Frankfurt University of Applied Sciences

–Faculty of Computer Science and Engineering–

Detection and Visualization of Anomalies in Cloud-Based Enterprise Applications

Master Thesis for High Integrity Systems (M.Sc.)

By

Mohammad Faiz Usmani

Matriculation number: 1323197

First Supervisor : Prof. Dr. rer. nat. Josef Fink
Second Supervisor : Msc. Bastian Stock

DECLARATION

I hereby confirm that this thesis titled **"Detection and Visualization of Anomalies in Cloud-Based Enterprise Applications"** and the work presented are my own and completed independently without the use or help of others.

Figures in this thesis have an appropriate source reference or have been created by myself.

All passages from published and unpublished sources, appearing either verbatim or in adapted form, were properly identified as such.

The thesis has not been, in this form or in a similar form, submitted to any other examination authority.

Frankfurt, 15th November 2023

Mohammad Faiz Usmani

ABSTRACT

Anomaly detection is the process of analyzing data to identify patterns and observations deviating from the normal behavior. The process of anomaly detection is a reiterating and daunting task because usually the nature of anomalies is unknown, rare, evolving, and diverse. Companies usually have a designated team or a solution in place and invest a considerable amount to detect the anomalies as they may lead to significant downtime and revenue loss. Major sectors where anomaly detection is employed frequently include Manufacturing, Cybersecurity, IoT and Event Communication amongst others. This thesis briefly describes the real-world use cases of some big-tech companies. The concepts and definitions associated with anomaly detection along with some of the most used techniques are then explained. Furthermore, the methodologies used in this thesis for checking time series, correlations, finding anomalies, and the tools/libraries used are illustrated. Afterwards, the dataset used for this thesis is explored which consists of data about millions of events between various cloud services for a time period of 1 month (provided by a major energy company in Germany). The procedure and its results are discussed in detail and proof is provided of valid and explainable anomalies using Python and Azure Data Explorer. Finally, the thesis concludes with the findings along with the recommendations for future work.

"In God we trust, all others must bring data."
"The ultimate purpose of collecting
the data is to provide a basis for
action or a recommendation."

— W. Edwards Deming

ACKNOWLEDGMENTS

I would like to express my gratitude to my university supervisors Prof. Dr. rer. nat. Josef Fink and Msc. Bastian Stock for providing me with the opportunity to work on this thesis. Their guidance and valuable feedback enabled me to complete this work on time.

I would also like to thank my family and friends for their constant support and encouragement in all of my pursuits.

CONTENTS

LIST OF FIGURES

LISTINGS

ACRONYMS

RAD	Robust Anomaly Detection
IQR	Interquartile range
STL	Seasonal Trend Decomposition
ADX	Azure Data Explorer
KQL	Kusto Query Language
ADTK	Anomaly Detection Toolkit
RPCA	Robust Principal Component Analysis
ESD	Extreme Studentized Deviate
S-H-ESD	Seasonal Hybrid Extreme Studentized Deviate
LOESS	Locally Estimated Scatterplot Smoothing
MCD	Minimum Covariance Determinant
MD	Mahalanobis Distance
SVM	Support Vector Machine
DBSCAN	Density-based Spatial Clustering of Applications with Noise
SR-CNN	Spectral Residual Convolutional Neural Network
LSTM	Long Short-term Memory
RNN	Recurrent Neural Network
LOF	Local Outlier Factor
IF	Isolation Forest
ARIMA	Autoregressive Integrated Moving Average
SARIMA	Seasonal Autoregressive Integrated Moving Average
ADF	Augmented Dickey-Fuller Test
MAD	Median Absolute Deviation
LRD	Local Reachability Density
AR	Autoregressive
IoT	Internet of Things

1 INTRODUCTION

Anomaly detection is a process that has been present in the nucleus of data analysis in one form or the other for the past few decades if not centuries. It is a process of identifying unusual patterns or data points in a dataset that do not conform to the expected pattern of the data usually observed over time (time series). Anomaly detection nowadays has applications in numerous fields but is most widely used for Fraud and Intrusion detection. One such example of the former being mainly used by banks and payment processing companies is to analyze the data of various payment methods (debit, credit cards, etc.) and identify fraudulent behavior. For the latter, it involves monitoring network traffic and differentiating between normal behavior and malicious attempts to compromise the system, used in the field of cybersecurity [25]. So, in the terms of data science, it is a process that involves analyzing and modeling the data to determine its normal behavior and then identifying the deviations as anomalies.

The importance of anomaly detection can be asserted with the fact that companies like Uber, LinkedIn, and Netflix have their own anomaly detection solutions in place. The growing nature of Uber along with the dynamic nature of their system architecture combined with peak traffic on certain days in their system, which resulted in false positives and false negative alarms lead them to develop ARGOS [8]. Thanks to their streaming anomaly detection algorithm, Uber engineers are alerted to unidentified outages less frequently. Similarly, Netflix developed its outlier detection function called Robust Anomaly Detection (RAD). This helped them in identifying anomalies in failures in the payment network at a bank level, and also in their sign-up process which helped their engineers understand and react to customer sign-up problems on time [28].

There are a plethora of techniques present to detect an anomaly whose nature is not known beforehand, with the respective libraries present in Python, R, etc. They can range from simply finding the z-score and InterQuartile Range (IQR) and seeing the points that fall outside a set range, to unsupervised methods like Local Outlier Factor (distance and density based method), Isolation Forest (Method based on Decision trees) and Autoencoders (Deep learning approach). There are also time-series specific methods like Seasonal Trend Decomposition (STL) which decomposes the time-series into its Seasonal, Trend and Residual components and then finds the anomalous points.

As more and more companies are shifting their whole solutions to cloud providers like AWS, Azure, GCP, and others; combined with the latest application development and deployment processes like microservices, containerization increases the chances of error or an anomaly occurrence when multiple applications communicate with each other using event hubs, API gateways, etc. The exact source of anomaly can be pretty hard or time-intensive to identify which can make it difficult to plan corrective actions. However, there are well-researched and effective supervised, and unsupervised anomaly detection methods that can be used on-premise. But to really leverage the power of cloud, major cloud providers have developed their own data analytics platform. Azure Data Explorer (ADX) is one such platform that can easily integrate with numerous services and can perform operations like data collection, ingestion, storage, indexing, querying, and visualization with ease on terabytes of data. One of its major advantages lies in carrying out time series analysis very efficiently. ADX uses Kusto Query Language (KQL) which is used to explore data and discover patterns, identify anomalies and outliers, create statistical modeling, etc. It is similar to SQL as it is used for querying the data, but unlike SQL it cannot be used for updation or deletion.

1.1 MOTIVATION

The motivation for this thesis firstly includes understanding the techniques employed by big Tech companies to understand their rationale. Secondly, to gather knowledge about the workings of State-of-the-art methods used for Anomaly Detection. Lastly, to select and implement methods for the dataset presented in this thesis and draw conclusions about the anomalies only from the understanding of the results, without extra business domain knowledge.

1.2 THESIS GOAL

This thesis is conducted on data provided by a major Energy and Electricity provider company in Germany. The data is event-based and structured involving multiple sources in which millions of events are present per month. One such process in the company is the 'Order to Contract' process in which a new contract is filled out by the potential customer and then sent for approval using their online portal. Sending a new contract constitutes an event that may not reach the desired authority due to an anomaly. There are also other types of events present in the data. The goal is to identify anomalous behavior and the reasons for that. The process of anomaly detection will be applied using two different techniques:

- The first approach would be using Python libraries (sklearn, ADTK, etc.). The algorithms are Outlier-based (Isolation forest), time-series based (STL Decomposition), Statistical Measures (Median Absolute Deviation), and others.

- Using the in-built functions in ADX to detect anomalies and identify patterns for anomalous behavior.

These two methods are then compared based on certain metrics which include complexity, time required, and cost. Finally, the output data will then be visualized in PowerBI and ADX's dashboard to see the results.

1.3 RESEARCH QUESTION

Based on the Thesis goal, a generalized research question can be phrased as: *"Is it possible to detect and visualize so far unknown, valid, explainable and actionable anomalies in event communication between multi cloud-based enterprise applications?"*

1.4 STRUCTURE

The documentation of this thesis contains the following sections. The 1st section after the introduction deals with the real and general use cases of how Anomaly Detection is being used in various companies and businesses along with its impact. The focus is narrowed down to the relevant terminologies along with the existing techniques and research in the 2nd section. The next section(3rd) describes in detail the techniques and tools used in the thesis. The 4th section describes the procedure implementation in a step-by-step way (including data processing, analysis and challenges faced), allowing the reader to assess the validity of the results. The 5th section discusses the findings of various methods applied from the previous section in detail. Finally, the last section concludes the work carried out along with the scope for further research. The sections mentioned above can be outlined as follows:

- Business Use Cases
- Approaches for Anomaly Detection
- Methods, Techniques, and Tools
- Case Study
- Discussion
- Conclusion and Future Scope

2

BUSINESS CASES

This chapter briefly goes over the techniques that are used by the leading companies in the present time for Outlier/Anomaly Detection. Moreover, it presents the general approaches that are or have been used in different types of industries.

2.1 NETFLIX [28]

Netflix- One of the most popular Movies and TV shows streaming service. Their datasets grow by a number which is in Billions per day. With this huge amount of data, they were faced with some generic challenges to identify anomalies in their datasets. Those challenges were - multiple unique values across a large number of columns (high cardinality) and grouping of these values through various permutations and combinations which made manual investigation very difficult, data was not always normally distributed, presence of multiple seasonality effects like daily, weekly, etc. which could have been recognized as outliers, and finally, decreasing the number of false positive anomalies.

To tackle the above challenges, they developed the Robust Anomaly Detection (RAD) function. They open-sourced it in 2015 as part of their Surus project, Surus is a Maven project that consists of Netflix's internal user-defined functions available on Github [78]. The engineering team tried Moving averages and other regression models but was not satisfied with their results, especially for datasets with high cardinality. They finally settled on a Robust version of Principal Component Analysis (RPCA). Principal Component Analysis (PCA) basically reduces the dimensionality of the data by transforming a set of variables into smaller ones but also retaining the information that was originally present in the dataset. RPCA is used in cases when the data is affected by outliers, it runs for a limited number of iteration and in each step it identifies the low dimensional representation, outliers and random noise. It also calculates thresholds for values and errors and apply them in the next iteration. Netflix's team responsible for RAD wrote a wrapper in Apache Pig as it is their primary ETL language.

They used it to detect anomalies in two cases. First one is the sign-up flow, as Netflix has customers across the world who use a lot of different devices and browsers to sign up. RAD helped their engineering team to identify anomalies across unique combinations of country, language and devices. Secondly, they used it to detect anomalies in the case of failed transactions at the bank level. As the transactions are in the form of both batch processing

and real-time between thousands of financial institutions and Netflix every day, the finance and business team were able to identify and follow up for the payment issues with the banks rather than the customer in due time.

2.2 X (FORMERLY TWITTER) [69]

Twitter developed its own open-sourced Anomaly Detection package in R [76]. The reason for them doing this can be summarized in the following points :

- Anomalies in a lot of places are contextual in nature, data in the social media domain also have trend and seasonal components in it which can be misclassified as an outlier. So the technique that works in one domain can be of not so much use in the other.
- Presence and identification of global and local anomalies in the data at the same time. Global anomalies are easier to identify as they overextend both above and below thresholds and are not affected by seasonality and trend in the data. Whereas local anomalies are robust in nature and are difficult to identify as they are often masked by seasonality and trend.
- Identifying positive (too high) and negative (too low) anomalies, so the engineering team can prepare itself in a better way for the future whether it be efficient capacity planning in case of positive anomalies or potential hardware and data collection issues for the negative ones.

Their main algorithm is S-H-ESD, it can be considered as an improved version of the Extreme Studentized Deviate (ESD) method. ESD is based on Grubb's test which is used to find exactly one outlier [65]. In it, the maximum deviation is taken between the values and the mean, and then the test statistics are calculated. Finally, it is checked that if the maximum critical value is higher than the test statistics for it to be considered as an outlier. The ESD method is a generalization of Grubb's test as it can handle more than one outlier. In each iteration, one point is removed and the procedure of the Grubb's test is performed until the upper bound provided by the user for the number of anomalies is reached. Twitter proposed an advanced version of ESD which decomposes a time series into its seasonal, trend, and residual components like STL but with a key difference which is using the median to represent a steady trend instead of the trend component like in STL. It also uses Median and Median absolute deviation (MAD) in place of mean and standard deviation to calculate the z-score. As mean and standard deviation are easily affected by a large number of anomalies, the Twitter team argued that a better measure of central tendency can be calculated with median and MAD. The approach is called Seasonal Hybrid ESD (S-H-ESD).

2.3 UBER [8]

For monitoring its system health and generating alerts for the respective personnel, Uber uses user-defined static thresholds across its metrics. Along with it, they also have dynamic thresholds because of their growing user base which they update on a regular basis. The static thresholds posed some basic but important challenges: the generation of too many false positives and false negatives based on setting the thresholds too tight or too loose. Also, the generation of too many alerts based on false cases might lead to the responsible person ignoring the real ones. To add to the complexity, Uber has a dynamic and flexible Backend architecture in which any service can communicate with any other service. So, root cause analysis required extensive knowledge of the system which was not an easy feat to achieve.

To mitigate the above-mentioned problems along with several more, Uber engineering team developed a streaming anomaly detection algorithm that had to adhere to certain specifications:

- A computationally robust algorithm that accounts for daily, weekly, or any other kind of seasonality.
- The algorithm should not always depend on the labeled historical data. It should not also miss a real outage even at the expense of false alerts.
- Choosing a minimum number of data points to forecast as accurately as possible, as streaming data can be considered infinite in length. Likewise, past outliers must not affect the outlier score for the in-streaming data points.
- The algorithm should distinguish between outages and high demand based on the multiple data streams and be able to adjust itself accordingly.
- It should be able to identify if the data point is an outage before the arrival of the next data point. Therefore, it should not be algorithmically too complex.

The algorithm was designed to have 2 parts to solve the issue of algorithmic complexity- outlier and outage detector. The outlier part which is hosted online detects outliers based on pre-calculated thresholds which are updated hourly. After that, these outliers are passed onto a more computationally complex outage detection algorithm to be analyzed. The algorithm in general has a median-based approach. The two user-defined inputs it takes are the list of metrics to be taken into consideration along with the channels (email, ticket, SMS, etc.) by which the responsible person should receive an alert. The algorithm calculates upper and lower thresholds for a time-series ahead of time (Figure 2.1), these thresholds dynamically adjust

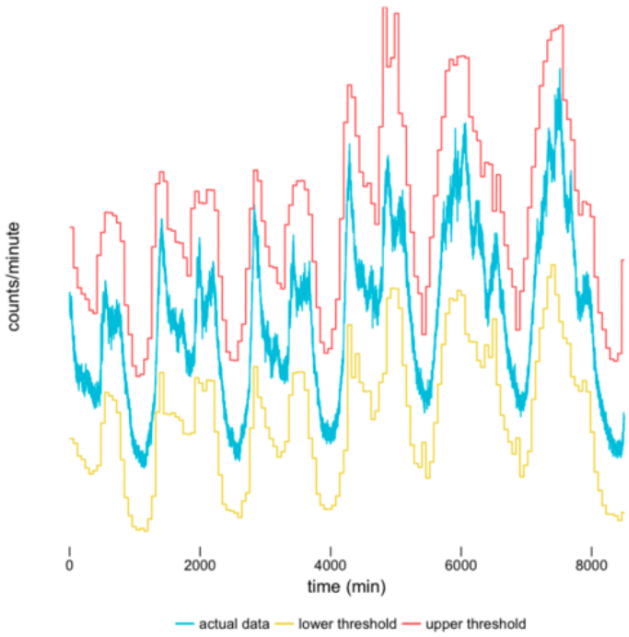


Figure 2.1: Hourly Thresholds. Adapted from [8].

themselves as time advances, this is done for individual time-series per metric(s).

Once the outliers are identified, the different time-series are clustered together based on their similarities to understand the dependencies of various services. The length of the edges between the nodes(different time-series) represents how much similar are they to each other. The color denotes the probability of a time-series being in an anomalous state. Positive alerts denote values that are more than the upper threshold and Negative alerts denote values that are less than the lower threshold. This whole solution of algorithms, clustering, and dashboard was named ARGOS as shown in Figure 2.2.

Thanks to the algorithm, unidentified outages are less frequent. Moreover, because of the clustering representation, an issue with a single metric is conveyed to the responsible person (Example- On call Engineer). But, an issue affecting a number of connected metrices is escalated higher up the authority (Example- Engineering Director).

2.4 LINKEDIN [56]

LinkedIn developed 'ThirdEye', which is their business-wide monitoring platform. The platform assists and tracks the performance and health of LinkedIn's various services and systems. In order to maintain a good user experience, it utilizes machine learning and data analytics to identify anoma-

Figure 2.2: ARGOS dashboard showing clustering and alerts. Adapted from [8].

lies, provide insights, and carry out proactive actions.

The reasons LinkedIn developed ThirdEye can be listed as follows:

- To maintain a high-quality user experience by quickly detecting and resolving issues that could negatively affect user interactions, such as slow loading times, errors, or downtime.
- To have an automated solution to keep track of various aspects of the platform because monitoring such a complex environment manually is extremely challenging.
- To identify issues and anomalies in real-time or even before they impact users. Allowing them to proactively address potential problems before they impact user experience.
- To have insights and alerts that provide actionable data that can guide decision-making for improvements and optimizations across various services and systems.
- To manage the complexity of LinkedIn's platform by providing a comprehensive view of the system's health and performance, allowing them to tackle issues more effectively.

ThirdEye collects data from various sources within the LinkedIn infrastructure. This data can include metrics such as server performance, user interactions, response times, error rates, etc. An appropriate model from various available ones is then selected based on the use case. Historical data with no anomalies is then used to train the selected machine learning model, the thresholds and rules are then set based on the training. When the data arrives in real-time, ThirdEye compares it to the established thresholds. If a data point is significantly above the thresholds, an alert is generated for an

anomaly that is sent to the responsible person along with the relevant details. The generated alerts are continuously reviewed for false positives and false negatives by subject matter experts, these feedbacks are then used to refine the model and redefine the thresholds (Figure 2.3). ThirdEye also analyzes correlations between different metrics to identify complex anomalies.

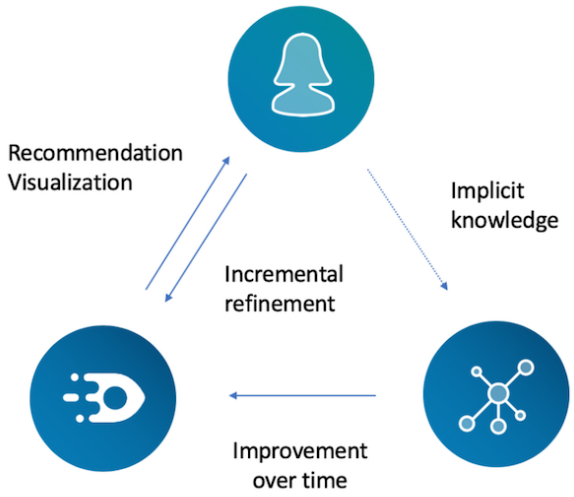


Figure 2.3: ThirdEye's Feedback Loop. Adapted from [56].

2.5 PINTEREST [12]

Pinterest tried moving away from its static rule-based alerts for the following reasons:

- Any kind of seasonality is not taken into account by static rules and will result in unnecessary false positives.
- If a rule is made for a particular period of time say for 12 am to 12 pm, 12:01 or 12:02 pm will be similar in nature but will not be accounted for by the static rule.
- For predicting anomalies in the future, historical anomalous data should not be considered to configure static rules.

After carefully considering various algorithms, the Pinterest team found the Decomposition algorithms best for their domain. To build their system, they had to keep several requirements in mind which were similar in nature to the companies mentioned before- minimizing false positives, being robust to missing data and previous anomalies, autoscaling to millions of time series when required, prioritizing actionability based on the nature of anomaly and sending alerts to the responsible person within minutes.

