FinOps :
Monitoring and Controlling GCP costs

Saad AIT CHIKH - 2201837
Master’s Thesis in computer science
ÅAU Supervisor : Sébastien Lafond
INSA Supervisor: Laurence Rozé
Faculty of Science and Engineering
Double Degree INSA Rennes - Åbo Akademi University (ÅAU) - 2023

Abstract

Cloud computing has gained significant popularity in today's digital landscape, with companies relying on cloud-based solutions to manage their data, applications, and infrastructure. The cloud offers several advantages, including scalability, flexibility, and cost-effectiveness, making it a popular choice for businesses of all sizes. However, with the increasing adoption of cloud technologies, it is important for companies to keep a close eye on their cloud usage costs to ensure they are using the cloud efficiently and effectively. This is where the discipline of Financial Operations (FinOps) comes into play. FinOps seeks to optimize cloud spending, and it has become increasingly important for organizations that utilize cloud computing. By implementing FinOps practices, companies can achieve better cost visibility and control, leading to more efficient and effective cloud usage. While several cloud providers are available in the market, such as Amazon Web Services (AWS) and Microsoft Azure, this work will focus specifically on Google Cloud Platform (GCP).

The goal of this thesis is to present two implemented solutions for managing GCP costs: proactive anomaly detection and cost forecasting using machine learning (ML) algorithms. Thanks to anomaly detection, companies can detect unusual patterns in their cloud billing data and proactively alert teams to investigate and address any issues. Furthermore, forecasting future costs can help companies anticipate potential cost spikes and take proactive measures to avoid them.

Keywords : Machine Learning, Anomaly Detection, Cost Forecasting, Google Cloud Platform (GCP), API

Preface

This thesis is the outcome of my end-of-studies internship that I had the pleasure to complete at *Neoxia*. Before starting talking about this professional experience, it seems natural to me to first express my gratitude to all the people who have been able to participate in the smooth running of this internship and the achievement of this thesis.
First, I would like to warmly thank my internship tutors, *Mrs Audrey KAMTA DJAKOU* and *Mr Kais ARBI*, Data Full Stack Consultants at *Neoxia*, for their monitoring of my internship, their continuous support, their contribution to my mission and their availability throughout this period.
A special thanks to *Mr Mohamed Amine BERGAOUI*, the Data Factory Director and *Mr Jean-Baptiste PACCOUD*, the CEO of *Neoxia*, for the importance they attach to my work.
At the same time, I would like to thank the entire *Neoxia* technical team. It was a real pleasure to work alongside them, both at office and remotely.
I also like to express my deep gratitude to all my professors at *INSA Rennes*, and my professors of my double degree at *Åbo Akademi University* for the quality of the training they provided.
Finally, I also express my appreciation to *Laurence Rozé*, my supervisor at *INSA Rennes* and *Sébastien Lafond*, my tutor at *Åbo Akademi University*, for their support.

Contents

List of Abbreviations

GCP	Google Cloud Platfrom.
AI	Artificial Intelligence.
API	Application Programming Interface.
ML	Machine Learning.
R&D	Research and experimental Development.
HTTP	Hypertext Transfer Protocol.
IQR	Inter-Quantile Range.
ARIMA	AutoRegressive Integrated Moving Average.
ACF	Autocorrelation Function.
PACF	Partial Autocorrelation Function.
NN	Neural networks.
LSTM	Long-Short-Term-Memory.

List of Figures

List of Tables

Introduction

In this first chapter, we will present the company, Neoxia, where I did my internship. Then, we will introduce the context of the internship as well as its objectives.

1.1 The company : Neoxia

Neoxia is a digital services company founded in 2000 in Paris and is now established in seven different offices that span three countries : France, Morocco and Canada. *Neoxia* now includes more than 300 highly skilled professionals, working on three core activities of the company which are:

- Business Technology Consulting: offering services in digital strategy, up-to-date architecture, technical audit, experiments and Proof of Concept.
- Digital and Data Factory: working on Design and Delivery of digital platforms (web, mobile, IoT, data driven) and offering solutions related to different Data disciplines: Data Science, Machine Learning and finally Data Engineering.
- DevOps Run 24/7 : integration, industrialization and 24/7 cloud managed services.

Through those three units, *Neoxia* tries to respond to its customers' business issues by deploying operational solutions as quickly as possible. The services offered include information systems architecture design, information systems audit, applications development and deployment. Also, the company spends 20% of its time on R&D projects and does not hesitate to take risks, partnering with the best entrepreneurs or business leaders to design, build and even fund highly innovative solutions. During this internship, I integrated the Digital & Data factory unit, and especially the Data team in the Paris office.
Since 2000, *Neoxia* has been a key player in the digital transformation of all cloud pioneering companies, from start-ups to large enterprises. For instance, *Neoxia* has recently worked with several clients such as: Eurosport, Veolia, Intermarché, Fnac-Darty, Air Liquide, Bouygues Construction, L'Equipe, Pernod-Ricard and others.
In the following, we will present the clients I worked for, as well as my mission.

1.2 Internship context and goals

Neoxia has several clients who use Google Cloud Platform (GCP) to host their data, applications and projects. These customers want to have an alert system to detect high costs, which can be caused by GCP services charged at high rates. They would also like to have a detailed monitoring of their GCP hosting invoices and anticipate future costs to avoid budget overruns.
Two of Neoxia's clients, *Ouest France* and *Stime*, expressed the need for a detailed monitoring of their GCP costs. *Ouest France* is a French newspaper company, founded in 1944, which publishes regional and national daily newspapers. *Stime*, in turn, is the Information Systems Department of the Groupement Les Mousquetaires, which has seven brands (Intermarché, Netto, Bricomarché, Bricorama, Brico Cash, Roady, Rapid Pare-Brise).
In this context, the objectives of this internship can be divided into three main parts:

- The first goal is to develop an anomaly detection method to alert customers when they detect high costs.
- The second objective is the development of a model that forecasts GCP costs for the upcoming months.
- The third objective is the deployment and industrialization of the two solutions, i.e. the anomaly detection method and the GCP cost forecasting model.

This internship will focus on *Stime* data, but the main goal is to develop a generic solution that can be deployed for *Ouest France* and other clients.

Before diving into the details of the proposed solutions, it is appropriate to present an overview of the GCP services relevant to our study in chapter 2. Then, we will continue with a state of the art on time series in chapter 3. In chapter 4, the accomplished work will be presented.

Google Cloud Platform

Google Cloud Platform (GCP) [5] is a major player in the cloud market. It allows companies to host their data and applications in the cloud. GCP offers a suite of cloud computing services including storage, machine learning, big data, security, and networking. Companies are only charged for the amount of time and resources that they consume, rather than having to pay for pre-allocated amounts of computing power. This chapter provides a short review of some GCP services that will be used in this project.

2.1 Cloud Compute

Google Cloud Compute services are virtual machines that are provisioned and physically running on Google's infrastructure. As part of this project, we will be using two main services : Compute Engine and Cloud Functions.

2.1.1 Compute Engine

Google Compute Engine (GCE) [4] provides a scalable number of virtual machines called instances that allow clients to run workloads. GCE can be managed through an API, the Google Cloud console or the Google command line interface. When creating instances, some properties can be specified, such as the number of virtual CPUs and the amount of memory, by using a set of predefined machine types or by creating custom machine types.

2.1.2 Cloud Functions

Google Cloud Functions [13] are serverless cloud-based functions for connecting and extending cloud services. With cloud functions, developers can write code which will be executed in a fully managed environment. These functions are triggered by events emitted from other cloud services. Their code is typically stored in a *Google Cloud Storage bucket*. Buckets are containers for storing data and code in GCP. Cloud functions supports various programming languages, with Python being the main programming language for this project.

2.2 Big Data - Analytics

GCP offers a wide variety of Big Data[1] and Analytics services to manage and analyze data. These services include solutions for data warehousing[2], data exploration, stream, and batch analytics. Out of all the offered services, the two that we will be use are BigQuery and Looker studio.

2.2.1 BigQuery

BigQuery [1] is a highly scalable, and cost-effective analytics and data warehousing platform. BigQuery allows oraganizations to store massive amounts of data. Its serverless architecture allows it to operate at scale and speed to provide incredibly fast SQL analytics over large datasets.

2.2.2 Looker Studio

Looker Studio [9] , formerly known as Data Studio, is a powerful data expolaration and visualization platform to create fully customizable dashboards and interactive reports. Users can easily collaborate on reports and share insights with other team members. Looker Studio can be connected to a wide variety of data sources, including GCP services like BigQuery or third-party data, like Facebook and LinkedIn Ads.

2.3 Cloud Management

Google Cloud Management is a wide range of tools provided by Google to simplify application management tasks. Customers have access to different services and APIs to manage their Google Cloud projects, schedule tasks and monitor cloud resources such as Compute Engine instances. Billing API and Cloud scheduler are our point of emphasis among the suite of Cloud Management services.

[1]The massive amount of data available to organizations

[2]Collecting data from a wide range of sources into a single database

2.3.1 Billing API

Billing API [2] is a set of tools to plan, monitor, and control costs of cloud resources. This API provides the Billing export feature which allows organizations to export billing data automatically to a BigQuery dataset. The billing data contains detailed usage and billing information for all the used cloud resources. This data helps identifying specific resources that might be driving up costs. It can be accessed from BigQuery for detailed analysis, or using Looker Studio for visualization. The billing export will be the primary data we use for this project.

2.3.2 Cloud Scheduler

Google Cloud Scheduler [3] is a fully managed service that enables users to schedule and automate the execution of recurring tasks at defined times or regular intervals. Each scheduled task is commonly known as *cron* job. *Cron* jobs are sent to a target in a specified schedule to accomplish a task. Typical targets are HTTP endpoints like Cloud Functions.

2.4 Cloud AI

Cloud AI is a suite of artificial intelligence services (AI) offered by Google. The AI services include Cloud Natural Language to analyze documents, Cloud Speech API to transcribe audio to text, Cloud Video Intelligence API for video analysis, Cloud AutoML to build high-quality custom machine learning models automatically and, finally, Vertex AI which is the tool used for this project.

2.4.1 Vertex AI

Vertex AI [12] is a fully managed Machine Learning (ML) platform for building, deploying, and scaling ML models. In Vertex AI, users can easily train and compare models using Cloud AutoML or custom code training; it integrates with open source frameworks such as *TensorFlow* and *scikit-learn*.
Once the relevant GCP services are presented, we will continue with a state of the art of time series.

Time Series: state-of-the-art

A time series is a set of observations x_t, each recorded at a specific time t. This set of data points is indexed in chronological order with a fixed frequency. Time series appear in many different domains: economics, finance, engineering, environmental modeling, meteorology, and more.

Each time series has certain characteristics that need to be considered before creating time series models. These characteristics will be discussed in the first section (3.1) of this chapter, before moving on to the second section (3.2) to discuss time series anomaly detection and then forecasting models in the last section (3.3).

