline comments explaining the logic of the code step by step. There is a readme.txt in each code repository to explain the order of notebooks.

The flow of work is best represented as: init_bounds.ipynb (we create bounds_name_finale.csv) ⟶ anom_prophet.ipynb (we create anomalies.csv) ⟶ fetch.ipynb (we create 'filename_ab.csv') ⟶ anom_graphfinale.ipynb (we create 'filename_year_zählfelde_final.html').

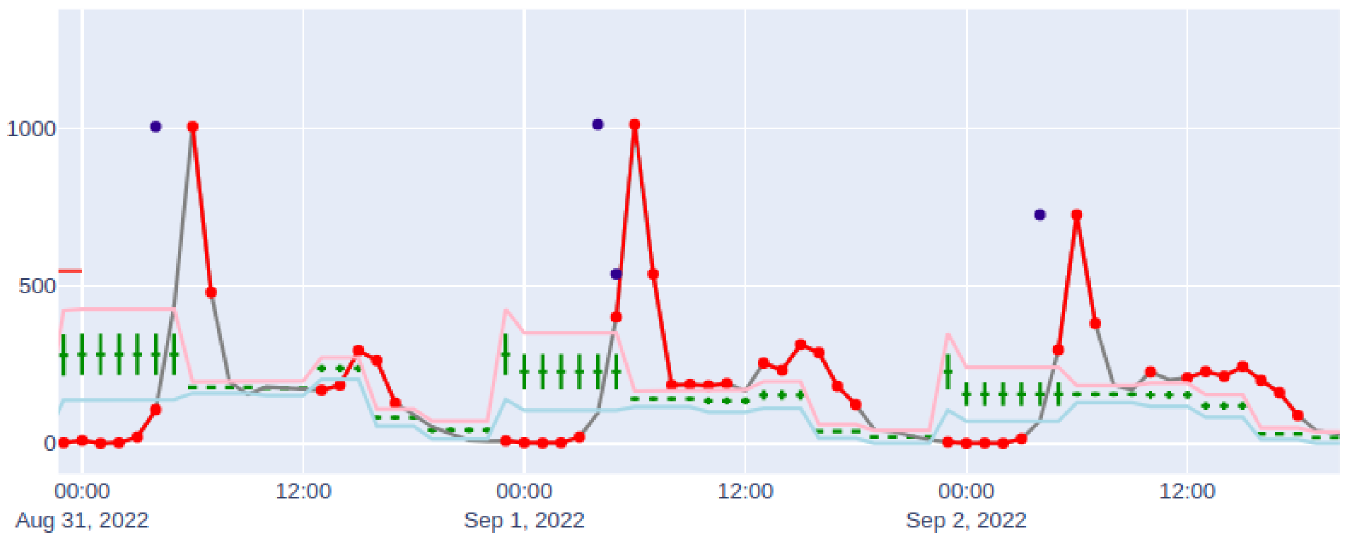
Data Source:

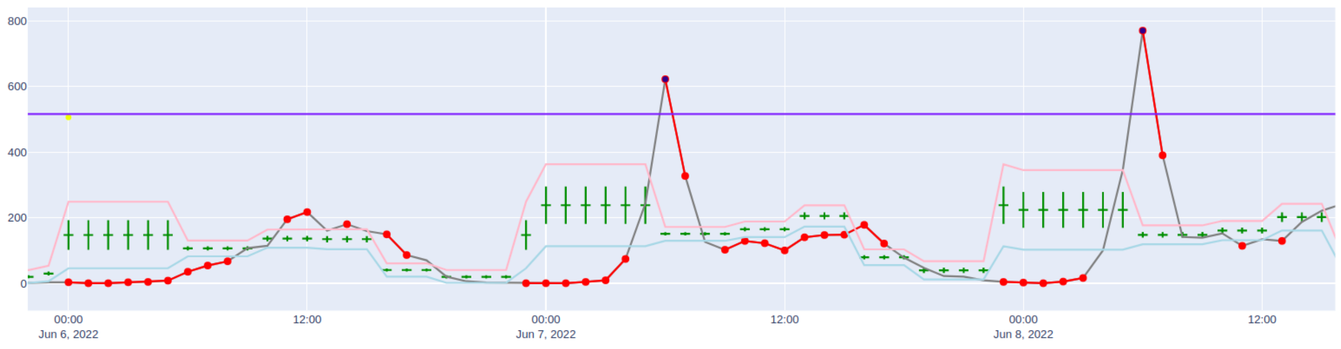
- raw1hr.csv : It is the raw csv file downloaded from mongodb collection which holds hourly sensor data for different sensors with their ID. It is required for processing in init_bounds.ipynb.
- daquai_re/mq_x/filename.xlsx : It is required in fetch.ipynb to create 'filename_ab.csv' and 'filename_year_zählfelde_final.html'.