To keep their model updated and find anomalies in real-time, Pinterest team employed two forms of updates: Updating the model online as and when a new data point arrived and changing the parameters if required, and updating the model by Brute force whenever it seemed fit. For real-time analytics they found out that LOESS method in STL was not scalable, instead they used simpler techniques i.e. Fourier and historical sampling. They also replaced the trend component of STL with the median. For marking a point as an anomaly they used the classical three-sigma rule which states that 99.73% of observations lie within three standard deviations of the mean in a normal distribution. If the data was non-normal, power transformations were used to get the normal one.

As shown in the Figure 2.4 their anomaly detection system's architecture consists of a forecasting server that generates 'one-step ahead forecasts' in real-time for their metrics dashboard (known as Statsboard) and stores them in their time-series database (TSDB). The server consists of a scheduler that submits 'one-ahead forecast' jobs every 60 seconds, a job queue, and numerous autoscaling workers. The alerts in case of an anomaly are handled by the Alert server.

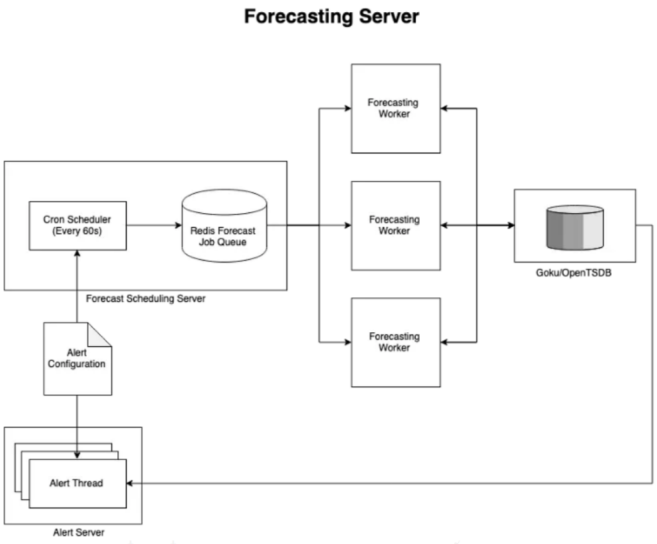


Figure 2.4: Forecasting Server Design. Adapted from [12].

2.6 GENERAL USE-CASES

- Fraud detection in the Banking sector involves detecting anomalies that are usually deceitful in nature like looking into debit/credit card transactions to identify the fraudulent ones. There are two major techniques for this: The by-user technique monitors each credit card used by its usage patterns over time, on the other hand by-operation technique monitors transactions that have been carried out at the same location [54].

- Anomaly detection in the healthcare sector is used to detect anomalies for the anticipation of diseases and outbreaks [24]. Also, historical patient records are analyzed to see any instrumentation or recording errors. Mutations in DNA are also identified using Genomic Anomaly Detection techniques. Real-time data is continuously monitored through wearable devices and using statistical methods alerts are generated in case a reading crosses a certain threshold based on the person's age, gender, etc.

- In the manufacturing industry anomaly detection is used to collect sensor data from machines and other equipments [54], sudden deviations in parameters like temperature, pressure, and vibration can indicate potential equipment malfunctions or breakdowns. In the supply chain management sub-sector it is applied to identify unusual patterns in demand, inventory levels, and supplier performance. Historical data related to anomalies in machines and equipment is used for better predictive maintenance to prevent breakdown and malfunctioning. Wear and tear of equipments/gears are anomalies that trigger alerts for maintenance by technicians before irreversible damage. Deviations from the optimal parameters are identified for a manufacturing process which helps manufacturers fine-tune their operations for efficiency and quality.

- In the field of cybersecurity, computer networks and applications are monitored to identify unauthorized or suspicious activities that could potentially compromise the security of the system. A baseline is learned and established for normal behavior of network and system activities. Any deviations from the baseline may indicate malicious behavior, this process is known as anomaly-based Intrusion detection. An organization's employee log-on patterns, activities, and behavior are profiled, any deviations from these behaviors for example accessing sensitive information at odd hours and doing unauthorized activities are flagged and the employee or the managers are contacted or worse user access rights are suspended based on the severity of the activity [32].

3

APPROACHES FOR ANOMALY DETECTION

This chapter describes the different terminologies associated with the concept of Anomaly Detection, which are important to understand when it comes to the use case that is being dealt with. Then, the most widely used approaches for Anomaly Detection are briefly explained.

3.1 TERMINOLOGIES

3.1.1 *Outlier and Novelty detection*

Outlier is a term that is synonymously used with the word anomaly by Data scientists. In the majority of cases, they both refer to the same thing. But sometimes, outliers refer to expected abnormality while anomalies are rare and unexpected. In the case of Novelty detection, the training dataset used for a model contains no outliers i.e. the good data. The goal of this technique is to check if a new data point behaves like the good data or not [17].

3.1.2 *Point, Collective and Contextual anomalies*

Point anomalies are single points that are very far away from the rest of the data in a dataset. Collective anomaly as the name suggests is a collection of many points together deviating from the normal behavior of the dataset. Contextual anomalies are the ones that may seem normal at first but with added context or metadata, the point turns out to be an anomaly [23].

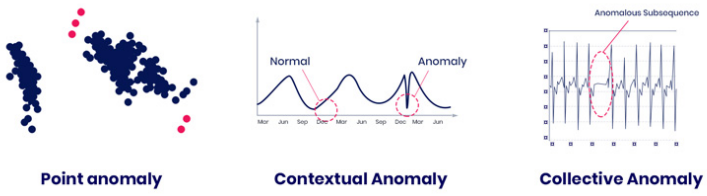


Figure 3.1: Type of anomalies. Adapted from [46].

3.1.3 *Global and Local anomalies*

Global anomalies are data points that fall outside of the normal range for the entire dataset. Local anomalies are points that are not outside the normal range for the entire dataset but are outliers when compared with the surrounding data points. In the context of anomaly detection, local outliers

are harder to identify because they are often masked by the presence of normal points.

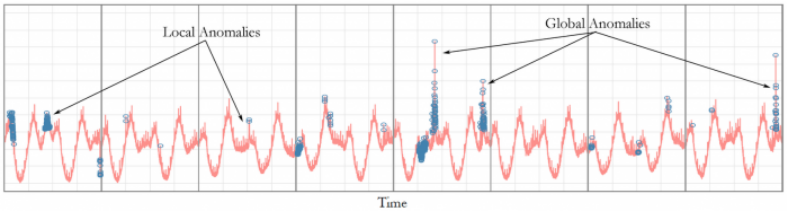


Figure 3.2: Local and Global anomalies. Adapted from [66].

3.1.4 *Unsupervised, Semi-supervised, and Supervised anomaly detection*

In the unsupervised domain, the data used for training is unlabelled and contains both the normal and outlier points. The idea behind using unsupervised techniques is to learn more about the data and the nature of the anomalous points. Outliers identified by the models are usually points that exist in the low-density regions of the data distribution and are detected during the model fitting process [49]. Semi-supervised techniques also known as Novelty detection deal with training data that consists of only normal points, new data points are classified after the model is fitted on the 'good' training data. The outliers are identified as the points differing from the distribution of the fitted data even if they are in the high-density region. In supervised anomaly detection techniques, both the normal and outlier points are labeled in the training data. The model is fitted to the training data and it is assumed that the new outliers if any identified will follow the same distribution as in the training data [49].

3.1.5 *Univariate and Multivariate anomaly detection*

As the name suggests the univariate methods looks at the outlier values for only one variable while multivariate anomaly detection looks at the values of two or more variables and how they correlate with each other, finally a correlated value is compared with other correlated values for it to be marked as an outlier [21]. Whether it be uni or multivariate these variables can be observed over time and then the method of detecting outliers will fall under the category of time-series analysis (Figure 3.3).

3.2 APPROACHES

The approaches for anomaly detection range from simple statistical methods to complex deep learning models and anything in between. The approach/method selection in itself is a task because a lot of times usage of a complex method does not always translate to better detection of outliers,

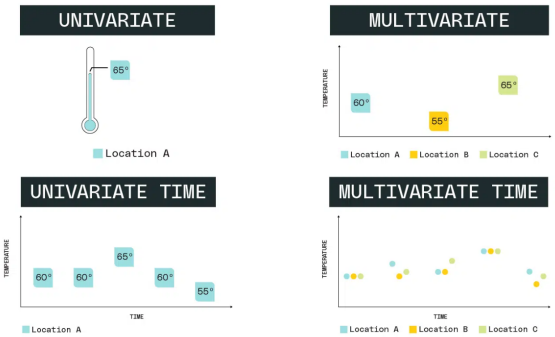


Figure 3.3: A simple example depicting Univariate and Multivariate analysis. Adapted from [21].

more often than not it all depends on the use case and the nature of the data. That is why in some cases a hybrid approach combining two or more methods is also used to yield better and accurate results. Nevertheless, various proven and widely used approaches for anomaly detection are discussed in brief in this section.

3.2.1 *Z-score*

Z-score is a numerical measurement that identifies how far a point is from the dataset's mean. It is measured in terms of standard deviation and is calculated as follows :

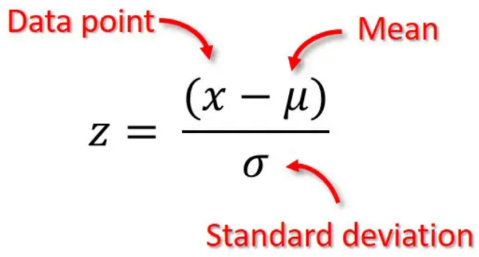


Figure 3.4: Z-score formula

A negative z-score indicates that the data point is below the mean. Similarly, a positive one shows that it is above the mean. And if it is close to zero then that indicates the data point's value is very close to the mean. Z-score works best if the data is normally distributed and as it is derived from the mean, it is sensitive to extreme values [1]. The threshold values for z-score are usually above +2, +3 and below -2, -3 as in a normally distributed dataset around 2% of the points usually lie 2 standard deviations away from the mean and 0.15% of the datapoints are 3 standard deviations apart from the mean.

3.2.2 *Interquartile Range (IQR)*

For an observation or numerical feature in a dataset, quartiles are three points that divide the data into four equal quarters. The first(Q1) and third points(Q3) are known as one-fourth and three-fourth quartiles as 25% and 75% points have values less than them respectively. The second point(Q2) denotes the Median. Interquartile Range (IQR) is calculated by subtracting Q1 from Q3 (IQR = Q3 - Q1) [67]. The IQR essentially represents the range of the middle 50% of the data. It provides information about the spread of values around the median and helps to identify the central distribution of the data without being heavily influenced by extreme values. Then outliers are identified as the points above (Q3 + 1.5x IQR) and below (Q1 - 1.5x IQR) (Figure 3.5). The number 1.5 is chosen because it is just the right factor that identifies outliers in most cases- If it is high, outliers would be considered normal data points and if it is low, normal data points would be perceived as outliers [11]. However, the number 1.5 can be adjusted for fine-tuning and improving the results of IQR based on the distribution of the data. As IQR is based on the median it is insensitive to extreme values or small changes in the dataset.

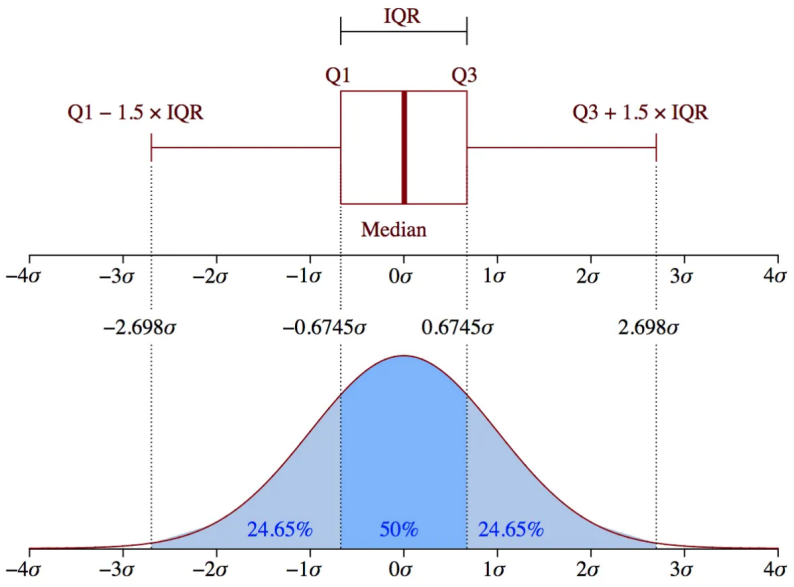


Figure 3.5: IQR depiction. Adapted from [67].

3.2.3 *Minimum Covariance Determinant (MCD)*

The understanding of MCD requires basic knowledge of two important concepts i.e. Mahalanobis distance and Covariance matrix. Mahalanobis distance in simple words is the distance between two points in multivariate space, it is not as simple as measuring the distance between two points with

a ruler like in Euclidean space. If there are two or more variables having correlations with each other, the distance between the points is not straight as the axes are no longer at right angles. A centroid point is found which can be thought of as an overall center of mass of the multivariate data, the Mahalanobis distance (MD) is calculated by taking the distance between the given point and the centroid divided by the width of the ellipsoid in the direction of the given point. Large MD means the point is far away from the centroid and if it is greater than a certain threshold, it is considered as an outlier [26]. For variables that have no correlation MD is equal to the Euclidean distance. A covariance matrix is a square matrix representing covariance values between multiple variables. In it, the diagonal element shows the variance of a single variable, and all the other elements constitute covariance (Figure 3.6). It can have positive, negative, and zero values signifying positive, negative and no correlation [38]. Large values mean a greater correlation between variables.

$$C(x, y, z) = \begin{bmatrix} var_x & covar_{x,y} & covar_{x,z} \\ covar_{y,x} & var_y & covar_{y,z} \\ covar_{z,x} & covar_{z,y} & var_z \end{bmatrix}$$

Figure 3.6: Covariance matrix for 3 variables x,y and z. Adapted from [38].

For calculating Mahalanobis threshold distance, the just right mean and covariance matrix are required so that the anomalies are excluded. This is done by choosing multiple subsets of the data one at a time and estimating the mean and covariance matrix for them. Then the estimates for the subset in which the covariance matrix's determinant was smallest are taken. The smallest determinant signifies densely distributed data which hopefully does not include anomalies [68]. This method works best when the data follows an elliptical (Gaussian) distribution.

3.2.4 *One-Class SVM*

Support vector machines (SVM) are algorithms that are highly used for classification and regression tasks. SVM finds a hyperplane that classifies/separates the datapoints in an N-dimension space (N = Number of features/-variables). Support vectors are points that are near the hyperplane on both or all sides of it and are responsible for the orientation and position of the hyperplane [22]. The end goal of SVM is to find the hyperplane with the maximum margin possible with the support vectors.

Using SVM data can be classified into multiple classes based on the position of optimal hyperplane. In one-class SVM there is only one class, specifying the boundary. Points inside this boundary are classified as inliers and hence the points outside are outliers. One-class SVM has two implementations, one uses a hyperplane similar to SVM for the boundary (Figure 3.7). The other widely used approach utilizes a hypersphere for the decision

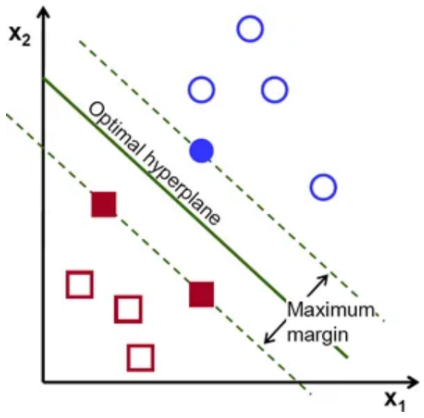


Figure 3.7: SVM finding optimal hyperplane with maximum margin. Adapted from [22].

boundary [77]. Here margin refers to the outside of the sphere and hence the largest margin means finding the smallest possible hypersphere encompassing all the possible inliers (Figure 3.8). The points touching the hypersphere (circle in a 2D space) are called support vectors in this case. One-class SVM is not very suitable for large datasets as it takes some time to generate results.

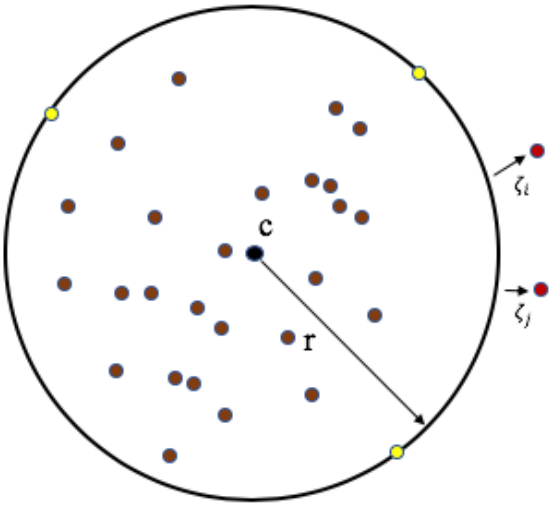


Figure 3.8: Hypersphere depicting inliers, support vectors, and outliers. Adapted from [72].

3.2.5 *K-means Clustering*

Clustering is a technique in which clusters/sub-groups of data are identified and each point is assigned to them such that data points in the same cluster are similar while being different than the points in another cluster [14] based on some metric like Euclidean distance or any other. K-means is an unsuper-

vised technique that uses the clustering approach to iteratively divide the data points into a pre-defined K number of clusters. Initially, a K number of random centroids are selected, and then each data point is assigned to a particular cluster where the sum of the squared distance between the centroid and the point is minimum. After the assignment of all the points to a particular cluster, instead of random selection, the centroid is properly calculated by taking the mean of all the points belonging to the same cluster. This process is repeated until the process becomes stable i.e. there is no change in points being assigned to clusters and the value of centroid [14]. For anomaly detection, a threshold value for distance is identified, and if a point has a distance more than this value from the K centroids, it will be considered an outlier. Limitations of K-means are that it always tries to assume that the clusters are spherical in shape which may lead to wrong classifications [59]. It is also susceptible to local outliers and finally, may form a cluster of anomalous points if there are several of them nearby identifying them as normal points.

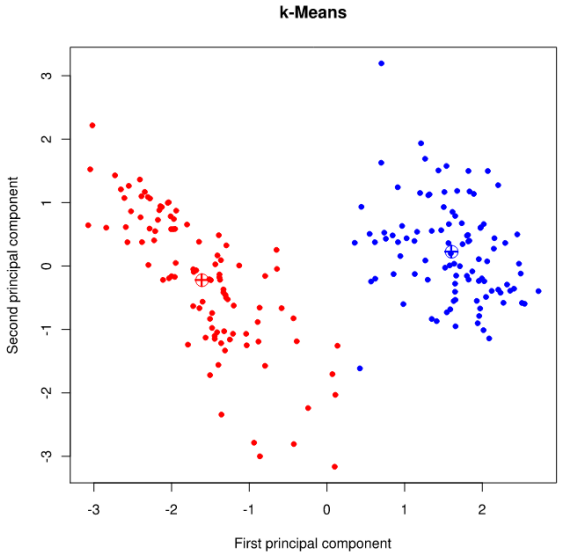


Figure 3.9: K-means representation with K = 2. Adapted from [33].

3.2.6 *DBSCAN*

Density-based spatial clustering of applications with noise (DBSCAN) is a clustering-based algorithm similar to K-means but instead of using a centroid-based approach, it uses a density-based approach. It forms clusters of dense points that are close to each other, ignoring the scarcely distributed points as noise. It needs two parameters initially- Eps (epsilon): The maximum distance between two points so that they can be considered as neighbors, min_samples or minPts: Minimum number of points to identify a region as dense and mark the point as a core data point [63]. After these two parameters are defined, for each point its distance is calculated from other data

points. Once the distance is less than or equal to epsilon, it is marked as the neighbor of that particular data point. If a data point gets neighbors greater than or equal to minPts, then it is marked as a core data point. Finally, each core point is assigned to an existing or new cluster and recursively its neighboring points are marked and then assigned to the same cluster [63]. The points that do not belong to any cluster are marked as noise or outliers. The advantage of DBSCAN is that clusters can have any shape unlike K-means but does not work well for thinly distributed datasets.