3.1 Time Series Analysis

Time series analysis is a mathematical approach for identifying characteristics and extracting knowledge from time series. This approach highlights the time series patterns, components and certain correlations that may be difficult to detect with the naked eye.

3.1.1 Time series patterns

Time series patterns are recurring behaviors that can be observed in many different types of data. Here are three important time series patterns:

- **Trend:** a long-term movement in the data that follows a consistent direction, either upwards or downwards.
- **Seasonality:** refers to patterns that repeat themselves over a fixed period, such as weeks, months, or quarters.
- **Cycle:** refers to patterns that repeat themselves over a period of time that is longer than seasonality.

Overall, identifying and understanding these time series patterns is essential for understanding the components of a time series.

3.1.2 Time series components

A time series X_t can be decomposed into :

- **Trend** T_t corresponding to the trend-cycle component.
- **Seasonality** S_t corresponding to the seasonal component.
- **Error** ε_t which is the random part of the series.

This decomposition can be additive or multiplicative. A multiplicative decomposition would be written as in equation (3.1):

$$X_t = T_t * S_t * \varepsilon_t \tag{3.1}$$

Alternatively, if we assume an additive decomposition which is commonly used for financial data, then the mathematical equation is written as follows in equation (3.2):

$$X_t = T_t + S_t + \varepsilon_t \tag{3.2}$$

There are several methods to decompose a time series; we will discuss the most robust one : STL.

3.1.3 STL decomposition

STL (Seasonal-Trend decomposition using LOESS[1]) [16] is a robust method for time series decomposition. STL uses regression models to decompose the time series into its three components (trend, seasonality and error).
Figure 3.1 shows an example of the STL decomposition of an additive time series. Notice that if we add the three components (random, seasonal and trend) together, the time series (observed data) will be reconstructed.

[1]A method for estimating nonlinear relationships.

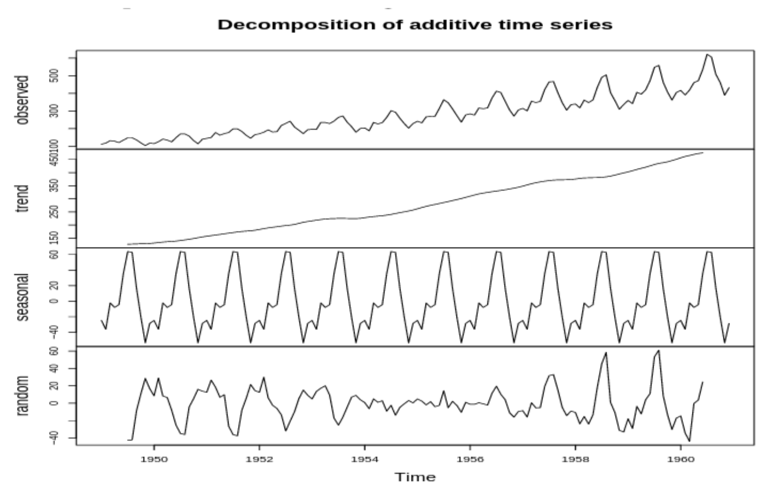


Figure 3.1: Example of a time series and its STL decomposition

3.1.4 Autocorrelation

Autocorrelation function (ACF) [10] measures the degree of similarity between the current values of a variable and the historical data of that same variable —hence, the name. Autocorrelation r_k refers to the relationship between a time series X_t, and a lagged version of itself X_{t-k}, where X_{t-k} is a version of the original time series X_t that is k periods behind in time. k is the lag.
When k=1, autocorrelation is assessing adjacent observations. For each lag, there is a correlation. For example, the autocorrelation at lag 2 measures how correlated is this month's value with the value two months ago.

3.1.5 Partial Autocorrelation

The partial autocorrelation function (PACF) [10] is a measure of the direct relationship between a time series and its lagged counterparts while removing the effects of intermediate lags. It provides a way to determine the direct influence of a specific lag on the current value of the time series, independent of any indirect effects from other lags. For instance, the PACF can be used to identify the direct impact of the number of passengers from six months ago on this month's number of passengers, by removing the effects of the number of passengers from each of the intervening months. The process involves iteratively removing the effects of the intermediate lags until only the direct influence of the specified lag remains in the calculation.
We will not delve into the technical details of how to calculate ACF and PACF in this context, as it involves complicated computations and they are not essential for our purposes.

Now that we have covered the basics of time series analysis, we can move on to the next step: time series anomaly detection.

3.2 Time Series Anomaly Detection

While analyzing time series data, we have to make sure of unusual data points. These observations are often referred to as anomalies. In this section, the main concepts are defined. Then, anomaly detection approaches will be presented.

3.2.1 Anomaly definition

In time series, an anomaly [6] is a data point that is significantly different from the overall nature of the time series. Commonly named as an outlier, this data point deviates considerably from the patterns of the rest of the data. Thus, in a set Ω containing N observations denoted x_i, then $x_k \in \Omega$ will be considered as an anomaly if it differs significantly from the observations contained in the set $\Omega \setminus \{x_k\}$.
In the context of this work, a data point is considered as an anomaly when it does not conform to the expected behavior. Classifying a data point as abnormal depends a lot on the context. Thus, anomalies are often divided into types.

3.2.2 Anomaly types

There are multiple types of anomalies, but we will focus on the most important ones [6]:

- **Point anomaly** : otherwise called global anomaly, corresponds to a single data point that deviates with respect to a given metric (distance for example). After the anomaly has occurred, the time series recovers its normal behavior. In Figure 3.2, the red dot is a point anomaly, so it is outside the "normal" behavior of the series.

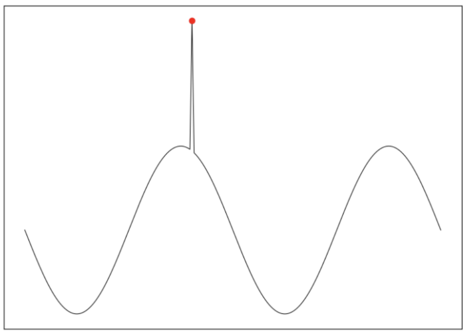


Figure 3.2: Example of a point anomaly

- **Contextual anomaly** : An anomaly is contextual if it depends on the context where it occurs. In this case, it will be difficult to decide correctly on the anomalous aspect without having information about the context. We can take the example of a time series modeling the temperature over a year. In figure 3.3, at time t_1 a temperature of 5 °C can be considered as normal in the winter but abnormal at time t_2 in the summer.

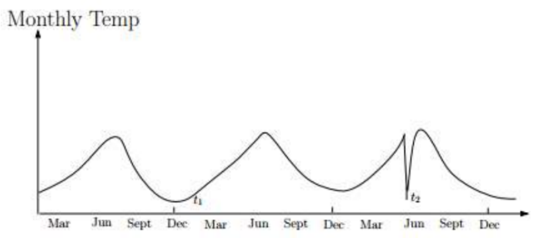


Figure 3.3: Example of a contextual anomaly

- **Collective anomaly** : A single observation will not be considered as a collective anomaly. When analyzing a complete time series, a subsequence[2] will reveal abnormal behavior relative to the rest of the series. We illustrate this example in figure 3.4.

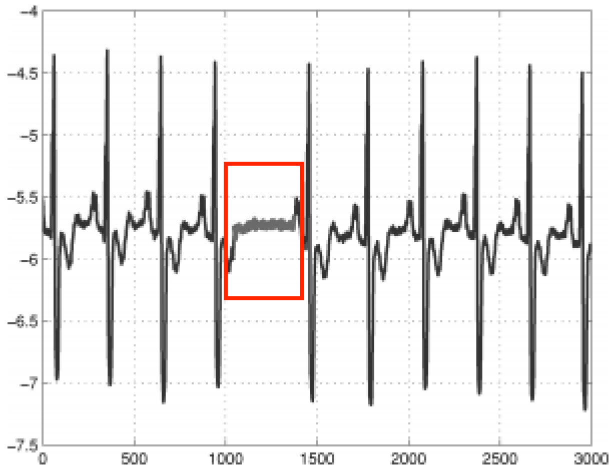


Figure 3.4: Example of a collective anomaly

In this project, we will be facing point anomalies. Understanding the type of anomalies will guide us in choosing the appropriate approaches to detect them.

3.2.3 Anomaly Detection Approaches

In literature, anomaly detection has been extensively studied, starting from statistical methods to sophisticated machine learning algorithms. This section will explore three different techniques. The first one is Inter-Quantile Range.

[2]A segment/sequence of the time series

3.2.3.1 Inter-Quantile Range

To explain the Inter-Quantile Range (IQR) method [15], we will start by understanding the concept of a box plot. Also known as a box-and-whisker plot, it is a graphical representation of a dataset that provides information on the spread, skewness, and range of the data. A box plot is shown on the top part of figure 3.5 :

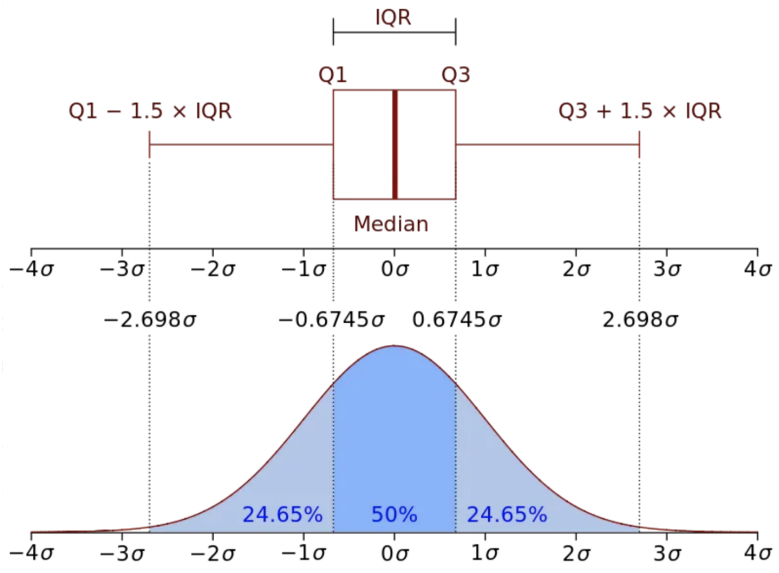


Figure 3.5: IQR method

- **Median** is the center point of the data; it is the value separating the higher half of the data from the lower half.
- **Q1** is the first quartile of the data, meaning that 25% of the data lies below Q1.
- **Q3** is the third quartile of the data, meaning that 75% of the data lies below Q3.