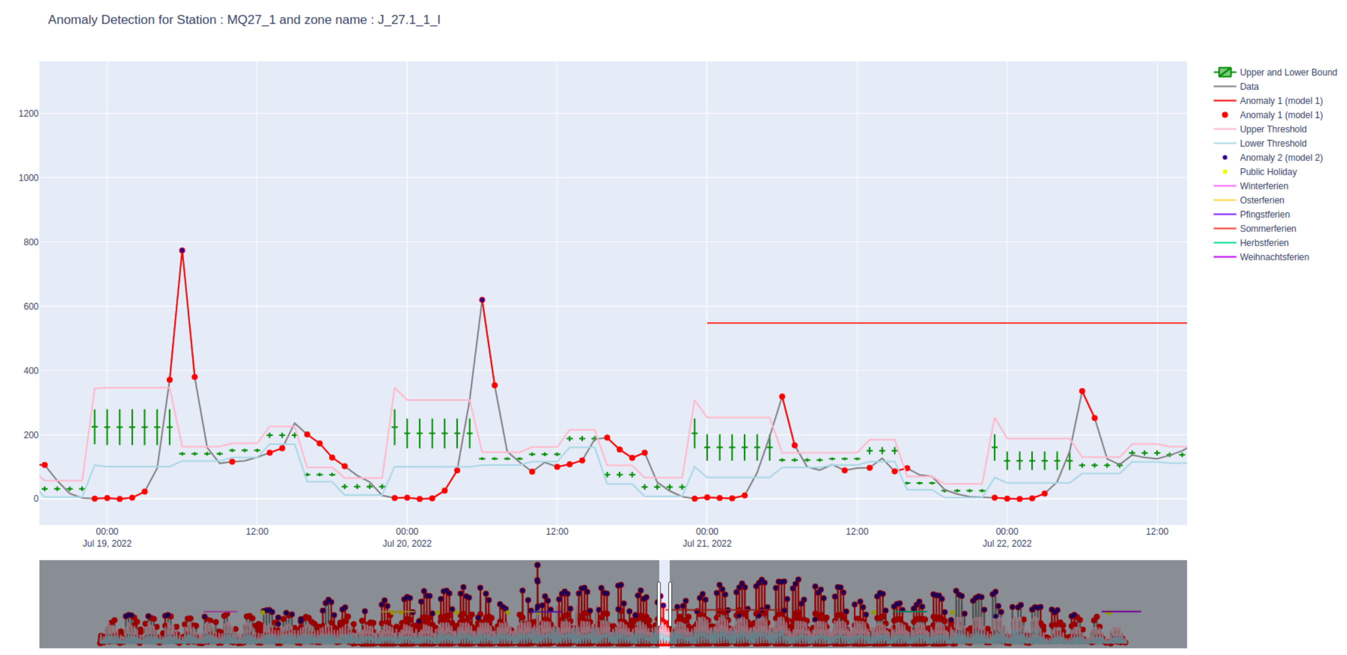
Visualization

This study presents a dynamic visualization approach for showcasing the final results of anomaly detection in time series sensor data, incorporating anomalies detected by both a custom statistical model and Facebook Prophet. Leveraging Plotly Graph Objects, an interactive and visually engaging platform, the research integrates additional contextual information such as school holiday data and public holiday data, enriching the visualization.

The interactive Plotly plot offers an insightful display of the sensor data, with distinct features to enhance comprehension:







+ **Upper and Lower Bounds**: The upper and lower bounds of the data, representing expected variation, are depicted in green, providing a reference for normal data patterns.

+ **Thresholds**: Upper thresholds are highlighted in pink, while lower thresholds are denoted in light blue, offering clear demarcations for the expected data range.
Thresholds are dynamic and are calculated on the absolute difference of upper and lower bounds.

+ **Anomalies from Model 1**: Anomalies detected by the custom statistical model are prominently marked as red data points, serving as visual cues for potential irregularities.

+ **Anomalies from Facebook Prophet**: Anomalies detected by the Facebook Prophet model are distinguished by navy blue data points, providing a contrasting representation of detected anomalies.

+ **Holiday Context**: Overlaying school holiday and public holiday data adds a contextual layer to the visualization, allowing viewers to observe potential correlations between holidays and anomalous data points.

This dynamic and informative visualization not only offers a comprehensive overview of anomaly detection results but also facilitates a deeper understanding of the data's temporal and contextual intricacies. It empowers stakeholders to make data-driven decisions by providing a visually intuitive platform for exploring patterns, anomalies, and their relationships with external factors.