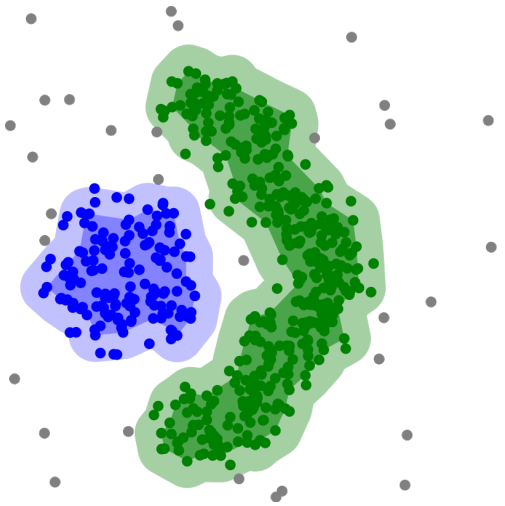


Figure 3.10: DBSCAN identifying two clusters and noise. Adapted from [73].

3.2.7 *SR-CNN*

Spectral Residual Convolutional Neural Network (SR-CNN) is a type of Neural network used for identifying visual saliency. CNNs are typically used to analyze visual imagery. They use an operation called convolution which in mathematical terms takes two functions and produces a third one that describes how the shape of one is modified by the other [43]. In image processing, convolution is used to reduce the size and dimension of an image by noise reduction, edge detection, sharpening, etc. so that they are easier to process meanwhile, also retaining the important information that defines the image. SR-CNN is used to identify patterns or components in an image that stand out from the rest of the features in an image. A similar philosophy can be used to detect outliers in a dataset as normally they stand out from the rest of the data points. SR-CNN is used for outlier detection apart from its usual image processing domain because it is unsupervised, more efficient, and do not depend on hyperparameters [50].

As seen in Figure 3.11 the Spectral residual (SR) part consists of three steps: Fourier transform to get the log amplitude spectrum, calculating the spectral residual, and finally inverse Fourier transform to transform the output back to the original form along with the added info of the spectral residual [74]. The anomalies are magnified and then the whole dataset is used to

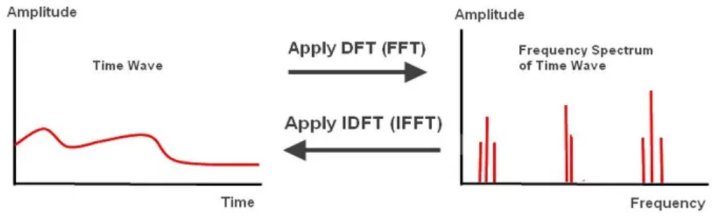


Figure 3.11: Fourier and Inverse Fourier transform applied on time-series. Adapted from [50].

train CNN with the synthetic approach by randomly selecting points from the spectral saliency map and injecting them into the original dataset (Figure 3.12) [74].

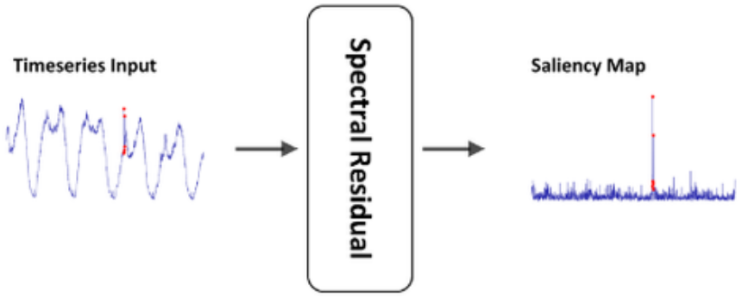


Figure 3.12: Saliency map for a time-series. Adapted from [74].

3.2.8 *LSTM Autoencoder*

Autoencoders are a type of neural network that encodes and compresses the input data retaining the most important information that describes the data, then attempts to reconstruct the original data from the encoded representation [6]. The main advantage of autoencoders is dimensionality reduction i.e. to identify the most important features, points and ignore the noise.
LSTM (Long Short-term memory) is a type of RNN (Recurrent Neural Network) used for learning and processing sequential data and not just the individual points. Normal RNN can be used for the same task but it suffers from the 'vanishing gradient problem' [61] which essentially means it forgets the important info after a short time and cannot be used for longer dependencies. LSTM is similar in working but has a complex cell structure that allows it to decide whether to forget the previous information or process it and pass it to the next cell. So an LSTM cell has three components: 1-Forget Gate, 2-Input Gate, and 3-Output Gate as shown in the Figure 3.13. Like RNN, it has a 'hidden state' also known as short-term memory, and additionally a 'cell state' called long-term memory [61]. t-1 represents the previous point or timestamp and t represents the current timestamp.

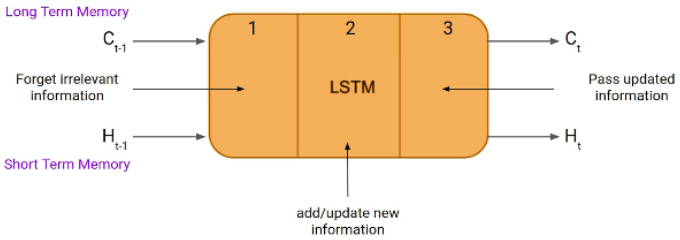


Figure 3.13: LSTM cell structure. Adapted from [61].

As LSTM considers the effect of previous points while processing the current one i.e. it is capable of remembering the complex dynamics between points. This combined with the capability of an autoencoder to ignore the noise, can be used for anomaly detection. In an LSTM autoencoder, the Encoder and Decoder modules have several layers of LSTMs inside it based on the configurations (Figure 3.14). It reads the sequence of data i.e. the normal dataset and encodes it while remembering the important information and its effect throughout [60]. Finally, it recreates the sequence from the latent representation (enoded/compressed data). An LSTM Autoencoder comes under the category of semi-supervised learning because while training it the normal dataset is provided and attempts are made to reconstruct the dataset as close to the original as possible by looking at the loss function and tuning the hyperparameters. The main idea behind doing this is when a dataset with anomalies is fed into the LSTM Autoencoder, the reconstruction will defer from the original one where the points are anomalous because the process will occur considering the normal values the model expected resembling the training dataset [62]. Both the reconstruction and the original dataset can be plotted on top of each other to identify the anomalous points. It is a complex approach that requires a good understanding of both the LSTM and autoencoders to effectively control the parameters and set the layers based on the use case.

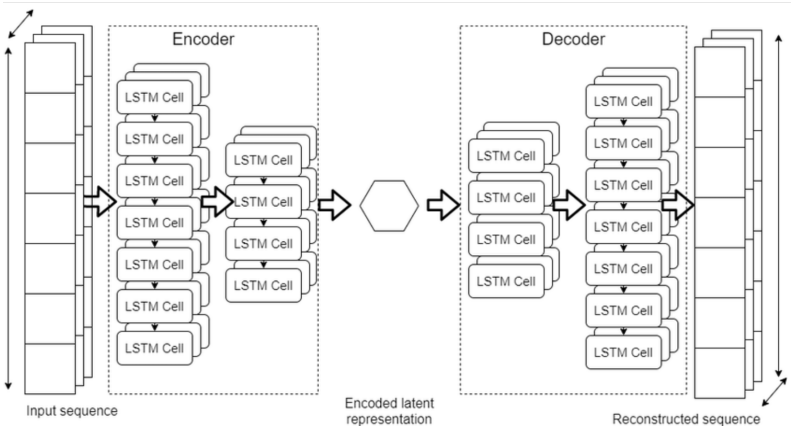


Figure 3.14: LSTM Autoencoder high-level architecture. Adapted from [31].

Note: Since the algorithms Local Outlier Factor, Isolation forest and STL decomposition have been explored in this thesis. They are briefly introduced here and their working is explained in-depth in the Section 4.3.

3.2.9 *Local Outlier Factor (LOF)*

Local Otlier factor belongs to the class of density-based outlier detection algorithms. The main concept on which it is built is local density. A point is looked one at a time and then a k-number of neighbors are used to estimate its density [35]. The local density of neighbors is then compared to the local density of a point and if it has a very low density, then the point is considered to be an outlier. It uses a concept known as 'reachability distance' in its algorithm. An outlier score is assigned to every point indicating whether it is an anomaly or not, and a score greater than 1 usually indicates an anomalous point [35]. As the name of the algorithm suggests, it is good for detecting local outliers as it only considers the direct neighborhood of a point. Meanwhile, it should also be noted that if a use case requires only extreme values to be marked as anomalies, LOF will generate a lot of false alarms and might not be the best candidate.

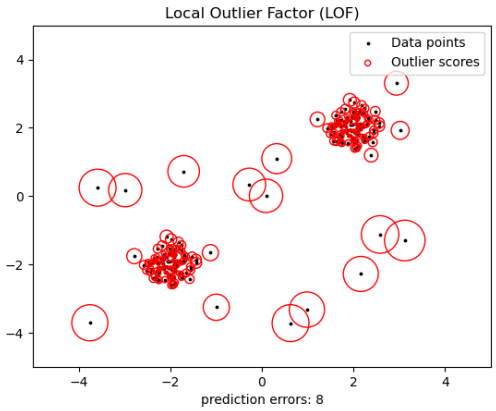


Figure 3.15: LOF depicting points and circles around them representing outlier score. Adapted from [75].

3.2.10 *Isolation Forest (IF)*

Isolation forest is an algorithm based on the idea of random forests where multiple decision trees are used to predict the anomaly score of a given point. It randomly splits the data along a randomly chosen dimension (in case of multiple features), this split forms two nodes of a tree [42]. IF continues this process until each point is isolated and forms a node. As seen in Figure 3.16, the main idea behind IF is an anomalous point will split and isolate itself earlier on in the process. Hence, the average tree depth score of an anoma-

lous point across multiple decision trees will be smaller than the normal points [42]. It is an unsupervised algorithm that is fast and efficient but not so skillful when it comes to the detection of local outliers. Plus it requires some knowledge of the contamination percentage of the data beforehand so that the correct number of points are marked as outliers.

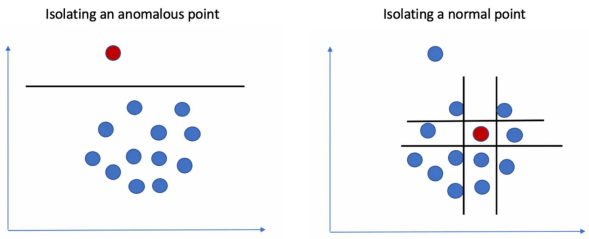


Figure 3.16: Isolation of an anomalous and a normal point in an Isolation Forest. Adapted from [45].

3.2.11 *STL decomposition*

STL decomposition which is a technique used for the analysis of time series data decomposes a time series into its seasonal, trend, and the remaining residual components using a technique known as LOESS smoothing [64]. Trend denotes the overall increase or decrease in a time-series data whereas seasonality is a predictable pattern that repeats itself at regular intervals (hourly, weekly, monthly, etc.). Removing, these two components from a time series data leaves us with the residual component which is generally considered as noise (Figure 3.17) [15]. The idea behind using STL is the residual component contains noisy data including outliers. A threshold is applied to the residual data to identify the points that are far away from the random points which are generally scattered around the baseline mean. It's a good method to analyze a time-series having some sort of trend and seasonality. However, if these components are missing in it, it's hard to identify outliers through this method.

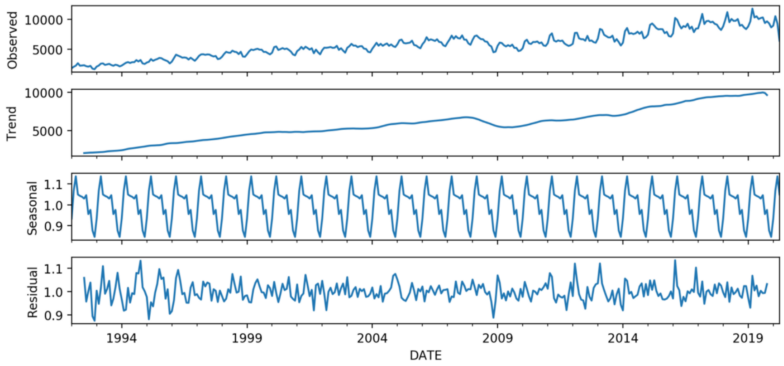


Figure 3.17: STL performed on a time-series dataset. Adapted from [15].

3.2.12 *Forecasting methods for Anomaly detection*

Forecasting a time-series is a domain of its own as there are a plethora of methods to do the same for example- Naive approach, Autoregression, Moving Average, Exponential smoothing, Holt's Winter method, etc. One of the popular classes in this area are methods that combine two or more techniques mentioned above like ARMA (Autoregressive Moving Average) or SARIMA (Seasonal Autoregressive Integrated Moving-Average). One such popular method is ARIMA (Autoregressive Integrated Moving Average). The main idea used in this method is a point in time can be expressed in some form by the points in the past as there is some direct or indirect correlation between them [57]. The AR part of ARIMA represents a regression model in which a point in time is defined by its own values in the past. The AR considers only the direct effects from the previous points (partial auto-correlation). The MA bit considers the errors that happened in predicting the previous time points to have a better estimate of the current time point in question. It considers both the direct and indirect effects of the previous points (auto-correlation) [57]. AR and MA together represent the effect of previous observations and errors and can be used to forecast the time-step in the future. But they have a requirement that the time series should be stationary (i.e. mean and standard deviation are constant throughout). But in reality, it's not the case as more often than not time series have some kind of a trend. I's (Integration) job is to convert a non-stationary time series to stationary. It is done by taking a difference of two consecutive time steps. When done for the first time for the whole series, it is called first order. It can done for n number of orders until the time series becomes stationary, usually it's 1 or 2. This forecasting approach is used to predict a range of points from the past data, these points are then compared with the real ones, and the deviation between them is observed. If they are greater than a certain threshold or going out of the confidence interval (Figure 3.18) [7], the real data points are marked as anomalous.

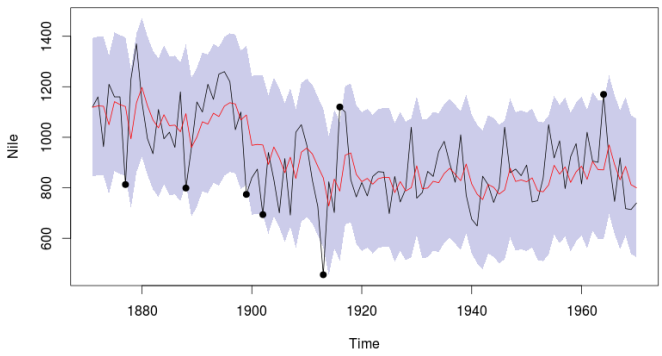


Figure 3.18: Predicted(Red) series by ARIMA and Real(Black) time series, Violet area represents 95% confidence intervals with outleirs marked. Adapted from [79].

4

METHODS, TECHNIQUES, AND TOOLS

This chapter goes over the methods, techniques, and tools explored in this thesis. They are methods for checking white noise and random walk, finding correlations, anomaly detection techniques, ADX, Python libraries, and finally PowerBI for Visualization.

4.1 WHITE NOISE AND RANDOM WALK

A lot of time series follow a pattern including trend, seasonality, and even cyclicity. However, some of them do not have these components and are harder to forecast and detect anomalies amongst them as they do not follow a particular order. There are two kinds of time series belonging to these categories i.e. White Noise and Random Walk. White Noise as the name suggests is a sequence of random numbers at different timestamps following no pattern, it has some characteristics which are the mean is zero and the standard deviation remains constant over time. A white noise series can be detected by an autocorrelation test [9]. Random Walk on the other hand is not so straightforward to identify as it looks like a 'good' time series and is not a sequence of random numbers. It is a summation of a previous value and a random number for a point in a time series. Random walk has a non-zero mean and non-constant standard deviation. Random Walk can be detected by differencing and augmented Dickey-Fuller test [9].

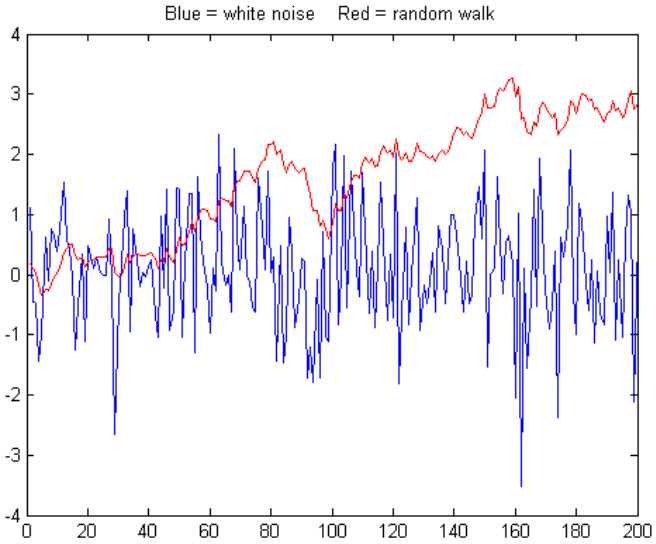


Figure 4.1: White Noise and Random Walk. Adapted from [48].

4.1.1 *AutoCorrelation test*

As explained briefly in the previous chapter, Autocorrelation is a technique that measures both the direct and indirect effects of previous points on the value of the current one. The correlation is measured in terms of its lagged counterpart [39]. So lag = 1 means the correlation with the previous point and lag = 2 means the correlation with previous 2 points. The autocorrelation formula is given by :

$$r_k = \frac{\sum_{t=k+1}^{T} (y_t - \bar{y})(y_{t-k} - \bar{y})}{\sum_{t=1}^{T} (y_t - \bar{y})^2}$$

Figure 4.2: Autocorrelation formula

where $\bar{y}$ represents the mean of the time series at lag k, for k greater than or equal to 0. The function in the numerator is called the autocovariance function and in the denominator variance for the original time series is present.

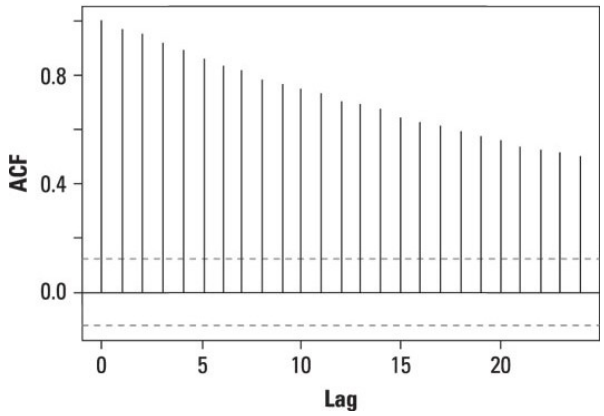


Figure 4.3: Autocorrelation plot

In the Figure 4.3, the autocorrelation upto lag 24 is shown. The dotted horizontal lines above and below 0 represent significance boundaries, meaning any vertical line falling under them is not significant. The autocorrelation of the first line is a perfect 1 because it correlates each term with itself. This test is done to confirm the possibility that a time series is not white noise since the lags are significant. For random numbers, they would lie inside the significance boundary.

4.1.2 *Differencing and Augmented Dickey-Fuller test*

Differencing is a process in which the previous observation is subtracted from the current one, doing this will reveal the random numbers that are present in a random walk that can be confirmed by plotting the differencing of the time series on a graph. The second method is a statistical method that goes by the name of Augmented Dickey-Fuller test (ADF). It is a type of unit root test that checks for non-stationarity of a time series. The null hypothesis states that there is a unit root present and hence the series is non-stationary, rejection of the null hypothesis confirms that the time series is stationary. The null hypothesis is rejected if Test Statistic value < Critical Value and p-value < 0.05 [20]. The equation for ADF is:

$$y_t = c + \beta t + \alpha y_{t-1} + \phi_1 \Delta Y_{t-1} + \phi_2 \Delta Y_{t-2} .. + \phi_p \Delta Y_{t-p} + e_t$$

Figure 4.4: Augmented Dickey-Fuller equation

It is the 'augmented' version of the original Dicky-Fuller equation adding more than one differencing term to it. y(t-1) represents the first lag and delta Y(t-1) shows the first difference of the series at the time (t-1) [20]. ADF test initially assumes that alpha- the coefficient of y(t-1) is 1 and the series is non-stationary.