The difference between Q3 and Q1 is called the Inter-Quartile Range or IQR (3.3):

$$IQR = Q3 - Q1 \tag{3.3}$$

In order to identify anomalies using this method, a new range known as the decision range is defined. Any data point that falls outside this range is classified as an anomaly. The decision range $[\texttt{Lower Bound}, \texttt{Upper Bound}]$ is determined as follows:

- **Upper Bound** $= Q3 + \beta * IQR$
- **Lower Bound** $= Q1 - \beta * IQR$

IQR is denoted as $IQR(\beta)$. To determine the decision range, generally we set β at 1.5 in the bounds. This is because any data point that falls beyond 2.7 standard deviations σ from the mean μ in either direction is considered an anomaly. This decision range aligns closely with the threshold of 3σ specified by the Gaussian distribution approach for outlier detection. Refer to figure 3.5 for a visual representation.
After understanding the Inter-Quantile Range (IQR) method for anomaly detection, we will now move on to another popular technique called Local Outlier Factor (LOF).

3.2.3.2 Local Outiler Factor

The Local Outlier Factor (LOF) algorithm [8] is an unsupervised ML method for finding anomalies in a dataset by measuring an individual's local deviation from its neighbors [8]. The approach involves comparing the local density of each individual with respect to the densities of its k-nearest neighbors. Outliers are identified as individuals with a lower density compared to their neighbors.
Let $dist_k(i)$ be the distance between data point X_i and its k^{th} nearest neighbors, and let $N_k(X_i)$ be the set of X_i's k-nearest neighbors; $N_k(X_i)$ includes a set of points that lie in the circle of radius $dist_k(i)$. This distance is used to define what is known as reachability distance (3.11):

$$dist_{reach}(X_i, X_j) = max(dist_k(X_j), d(X_i, X_j)) \tag{3.4}$$

In the example below 3.6, $k = 2$ and $||N_k(X_j)|| = 3$. In regards to reachability distance, if a point Xi lies within $N_k(X_j)$, $dist_{reach}(X_i, X_j)$ will be $dist_k(X_j)$ (blue line), else $dist_{reach}(X_i, X_j)$ will be the distance between X_i and X_j (orange line).

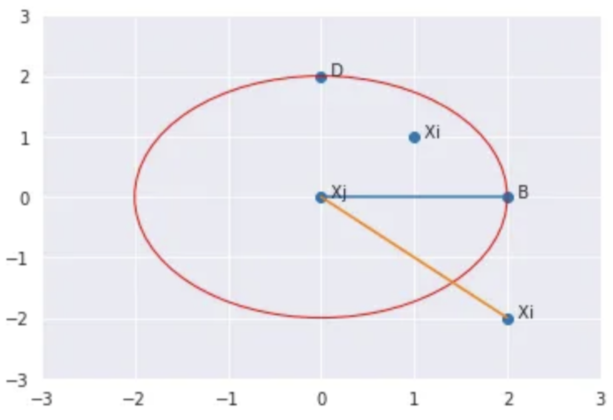


Figure 3.6: Illustration of reachability distance

The reachability distance is then used to calculate another concept called local reachability density (LRD). The LRD of a data point A is calculated as follows (3.5) :

$$LRD_k(A) = \frac{1}{\sum_{X_j \in N_k(A)} \frac{\text{dist}_{reach}(A, X_j)}{k}} \tag{3.5}$$

Intuitively, the LRD_k (Distance to Nearest Cluster) informs about the distance that needs to be covered from A in order to reach the nearest cluster of points. The LRD_k of each individual can then be compared to the LRD_k of its k-nearest neighbors using *LOF* formula (3.6):

$$LOF_k(A) = \frac{\sum_{X_j \in N_k(i)} LRD_k(X_j)}{k \times LRD_k(A)} \tag{3.6}$$

A value of LOF ≈ 1 indicates that the data point is similar to its neighbors, while a value of LOF > 1 indicates that the data point is an anomaly. Local Outlier Factor model will be denoted as *LOF(k)*, where k represents the parameter for the number of neighbors used in the model.
After discussing the Local Outlier Factor method, we will explain the last method, Isolation Forest, which is a tree-based algorithm for anomaly detection.

3.2.3.3 Isolation Forest

Isolation forest [7] is an unsupervised anomaly detection algorithm. The algorithm selects a set of randomly drawn samples. Each sample is separated according to a randomly chosen threshold value between the maximum and minimum of the time series. Data points of each sample are classified according to this threshold to build a decision tree, all the lower values will be on the left and the higher ones on the right. This operation is repeated according to another threshold value in each sub-tree created, until we have isolated each data point. Figure 3.7 shows a fitted tree on a given sample. We can notice that the anomaly (Point G) is close to the root node (i.e. at a lower depth); however, normal instances are not.

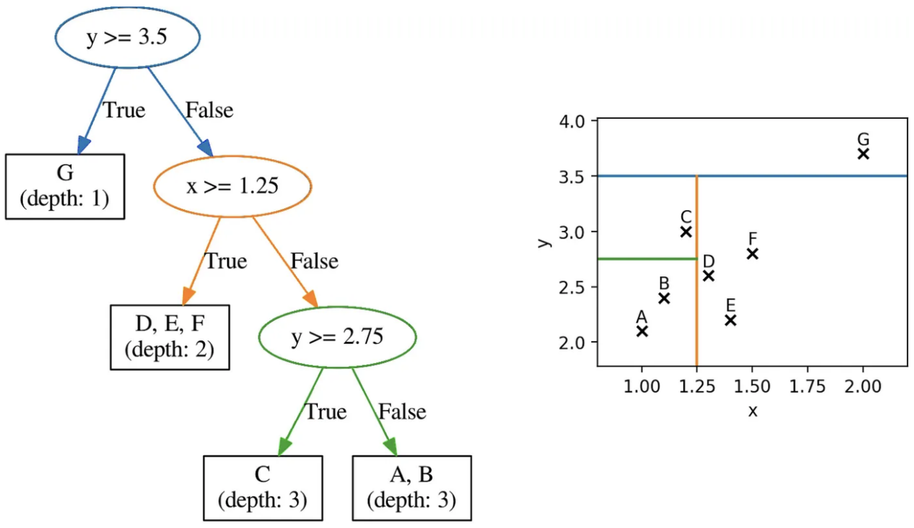


Figure 3.7: An example of a fitted tree with Isolation Forest

As often in machine learning, the key is iteration. In fact, if we randomly fit many decision trees, and then take an average of the depth of each data point over the different trees, we find an *average depth* that represents an *anomaly score*. Thus, the *p* data points with lower average depth are more likely to be anomalies. Therefore, Isolation Forest is denoted as *IF(p)*, where *p* represents the percentage of anomalies in the dataset and is set at 0.01 by default. Figure 3.8 shows a conceptual illustration of Isolation Forest.

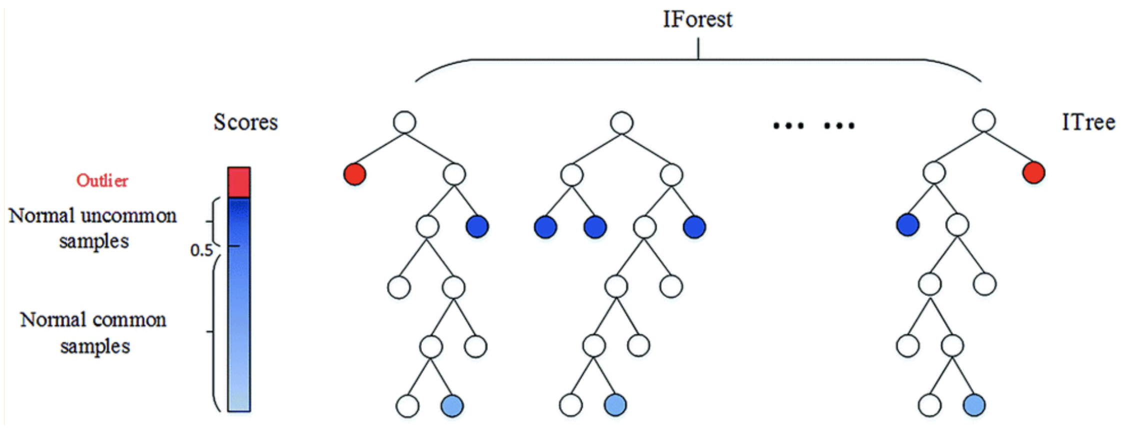


Figure 3.8: Isolation Forest algorithm

Now that we have explored different algorithms for anomaly detection, it is important to evaluate their performance using appropriate metrics.

3.2.4 Evaluation metrics

Confusion matrix is a useful tool to evaluate the performance of a binary classification model. It represents the classification results, where normal instances are classified as negative (0) and anomalous instances are classified as positive (1).
The confusion matrix is defined as follows 3.9:

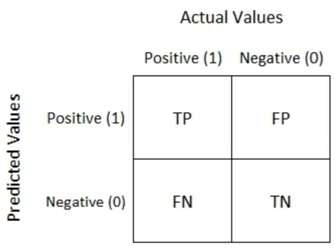


Figure 3.9: Confusion matrix

In the confusion matrix, True Positives (TP) represent the number of correctly classified anomalous instances, False Positives (FP) are the number of normal instances that are incorrectly classified as anomalous, True Negatives (TN) are the number of correctly classified normal instances, and False Negatives (FN) are the number of anomalous instances that are incorrectly classified as normal. Since we are primarily interested in identifying anomalies, TN doesn't provide us with useful information about the performance of the model in this context. From the confusion matrix, various metrics can be derived to evaluate the model's performance, such as precision, recall and F-1 score :

- **Precision** measures the proportion of correctly classified anomalous instances over all instances classified as anomalous.

$$Precision = \frac{TP}{TP+FP} \tag{3.7}$$

- **Recall** measures the proportion of correctly classified anomalous instances over all anomalous instances in the dataset.

$$Recall = \frac{TP}{TP+FN} \tag{3.8}$$

- **F1-score** provides a balanced measure between precision and recall.

$$F1-score = 2*\frac{precision*recall}{precision+recall} \tag{3.9}$$

Now that we have covered a state of the art of time series anomaly detection techniques, it is time to explore the topic of time series forecasting.

3.3 Time Series Forecasting

The usual objective of time series analysis is to predict the future values of a series: given a series of observations $x_1, ..., x_T$, the objective is to predict (at time T) the value of the series at time $T+h$, noted x_{T+h}. See figure 3.10 for the time series forecast of the same example 3.1.3.

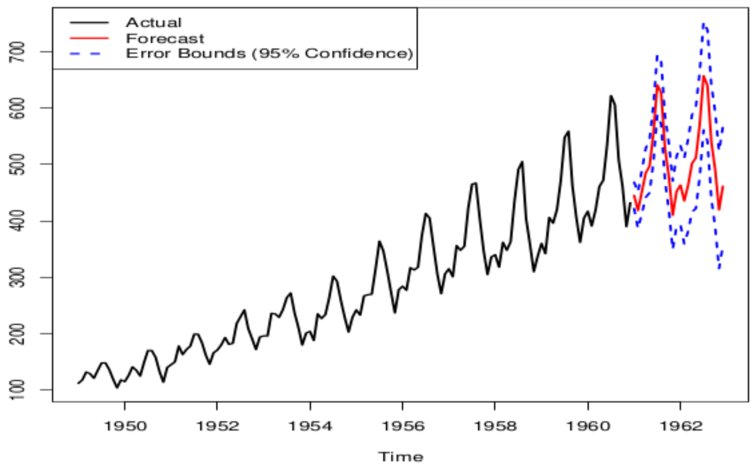


Figure 3.10: Forecasting a time series

This section provides an overview of the main models available for time series forecasting.