Folder Structure:

There are 2 directories named ‘daquai_final’ and ‘daquai_res’ where the code and results are stored respectively. The ‘daquai_final’ directory structure looks like:

/daquai_final/

- anomalies_model2/ (experimental anomaly plots)
- experiments/ (experiments with data for anomaly detection)
- fig_rawsensordata / (data plots of raw sensor data)
- anom_graphfinale.ipynb
- anom_prophet.ipynb
- anomalies.csv (created data : anomalies from prophet)
- mergedata.csv (data: input to anom_prophet.ipynb)
- bounds_name_finale.csv (created data : bounds from init_bounds.ipynb)
- fetch.ipynb
- for_res.zip (raw data source)
- init_bounds.ipynb
- MQ27_1_FORECAST_2022_J_27.1_1_I_final.html (example visualization)
- public_holiday_data.csv (data: public holiday data (external))
- raw1hr.csv (data: hourly sensor data)
- readme.txt
- school_holiday_data.csv (data: school holiday data (external))

The second directory ‘daquai_res’ holds results of work done until this point in the form of “filename_ab.csv” for each Zählfelde in the MQ stations. The raw data of MQ stations are in ’for_res.zip’ in the ‘daquai_final’ directory. The ‘daquai_res’ directory structure looks like:

/daquai_res/

- MQ_X/
  - filename_year_ab.csv (final data sheet for anomalies and bounds (_ab))
  - filename_year_zählfelde_final.html (final visual plot for MQ_X)
- MQ_Y/
  - filename_year_ab.csv (final data sheet for anomalies and bounds (_ab))
  - filename_year_zählfelde_final.html (final visual plot for MQ_X)
- MQ_Z/
  - filename_year_ab.csv (final data sheet for anomalies and bounds (_ab))
  - filename_year_zählfelde_final.html (final visual plot for MQ_X)

CODE SAMPLES:

anom_graphfinale.ipynb:

```
#filter data as per target date range
if year_now == 2023:
    date_ti = datetime.strptime('2023-01-01 00:00:00', date_format)
    date_to = datetime.strptime('2023-03-31 00:00:00', date_format)
else:
    date_ti = datetime.strptime('2022-01-01 00:00:00', date_format)
    date_to = datetime.strptime('2022-12-31 00:00:00', date_format)



# process and filter public holiday data

df_xo = pd.DataFrame(columns=['date', 'p_name'])


for fo in range(len(df_p["date"])):
    date_t = df_p.iloc[fo]["date"]
    date_t = datetime.strptime(date_t, date_format2)
    d_name = df_p.iloc[fo]["name"]
    new_dat = {'date':date_t, 'p_name':d_name}
    df_xo = pd.concat([df_xo, pd.DataFrame([new_dat])], ignore_index=True)

df_xo = df_xo[df_xo["date"]>date_ti]
df_xo = df_xo[df_xo["date"]<date_to]
```

```
#find upper and lower threshold
bound_gup = [ abs(el1 + el2) for (el1, el2) in zip(list(df['Mean']), final_thres)]
bound_glow = [ abs(el1 - el2) for (el1, el2) in zip(list(df['Mean']), final_thres)]

bounds_glow2 = []
for xu in bound_glow:
    if xu < 0:
        xu = 0
    bounds_glow2.append(xu)
```

init_bounds.ipynb

```
# cluster time series data as per decided variable name convention
for p2 in NUM22:

    t1_da1 = y2+'_'+z2+p2
    val_h = eval(t1_da1)
    dat_name.append(t1_da1)

    t1_da2 = val_h[val_h["d_hour"] > 5]
    t1_da3 = t1_da2[t1_da2["d_hour"] < 10]

    t2_da2 = val_h[val_h["d_hour"] > 9]
    t2_da3 = t2_da2[t2_da2["d_hour"] < 13]

    t3_da2 = val_h[val_h["d_hour"] > 12]
    t3_da3 = t3_da2[t3_da2["d_hour"] < 16]

    t4_da2 = val_h[val_h["d_hour"] > 15]
    t4_da3 = t4_da2[t4_da2["d_hour"] < 19]

    t5_da2 = val_h[val_h["d_hour"] > 18]
    t5_da3 = t4_da2[t4_da2["d_hour"] < 23]

    t6_da2 = val_h[val_h["d_hour"] > 22]
    t6_da3 = val_h[val_h["d_hour"] < 6]
    t6_da4 = pd.concat([t6_da2, t6_da3], ignore_index=True)

    for x2 in time:
      name = x2+'_'+y2+'_'+z2+p2
      print(name)

    t1 = time_t[1]+'_'+t1_da1
    t2 = time_t[2]+'_'+t1_da1
    t3 = time_t[3]+'_'+t1_da1
    t4 = time_t[4]+'_'+t1_da1
    t5 = time_t[5]+'_'+t1_da1
    t6 = time_t[6]+'_'+t1_da1

    t1a = t1_da3
    t2a = t2_da3
    t3a = t3_da3
    t4a = t4_da3
    t5a = t5_da3
    t6a = t6_da4

    tup = ((t1, t1a), (t2, t2a), (t3, t3a), (t4, t4a), (t5, t5a), (t6, t6a))
    data2[t1_da1] = tup
```