4.2 CORRELATION COEFFICIENTS

Correlation coefficients are statistical measures used to quantify the strength and direction of the relationship between two variables. There are various types of correlation coefficients, the ones used in this thesis are discussed below.

4.2.1 *Pearson Correlation Coefficient*

Pearson Correlation Coefficient (r) is one such coefficient that is widely used which measures the strength of the relationship between two variables. Its value ranges from -1 to +1 where -1 denotes a strong negative relationship (as one variable increases, the other variable decreases proportionally), 0 denotes no correlation, and +1 shows a strong positive correlation (when one variable increases, the other also increases proportionally). The formula for calculating 'r' is given by:

$$r = \frac{\sum (x - m_x)(y - m_y)}{\sqrt{\sum (x - m_x)^2 \sum (y - m_y)^2}}$$

Where x and y are individual data points belonging to the two different sets of variables X and Y, and 'mx' and 'my' are the means of variables X and Y.

4.2.2 *Kendall's Rank Correlation Coefficient*

Also known as Kendall's Tau, is a statistical measure similar to Pearson as it measures the strength and direction of the relationship between two variables. The key differences between the two are, Pearson's coefficient measures linear relationships and works best for a normally distributed continuous dataset, it is also sensitive to outliers. While Kendall's Tau is non-parametric in nature i.e. it does not assume any specific distribution and is suitable for both measuring monotonic relationships for continuous and ordinal data. Also, it is less susceptible to outliers.
Kendall's Tau value also ranges in from -1 and +1, where -1 represents a negative correlation (ordering of pairs in the two datasets is in the opposite order), +1 indicates a positive correlation (pairs are in the same order), and 0 shows no correlation. The formula for Kendall's Tau is expressed as follows:

*Kendall's Tau = ((number of concordant pairs) - number of discordant pairs))/(n(n-1)*1/2)*

Concordant Pairs are data points (Xi, Yi) and (Xj, Yj) at locations i and j for variables X and Y that follow the same order. If the order is opposite between the points they are known as discordant pairs.

4.3 ANOMALY DETECTION

Although some of the techniques discussed here (IF, LOF and STL) are briefly described in the previous chapter. This section will specifically go over their working mechanisms along with others.

4.3.1 *Median and Median Absolute Deviation (MAD)*

Median is a statistical measure of central tendency that denotes the middle value for a distribution. The observations are firstly arranged in numerical order and the middle value is picked if the number of observations is odd. If the number of observations is even, the average of the two middle values is denoted as the median. It is a better measure of central tendency than the Mean for a skewed distribution or data with lots of outliers as only a few outliers can shift the mean towards themselves, giving a not-so-good picture of the true central tendency.

Like Standard deviation which denotes the spread of data around the mean, Median Absolute Deviation (MAD) denotes the spread around the median. As it is derived from the median, MAD is also not susceptible to outliers and hence is a better measure than standard deviation for a dataset containing outliers. It is calculated by firstly finding the median of the dataset. Then for each point, the absolute deviation is calculated by subtracting the median from each point. Finally, the median for these absolute deviations is computed which denotes MAD. The formula is given by:

MAD = median(| Xi - M |) for i= 1 to n

where M is the median of the dataset and Xi denotes each data point. Using MAD a boundary can be set for identifying outliers which can be given by the formula :

*MAD Outlier Threshold = Median + i*MAD*

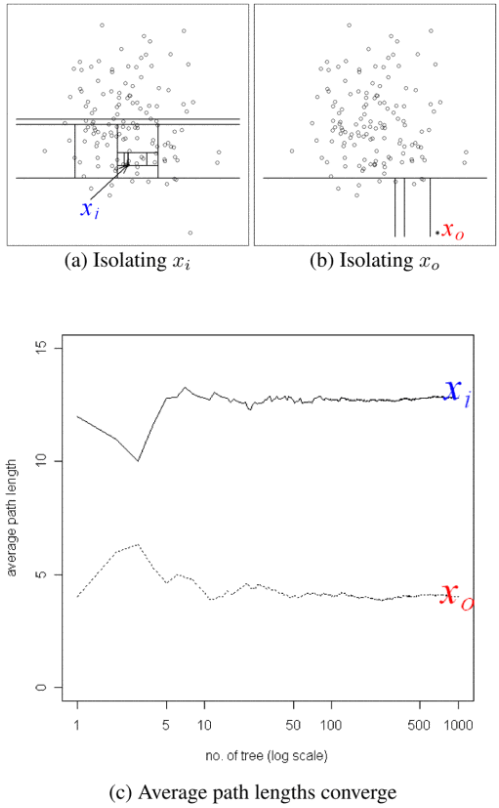
where i usually is in between 2 and 3, depending on the nature of distribution and the number of outliers to be identified. Leys et al. [40] gave various examples through their research and argued why the outlier threshold set by using Median and MAD is better than setting it using Mean and Standard Deviation. It is being used in the finance sector to identify outliers in stock market, and in the healthcare sector to detect outliers in patient data and readings.

4.3.2 *Isolation Forest*

Isolation Forest (IF), an algorithm proposed by Fei Tony Liu, Kai Ming Ting and Zhi-Hua Zhou in 2008 [41] was made on the condition that the most notable anomaly detection algorithms at that time were built to construct profiles of normal instances and not outliers, leading to false alarms. This algorithm profiled outliers based on the properties that they are few and different. As discussed previously, it does random partitioning over attributes and anomalous points are isolated in fewer number of steps via partitions compared to normal points. The algorithm does this in a random forest as each iteration of a decision tree recursively divides the points until the end nodes are reached. As seen in Figure 4.5 a normal point takes 12 random partitions compared to an anomalous point which takes 4 [41]. Also, as Isolation forest does this random partitioning for a number of decision trees to have a better and confirmed outcome, average path length (number of partitions) converges when the number of trees increases.
An anomaly score is generated for each point based on the fact that external nodes have no children and internal node have exactly two children, making them a binary search tree. The calculation of the average path length of an external node termination is same as the average path length of an unsuccessful search in BST [70].

The formula for a point 'x' in a sample size 'm' with path length h(x) is given in Figure 4.6:

Formula in (a) represents average of path length c(m) for h(x) given m. H represents Harmonic number, calculated by the formula in (b). Finally in (c), E(h(x)) is the average value of h(x) from a collection of decision trees used to calculate the anomaly score s(x,m). If the value of s is close to 1 then the point x is an anomaly, if it is less then 0.5 then x is a normal point.



(a) Isolating x_i (b) Isolating x_o

(c) Average path lengths converge

Figure 4.5: Partitioning of a normal and anomalous point in (a) and (b), (c) shows convergence of path lengths. Adapted from [41]

Isolation forests have low memory requirements and a linear time complexity working well with uni and multivariate data. Isolation forests also perform better with sub-sampled data unlike other algorithms whose efficiency increases with the data size but struggle in detecting outliers when they are too close to the normal data points (swamping) [70]. Also in rare cases, IF suffers from bias because of the way partitions take place in a dataset. As shown in the Figure 4.7, the partitions are done in horizontal and vertical cuts and points lying on these cuts or intersection of them will have low anomaly scores if they are outliers also. To tackle this bias extended isolation forests were suggested by Sahand Hariri, Matias Carrasco Kind, Robert J. Brunner which make cuts at random slopes [27].

4.3.3 *Local Outlier Factor (LOF)*

As briefly explained in the previous chapter, LOF uses the concept of local density and K-neighbors for outlier detection. Let's have a look in detail at its working. LOF was proposed by Markus M. Breunig, Hans-Peter Kriegel, Raymond T. Ng, and Jörg Sander in 2000 [10]. There are some concepts that need to be understood to get a holistic view of LOF. K-distance denotes a distance between a point and its Kth nearest neighbor. A circle is drawn with a radius 'K-distance' and all the points that lie inside or the boundary

$$c(m) = \begin{cases} 2H(m-1) - \frac{2(m-1)}{n} & \text{for } m > 2 \\ 1 & \text{for } m = 2 \\ 0 & \text{otherwise} \end{cases}$$

(a)

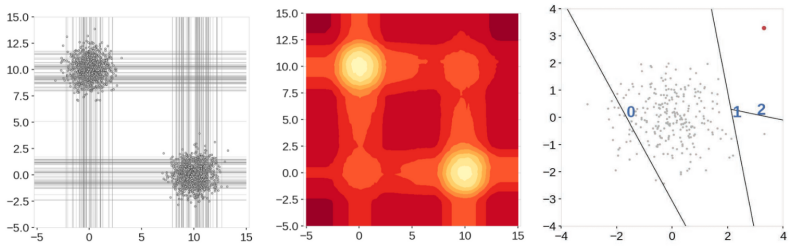
$$H(i) = ln(i) + \gamma, \text{ where } \gamma = 0.5772156649$$

(b)

$$s(x, m) = 2^{\frac{-E(h(x))}{c(m)}}$$

(c)

Figure 4.6: Formulas for calculating Anomaly score in Isolation Forest

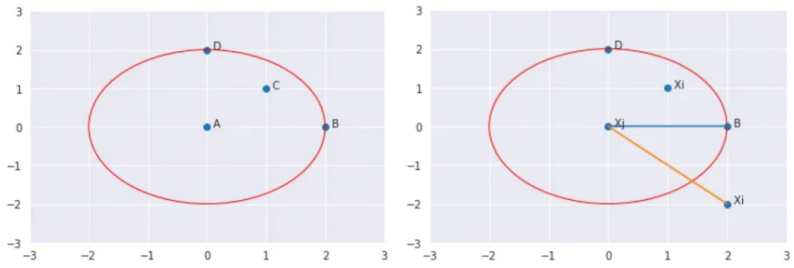


(a) (b) (c)

Figure 4.7: (a) Partitions on two normally distributed clusters, (b) Anomaly score map, (c) Extended Isolation Forest. Adapted from [19].

of that circle are denoted by $N_k(A)$ for the point A. By convention, $N_k(A)$ can be more than or equal to the value of K as there can be more points lying on the boundary of circle having the same K-distance. As shown in the Figure 4.8(a), for K=2, $N_2(A)$=3 [30]. Then there is reachability distance that is given by the formula reachability-distance$_k$(X_i,X_j) = max{k-distance(X_j), d(X_i,X_j)}. It can be expressed as the distance between two points X_i and X_j. If X_i lies within the circle of radius k-distance, it is k-distance(X_j) otherwise it's d(X_i,X_j) as shown in Figure 4.8(b). Using the above info Local Reachability Density (LRD) is calculated which is inverse of the average reachability distance of a point A from its neighbors (Figure 4.9(a)). Reachability distance and LRD are inversely proportional to each other implying less reachability distance means more density of points around point A. Finally, Local Outlier Factor (LOF) is computed which is the ratio of the average LRD of neighbors of A divided by LRD of A (Figure 4.9(b)) [10]. If LOF is less than 1 it means the point lies in a dense region and is an inlier. If it is equal to 1, the point has a similar density as its neighbors. And if LOF is greater than 1 then the point lies in a sparse region and is usually an outlier.

Obviously, its main advantage is that it can detect outliers locally which a lot of algorithms fail to do so. However, LOF might generate false cases in detecting global outliers and a value greater than 1, this doesn't always mean that the point is a clear outlier. In these kinds of cases, various LOF scores are compared and the points with high LOF scores are marked as outliers. Apart from that there are other methods developed on top of LOF,



(a) (b)

Figure 4.8: (a) Point A with K=2 and (b) Reachability distance. Adapted from [30].

$$LRD_k(A) = \frac{1}{\sum_{Xj \in N_k(A)} \frac{\mathrm{RD}(A,Xj)}{||N_k(A)||}} \qquad LOF_k(A) = \frac{\sum_{Xj \in N_k(A)} LRD_k(Xj)}{||N_k(A)||} \times \frac{1}{LRD_k(A)}$$

(a) (b)

Figure 4.9: (a) LRD formula. (b) LOF formula.

aimed at improving its efficiency. One such method is LoOP: Local Outlier Probabilities in which scores are in the range of [0,1] and are directly interpretable as the probability of a sample being an outlier [34]. It is also less sensitive to the choice of parameter K as compared to LOF.

4.3.4 *STL Decomposition*

As discussed in the previous chapter, STL decomposition (Seasonal Trend Decomposition using LOESS) breaks down a time/series into its seasonal, trend, and residual components. To understand the workings of STL, we need to understand LOESS (locally weighted smoothing) as it is the main technique used in STL decomposition. LOESS is a non-parametric technique (a technique that does not make any assumption about the type of distribution for a dataset) that uses weighted regression techniques (linear or parabola) of k-nearest neighbors to generate a best-fitted curve Figure 4.10, using this technique for every point a smooth curve is generated [18]. LOESS is applied to datasets that are noisy, sparse and when other kinds of regression does not seem to produce a good fitting line for it. For example, there is a point in 2-D space having coordinates x and y, a window of size k is taken around x and the points lying inside this window are used to calculate an estimate y' for the point x. A weight is also assigned to every point inside the window based on the fact that the points which are near to the point in consideration will have higher weights and will affect its estimate more. These 'distance-weights' are calculated using the tri-cubic function [18]. In a tri-cubic function, the points lying between -1 and 1 have a positive value, for the rest, it's 0 as shown in Figure 4.11(a). Using the same concept in the

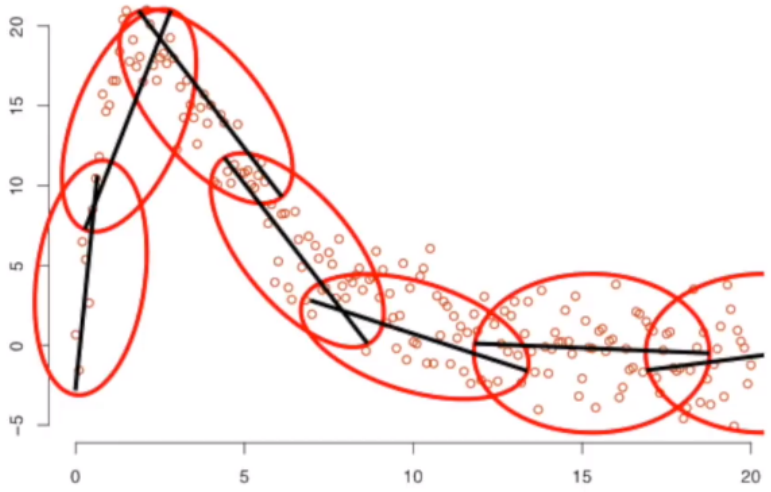


Figure 4.10: Simple Representation of how LOESS works. Adapted from [51]

window, the points that are far off i.e. distance greater than or equal to 1 have a weight of 0 and as the distance gets closer to 0 the weight gets closer to 1. As the distance in the formula is considered between 1 and 0 and in actuality it can be any value, the distance is normalized by dividing the distance between the point in consideration and the neighboring point by the distance between the point in consideration and the farthest point inside the window i.e. maximum distance as shown in Figure 4.11(b). After estimation for every point in the dataset, the estimation process is repeated again for the new points, and weights are assigned based on the fact that new estimated points that are closer to their original counterparts will have a higher weight. This process of estimating a new position of every point is repeated until a smooth curve comes into shape. LOESS requires a large enough dataset to produce good results. Moreover, the regression function produced by it cannot be represented by simple equations [71].

$$w(x) = \begin{cases} (1 - |x|^3)^3 & |x| < 1 \\ 0 & |x| \geq 1 \end{cases} \qquad w(x) = \begin{cases} \left(1 - \left|\frac{d(x,x')}{max_i\, d(x_i,x')}\right|^3\right)^3 & |x| < 1, x_i \in D \\ 0 & |x| \geq 1 \end{cases}$$

(a) (b)

Figure 4.11: (a) Tri-cubic function. (b) Weighting function.

In STL decomposition LOESS is performed in two nested loops. The inner loop iterates between seasonal and then trend smoothing and the outer loop is responsible for damping the effect of outliers [13]. Inside the inner loop, the calculation of seasonal component is done first and it is removed to calculate the trend component next. The window sizes for trend and seasonality are mentioned by the user. A default value for trend window is used if there is no value, but for seasonality, an odd number for the window must

be specified. This gives an idea to the algorithm after how many points the seasonality repeats itself.

4.3.5 *Persistence, Autoregression, and Rolling Windows.*

Persistence model is a simple model in which the next value for a time-series data is predicted to be the same as the last observed value, a time window can also be used for which the next value is the average of the values present in the time window. For outlier detection, if a point in consideration is deviating from the prediction by the Persistence model it is considered as an outlier. Although not a complex and recommended model for outlier detection, it is often used as a baseline model to compare other outlier detection models and assess their significance scores.

Autoregressive model (AR) is typically used for time-series forecasting in which a set number of previous time steps (lagged observations) are used to predict the future value/values. It is usually a part of other complex time-series forecasting models like ARIMA, SARIMA, etc. Its equation is given by:

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

Figure 4.12: Autoregression Equation

where Yt is the value at a time 't', c is a constant, phi1, phi2...phi p are coefficients of lagged values, Yt-1, Yt-2, Yt-3 are lagged values at time t-1, t-2...t-p, Epsilon is the error term at time t. The phi coefficients are estimated from historical data using least squares estimation or other methods. Terminology like AR(4), AR(5) denotes an AR model where the previous, most recent 4 and 5 values respectively are used to predict the next point. For outlier detection, the point in consideration is predicted using the past values and then the result is compared to the original point. If it's too high or too low, it is considered an outlier.

The Rolling window approach involves using a contiguous block of fixed size and moving it along the time axis (Figure 4.13) to calculate a statistical measure for the observations falling in that window/block (mean, median, standard deviation etc.). The main idea behind this approach for outlier detection is that if there are outliers in a particular window then its statistical measure will be significantly different compared to the other windows.

Similarly in the Double Rolling window approach, two windows of equal or different lengths are moved side by side, and the difference between their statistical measures (right minus left) is calculated. If the difference is large on the positive or negative side then that indicates the presence of an outlier in the right or left window respectively.

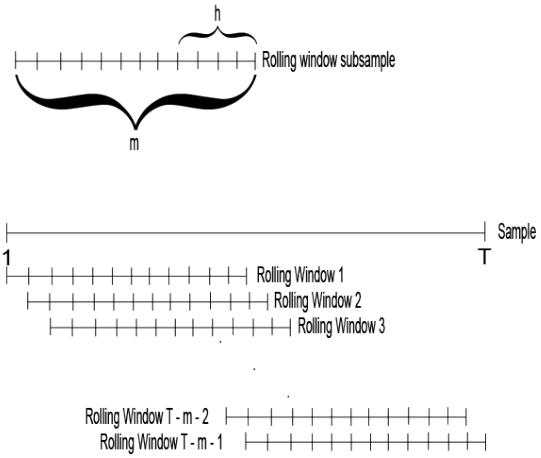


Figure 4.13: Rolling Window Example. Adapted from [16].

4.4 TOOLS

4.4.1 *Azure Data Explorer (ADX)*

Azure Data Explorer [3] is a data exploration service and big-data analytics platform provided by Microsoft Azure. Due to its nature and functions provided, it is considered very efficient for time-series analysis. It is capable of performing real-time analysis on streaming data coming in from various sources like IoT devices, web applications, etc. Being a product of Microsoft, it integrates well with other Azure services and tools. The instances of ADX can be scaled up and down easily either automatically or manually based on the amount of data ingested. It has its own well-built dash-boarding capabilities but also can integrate well with other services like PowerBI. In terms of data security, the data is always encrypted when it is transferred or when it is just residing in a table. ADX also has role-based access control that allows an admin/owner to control the type of access various users can have on the cluster where ADX is hosted [29]. Based, on the statements discussed above ADX is an end-to-end solution where operations like data ingestion, querying, visualization, and user management can be performed under one hood.