3.3.1 Prophet

Prophet [11] [14] is an algorithm based on an additive model for time series forecasting. This open source library is developed by *Facebook* with the aim of democratizing time series forecasting. This model is integrated into *Stan*, a programming language for statistical inference written in C++. Prophet fits linear and non-linear trends with three main model components: trend, seasonality, and holidays. The equation (3.10) shows the mathematical representation of this algorithm :

$$y(t) = g(t) + s(t) + h(t) + \varepsilon_t \tag{3.10}$$

- The **trend** $g(t)$ models non-periodic changes : continuous linear or piecewise logistic. The change points of the trend are automatically detected by the model.
- The **seasonality** $s(t)$ (i.e. daily, weekly, monthly, yearly) is approximated by a Fourier series which given in (3.11):

$$s(t) = \sum_{n=1}^{N} (\alpha_n cos(\frac{2\pi nt}{P}) + \beta_n sin(\frac{2\pi nt}{P})) \quad (3.11)$$

where :

- P is the period (365.25 for annual data, 30.5 for monthly data, 7 for weekly data)
- N is the length of the time series

- A function of **holidays** $h(t)$ integrated by Prophet which allows the capture of vacations and recurring events. This component will not be used in this project.
- ε_t is the **error** component accommodated by the model. ε_t is normally[3] distributed.

One notices that this approach is simply fitting a curve without analyzing the time based dependency between individual data points. One of the most popular models for analyzing this type of relationships is the ARIMA family models.

3.3.2 ARIMA models

ARIMA [18], for AutoRegressive Integrated Moving Average, is a set of statistical models that are widely used for time series forecasting. This section explores the literature of these different models: *I*, *AR*, *MA*, *ARIMA* and *seasonal ARIMA*.

3.3.2.1 Stationarity and differencing

A stationary time series is one whose statistical properties, such as mean and variance, do not change over time. In other words, the series does not have any trend component. ARIMA is based on the assumption that the data is stationary, as it simplifies the modeling process and allows for more accurate forecasts. However, it is common for real-world time series data to exhibit trend patterns, which must be removed before modeling.
To make a non-stationary time series stationary, we can use differencing. By taking the difference between consecutive observations, we can remove the trend component and obtain a stationary series. In ARIMA models, this differencing process is presented by the $I(d)$ model where d is the degree of differencing.

[3]Normal distribution, also known as the Gaussian distribution

3.3.2.2 Autoregressive models

In an autoregressive model (AR), the variable of interest is forecasted by using a linear combination of its past values. This type of model is called autoregression because it is a regression of the variable against itself. Specifically, an autoregressive model of order p is represented by the equation (3.12) :

$$y_t = \phi_1 y_{t-1} + \phi_2 y_{t-2} + \cdots + \phi_p y_{t-p} + \varepsilon_t \tag{3.12}$$

Similar to multiple regression, the autoregression model also uses lagged values of y as predictors. Therefore, we denote it as an $AR(p)$ model, where p represents the order of the autoregression.

3.3.2.3 Moving average models

A moving average (MA) model takes a different approach than an autoregressive model in time series forecasting. Instead of using past values of the forecast variable as predictors, it utilizes past forecast errors in a regression-like model:

$$y_t = \varepsilon_t + \theta_1 \varepsilon_{t-1} + \theta_2 \varepsilon_{t-2} + \cdots + \theta_q \varepsilon_{t-q} \tag{3.13}$$

An MA model of order q is denoted as $MA(q)$, where q represents the number of past forecast errors that are included in the model.

3.3.2.4 Non-seasonal ARIMA models

By combining differencing with autoregression and a moving average model, we can create a non-seasonal ARIMA model. This model can be expressed as a combination of the AR, I, and MA models. The equation is written as follows (3.14) :

$$y'_t = \phi_1 y'_{t-1} + \cdots + \phi_p y'_{t-p} + \theta_1 \varepsilon_{t-1} + \cdots + \theta_q \varepsilon_{t-q} + \varepsilon_t \tag{3.14}$$

where the differenced series is denoted as y'_t and the predictors on the right-hand side of the equation include both lagged values of y_t and lagged errors. This model is denoted as an ARIMA(p, d, q) model, where:

- p is the order of the autoregressive part
- d is the degree of differencing
- q is the order of the moving average part

We can determine the appropriate values of p and q that capture the autoregressive and moving average components of the series by using the *ACF* and *PACF* discussed in 3.1.4 and in 3.1.5. The value of d is determined by the number of times the series needs to be differenced to achieve stationarity.

3.3.2.5 Seasonal ARIMA models

So far, we have focused on analyzing and modeling non-seasonal data using non-seasonal ARIMA models. Nonetheless, ARIMA models are also applicable for modeling seasonal data. To incorporate the seasonal patterns present in the data, we can include additional seasonal terms in the ARIMA models. This results in a seasonal ARIMA model, expressed as (3.15):

$$ARIMA \quad \underbrace{(p,d,q)}_{\text{Non-seasonal part}} \quad \underbrace{(P,D,Q)_m}_{\text{Seasonal part}} \tag{3.15}$$

where m represents the number of observations per season. To differentiate between the seasonal and non-seasonal components, uppercase notation is used for the seasonal parts and lowercase notation is used for the non-seasonal parts. The seasonal part of the model involves terms that are similar to the non-seasonal components, but incorporate backshifts of the seasonal period to account for the seasonal variation.
Moving on from the state-of-the-art ARIMA model, we now shift our focus to a more advanced approach in time series forecasting: Long Short-Term Memory (LSTM) neural networks.

3.3.3 LSTM

LSTM [17] is an artificial recurrent neural network (RNN) architecture used in deep learning. RNNs take as input a sequence of vectors $(x_1, x_2, ..., x_n)$ at time t and return another sequence of vectors $(h_1, h_2, ..., h_n)$, the hidden state, which stores all useful information at (and before) time t. Although RNNs can theoretically learn long-term dependencies, in practice they fail to do so and tend to become biased towards the most recent inputs of the sequence. LSTMs were designed to address this problem by incorporating a memory cell, and it has been shown that they are capable of detecting long-term dependencies. This is achieved through the presence of multiplicative gates that control the amount of information from the previous state to forget and the information from the inputs to let through to the next memory cell; the state of the LSTM cell can be modified through an input gate that allows or blocks updates. Similarly, an output gate controls whether the cell state is communicated as output from the LSTM unit. The most commonly used version of LSTM also includes a forget gate that allows for resetting the cell state. Figure

3.11 shows the basic schema of an LSTM unit.

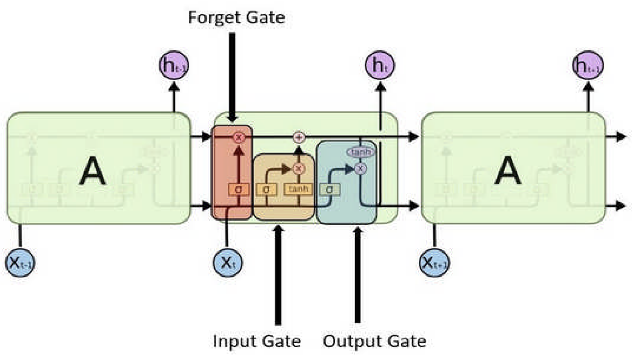


Figure 3.11: LSTM unit

The formulas for updating an LSTM unit at time t are:

- Input gate :

$$i_t = \sigma(W_i h_{t-1} + U_i x_t + b_i) \tag{3.16}$$

$$\tilde{c}_t = \tanh(W_c h_{t-1} + U_c x_t + b_c) \tag{3.17}$$

$$c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t \tag{3.18}$$

- Forget gate :

$$f_t = \sigma(W_f h_{t-1} + U_f x_t + b_f) \tag{3.19}$$

- Output gate :

$$o_t = \sigma(W_o h_{t-1} + U_o x_t + b_o) \tag{3.20}$$

$$h_t = o_t \odot \tanh(c_t) \tag{3.21}$$

where :

- σ is the sigmoid function
- $\odot$ is the product function
- x_t is the input vector at time t
- h_t is the hidden state vector that stores all the useful information at (and before) time t
- U_i, U_f, U_c, U_o denote the weights of the matrices for the different gates for input x_t
- W_i, W_f, W_c, W_o are the weight matrices for the hidden state h_t
- b_i, b_f, b_c, b_o denote the bias vectors

In the following section, we will introduce and discuss some commonly used forecast performance metrics to assess the accuracy of our models.

3.3.4 Forecast Performance Metrics

To evaluate the quality of predictions, the two criteria commonly used in ML are the root mean squared error (RMSE) and the mean absolute error (MAE).
The RMSE measures the difference between the predicted values and the actual values, taking into account the square of the differences. The RMSE is defined as (3.22):

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i - \hat{x}_i)^2} \tag{3.22}$$

where :

- x_i is the actual value of the i^{th} data point
- $\hat{x}_i$ is the predicted value
- n is the total number of data points

The MAE measures the average absolute difference between the actual and predicted values. The MAE is defined as (3.23):

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|x_i - \hat{x}_i| \tag{3.23}$$

where :

- y_i is the actual value of the i^{th} data point
- $\hat{x}_i$ is the predicted value
- n is the total number of data points

Now that we have discussed the state-of-the-art of time series, it is time to move on to the implementation phase of our project.

Stime : a Proof of Concept

As previously mentioned in the internship context and goals in section 1.2, two of Neoxia's clients, *OuestFrance* and *Stime*, expressed the need for detailed monitoring of their GCP costs. While this thesis will focus on *Stime* data, the overall goal is to develop a generic solution that can be deployed for Ouest France and other clients. To achieve this, a proof of concept will be conducted with *Stime*, and the chosen solution will be adapted for *OuestFrance* and other clients as needed. This proof of concept will encompass both the development and deployment phases of the project. First, we will begin by focusing on the development phase.

4.1 Development phase

The development phase is a crucial step in achieving our project objectives of developing an anomaly detection method and forecasting GCP costs for *Stime*. This phase will involve exploring and understanding the *Stime* data, building anomaly detection and forecasting models, and finally evaluating their performance. By the end of this phase, we aim to deliver a robust and effective solution that enables *Stime* to forecast their GCP costs and detect any anomalies that require attention.