- In Figure 4.14, the data sources on the leftmost side are the ones where the data can be pushed to and pulled from. It includes the conventional API approach, Logstash, Kafka, and the approaches native to Azure like Event Hub/Grid and Data Lakes residing on Azure among others.

- The data can be uploaded in real-time (stream) or using the batch approach, the data can be structured and unstructured. While ingesting the data, it is profiled and the parameters like data types, removal of features, etc. can be done on the fly.

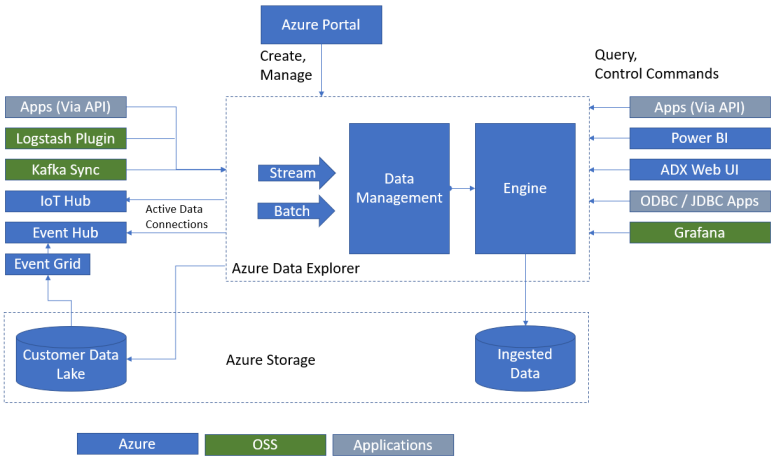


Figure 4.14: Azure Data Explorer Architecture. Adapted from [58].

- ADX mainly has two services that are Data Management and Engine. The Data Management service is responsible for ingesting and pushing the data to external sources, while the Engine is mainly responsible for querying the data and storing it. These two services are clusters of compute nodes hosted on virtual machines that can be scaled up or down depending on the volume and complexity of the data and operations.

- The queries on ADX can be run through its Azure integrated or separate web UI, Kusto Desktop app, APIs, and other visualization services like Power BI and Grafana.

- The creation, management, and tuning of parameters for ADX and other Microsoft services is done using the Azure portal.

Kusto Query Language (KQL) [36] is a querying language similar to SQL that processes read-only requests in ADX and returns the results. The structure of KQL statements can be basically divided into 3 parts which include the main tabular expression statement in which the input and output are both in table format, the statements are combined using the pipe | operator and include functions like aggregation, filtering, time series functions, geospatial functions, joins, unions, and more. The other two are let and set statements used for initializing variables and setting request properties respectively. KQL also supports running database management commands that are essentially not a part of its core syntax. Apart from its documentation, a nice cheat sheet summarizing its commands can be found here [80].

Azure also provides role-based access control that can be modified using the Azure portal or Management commands in KQL by the admin. They include mainly Admin, User, Viewer, and Monitor roles at Cluster and Database levels. The different roles are explained briefly here [37].

Finally, the pricing model of ADX depends on several factors that are: Firstly, the region of availability of Azure (Ex- East US, UK West, North Germany, etc.). Secondly, whether the ADX instances are storage-optimized (fewer queries over a large volume of data) or compute-optimized (high rate of queries over a smaller data size), there are various types within these two optimization categories based on the storage capacity and the number of virtual CPU's. And finally, the data ingestion rate, cache and data retention period, and the number of queries ran per day/month. The detailed breakdown can be found here [4].

4.4.2 *Python libraries*

When it comes to Statistical Analysis and Machine Learning problems the support and the range of libraries provided for Python are second to none, which is why it ranks at the top when it comes to Data Science and Analytics related tasks [84]. The libraries used in this Thesis are mentioned below:

Pandas [81] is an open-source Data manipulation and analysis library with support for mainly two kinds of data structures i.e. Dataframe (two-dimensional table-like data structure with rows and columns) and Series (one-dimensional array-like object). It is used for tasks like Data I/O (CSV, Excel, Databases, etc.), Cleaning, Manipulation, and Analysis.

Numpy [47] is a powerful numerical computing library in Python that is used to perform mathematical and logical operations on matrices and multi-dimensional arrays. It is mainly used for data analysis and computational tasks.

Matplotlib [44] is a library used for creating static or interactive data visualizations like line charts, bar charts, scatter plots, histograms, 3D plots, etc. These plots can be customized and exported to various formats (jpg, png, svg, etc.).

Plotly [52], a library similar to Matplotlib used for creating, customizing and exporting different types of data visualizations. The main difference between the two is that Plotly is more user-friendly and provides better interactive capabilities without additional configurations.

Scikit-learn [82] also known as sklearn is a machine-learning library built on top of Numpy, Scipy, and Matplotlib. It is used for tasks like regression, classification, clustering, etc. and provides a multitude of in-built functions for various algorithms like Support Vector Machines, Decision trees, K-means, etc. Machine learning models can be selected and evaluated using sklearn along with tuning of hyperparameters.

Statsmodels [83] is a Python library used for conducting statistical tests and doing data exploration tasks. It provides functions for doing time-series analysis like ARIMA models, STL decomposition and many others.

Anomaly Detection Toolkit (ADTK) [2] is a specialized Python package for unsupervised/rule-based time series anomaly detection. It includes models like detectors (detection algorithms), transformers (feature engineering methods), and aggregators (ensemble methods) to detect anomalies. It also provides functions to process and visualize time series and anomaly events.

Jupyter Notebook [55] is an open-source web-based IDE that allows a user to create and share documents containing code, equations, and visualizations. It is one of the most preferred IDEs by data analysts and data scientists and supports various programming languages like Python, R, Julia, etc.

4.4.3 *Power BI*

Power BI [53] is a powerful business intelligence and data visualization product developed by Microsoft. In it, data can be read from a plethora of sources, manipulated, modeled, analyzed, visualized using reports and dashboards, and shared through various channels with the team members and stakeholders for making data-driven decisions. Power BI mainly provides a cloud-based online service widely known as 'Power BI Service' and a Windows application 'Power BI Desktop' that is primarily used for designing and publishing reports to the Power BI service.

5 CASE STUDY

This chapter explains in detail the procedure applied to find out the anomalies. Based on the research question the methodology mainly consists of Exploratory Data Analysis (EDA) which means finding correlations and making sense of the data through visualizations so that a suitable anomaly detection method can be applied and its results are interpreted in a better way. The whole approach is divided into two parts i.e. Python-based and ADX-based. The CSV used for analysis contains the following columns:

- eventID - A unique ID according to the type of event. It can be considered as a categorical datatype with 40 distinct values.
- eventIDCompact - Some eventIDs are similar in nature/function and hence they are combined and represented by one eventIDCompact (ex-16all represents 16, 16a, 16b, etc). It has 22 distinct values.
- eventPhase - Splits the events into two phases Order and Contract.
- system - The relevant system from either System1, System2, or System3.
- timestamp - Timestamp when the event occurred including data and time till microseconds.
- eventIDSteps - It can be understood as a ticket number within a service request system assigned per eventID per customer at a particular moment in time. This allows the processing within an eventID to be distinguished.
- retryCounter - As soon as there is a problem processing an event, a retry is attempted, up to a maximum of 15 retries per eventIDSteps are allowed.
- latencyInSeconds - Results from the start time per eventIDSteps up to the highest possible retryCounter at the respective time.

There are listings in every subsection containing code in Python and then KQL. It is to be noted that not every line of code but necessary excerpts are displayed under the appropriate points.

5.1 METHODOLOGY IN PYTHON

The whole Python approach is segregated into three Jupyter notebooks. The first one deals with finding correlations, in the second one various anomaly detection methods are applied, and finally, in the third one, outlier thresholds are set using Median Absolute Deviation.

5.1.1 *Finding Correlations*

- After importing the necessary packages, the CSV was read and stored in a pandas dataframe and a standard time format '%Y-%m-%d %H:%M:%S.%f' for the timestamp column was set.
- As there is no primary key column to uniquely identify every row, the duplicate rows were removed. And timestamp was set as an index column.
- To get a feel of the data distribution, all the events that occurred on the same day were grouped together (eventCount) and visualized using Plotly.
- Similarly, the process was applied to the events and Retry counts that occurred in the same hour. The retry counter has a value from 1 to 15 unique to every EventID Step, it should be noted that the value 14 doesn't imply 14 retries occurring at a particular moment in time and actually signifies the 14th retry at a particular moment in time. Hence, the occurrence of a retry (no blank value) was considered as 1(singular) rather than the value in the RetryCounter column for calculating the retries per hour. The EventCount and RetryCounter per hour were plotted on the same graph to view if there were any similar patterns.

```
df_hour = df_main.groupby(pd.Grouper(level='timestamp', freq='1H'))['
    eventIDCompact'].agg('count')
df_hour = df_hour.to_frame().reset_index()
df_hour.rename(columns={"eventIDCompact": "eventCount"}, inplace=True)
df_hour.info()

df_retry_hour = df_main.groupby(pd.Grouper(level='timestamp', freq='1H'))[
    'retryCounter'].agg('count')
df_retry_hour = df_retry_hour.to_frame().reset_index()
df_retry_hour.info()
```

Listing 5.1: Grouping Events and Retries per hour and counting their occurences

- In the case of Latency, the data was grouped per EventIDSteps, and the max number per group for latencyInSeconds was found out since it signified the maximum delay for an EventIDStep. Then all the max values in latencyInSeconds for an hour were summed up. This approach presented a picture of the maximum delay unique to every EventIDStep summed per hour and when latencyInSeconds was high. Secondly, all the occurrences in latencyInSeconds were grouped and summed up to get a picture of a timeframe when latencyInSeconds was triggered the most.

```
df_l2 = df_l1.loc[df_l1.groupby('eventIDSteps')['latencyInSeconds'].idxmax
    ()]
```

```
df_l2.sort_values(by='timestamp', inplace = True)
df_l2.set_index('timestamp', inplace=True)
df_hour_latency = df1.groupby(pd.Grouper(level='timestamp', freq='1H'))['
    latencyInSeconds'].agg('sum')
df_hour_latency = df_hour_latency.to_frame().reset_index()
df_hour_latency.rename(columns={"latencyInSeconds": "latencyMaxSum"},
    inplace=True)
df_hour_latency.info()
```

Listing 5.2: Summing Max Latency per EventIDStep per Hour

- Other aggregation functions like Sum, Mean, and Standard Deviation were also applied to retryCounter and latencyInSeconds by grouping them in an hour.
- All the aggregated results of eventCount, retryCounter, and latencyInSeconds were combined in one dataframe and correlation were found out between them using Pearson and Kendall's Tau methods (mentioned in Section 4.2.1 and Section 4.2.2)

```
correlations_hour = inner_merged_hour.corr(method='pearson')

fig, ax = plt.subplots(figsize=(10,10))
sns.heatmap(correlations_hour, vmax=1.0, center=0, fmt='.4f', square=True,
     linewidths=.5, annot=True, cbar_kws={"shrink": .70})
plt.show();
```

Listing 5.3: Finding Correlations between Aggregated Parameters

- The 4 steps mentioned above were then executed again for every minute (instead of an hour). Then the whole process was applied to records belonging to every eventIDCompact to see if the correlation holds separately for them.

```
selectEvent = '99s'
df_event = df_main[df_main['eventIDCompact'] == selectEvent]
df_event.info()
```

Listing 5.4: Filtering rows based on eventIDCompact

5.1.2 *Detecting Anomalies*

- This approach builds on the dataframes containing grouped data for eventCount and retryCounter per hour and per minute built in the previous subsection. Either those dataframes can be imported as CSV's or the steps to make them can be followed again.
- The anomaly detection models mentioned in Section 4.3.5 were applied using the ADTK library (Section 4.4.2) and the results were plotted.

The detectors were PersistAD, AutoregressionAD, and DoubleRollingAggregate transformer.

- As the Persistent model is short-sighted, different window sizes based on the seasonality of the data were tried and a size of 168 (24hours * 7days) was chosen after evaluation of results for hourly data. For per minute 1440 (60*24) was chosen.
- For AutoRegression model based on the seasonality, a step size of 24 and 7 number of steps were chosen. So for predicting the next point in the hourly data and finding if it is anomalous or not the model will consider 7 points in the past which are 24 steps away from each other.
- For DoubleRollingAggregate a window size of 24 was chosen for the left and right rolling windows for the hourly data. And the difference between their mean and median was calculated. For the per minute data, 60 was chosen as the size for windows.

```
eventCount_time_series_hour = validate_series(eventCount_time_series_hour)
persist_eventCount_hour = PersistAD(window=168, c=3.0, side='positive')
anomalies_eventCount_hour = persist_eventCount_hour.fit_detect(
    eventCount_time_series_hour)
plot(eventCount_time_series_hour, anomaly=anomalies_eventCount_hour,
    ts_linewidth=1, ts_markersize=3, anomaly_color='red');

autoregression_ad_hour = AutoregressionAD(n_steps=7, step_size=24, c=3.0)
anomalies_hour = autoregression_ad_hour.fit_detect(
    eventCount_time_series_hour)

eventCount_time_series_hourRolling = validate_series(
    eventCount_time_series_hourRolling)
s_transformed_hourRolling = DoubleRollingAggregate(
    agg="median",
    window=24,
    diff="diff").transform(eventCount_time_series_hourRolling)
s_transformed_hourRolling.rename(columns={'eventCount': '
    eventCountDifference'}, inplace=True)
```

Listing 5.5: Persistent; AutoRegression and Double Rolling Windows

- Local Outlier Factor mentioned in Section 4.3.3 and Isolation Forest mentioned in Section 4.3.2 from Scikit-learn were applied in combination with the OutlierDetector method from ADTK. The methods can be applied to a univariate time-series, but in here they were used with a multivariate time-series of Event Counts and Retry Counter to see if using a combination of both the time-series, deviations that occurred in similar timeframes in them were identified correctly.

```
outlier_detector_multiHour = OutlierDetector(IsolationForest(contamination
    =float(0.05), random_state=0))
anomalies_multiHour = outlier_detector_multiHour.fit_detect(
    inner_merged_hour1)
```

```
plot(inner_merged_hour1, anomaly=anomalies_multiHour, ts_linewidth=1,
    ts_markersize=3, anomaly_color='red', anomaly_alpha=0.3, curve_group='
    all');
```

Listing 5.6: Isolation Forest combined with OutlierDetector(ADTK).

- The above steps were repeated by grouping the data per minute instead of per hour. Then the whole process (per hour and per minute) was applied to records belonging to every eventIDCompact for detecting anomalies in them.

5.1.3 *Setting Outlier Thresholds*

If the data exhibits skewness and has outliers, it's not mean but median that is a better representation of central tendency. Similarly, Median Absolute Deviation (MAD) is a better parameter instead of standard deviation to identify outliers.

- Because of the nature, and seasonality of the data. Data aggregated on an hourly basis was identified as a good candidate to identify thresholds. Data aggregated by days may turn out a bit hazy for decision-making and data aggregated per minute might generate too many false positives.
- The data was aggregated hourly based on eventCount.
- As the data exhibited weekly and hourly seasonality. Mean and median were calculated for every hour based on the day, for example- If in the data Monday occurred 4 times, 4 entries were considered for the mean and median for 09:00 (Monday). The mean and median were plotted together and observed.

```
df_day_hour['dayofweek']=df_day_hour['timestamp'].dt.day_name()
df_day_hour['hour']=df_day_hour['timestamp'].dt.strftime("%H:%M")
df_day_hour_median = df_day_hour.groupby(['dayofweek','hour'])['eventCount
    '].agg('median')
df_day_hour_mean['dayofweek'] = pd.Categorical(df_day_hour_mean.dayofweek,
    categories=weekdays)
df_day_hour_mean['hour'] = pd.Categorical(df_day_hour_mean.hour,categories
    =hours24)
df_day_hour_mean = df_day_hour_mean.sort_values(['dayofweek','hour'])

fig_day_hour_median = px.line(df_day_hour_median,
             x='day_hour',
             y='eventCount',
             title='Time Series for Median of Total Events per Hour',
             color_discrete_sequence=["#ff97ff"]
            )
```

Listing 5.7: Calculating and Plotting Median based on Hourly and Weekly Seasonalities.

- MAD and outlier thresholds were calculated based on the formulas mentioned in Section 4.3.1. The values exceeding the outlier thresholds were identified as anomalies and their difference with the threshold was calculated.

```
s1 = df_threshold.groupby('dayHour')['eventCount'].transform('median')
s2 = df_threshold['eventCount'].sub(s1).abs()
df_threshold['Median'] = s1
df_threshold['MAD'] = s2.groupby(df_threshold['dayHour']).transform('
    median')

df_threshold['outlierThreshold'] = df_threshold['Median'] + 2.5*
    df_threshold['MAD']
df_threshold['Outlier?'] = np.where(df_threshold['eventCount'] >
    df_threshold['outlierThreshold'], 'Yes', 'No')
df_threshold['valueDifference'] = np.where(df_threshold['eventCount'] >
    df_threshold['outlierThreshold'], df_threshold['eventCount'] -
    df_threshold['outlierThreshold'],  float('NaN'))
```

Listing 5.8: Calculating MAD and Outlier Thresholds.

- A severity level of low, medium or high was set based on the IQR (Section 3.2.2) of the differences between the thresholds and outliers. Values equal to or above 3rd Quartile were denoted with high severity, values in between 1st and 3rd were classified as medium, and finally values equal to or below 1st quartile were denoted with low severity.

```
conditions  = [ df_threshold['valueDifference'] >= df_threshold['
    valueDifference'].quantile(0.75), np.logical_and(df_threshold['
    valueDifference'] > df_threshold['valueDifference'].quantile(0.25),
    df_threshold['valueDifference'] < df_threshold['valueDifference'].
    quantile(0.75)),df_threshold['valueDifference'] <= df_threshold['
    valueDifference'].quantile(0.25) ]
choices     = [ "high", 'medium', 'low' ]

df_threshold["Severity"] = np.select(conditions, choices, default=np.nan)
```

Listing 5.9: Assigning severity to outliers.

- The Outliers identified using MAD were then plotted with Actual Retry Counters to see the similarities between them.

```
fig_mad_outliers = px.scatter(df_threshold[df_threshold['Outlier?'] == '
    Yes'],
             x='timestamp',
             y='eventCount',
             title='Time Series of Anomalous Events per Hour',
             color='Severity',
             color_discrete_map={
                "high": "red",
                "medium": "orange",
                "low": "yellow"}
```

```
            )

fig_retry = px.scatter(df_retry_hour,
             x='timestamp',
             y='retryCounter',
             title='Time Series for Retries per Hour'
            )

fig_anomaly_retry_combined = go.Figure(data = fig_mad_outliers.data +
    fig_retry.data)
fig_anomaly_retry_combined.show()
```

Listing 5.10: Plotting Actual Retries and Identifed Outliers.

- The whole process was then repeated for records belonging to different eventIDCompacts.

5.2 METHODOLOGY IN AZURE DATA EXPLORER

This section discusses the approach followed in Azure Data Explorer for outlier detection. Firstly, Exploratory Data Analysis was carried out on the dataset using queries and different visualizations. Secondly, STL decomposition, clustering, and diffpattern techniques were implemented on the dataset to identify and narrow down the outlier instances and causes.

5.2.1 *Exploratory Data Analysis*

- For exploratory data analysis, using the functionalities of the dashboard three separate reports were created. In these reports different types of visualizations can be added and customized (bar, line, pie chart, etc.). These visualizations are made through queries that fetch the results directly from the table stored in the ADX database.
- In the first report, the events were firstly separated and summarized by various features like total event count, eventIDCompact, system and phase.

```
['OTCevents']
| summarize EventCount = count() by eventIDCompact
| sort by EventCount

['OTCevents']
| summarize EventCount = count() by eventPhase
| sort by EventCount
```

Listing 5.11: Calculating total event counts by eventIDCompact and phase.

- Using the make-series function of KQL different time-series were made for eventCount, seggregated by various features like system, Phase and eventIDCompact for the hourly data.

```
let interval = 1h;
let stime = datetime(2023-07-31);
let etime = datetime(2023-08-30);
['OTCevents']
| make-series eventCount=count() on timestamp from stime to etime step
    interval
| render timechart

['OTCevents']
| make-series eventCount=count() on timestamp from stime to etime step
    interval by system
| render timechart
```

Listing 5.12: Time series of eventCount and eventCount segregated by system.

- In the second report which is related to retries, firstly different summary information was displayed using queries by total count, number of retries, and eventIDCompact.

```
['OTCevents']
| where retryCounter > 0
| summarize RetryCounts = count() by eventIDCompact
| sort by RetryCounts

['OTCevents']
| summarize Retries = count() by retryCounter
```

Listing 5.13: Number of Retries by eventIDCompact and retryCounter value.

- Then the top counts of retries per hour and per minute were displayed grouped by eventIDCompact, and eventPhase.

```
let interval = 1m;
let stime = datetime(2023-07-31);
let etime = datetime(2023-08-30);
['OTCevents']
| where timestamp  between (stime..etime)
| where retryCounter > 0
| summarize TotalRetries = count() by bin(timestamp, interval),
    eventIDCompact, eventPhase
| sort by TotalRetries desc
| top 30 by TotalRetries
```

Listing 5.14: Top 30 Retries per minute by eventIDCompact and eventPhase.

- Finally, like in the first report, using the make-series function of KQL different time-series were made for Retries, segregated by various features like system, Phase and eventIDCompact for the hourly data.
- In the third report that is related to latency, a similar approach as retry counters to generate various visualizations was followed.

```
['OTCevents']
| where latencyInSeconds > 0
| count

let interval = 1h;
let stime = datetime(2023-07-31);
let etime = datetime(2023-08-30);
['OTCevents']
| summarize TotalLatency = sum(latencyInSeconds) by eventIDCompact
| sort by TotalLatency
```

Listing 5.15: Events with Latency and Latency by eventIDCompact.

5.2.2 *STL Decomposition and Clustering*

- There are functions in KQL that allow a user to perform STL decomposition on a time-series out of the box.
- Using the principles of STL (Section 4.3.4) firstly the time-series is decomposed into its seasonal, trend, and residual components. Then using the series_decompose_anomalies function with proper hyperparameters, the anomalies are identified.

```
['OTCevents']
| make-series eventCount=count() on timestamp from stime to etime step
    interval
| extend (5_baseline, 2_seasonal, 3_trend, 4_residual) = series_decompose(
    eventCount, 168, 'linefit')
| render timechart with(title='STL Decomposition', ysplit=panels)
['OTCevents']
| make-series eventCount=count() on timestamp from stime to etime step
    interval
| extend (anomalies, score, baseline) = series_decompose_anomalies(
    eventCount, 1.5, 168, 'linefit')
| render anomalychart with(anomalycolumns=anomalies)
```

Listing 5.16: STL decomposition with KQL

- After finding out anonymous timeperiods, these periods are used to find top clusters of features responsible for high count. The function used in KQL for clustering is based on the Seed-expand algorithm [5]. Also diffpattern function is used to identify the features that are different between normal and anomalous time periods.

```
['OTCevents']
| where timestamp between (stime..etime)
| evaluate autocluster()
| project-away timestamp, eventIDSteps
['OTCevents']
| where (timestamp between (stime..etime)) or (timestamp between (s1time..
    e1time))
| extend AB = iff(timestamp>splitime,'Anomaly','Baseline')
| evaluate diffpatterns(AB,'Anomaly','Baseline')
```

Listing 5.17: Clustering and DiffPatterns with anomalous time period in KQL

6 DISCUSSION

This chapter discusses in detail the results of the methodologies applied in Python and ADX.

6.1 RESULTS IN PYTHON

The results of the three notebooks for finding correlations, anomalies, and thresholds are discussed in this section.

6.1.1 *Correlations*

- While finding out correlations on the per hour data, as shown in Figure 6.1 the features eventCount, retryCounter and latencyInSeconds(Sum) were found strongly and moderately(for latencyInSeconds) correlated with each other with both Pearson and Kendall's Tau correlation techniques. Other parameters based on aggregation of retryCounter and latencyInSeconds were also considered while finding out correlations. (For example- Mean of retryCounter and latencyInSeconds per hour). The correlations held for 'per minute' data but were not that strong between eventCount and retryCounter, latencyInSeconds(Sum).
- It is to be noted that the data (hourly) is not normally distributed but various aggregated parameters showed monotonic behavior when plotted against each other especially eventCount, retryCounter, and latencyInSeconds(Sum)(partly). Hence, the correlation between eventCount and retryCounter held when measured with Kendall's Tau. It is also to be noted in Kendall's Tau case that the correlation value is not as strong for latencyInSeconds(Sum) with eventCount and retryCounter as shown in Figure 6.1(b). The reason being there are events 16all, 30all and 97all that had latency without retries, also there are some particular cases of events where the latency was very high in individual cases affecting the whole correlation with eventCount and retryCounter and bringing it down from strong to moderate.
- There were three kinds of events based on eventIDCompact: events with no retries **(10, 31, 87, 99s)**, events with very less retries **(9, 11, 20, 95)**, and events with a significant number of retries **(16all, 19, 21, 22, 30all, 32, 36, 39, 86, 89, 91, 94, 97all, 99)** when compared to their respective total count.
- For events that had no retries triggered like Event 10, there were no retries and no latency present and hence there was no correlation

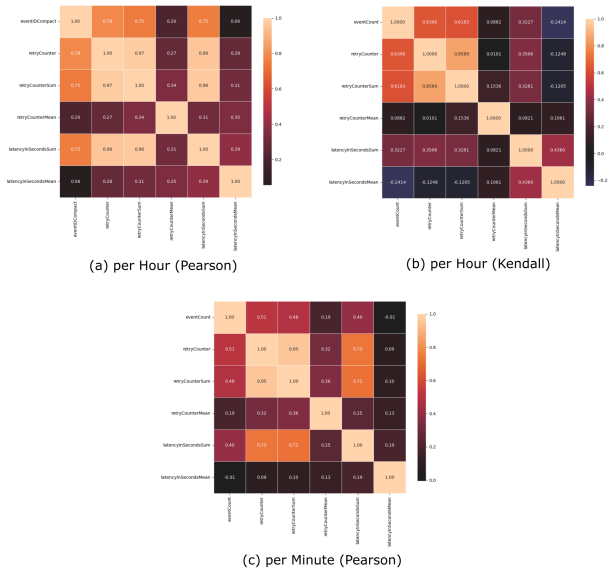


Figure 6.1: Correlations on EventCount, RetryCounter, and latencyInSeconds(Sum) along with other aggregated Parameters.

between eventCount, retryCounter, and latencyInSeconds(Sum). For events where there were fewer retries like Event 11 and Event 20, there was significantly no correlation (very less) between eventCount and retryCounter, latencyInSeconds(Sum). But there was a correlation present between retryCounter and latencyInSeconds(Sum). Finally, in events where there were a significant number of retries like Event 21, Event 36, and Event 97all, there was a strong correlation between eventCount, retryCounter, and latencyInSeconds(Sum). (All the correlation diagrams of the six events mentioned are depicted in Figure 6.2)

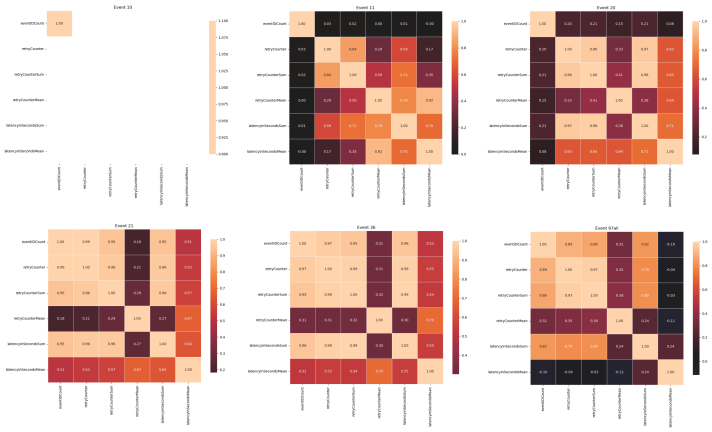
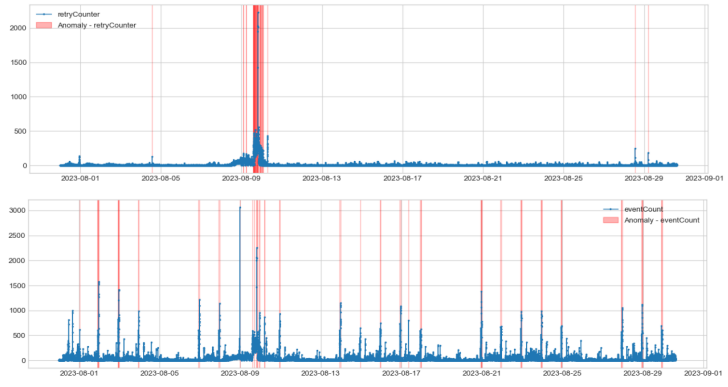


Figure 6.2: Correlations on eventCount, retryCounter, and latencyInSeconds(Sum) along with other aggregated Parameters.

6.1.2 *Anomalies*

- Finding out anomalies was based on the fact that there was a good correlation present between eventCount and retryCounter. Using two methods i.e. Persistent model and Outlier Detection method from ADTK combined with Isolation forest on the hourly data, it was found that it is possible to identify outlier cases that show similar kind of behavior for both the time-series.
- For both (eventCount and retryCounter) the time-series, outliers using Persistent model were compared side-by-side. While Isolation Forest was applied on the multivariate time series by combining the two.
- As seen in the Figure 6.3 (a), the anomalous cases of retries were identified correctly along with the identification of anomalous eventCount, and when overlapped with each other apart from a few false positives the anomalies identified in eventCount time-series aligned with the high retryCounter.

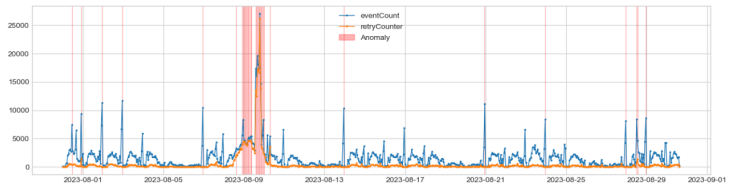
(a) per Hour (Persistent Model)



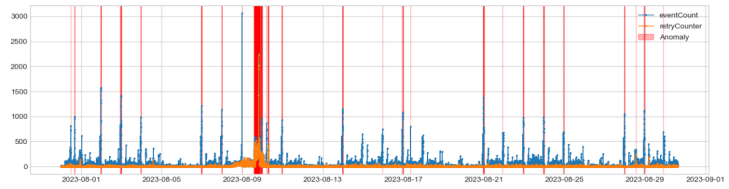
(b) per Minute (Persistent Model)

Figure 6.3: Persistent Model for EventCount, RetryCounter per Hour and Minute

- For the per minute data the results were not that good (Figure 6.3 (b)) for eventCount time-series as there were a lot of false positives and when the hyper-parameters were adjusted to decrease them, the timestamps which should have been kept as anomalies when compared with retryCounter time-series were identified as false-negatives very early-on in the process.
- From the Figure 6.4 same can be said about multivariate Isolation Forest consisting of eventCount and retryCounter. For the hourly data, the instances where the retry counter was high were identified correctly. But on the minutely data, there were a lot of instances marked as anomalies even when the retry counter was low solely based on peaks of eventCount.



(a) Multivariate Isolation Forest (per Hour)



(b) Multivariate Isolation Forest (per Minute)

Figure 6.4: Multivariate Isolation Forest for per hour and per minute data.

- In Isolation Forest multi-variate analysis latencyInSeconds(Sum) can also be considered as it is moderately correlated with the other two parameters.
- For Autoregression and Double Rolling windows only eventCount was looked at because of the possibility of identifying where anomalous retries would occur by only looking at the time series of eventCount.
- Using Auto-Regression with proper window and step size along with tuning hyperparameters a lot of false-positives and false-negatives were marked even in the hourly data in the eventCount time-series when compared to the actual high retries as seen in Figure 6.5. The situation got worse in the per minute data.
- Using Double-rolling windows on the hourly data did produce average results (Figure 6.6 (a)), however compared to the other previous results this approach presented better results in the 'per minute' data

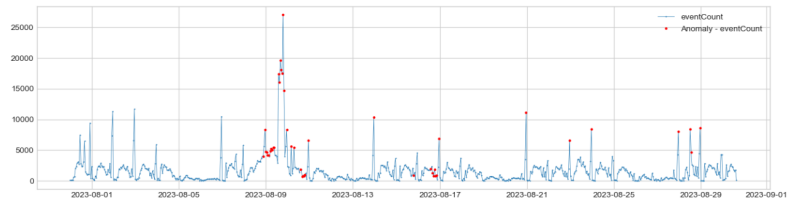
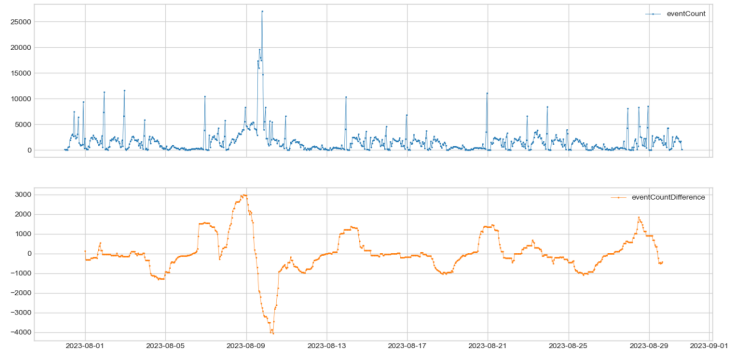
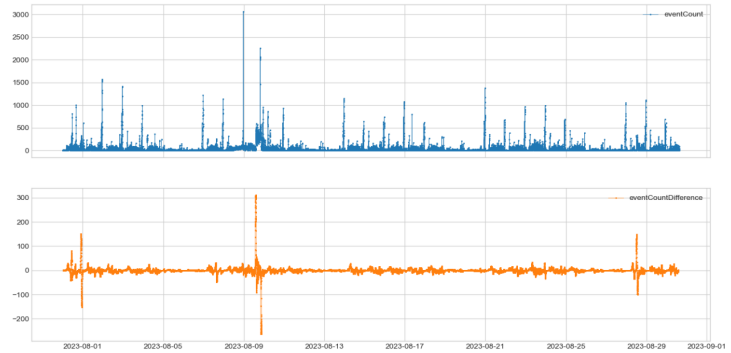


Figure 6.5: Autoregression for anomaly detection for eventCount (per hour).

(Figure 6.6 (b)). There were high counts of retries around July 31 22:53, Aug 9 (almost all day), and Aug 28 12:56. And these were depicted correctly through steep peaks and troughs in the visualization for double rolling windows on eventCount data.



(a) Double Rolling Windows (per Hour)

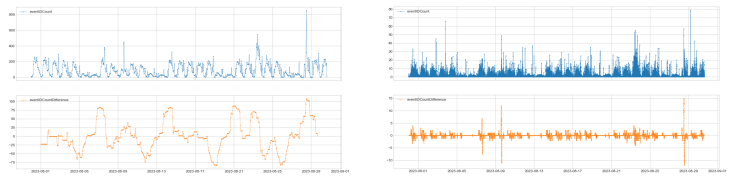


(b) Double Rolling Windows (per Minute)

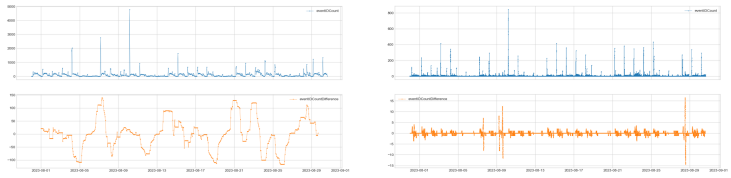
Figure 6.6: Double rolling windows for eventCount (per hour and per minute).

- Since double rolling windows seem to produce good results in identifying outliers visually especially for per minute cases. Their results for eventIDCompact 10, 11 and 21 are depicted in the Figure 6.7. These 3 events belong to three separate classes of events with no, less, and high retries. While eventCount for Event 10 had no retries, it's hourly seismic activity didn't show any anomalous behaviour. Although there were some peaks in the per minute data, which if needed can be investi-

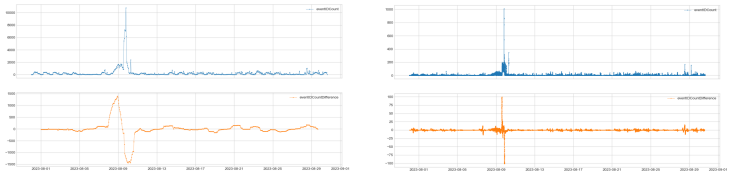
gated further, but as there were no retries there was no point. For Event 11 with less retries the hourly graph peaks were a little bit more steep as compared to Event 10 but nothing conclusive can be understood from it. However, in the per minute data steep peaks and troughs did align with the actual retries. For Event 21 with high number of retries, the conclusion can be drawn where the high retries occurred both from the per hour and per minute graphs for eventCount.



(a) Double Rolling Windows for Event 10 (per Hour and Minute)



(a) Double Rolling Windows for Event 11 (per Hour and Minute)



(a) Double Rolling Windows for Event 21 (per Hour and Minute)

Figure 6.7: Double rolling windows for Events 10, 11 and 21 (per hour and per minute)

- It is to be noted that in the initial version of the dataset Local Outlier Factor was also tried. But the understanding of the complete dataset made it clear that the anomalies were global in nature and Local Outlier Factor thrives in detecting local anomalies. Hence, it was not taken ahead in the final dataset.

6.1.3 *Thresholds*

- While calculating the thresholds firstly mean and median per hour were calculated based on the seasonality of the data (weekly and hour of the day). The mean/median of eventCount was high on the morning and afternoon part of weekdays but eventually got low from the night of Friday till the night of Sunday, which is an expected behavior. Also, the difference between mean and median can be seen in the Figure 6.8 where there was skewness and the presence of anomalies.

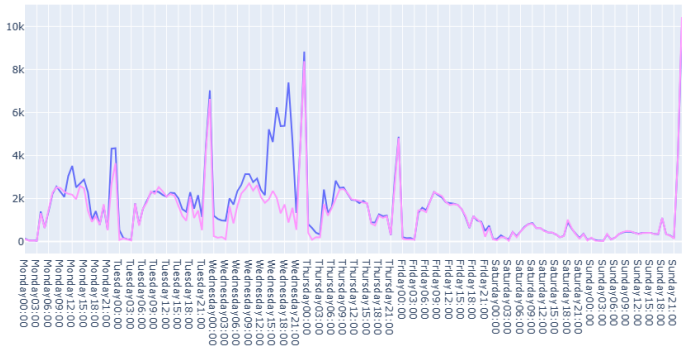


Figure 6.8: Mean, Median of Total Events per Day and Hour.

- However, from Sunday to Thursday in the 23:00 hour the mean/median of eventCount was the highest going against the usual behavior, especially on Sunday 23:00. The sole event which was responsible for shifting the mean/median at an hour of 23:00 was Event 20 (Figure 6.9), it is also to be noted that the retries when compared to eventCount were very less for Event 20 at 23:00. So maybe, it was some business requirement or a set process being executed at this hour.