4.1.1 Data Exploration

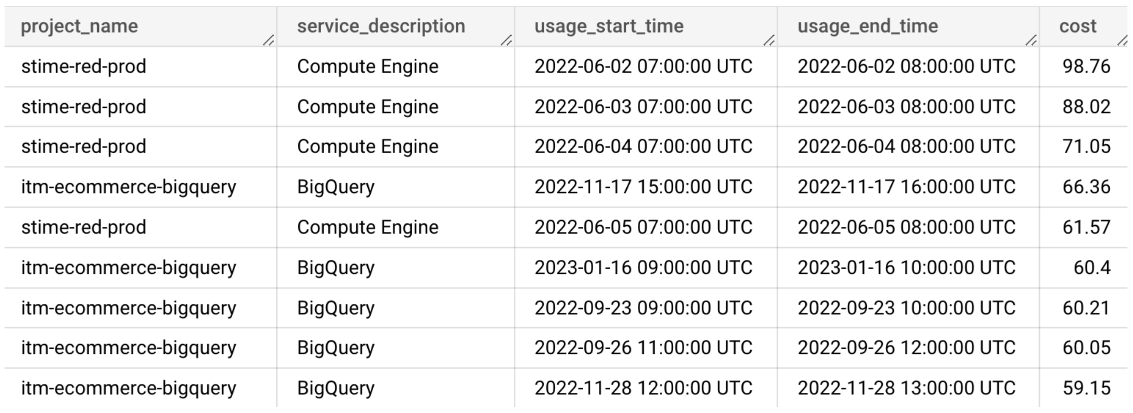
As mentioned in the Billing API section 2.3.1, the Billing export feature provided by GCP allows organizations to export detailed usage and billing information for all the used cloud resources. In this section, we will explore the structure of the billing export data and provide an example of what it looks like.
It is worth noting that organizations often create multiple projects in order to divide their resources into logical groups. However, when it comes to the monthly invoice for the organization, it represents the cumulative sum of resources across all projects.
The billing export data is structured as a table with many columns, each representing a

different aspect of the cloud resource usage and billing information. Some of the relevant columns for our analysis include:

- **Project Name**: The name of the project associated with the cloud resource usage.
- **Service Description**: A description of the used cloud service (e.g. Compute Engine, BigQuery).
- **Usage Start Time**: The start time of the cloud resource usage.
- **Usage End Time**: The end time of the cloud resource usage.
- **Cost**: The total cost of the cloud resource usage.

There are many other columns in the billing export data that provide additional information about the cloud resource usage and billing.

To illustrate what the billing export data looks like, we have included an example of *STIME* billing data in figure 4.1.



project_name	service_description	usage_start_time	usage_end_time	cost
stime-red-prod	Compute Engine	2022-06-02 07:00:00 UTC	2022-06-02 08:00:00 UTC	98.76
stime-red-prod	Compute Engine	2022-06-03 07:00:00 UTC	2022-06-03 08:00:00 UTC	88.02
stime-red-prod	Compute Engine	2022-06-04 07:00:00 UTC	2022-06-04 08:00:00 UTC	71.05
itm-ecommerce-bigquery	BigQuery	2022-11-17 15:00:00 UTC	2022-11-17 16:00:00 UTC	66.36
stime-red-prod	Compute Engine	2022-06-05 07:00:00 UTC	2022-06-05 08:00:00 UTC	61.57
itm-ecommerce-bigquery	BigQuery	2023-01-16 09:00:00 UTC	2023-01-16 10:00:00 UTC	60.4
itm-ecommerce-bigquery	BigQuery	2022-09-23 09:00:00 UTC	2022-09-23 10:00:00 UTC	60.21
itm-ecommerce-bigquery	BigQuery	2022-09-26 11:00:00 UTC	2022-09-26 12:00:00 UTC	60.05
itm-ecommerce-bigquery	BigQuery	2022-11-28 12:00:00 UTC	2022-11-28 13:00:00 UTC	59.15

Figure 4.1: Example of Billing Export Data

From this example, we can see that each row of the billing export data represents a specific usage instance of a particular resource, and contains detailed information about the resource usage and cost.
Now that we have explored the structure of the billing export data, we can move on to the pre-processing step, where we will extract and process the relevant information.

4.1.2 Data Processing

In this section, we will pre-process the billing export to create time series of daily costs for both the total cost of all resources used across all projects of STIME and the cost by project. This will allow us to analyze the organization's spending and identify any inefficiencies or opportunities for optimization.

To start, we will first focus on the total cost of all projects in the organization. Analyzing the total cost can provide a high-level overview of the organization's spending. We can pre-process the billing export data using the following SQL query to compute the total daily cost:

```
SELECT
  FORMAT_TIMESTAMP('%Y-%m-%d', usage_end_time) AS day,
  ROUND(IFNULL(SUM(cost),0),2) AS total_cost
FROM
  `billing_export_table`
GROUP BY
  day
```

This query calculates the daily cost for the total resources used across all projects of *STIME*. We can visualize the results of this query as a line plot from *January* to *May* 2023, as shown in Figure 4.2. The daily cost ranges from around 4000$ to 5000$, with a mean value of approximately 4500$. As we can see on the plot, the total cost exhibits a relatively stable seasonal pattern, with some fluctuations.

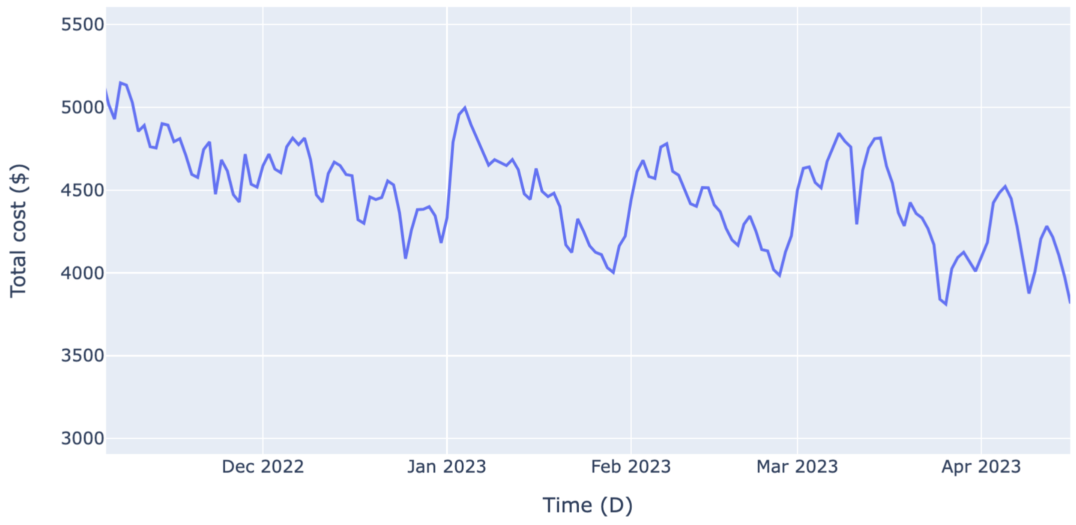


Figure 4.2: Daily total cost of all resources used across all projects of STIME from *January* to *May* 2023

To get a better understanding of how costs are distributed across projects, we can look at the cost by project. This will allow us to monitor the costs for individual projects and identify potential areas for optimization. We use the following SQL query to access the daily cost for each project:

```
SELECT
  project.name AS project_name,
  FORMAT_TIMESTAMP('%Y-%m-%d', usage_end_time) AS day,
  ROUND(IFNULL(SUM(cost),0),2) AS total_cost
FROM
  `billing_export_table`
GROUP BY
  day, project_name
```

However, as *STIME* has over 60 projects, covering all of them in this proof-of-concept would be time-consuming and difficult. Therefore, we have chosen to focus on three representative projects, *stime-red-preprod*, *itm-oneplace-prod*, and *stime-red-prod*, which were selected based on their size and complexity, and they are also among the projects that the business would like to closely monitor and track. These projects will serve as a guide for our analysis. We can visualize the daily cost from *January* to *July* 2022 for the specified projects in Figure 4.3. We can see that the three time series exhibit a monthly seasonal pattern.

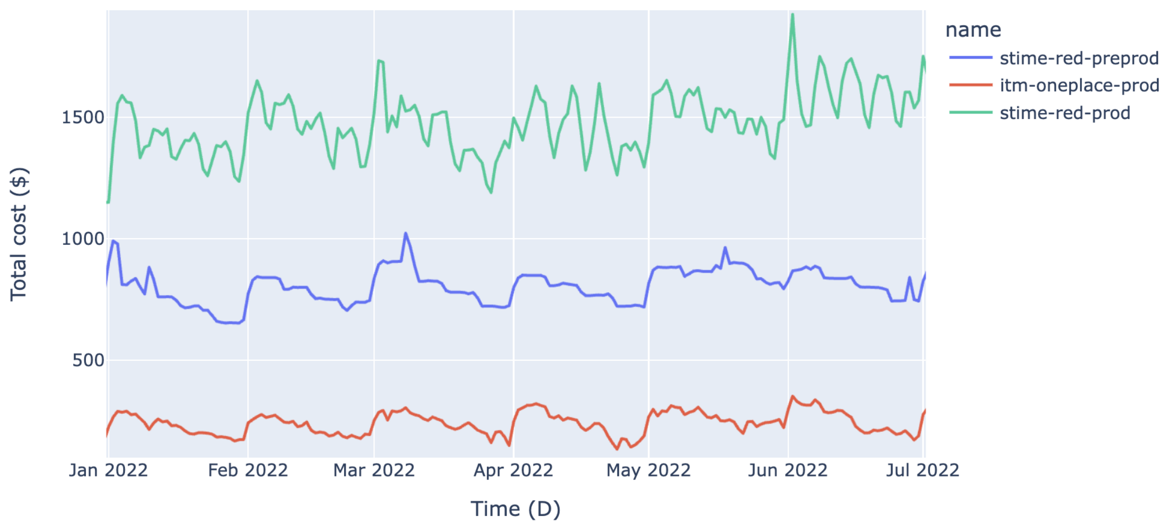


Figure 4.3: Daily total cost of different projects of STIME from *January* to *July* 2022

After constructing the time series for the daily cost of all resources and the cost by project, we are interested in detecting high costs using anomaly detection algorithms.

4.1.3 Anomaly detection models

In this section, we will compare different anomaly detection models to choose the best-performing one. Again, we will focus on the three projects, *stime-red-preprod*, *itm-oneplace-prod*, and *stime-red-prod*, over a certain period of time with a few anomalies.

For *STIME*, an anomaly is usually a higher GCP cost than usual. The client's objective is to receive timely alerts, allowing them to verify whether a high cost is a normal occurrence or if it indicates a potential resource failure. These alerts provide the client with the opportunity to investigate and determine the cause behind the elevated cost, ensuring proactive management of their resources. Firstly, we will present in the figure 4.4 the three projects with points that the client considers as true anomalies, which will serve as a ground truth to measure the performance of our models.

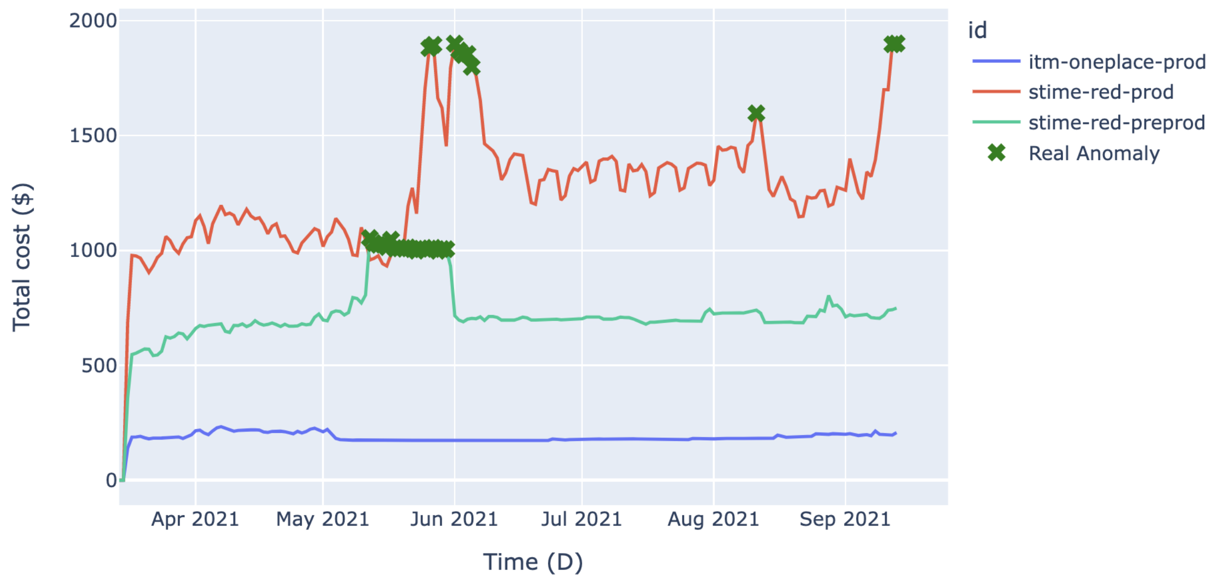


Figure 4.4: Ground truth anomalies in STIME's projects

Looking at the figure 4.4, we observe that the first project has several spikes with high values that are considered as anomalies. In contrast, the second project has a single prolonged period with a persistent anomaly, while the third project has no anomalies during this period.
Now, we will explore and compare the three different anomaly detection models presented in the state of the art 3.2 to draw conclusions.

4.1.3.1 Results and conclusions

First, we begin with the *Interquartile Range* (IQR) method.

1. **Interquartile Range (IQR)**:

We consider a point to be an anomaly if it is greater than the *upper bound* of the *decision range* discussed in 3.2.3.1, since we are only interested in high costs. Figure 4.5 shows the anomalies detected on the three projects using $IQR(\beta = 1.5)$:

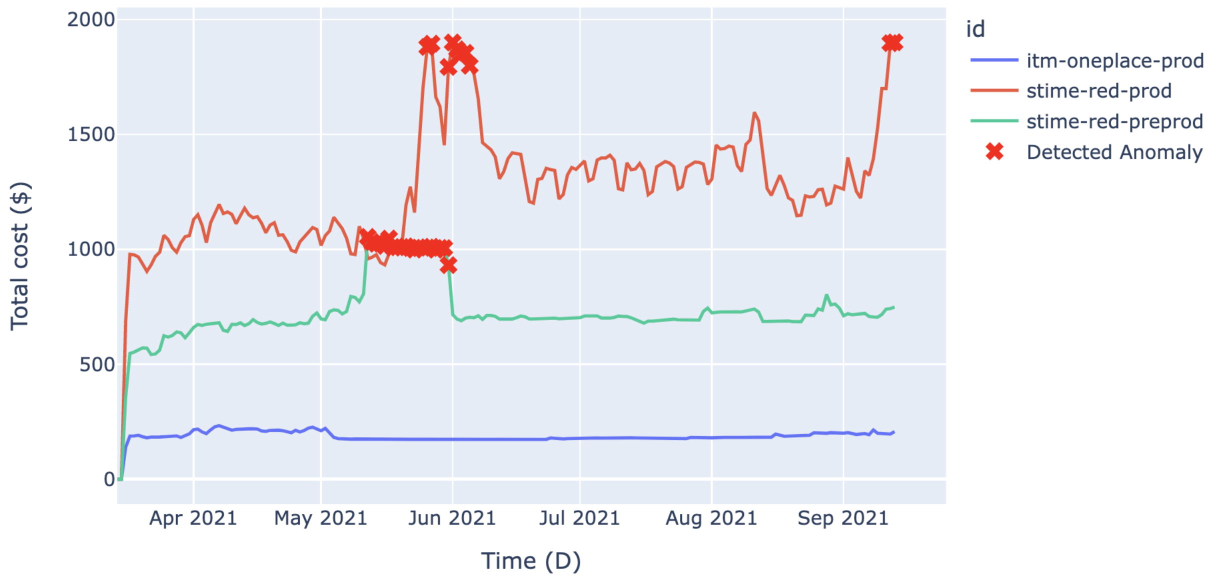


Figure 4.5: Detected anomalies using Interquantile range method

The following table shows the confusion matrix for the *IQR* model across the three time series:

TP = 28	FP = 1
FN = 2	TN = -

Table 4.1: Confusion matrix for IQR.

The following metrics are computed to evaluate the performance of the model:

- Precision = 0.97
- Recall = 0.93
- F1-score = 0.95

The computed metrics demonstrate a strong performance for the model, with an F1-score of 0.95, indicating a good balance between precision and recall. Now, we will delve into the second method.

2. **Local Outlier Factor (LOF)**:

The second method is Local Outlier Factor. The figure below 4.6 shows the detected anomalies using $LOF(k = 30)$ with the value of k tuned and set to 30:

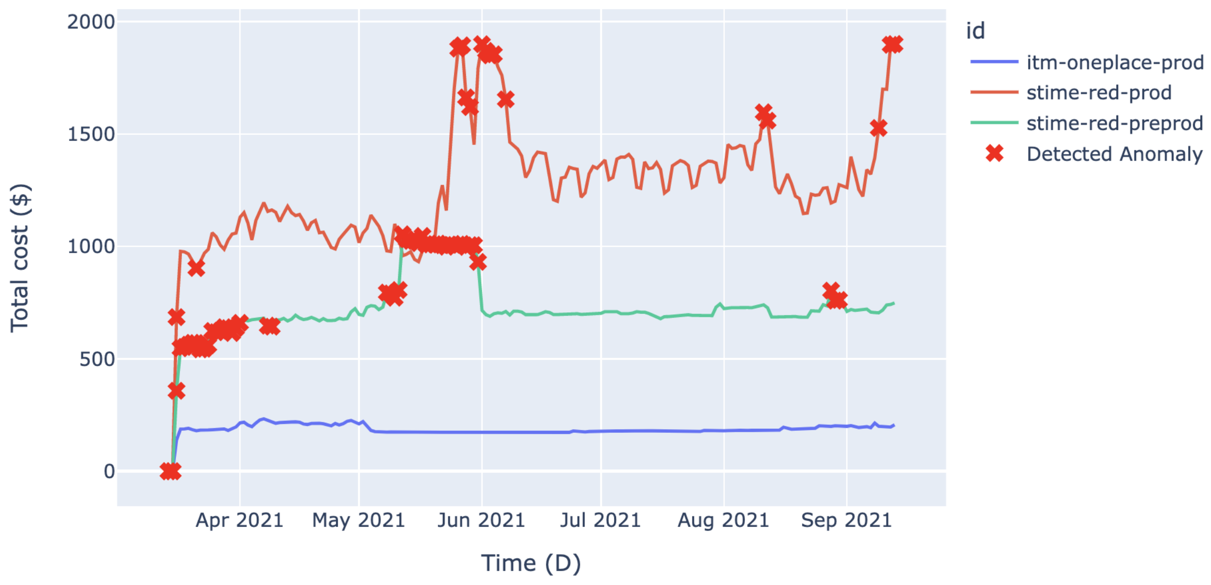


Figure 4.6: Detected anomalies using Local Outlier Factor method

The following table shows the confusion matrix for the *LOF* model across the three time series:

TP = 30	FP = 44
FN = 1	TN = -

Table 4.2: Confusion matrix for LOF.

The following metrics are computed to evaluate the performance of the model:

- Precision = 0.41
- Recall = 0.97
- F1-score = 0.57

The model's overall performance can be described as average, with a precision of 0.41, recall of 0.97, and an F1-score of 0.57. We will explore the potential of the third model.

3. **Isolation Forest (IF)**:

the last model is Isolation Forest. Figure 4.6 shows the detected anomalies using $IF(p = 0.02)$ with the anomaly percentage p tuned and set to 2%, :

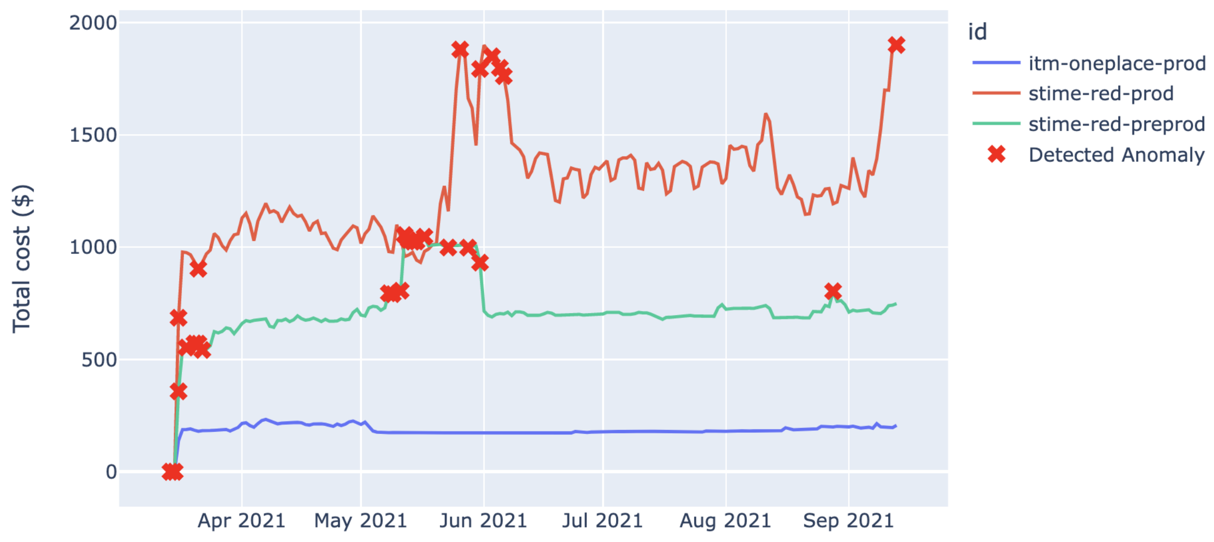


Figure 4.7: Detected anomalies using Isolation Forest algorithm

The following table shows the confusion matrix for the *IF* model across the three time series:

TP = 14	FP = 14
FN = 7	TN = -

Table 4.3: Confusion matrix for Isolation Forest.