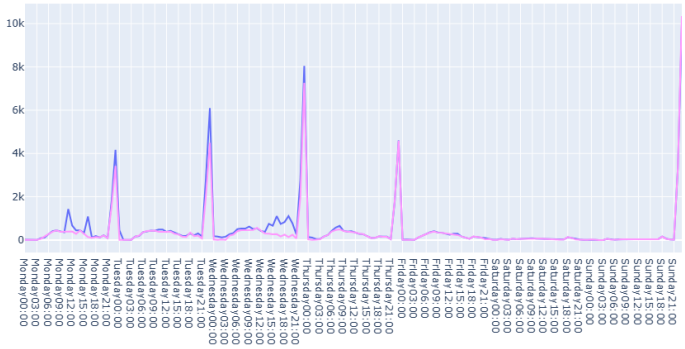
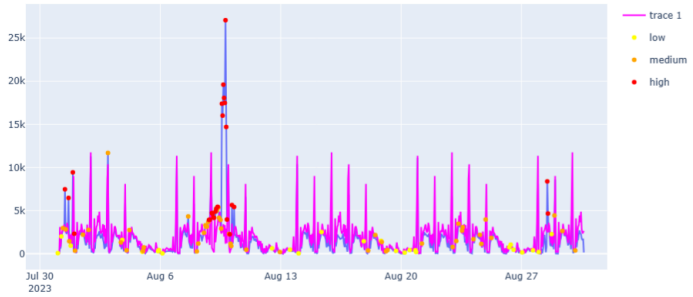
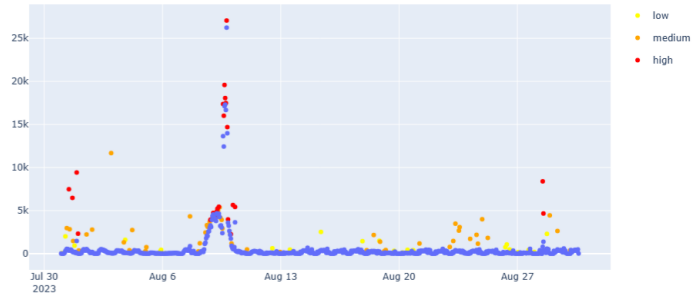


Figure 6.9: Mean, Median of Event 20 per Day and Hour.

- While plotting the MAD thresholds along with actual eventCounts and the severity level (Figure 6.10 (a)), and overlaying identified anomalies with actual retries (Figure 6.10 (b)) it was found that indeed there were similarities between high retry count and high severity anomalies. They did not completely overlay each other but the time frames were similar i.e. July 31st around 23:00, Aug 8th 12:00 - Aug 10th 06:00, and Aug 28th around 13:00.
- While looking at the three classes of the events (no, less and high retries) let's take the events which were mentioned previously i.e Events



(a) MAD threshold, anomalies with severity and actual eventCount in blue.



(b) Overlaying anomalies and actual Retries.

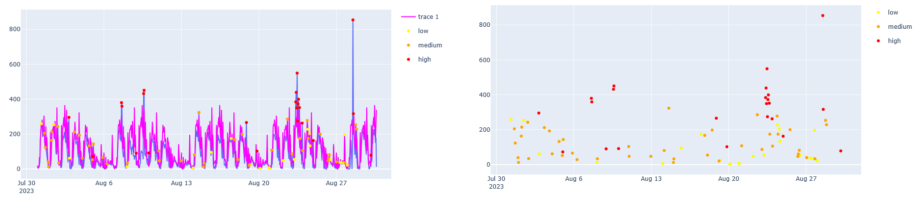
Figure 6.10: MAD Thresholds, Anomalies with Severity, eventCount (per hour) and RetryCount (per hour)

10, 11, and 21. For Event 10 (Figure 6.11 (a)), the threshold boundary is around 350 mark at its highest points and the high severity outliers identified are not very far off except for 1 or 2 instances. Since there are no retries for Event 10, there is no overlaying with the actual retryCounter. But the cluster of red outliers for August 24-25 and a very high singular incident at August 28th can be investigated further if needed.

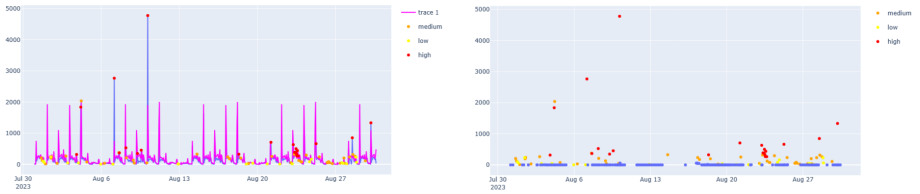
- For Event 11 with fewer retries (Figure 6.11 (b)), there are also fewer standalone high severity cases. The actual retry counter is around 1 or 2 for a given hour which usually lies around the similar timeframes as identified outliers.

- For Event 21 (Figure 6.11 (c)), the outliers are clearly visible in between August 8th and August 10th. As the eventCount is abnormally higher than usual and when viewed together with the retry counts, they kind

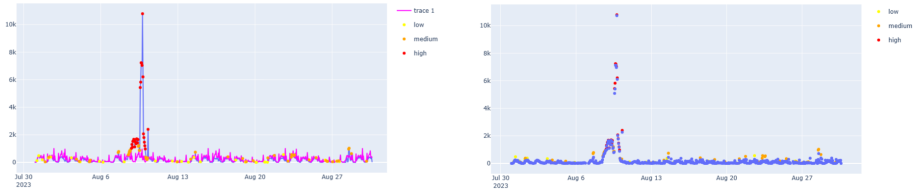
of perfectly overlay each other indicating problems during that time-frame.



(a) Event 10



(b) Event 11



(c) Event 21

Figure 6.11: MAD Thresholds, Anomalies with Severity, eventCount (per hour) and RetryCount (per hour) for different Events

- There is another technique of Time differencing currently being explored, in it the difference between consecutive steps of time-series based on a single eventIDCount is calculated. This difference shows some seasonality. Then, either moving average of this difference or MAD threshold based on the seasonality of the difference is calculated. Generally a lot of events occuring frequently indicates a problem associated with retries in this use-case. So, let's say when a 1000 consecutive events are below the threshold of this set time difference then an alert is generated.

6.2 RESULTS IN ADX

This section discusses the results achieved using the Exploratory Data Analysis method along with STL and Correlations.

6.2.1 *Exploratory Data Analysis*

- As seen in Figure 6.12 Event 20,21 and 94 are the top three eventIDCompacts with the most events making up for 50% of the total events for the whole time period in the dataset. For systems being targeted it is System 1 at the top along with Order for the phase. When the time series is made for the eventCount, the time period around 9th of August stands out with a very high count. When segregated by various features, the anomalous behavior can be seen because of the 'out of the blue' high count of events pertaining to System 2 (highlighted in orange in the 'Events per Hour by System' tile), Contract (highlighted in blue in the 'Events per Hour by Phase' tile), and Event 21 (highlighted in pink in the 'Events per Hour by EventIDs' tile).

Figure 6.12: Dashboard in ADX for eventCount

- In the Figure 6.13, the number of retries allowed decreased exponentially (1-15) indicating a good system behavior (highlighted in Number of events per Retry Counter (1-15) tile). The events 21, 20, 94, 39 and 16all made for about more than 90% of total retries based on eventIDCompact (highlighted in the tile Count of Retries by Event IDs), there were a small number of retries here and there but a very high number

of them came around the August 9th time period. The top occurrences of retry counter per hour and minute also showed different hours on August 9th and different minutes on August 9th around 19:30 timeframe with most of the entries belonging to Event 21. Finally, when the time series for retries were segregated by the features almost all of the entries came from System 2, for phase it was mostly the Contract phase and partly the Order phase. When viewed by eventIDCompact, it was Event 20 and 21 majorly responsible for the retries.

Figure 6.13: Dashboard in ADX for Retries

- When looking at the dashboard for latency (Figure 6.14), most of the latency came from 21, 16all, 20, 94, 39, 97all, and apart from some minor high instances, the prominent high count came in the time frame around August 9th. The top occurrences also belonged to 16all and 21 when seen per hour and minute, for 16all it was August 24th around 22:00 time frame while for 21 it was August 9th.

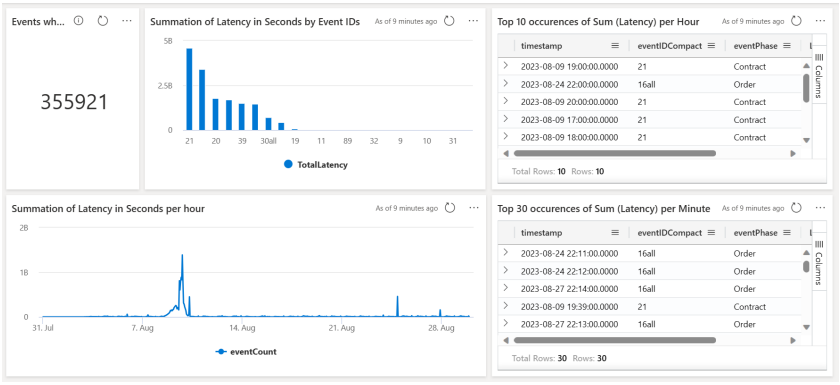


Figure 6.14: Dashboard in ADX for Latency

6.2.2 *STL Decomposition and Clustering*

As seen in the dashboard for STL decomposition and anomalies identified using it (Figure 6.15), they are concentrated mainly during the August 9th period (Identified Anomalies using STL tile). There are some others concentrated in the 2nd August and 28th August period. These are coming basically from the residual component (dark blue) in the first tile. In the clustering tile, the anomalous time period around August 9th was observed for finding top clusters of features contributing to the high count, and the events having Contract as the common Phase and System2 as the common system were identified as the main cluster for the high count. Pattern differencing was also done to compare an anomalous and non-anomalous time window and the main difference was 0 latency arising from the non-anomalous time window and System 2 as the main cause for the anomalous time window. Using segregation and removing all the events containing System 2 (orange line, last tile), the time series indeed shows a normal behavior (blue line, last tile).

6.3 SUMMARY

Using methodology in Python, a strong correlation was established between the features eventCount and retryCounter. Also, the timeframe between August 8th noon and August 10th morning especially August 9th evening was found to be anomalous in terms of really high retries. The events that showed abnormal behavior during this time were 21 (major) and 20, 94 (minor). Through the use of visualizations and clustering in ADX, System 2 was found to be the primary reason, and the Contract phase was the secondary reason for the anomalies. As seen in the table constructed in PowerBI (Figure 6.16), though there were events occurring at high count for all three systems, almost all of the retries came from System 2.

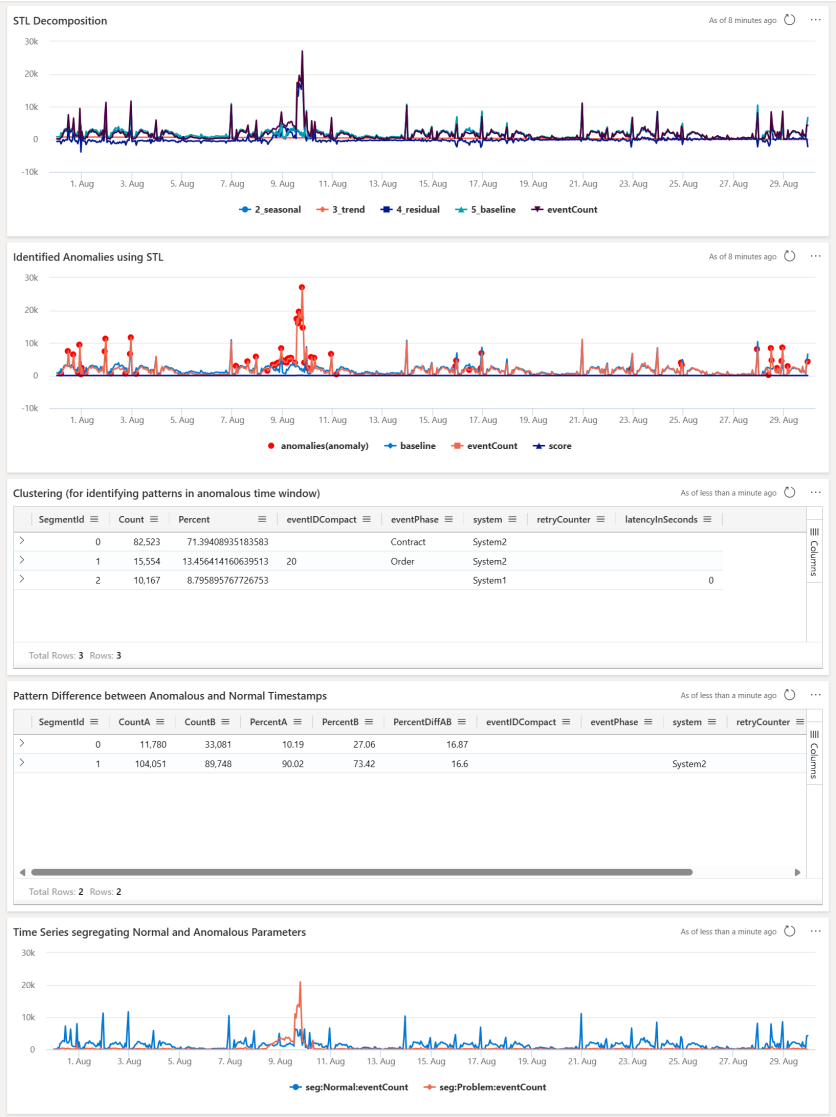


Figure 6.15: Dashboard in ADX for Anomalies using STL, and Clustering

system eventIDCompact	System1 Count of retryCounter	 Count of eventIDCompact	System2 Count of retryCounter	 Count of eventIDCompact	System3 Count of retryCounter	 Count of eventIDCompact	Total Count of retryCounter	 Count of eventIDCompact
21		60670	138964	138964			**138964**	**199634**
20	127	60036	52490	52490		250168	**52617**	**362694**
94		66619	46940	46940			**46940**	**113559**
39		44976	36083	36084			**36083**	**81060**
16all		18826	26800	48018			**26800**	**66844**
36		10281	10336	10336			**10336**	**20617**
97all		15920	8051	11853			**8051**	**27773**
22		43612	7118	7118			**7118**	**50730**
19		4153	4352	4352			**4352**	**8505**
99		3189	1199	1199			**1199**	**4388**
30all		2151	1004	3425			**1004**	**5576**
11		65303	658	658		30356	**658**	**96317**
86		783	430	430			**430**	**1213**
89		183	226	226			**226**	**409**
32		450	106	106			**106**	**556**
91		165	101	101			**101**	**266**
9		60567	4	4			**4**	**60571**
95		17655	2	2			**2**	**17657**
10		65268						**65268**
31		2053						**2053**
87		11						**11**
99s				34216				**34216**
Total	**127**	**542871**	**334864**	**396522**	**280524**		**334991**	**1219917**

Figure 6.16: PowerBI table for total events and retries by eventIDCompact and System

A dashboard (under construction) will present the stakeholders with an easy-to-understand summarized view of the data. In the Figure 6.17 a draft version made on the initial dataset can be seen where the user can see different visualizations and key figures under multiple reports. The user in this dashboard can filter data according to EventIDs, Source, and Target.

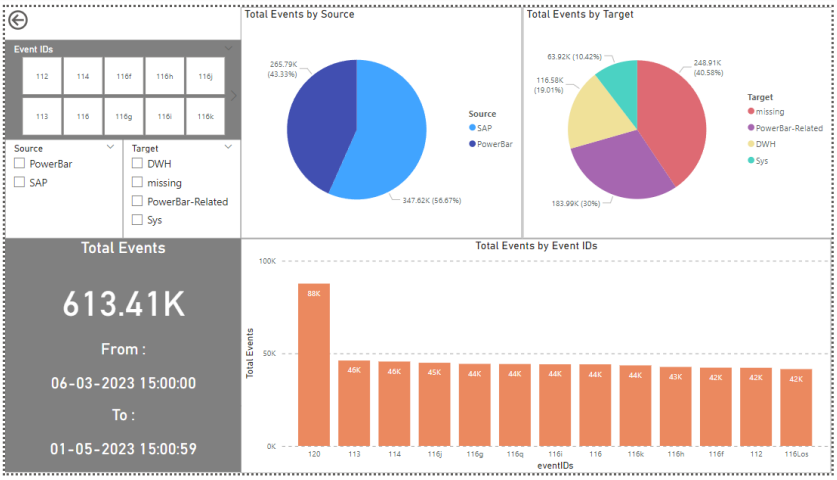


Figure 6.17: PowerBI report summarizing key figures made on the initial dataset.

7 CONCLUSION AND FUTURE SCOPE

The conclusion of the work done in the thesis is presented in this section. By recapitulating the thesis goal and research question, the results are presented. Finally, what can be improved and built upon the existing approach is discussed.

7.1 CONCLUSION

In this thesis, while discussing the real use cases (Chapter 2) and various techniques (Section 3.2), it was found out that no method for anomaly detection is foolproof and the technique being used depends on the quality of the data, characteristics of the features and the time-frame for which the data is available. It is entirely possible that the problem of identifying anomalies in a use case is being solved by using IQR instead of an LSTM Autoencoder.

For the case study, as the time series was unevenly sampled, to make it evenly sampled and get better idea of the parameters being measured. It was grouped per day, hour, and minute and aggregations like count and sum were performed on the features. Using this approach, the hourly group was found to be the most useful for setting thresholds and finding anomalies as an hourly seasonality based on the day of the week was present in the data. The grouping-per-minute approach is decent but is too fine-grained for setting thresholds, the opposite can be said for the grouping-per-day approach.

While finding out correlations, Kendall's Tau showed promising results when compared to the visualizations of various features. As it is non-parametric and does not depend on the type of distribution, using it the features that showed monotonic relationship showed strong correlation (i.e. eventCount and retryCounter) and the parameters where the relationship was not that strong indeed were not perfectly monotonic (i.e. latencyInSeconds with eventCount).

Various techniques were tried for anomaly detection, out of which multivariate Isolation forest was decent enough in the hourly approach. However, it is important to know beforehand what percentage of the data should be considered outliers in it, which can be problematic in streaming data or when the behavior of the data is not known. Exactly for this reason the double rolling window method was promising even for real-time data, it also performed better than other methods in the per-minute approach.

Median and Median Absolute Deviation is a proven and safe approach to set thresholds, that is also seen in various business cases where companies use them in one form or another incorporated in their outlier detection frameworks. This was also seen in this thesis when the high-priority outliers identified using it were in line with the actual retries.

In ADX, the approach of visualizing and Exploratory Data Analysis is on the mark as dashboards can be made in it similar to PowerBI directly using the SQL-like queries on the data stored in tables. Also, the in-built functions provided in it for time-series analysis like STL decomposition and clustering make it convenient to identify outliers and find patterns of the features that are responsible for these outliers. Comparing ADX with Python, when it strictly comes to time series analysis and anomaly detection.

1. ADX has built-in capabilities for time series and root cause analysis and a less steep learning curve compared to Python. Moreover, its excellent dashboarding capabilities makes it a lot easier to conduct exploratory data analysis and customize visualizations which is not easier to do with Python using Matplotlib and even Plotly.

2. The time required from ingesting the data to presenting the results using visualizations is less when compared with the same approach in Python. Additionally, there is a need for a BI tool to view the results by Python but all of this can be done in one place inside ADX. However, the functionalities provided by ADX are limited whether it be visualization capabilities compared to PowerBI or the flexibility and range of methods provided by Python libraries for analysis. Moreover, the data can only be read using KQL and cannot be added, or modified.

3. Lastly, there is a cost factor associated with ADX which is not the case with Python. As with all the cloud-provider services, it is not on the cheaper side. For this thesis, all the entry-level options while configuring the virtual machines, storage, etc. were used. The data of around 80 mb was ingested 3-4 times and retained for 2 weeks. All of these operations in ADX costed around 40 euros. But if an organization is already using various services provided by Azure, it is recommended to add ADX to that stack.

Reiterating the research question *"Is it possible to detect and visualize so far unknown, valid, explainable and actionable anomalies in event communication between multi cloud-based enterprise applications?"* and the answer is Yes to a plausible degree. By following an approach consisting of finding correlations, exploratory data analysis, and trying out various anomaly detection techniques. It is indeed possible to detect unknown, valid, explainable and actionable anomalies. The approach used in this thesis was investigative in nature and using it the reasons were pinpointed only from the dataset, without the extra business domain knowledge.