The following metrics are computed to evaluate the performance of Isolation Forest:

- Precision = 0.50
- Recall = 0.67
- F1-score = 0.57

It is evident from the performance metrics of the three methods that the IQR outperforms the other two with an F1-score of 0.95, while LOF and IF only have an F1-score of 0.57. One of the strengths of IQR is its flexibility, as it only considers upper bounds to detect large values and omits lower bounds. This characteristic is particularly suitable for our use case, where our main objective is to identify only high costs. On the other hand, LOF and IF have a limitation because they may also detect small values as anomalies, which is not relevant in our context. Therefore, IQR seems to be the most appropriate method for detecting anomalies in our data.
Having completed the first task of this project, which involved developing an anomaly detection model, we now move on to the second task, which is to develop a time series forecasting model.

4.1.4 Forecasting models

In order to predict future costs, we will build forecasting models for four different time series: the daily total cost of all resources used across all projects first, and the three selected projects. To achieve this, we trained three different models - *ARIMA*, *LSTM*, and *Prophet* - for each of the four time series.
During the training phase, we have experimented different historical data lengths (1 year, 6 months, 2 months), and found that using a 6-month training period provides good performance for our models.
While *Prophet* doesn't require any parameter tuning, for the *LSTM* model, we used the same architecture for all projects, with two *LSTM* layers of 100 dimensions and a Dense layer with one output neuron; we use a sequence length of 60 to predict one point in the future, which means we use the past 60 daily total cost values as input to predict the next day's cost.
As for the *ARIMA* model, one important thing to note is that we chose to use *auto-ARIMA* because it automatically determines the appropriate parameters (p,d,q) and the seasonal parameters (P,D,Q,m) for each time series. This is very useful because determining these parameters manually can be time-consuming and difficult. By using a generic approach like *auto-ARIMA*, we can save a lot of time and effort while still achieving accurate forecasts. we used the *auto-ARIMA* algorithm to obtain the best parameters for the different time series.
After presenting the different models and how they are trained. For all our training, we use a period of approximately 6 months of data and predict 30 days into the future. Now, we will give you an example of forecasting for the four time series. We will show the predicted values of each model (ARIMA, Prophet, and LSTM) for each time series and compare their performances. Before presenting each figure, we will provide the parameters used for the ARIMA model. This will give us a better understanding of how each model performs and how they compare to each other.
Figure 4.8 shows the predicted costs for the daily cost across all STIME projects using the three models: *ARIMA*, *Prophet*, and *LSTM*. For *ARIMA*, the resulting model is `ARIMA(0,1,0)(0,1,0,30.5)`.

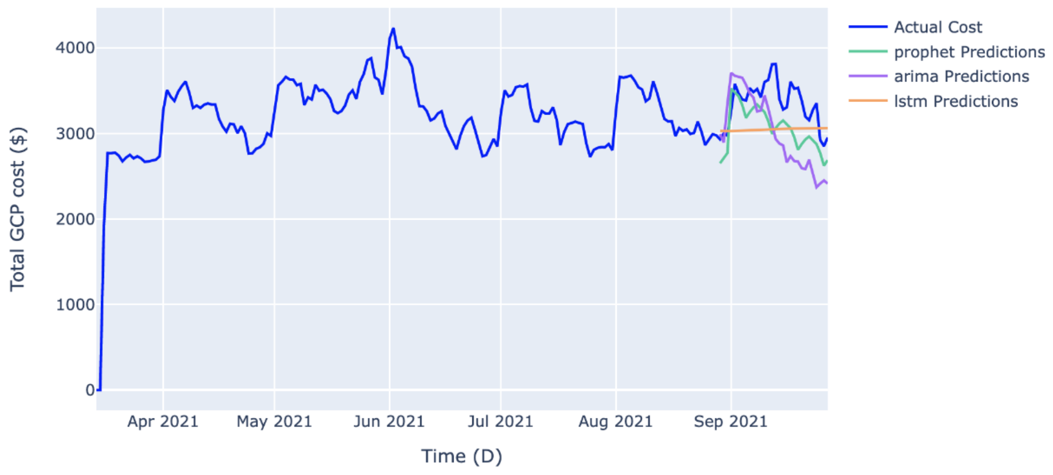


Figure 4.8: GCP cost forecasting for all the projects of STIME

Figure 4.9 shows the predicted costs for project *stime-red-prod* using the three models. Once again, the resulting *ARIMA* model is `ARIMA(2,1,1)(2,1,1,30.5)`.

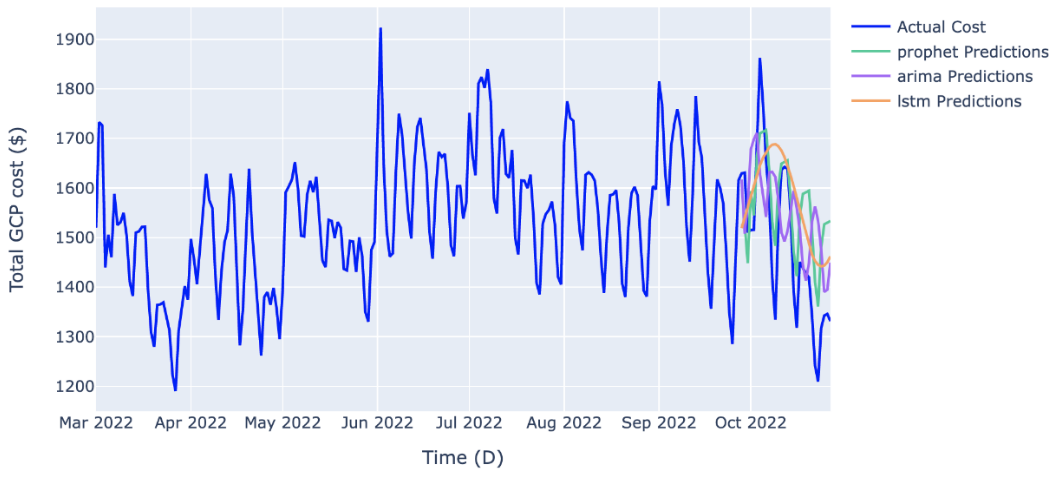


Figure 4.9: GCP cost forecasting for the project *stime-red-prod*

Figure 4.10 shows the predicted costs for project *stime-red-prod* using the three models: *ARIMA*, *Prophet*, and *LSTM*. The resulting *ARIMA* model is `ARIMA(1,1,1)(1,1,0,30.5)`.

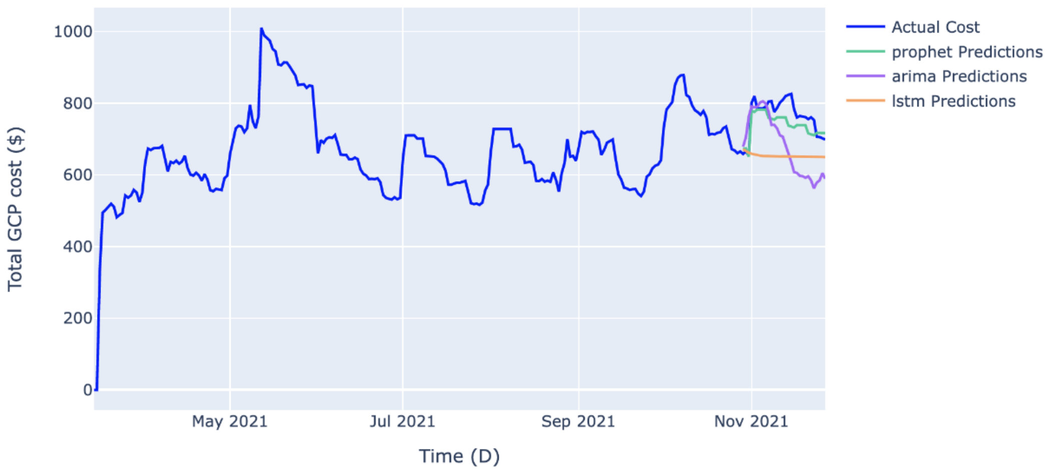


Figure 4.10: GCP cost forecasting for the project *stime-red-preprod*

Figure 4.11 shows the predicted costs for project *itm-oneplace-prod* using the three models. The resulting *ARIMA* model is `ARIMA(1,1,0)(0,1,0,30.5)`.

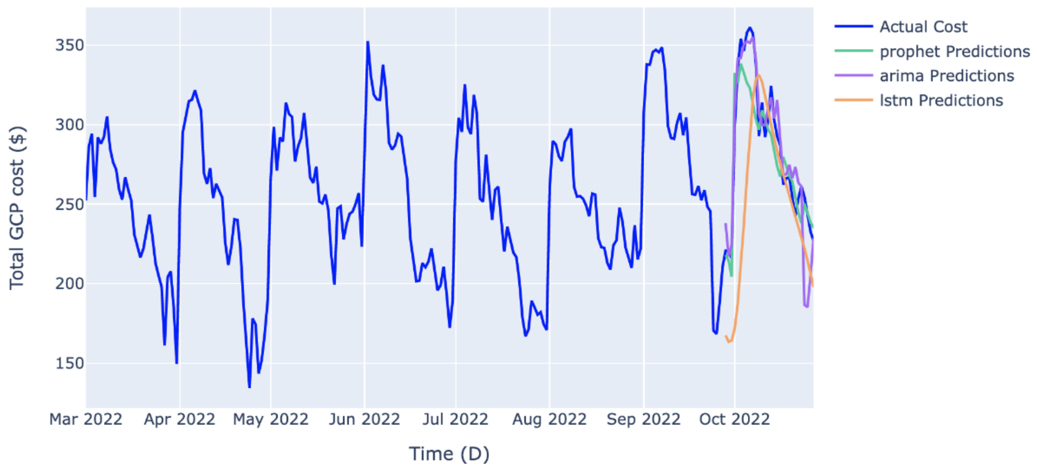


Figure 4.11: GCP cost forecasting for the project *itm-oneplace-prod*

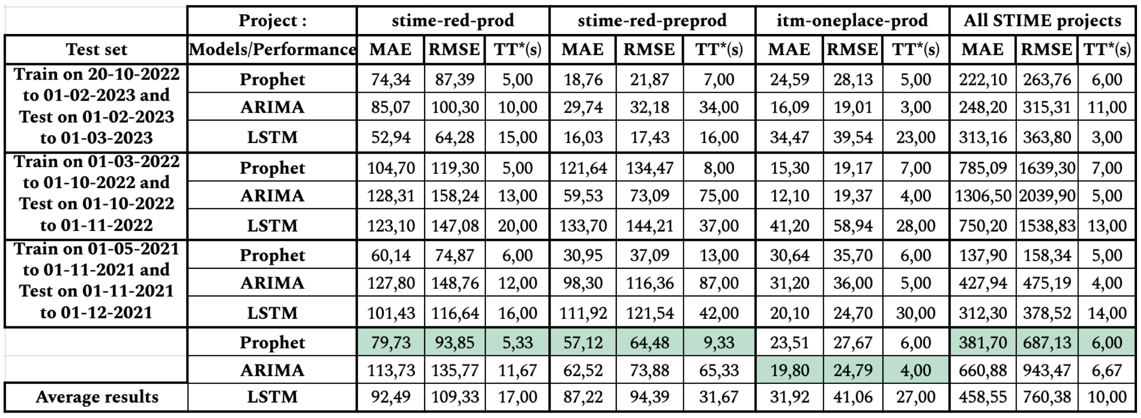
Based on these initial tests, we can conclude that the *ARIMA* and *Prophet* models perform well in detecting the seasonal pattern and capturing the overall trend of the time series. In contrast, the *LSTM* model predicts the average of the historical data for the next 30 days without detecting any clear pattern, except for two figures where a sinusoidal signal is observed. However, this signal is still far from resembling the actual curve.
Based on these initial tests, we cannot make a conclusive decision to choose the best model among the three presented. Therefore, we will conduct a testing phase and we will present the results and the conclusions in the next section.