7.2 FUTURE SCOPE

Carrying over the research in this thesis, the methodologies can be tested on a large dataset consisting of other months. For identifying thresholds the data in a calendar year should be used as it will contain multiple seasonalities (monthly or maybe season i.e. winter, summer, etc. along with the day of the week and hour of the day), this will make the thresholds more foolproof and they can be applied to real-time data. Also, methods where training and testing are required like LSTM Autoencoders a large dataset comes in handy, where a chunk of data is utilized to train the model and the rest to test it.

There are also two other features apart from retryCounter and latencyinSeconds, when available their correlation should be checked with other parameters to draw relations.

ADX's capabilities can be explored further. For example, running Python functions inside it is possible using sandbox images. How much of the anomaly detection libraries and functions are able to run inside it can be checked, along with the cost incurred.

By identifying anomalies and removing the abnormal ones, the clean data then can be used for forecasting using models like ARIMA, SARIMA, etc. The forecasting can be done for the whole dataset as one time-series or separate time-series segregated by features (i.e. system, eventIDCompact, phase).

APPENDIX

The complete code for this thesis work including Jupyter Notebooks (Python), ADX scripts, pictures of matplotlib and plotly visualizations and PowerBI reports are available in the Github repository :

https://github.com/faizusmani/Anomaly-Detection-in-Event-Communication.git

BIBLIOGRAPHY

[1] M. Alam. *Z-score for anomaly detection*. [Online; posted on 3-Sep-2020]. URL: https://towardsdatascience.com/z-score-for-anomaly-detection-d98b0006f510.

[2] *Anomaly Detection Toolkit (ADTK)*. [Online]. URL: https://adtk.readthedocs.io/en/stable/index.html.

[3] *Azure Data Explorer documentation*. URL: https://learn.microsoft.com/en-us/azure/data-explorer/.

[4] *Azure Data Explorer pricing*. URL: https://azure.microsoft.com/en-us/pricing/details/data-explorer/.

[5] R. B. Ofer., A. Eldar., A. Shalev., and Y. S. Resheff. "Algorithms for Telemetry Data Mining using Discrete Attributes." In: *Proceedings of the 6th International Conference on Pattern Recognition Applications and Methods - ICPRAM*. INSTICC. SciTePress, 2017, pp. 309–317. ISBN: 978-989-758-222-6. DOI: 10.5220/0006117903090317.

[6] W. Badr. *Auto-Encoder: What Is It? And What Is It Used For? (Part 1)*. [Online; posted on 22-Apr-2019]. URL: https://towardsdatascience.com/auto-encoder-what-is-it-and-what-is-it-used-for-part-1-3e5c6f017726.

[7] A. Bajaj. *Anomaly Detection in Time Series*. [Online; posted on 22-Aug-2023]. URL: https://neptune.ai/blog/anomaly-detection-in-time-series#:~:text=The%20procedure%20for%20detecting%20anomalies,based%20on%20that%20difference%20threshold..

[8] F. Bell. *Identifying Outages with Argos, Uber Engineering's Real-Time Monitoring and Root-Cause Exploration Tool*. [Online; posted on 24-Nov-2015]. URL: https://www.uber.com/en-DE/blog/argos-real-time-alerts/.

[9] T. Bex. *How to Detect Random Walk and White Noise in Time Series Forecasting*. [Online; posted on 16-Jul-2021]. URL: https://terpconnect.umd.edu/~toh/spectrum/AppendixO.html.

[10] M. M. Breunig, H.-Peter Kriegel, R. T. Ng, and J. Sander. "LOF: Identifying Density-Based Local Outliers." In: *SIGMOD Rec.* 29.2 (2000), 93–104. ISSN: 0163-5808. DOI: 10.1145/335191.335388. URL: https://doi.org/10.1145/335191.335388.

[11] S. Chaudhary. *Why "1.5" in IQR Method of Outlier Detection?* [Online; posted on 28-Sep-2019]. URL: https://towardsdatascience.com/why-1-5-in-iqr-method-of-outlier-detection-5d07fdc82097.

[12] K. Chen and B. Overstreet. *Building a real-time anomaly detection system for time series at Pinterest*. [Online; posted on 30-Jul-2019]. URL: https://medium.com/pinterest-engineering/building-a-real-time-anomaly-detection-system-for-time-series-at-pinterest-a833e6856ddd.

[13] R. Cleveland, W. Cleveland, J. McRae, and I. Terpenning. "STL: A Seasonal-Trend Decomposition Procedure Based on Loess." In: *Journal of Official Statistics* 6 (Jan. 1990), pp. 3–33.

[14] I. Dabbura. *K-means Clustering: Algorithm, Applications, Evaluation Methods, and Drawbacks*. [Online; posted on 17-Sep-2018]. URL: https://towardsdatascience.com/k-means-clustering-algorithm-applications-evaluation-methods-and-drawbacks-aa03e644b48a.

[15] S. Date. *How To Isolate Trend, Seasonality And Noise From A Time Series*. [Online]. URL: https://timeseriesreasoning.com/contents/time-series-decomposition/.

[16] De.Mathworks.Com. *Rolling-Window Analysis of Time-Series Models*. [Online; posted on 24-May-2019]. URL: https://de.mathworks.com/help/econ/rolling-window-estimation-of-state-space-models.html..

[17] Data Science Digest. *What's the difference between novelty detection and outlier detection?* [Online; posted on 15-Jun-2021]. URL: https://digestize.medium.com/whats-the-difference-between-novelty-detection-and-outlier-detection-bc22dd6de7b0#:~:text=In%20outlier%20detection%2C%20you%20dataset,semi%2Dsupervised%20anomaly%20detection%20algorithm..

[18] J. Paulo Figueira. *LOESS*. [Online; posted on 24-May-2019]. URL: https://towardsdatascience.com/loess-373d43b03564.

[19] V. Kumar G. *Anomaly detection using Isolation Forest – A Complete Guide*. [Online; posted on 26-Jul-2021]. URL: https://www.analyticsvidhya.com/blog/2021/07/anomaly-detection-using-isolation-forest-a-complete-guide/.

[20] V. Kumar G. *Statistical Tests to Check Stationarity in Time Series*. [Online; posted on 16-Jun-2021]. URL: https://www.analyticsvidhya.com/blog/2021/06/statistical-tests-to-check-stationarity-in-time-series-part-1/#How_to_Check_Stationarity?.

[21] J. Gamble. *A Beginner's Guide to Anomaly Detection*. [Online; posted on 17-Nov-2020]. URL: https://www.verytechnology.com/iot-insights/a-beginners-guide-to-anomaly-detection.

[22] R. Gandhi. *Support Vector Machine — Introduction to Machine Learning Algorithms*. [Online; posted on 7-Jun-2018]. URL: https://towardsdatascience.com/support-vector-machine-introduction-to-machine-learning-algorithms-934a444fca47.

[23] S. Garg. *Algorithm selection for Anomaly Detection*. [Online; posted on 2-Jun-2020]. URL: https://medium.com/analytics-vidhya/algorithm-selection-for-anomaly-detection-ef193fd0d6d1.

[24] GeakMinds. *Examples of Anomaly Detection in Major Industries*. [Online; posted on 11-Jan-2022]. URL: https://geakminds.com/examples-of-anomaly-detection-in-major-industries/.

[25] A. Gharakhanian. *Anomaly Detection: Its Real-Life Uses and the Latest Advances*. [Online; posted on 8-Nov-2022]. URL: https://aithority.com/machine-learning/anomaly-detection-its-real-life-uses-and-the-latest-advances/.

[26] S. Glen. *Mahalanobis Distance: Simple Definition, Examples*. [Online]. URL: https://www.statisticshowto.com/mahalanobis-distance/.

[27] S. Hariri, M. Kind, Carrasco, and Robert J. Brunner. "Extended Isolation Forest." In: *IEEE Transactions on Knowledge and Data Engineering* 33.4 (2021), pp. 1479–1489. DOI: 10.1109/TKDE.2019.2947676.

[28] E. Meeks J. Wong C. Colburn and S. Vedaraman. *RAD — Outlier Detection on Big Data*. [Online; posted on 19-Feb-2015]. URL: https://netflixtechblog.com/rad-outlier-detection-on-big-data-d6b0494371cc.

[29] Singhchawla Arunee and Myerscough Jason. *Scalable Data Analytics with Azure Data Explorer: Modern Ways to Query, Analyze, and Perform Real-time Data Analysis on Large Volumes of Data*. Birmingham, UK: Packt Publishing, 2022.

[30] V. Jayaswal. *Local Outlier Factor (LOF) — Algorithm for outlier identification*. [Online; posted on 31-Aug-2020]. URL: https://towardsdatascience.com/local-outlier-factor-lof-algorithm-for-outlier-identification-8efb887d9843.

[31] V. Julius, T. Povilas, and M. Jurgita. "Unsupervised marine vessel trajectory prediction using LSTM network and wild bootstrapping techniques." In: *Nonlinear Analysis: Modelling and Control* 26 (July 2021), pp. 718–737. DOI: 10.15388/namc.2021.26.23056.

[32] J. Kaur. *Anomaly Detection in Cyber Network Security | A Quick Guide*. [Online; posted on 21-Nov-2022]. URL: https://www.xenonstack.com/insights/cyber-network-security.

[33] B. Kizil. *Introduction to Anomaly Detection in Time-Series Data and K-Means Clustering*. [Online; posted on 30-Oct-2020]. URL: https://medium.com/swlh/introduction-to-anomaly-detection-in-time-series-data-and-k-means-clustering-5832fb33d8cb.

[34] H.-Peter Kriegel, P. Kröger, E. Schubert, and A. Zimek. "LoOP: Local Outlier Probabilities." In: *Proceedings of the 18th ACM Conference on Information and Knowledge Management*. CIKM '09. Hong Kong, China: Association for Computing Machinery, 2009, 1649–1652. ISBN: 9781605585123. DOI: 10.1145/1645953.1646195. URL: https://doi.org/10.1145/1645953.1646195.

[35] P. Kumar. *Understanding LOF (Local Outlier Factor) —perspective for implementation*. [Online; posted on 6-Jul-2020]. URL: https://medium.com/@pramodch/understanding-lof-local-outlier-factor-for-implementation-1f6d4ff13ab9.

[36] "Kusto Query Language (KQL) overview." In: (2023). URL: https://learn.microsoft.com/en-us/azure/data-explorer/kusto/query/.

[37] *Kusto role-based access control*. URL: https://learn.microsoft.com/en-us/azure/data-explorer/kusto/access-control/role-based-access-control.

[38] M. Lanhenke. *Understanding the Covariance Matrix*. [Online; posted on 29-Dec-2021]. URL: https://towardsdatascience.com/understanding-the-covariance-matrix-92076554ea44.

[39] E. Lewinson. *A Step-by-Step Guide to Calculating Autocorrelation and Partial Autocorrelation*. [Online; posted on 30-Jan-2022]. URL: https://towardsdatascience.com/a-step-by-step-guide-to-calculating-autocorrelation-and-partial-autocorrelation-8c4342b784e8.

[40] C. Leys, O. Klein, P. Bernard, and L. Licata. "Detecting outliers: Do not use standard deviation around the mean, use absolute deviation around the median." In: *Journal of Experimental Social Psychology* 49.4 (2013), pp. 764–766. ISSN: 0022-1031. DOI: https://doi.org/10.1016/j.jesp.2013.03.013. URL: https://www.sciencedirect.com/science/article/pii/S0022103113000668.

[41] F. T. Liu, K. M. Ting, and Z-H Zhou. "Isolation Forest." In: *2008 Eighth IEEE International Conference on Data Mining*. 2008, pp. 413–422. DOI: 10.1109/ICDM.2008.17.

[42] C. Maklin. *Isolation Forest*. [Online; posted on 15-Jul-2022]. URL: https://medium.com/@corymaklin/isolation-forest-799fceacdda4.

[43] M. Mandal. *Introduction to Convolutional Neural Networks (CNN)*. [Online; posted on 1-May-2021]. URL: https://www.analyticsvidhya.com/blog/2021/05/convolutional-neural-networks-cnn/.

[44] *Matplotlib 3.8.1 documentation*. [Online]. URL: https://matplotlib.org/stable/index.html.

[45] A. Mavuduru. *How to perform anomaly detection with the Isolation Forest algorithm*. [Online; posted on 24-Nov-2021]. URL: https://towardsdatascience.com/how-to-perform-anomaly-detection-with-the-isolation-forest-algorithm-e8c8372520bc.

[46] *Mystery of Predictions in Anomaly Detection*. [Online; posted on 10-Dec-2021]. URL: https://research.einar.partners/mystery-of-predictions-in-anomaly-detection/.

[47] *NumPy documentation*. [Online]. URL: https://numpy.org/devdocs/.

[48] Prof. T. O'Haver. *A Pragmatic Introduction to Signal Processing, Appendix O: Random walks and baseline correction*. [Online; Updated July-2022]. URL: https://terpconnect.umd.edu/~toh/spectrum/AppendixO.html.

[49] *Outlier Detection 101*. URL: https://pyod.readthedocs.io/en/latest/relevant_knowledge.html.

[50] J. Pandey. *SR CNN - Taking the error out of business*. [Online; posted on 19-Jul-2021]. URL: https://medium.com/@juhipandey4/sr-cnn-taking-the-error-out-of-business-a8b3cfa54835.

[51] A. Perlato. *Fitting a curve to data LOWESS and LOESS*. [Online]. URL: https://www.andreaperlato.com/theorypost/fitting-a-curve-to-data-lowess-and-loess/.

[52] *Plotly Open Source Graphing Library for Python*. [Online]. URL: https://plotly.com/python/.

[53] *PowerBI*. [Online]. URL: https://powerbi.microsoft.com/en-us/.

[54] K. Pradeep and H. Surana. *A Brief History of Anomaly Detection*. [Online; posted on 16-Aug-2021]. URL: https://www.chaosgenius.io/blog/a-brief-history-of-anomaly-detection/.

[55] *Project Jupyter Documentation*. [Online]. URL: https://docs.jupyter.org/en/latest/.

[56] A. Pucher. *Introducing ThirdEye: LinkedIn's Business-Wide Monitoring Platform*. [Online; posted on 9-Jan-2019]. URL: https://engineering.linkedin.com/blog/2019/01/introducing-thirdeye--linkedins-business-wide-monitoring-platfor.

[57] A. Rajbhoj. *ARIMA simplified.* [Online; posted on 26-Sep-2019]. URL: https://towardsdatascience.com/arima-simplified-b63315f27cbc.

[58] David Ramel. *Azure Data Explorer Heads Microsoft Cloud Analytics Updates*. [Online; posted on 03-Apr-2019]. URL: https://visualstudiomagazine.com/articles/2019/03/04/azure-data.aspx.

[59] U. Riswanto. *K-Means Clustering for Anomaly Detection*. [Online; posted on 30-Jan-2023]. URL: https://ujangriswanto08.medium.com/k-means-clustering-for-anomaly-detection-1bbbb0b20b52.

[60] R. Roy-II. *LSTM-AutoEncoders*. [Online; posted on 25-Jun-2021]. URL: https://bobrupakroy.medium.com/lstm-autoencoders-a45a04667346.

[61] S. Saxena. *What is LSTM? Introduction to Long Short-Term Memory*. [Online; posted on 16-Mar-2021]. URL: https://www.analyticsvidhya.com/blog/2021/03/introduction-to-long-short-term-memory-lstm/.

[62] A. Shah. *LSTM Autoencoder for Anomaly Detection for ECG data*. [Online; posted on 1-Jul-2022]. URL: https://medium.com/@jwbtmf/lstm-autoencoder-for-anomaly-detection-for-ecg-data-5c0b07d00e50.

[63] E. Sharma. *K-Means vs. DBSCAN Clustering — For Beginners*. [Online; posted on 27-May-2020]. URL: https://towardsdatascience.com/k-means-vs-dbscan-clustering-49f8e627de27.

[64] R. Sharma. *STL decomposition : How to do it from Scratch?* [Online; posted on 10-Nov-2019]. URL: https://towardsdatascience.com/stl-decomposition-how-to-do-it-from-scratch-b686711986ec.

[65] E. Shrestha. *Time-series Anomaly Detection with Twitter's ESD test*. [Online; posted on 6-Jan-2020]. URL: https://elisha32.medium.com/time-series-anomaly-detection-with-twitters-esd-test-50cce409ced1.

[66] D. Smith. *Twitter's new R package for anomaly detection*. [Online; posted on 7-Jan-2015]. URL: https://blog.revolutionanalytics.com/2015/01/twitters-new-r-package-for-anomaly-detection.html.

[67] K. Srinath. *Anamoly Detection: Techniques to detect outliers*. [Online; posted on 22-Jun-2020]. URL: https://towardsdatascience.com/anamoly-detection-techniques-to-detect-outliers-fea92047a222.

[68] D. Stepanov. *Anomaly detection using Minimum Covariance Determinant (MCD)*. [Online; posted on 29-Jun-2022]. URL: https://medium.com/geekculture/anomaly-detection-using-minimum-covariance-determinant-mcd-41dc28ccc8a.

[69] O. Vallis and J. Hochenbaum. *Introducing practical and robust anomaly detection in a time series*. [Online; posted on 6-Jan-2015]. URL: https://blog.twitter.com/engineering/en_us/a/2015/introducing-practical-and-robust-anomaly-detection-in-a-time-series.

[70] Wikipedia contributors. *Isolation forest — Wikipedia, The Free Encyclopedia*. [Online;] 2023. URL: https://en.wikipedia.org/w/index.php?title=Isolation_forest&oldid=1170095195.

[71] Wikipedia contributors. *Local regression — Wikipedia, The Free Encyclopedia*. [Online]. 2023. URL: https://en.wikipedia.org/w/index.php?title=Local_regression&oldid=1166437966.

[72] Wikipedia contributors. *One-class classification — Wikipedia, The Free Encyclopedia*. https://en.wikipedia.org/w/index.php?title=One-class_classification&oldid=1169307470. [Online]. 2023.

[73] In Wikipedia. *DBSCAN*. [Online]. URL: https://en.wikipedia.org/wiki/DBSCAN.

[74] T. Xing. *Overview of SR-CNN algorithm in Azure Anomaly Detector*. [Online; posted on 5-Nov-2019]. URL: https://techcommunity.microsoft.com/t5/ai-customer-engineering-team/overview-of-sr-cnn-algorithm-in-azure-anomaly-detector/ba-p/982798.

[75] Pedregosa et al. *Outlier detection with Local Outlier Factor (LOF)*. [Online]. URL: https://scikit-learn.org/stable/auto_examples/neighbors/plot_lof_outlier_detection.html.

[76] akejariwal et al. *AnomalyDetection R Package Twitter*. 2014. URL: https://github.com/twitter/AnomalyDetection.

[77] baeldung. *What Is One Class SVM and How Does It Work?* [Online; posted on 16-Jun-2023]. URL: https://www.baeldung.com/cs/one-class-svm.

[78] ccolburn. *Netflix Surus*. 2015. URL: https://github.com/Netflix/Surus.

[79] Rob Hyndman (https://stats.stackexchange.com/users/159/rob hyndman). *Fitted Confidence Intervals Forecast Function R*. Cross Validated. URL:https://stats.stackexchange.com/q/154392 (version: 2015-05-28). URL: https://stats.stackexchange.com/q/154392.

[80] marcusbakker. *KQL Cheat Sheet*. URL: https://github.com/marcusbakker/KQL/blob/master/kql_cheat_sheet_v01.pdf.

[81] *pandas documentation*. [Online; updated on 26-Oct-2023]. URL: https://pandas.pydata.org/docs/.

[82] *scikit-learnMachine Learning in Python*. [Online]. URL: https://scikit-learn.org/stable/.

[83] *statistical models, hypothesis tests, and data exploration*. [Online; updated on 05-May-2023]. URL: https://www.statsmodels.org/stable/index.html#citation.

[84] www.Datacamp.Com. *Top programming languages for data scientists in 2023*. [Online; updated on 01-Mar-2023]. URL: https://www.datacamp.com/blog/top-programming-languages-for-data-scientists-in-2022.