4.1.4.1 Results and conclusions

For each of the four time series, we will split the data into three periods and evaluate the performance of each model :

- Training on `20-10-2022` to `01-02-2023` and testing on `01-02-2022` to `01-03-2023`
- Training on `01-03-2022` to `01-10-2022` and testing on `01-10-2022` to `01-11-2022`
- Training on `01-05-2021` to `01-11-2021` and testing on `01-11-2021` to `01-12-2021`

The performance metrics used are *RMSE*, *MAE*, and training time (*TT(s)*), which is the time taken by the model to train. We have compiled the test results in the figure 4.12, (Highlighted in Green: Best Average Performance).



	Project :	**stime-red-prod**			**stime-red-preprod**			**itm-oneplace-prod**			**All STIME projects**		
Test set	**Models/Performance**	**MAE**	**RMSE**	**TT*(s)**	**MAE**	**RMSE**	**TT*(s)**	**MAE**	**RMSE**	**TT*(s)**	**MAE**	**RMSE**	**TT*(s)**
Train on 20-10-2022 to 01-02-2023 and Test on 01-02-2023 to 01-03-2023	**Prophet**	74,34	87,39	5,00	18,76	21,87	7,00	24,59	28,13	5,00	222,10	263,76	6,00
	ARIMA	85,07	100,30	10,00	29,74	32,18	34,00	16,09	19,01	3,00	248,20	315,31	11,00
	LSTM	52,94	64,28	15,00	16,03	17,43	16,00	34,47	39,54	23,00	313,16	363,80	3,00
Train on 01-03-2022 to 01-10-2022 and Test on 01-10-2022 to 01-11-2022	**Prophet**	104,70	119,30	5,00	121,64	134,47	8,00	15,30	19,17	7,00	785,09	1639,30	7,00
	ARIMA	128,31	158,24	13,00	59,53	73,09	75,00	12,10	19,37	4,00	1306,50	2039,90	5,00
	LSTM	123,10	147,08	20,00	133,70	144,21	37,00	41,20	58,94	28,00	750,20	1538,83	13,00
Train on 01-05-2021 to 01-11-2021 and Test on 01-11-2021 to 01-12-2021	**Prophet**	60,14	74,87	6,00	30,95	37,09	13,00	30,64	35,70	6,00	137,90	158,34	5,00
	ARIMA	127,80	148,76	12,00	98,30	116,36	87,00	31,20	36,00	5,00	427,94	475,19	4,00
	LSTM	101,43	116,64	16,00	111,92	121,54	42,00	20,10	24,70	30,00	312,30	378,52	14,00
	Prophet	79,73	93,85	5,33	57,12	64,48	9,33	23,51	27,67	6,00	381,70	687,13	6,00
	ARIMA	113,73	135,77	11,67	62,52	73,88	65,33	19,80	24,79	4,00	660,88	943,47	6,67
Average results	**LSTM**	92,49	109,33	17,00	87,22	94,39	31,67	31,92	41,06	27,00	458,55	760,38	10,00

Figure 4.12: Performance comparison of ARIMA, Prophet, and LSTM models

After analyzing the average results presented in the table 4.12, it can be concluded that *Prophet* generally outperforms the other two models in terms of *RMSE* and *MAE*, while also having a smaller training time (*TT(s)*). Therefore, based on these results, *Prophet* is the chosen model to be deployed for the time series forecasting task.
Now that we have completed the first two tasks of the project, namely developing the anomaly detection model using IQR and the forecasting model using Prophet, it is time to move on to the next step, which is deploying these solutions.

4.2 Deployment phase

In this section, we will outline the steps for deploying the two developed models:

- The anomaly detection model will be deployed using an API called *Alerteam*.

- The forecasting model will be deployed using an API that displays the forecasts on a *Looker studio* dashboard.

We will start by exploring the *Alerteam* API.

4.2.1 Alerteam - Alerting API

Alerteam is the chosen name for the API that monitors the cost of Google Cloud Platform (GCP) resources by detecting cost overruns and sending alerts. Initially, alerts are limited to email, which may not be convenient for some users. However, *Alerteam* is flexible and can be customized to suit send alerts to teams via different collaboration spaces (*Mattermost* - *Slack* - *Teams*).
In this section, we will explore the architecture of *Alerteam*, the prerequisites for deployment, and the different features and capabilities it offers.

4.2.1.1 Architecture

The proposed deployment architecture aims to integrate alerts with the collaboration space of the organization, enabling users to receive notifications about anomalies in GCP costs via their preferred communication channel. Whether the organization is using *Slack*, *Mattermost* or *Teams*, the process for integrating alerts is essentially the same. The alerting process involves four main steps:

1. **Exporting billing data to BigQuery:** GCP billing data is exported to a BigQuery **??** table, where it can be easily queried and analyzed.
2. **Triggering a Cloud Function:** A Cloud Function 2.1.2 is created to implement an anomaly detection approach. It is triggered by a Cloud Scheduler 2.3.2 *cron* job in a daily schedule.
3. **Getting billing data from BigQuery :** The Cloud Function queries data from the exported data in BigQuery and uses IQR to detect anomalies.
4. **Sending alerts:** When an anomaly is detected, the cloud function sends an alert to the collaboration space via a configured *webhook*. It is a user-defined HTTP callback which is configured in the collaboration space of the company (e.g., Slack, Mattermost, or Teams).

Figure 4.13 shows a visual architecture of *Aletream*:

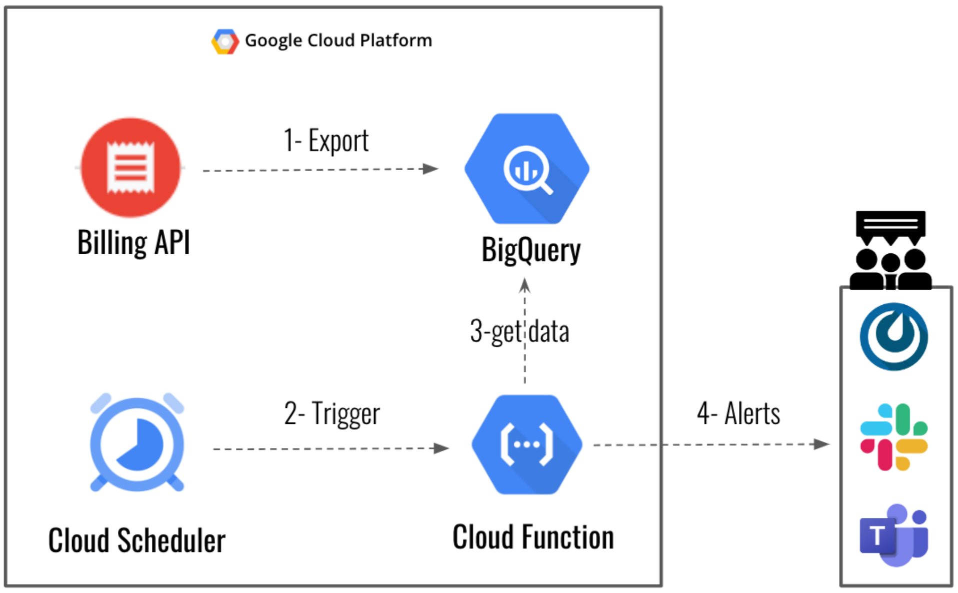


Figure 4.13: Architecture of *Alerteam*

The deployment process is achieved using *Terraform*, an infrastructure as code tool that enables the creation, modification, and versioning of cloud infrastructure. *Terraform* ensures that the entire deployment process is automated, making it easy to manage and maintain over time.
After describing the architecture of the API, it is important to understand the different parameters required to run it effectively. These parameters are mainly defined in the file *terraform.tfvars*. In this section, we will go through the different parameters and how they can be configured to deploy the *Alerteam* infrastructure.

4.2.1.2 Parameters

There are three main sections of parameters that we will cover. The first section is `params_deployment_project`, which contains parameters related to the project where *Alerteam* will be deployed. Finally, the second section is `default_webhook`, which is used to specify the default channel where alerts will be sent when high costs are detected on one or more projects. We will start by the first section of parameters :

1. `params_deployment_project`

The `params_deployment_project` section contains the following parameters:

- `project_name (string)`: the name of an existing project where we will deploy the API.

- `BQ_DATA_PATH (string)`: the path to the BigQuery table containing the billing data.
- `credentials (string)`: the path to the credentials JSON file for a service account. A service account is similar to a user login account, but it is used for applications and services rather than human users. *Terraform* will use a service account to authenticate with GCP and perform the necessary actions to deploy *Alerteam*. We need to create a service account and download its credentials in the form of a *JSON key* file, which will be saved locally.

An example configuration for `params_deployment_project` would be:

```
params_billing_export = {
    project_id = "stime-sandbox"
    BQ_DATA_PATH = "stime_billing_export_v1_987654_AB12CD_345F67"
    credentials = "../stime-key.json"
}
```

The second parameter is `default_webhook`:

2. `default_webhook`

The `default_webhook` parameter is used to specify the default channel where alerts will be sent when high costs are detected on one or more projects. By default, *Alerteam* tracks all active projects using an automated approach. If an alert is detected on one or more projects, it will be sent to the default webhook channel defined by the `default_webhook` variable:

```
default_webhook = "https://mattermost.stime.com/hooks/mj5u1h"
```

Alerteam comes with a default configuration that tracks all active projects using the Inter-Quantile algorithm. However, users may want to customize their configuration to meet their specific needs. we will see the difference between the default and customized configurations:

- Default Configuration

By default, *Alerteam* tracks all active projects using the *Inter-Quantile* algorithm. If an anomaly is detected on one or more projects, it will be sent to the default webhook channel defined by the `default_webhook` variable.

- Customized Configuration

In contrast, the customized configuration allows users to have more control over how alerts are triggered and where they are sent. Two additional features can be defined : `project_webhook_threshold` and `projects_to_ignore`.

The first feature is the variable `project_webhook_threshold`. It is a list of dictionaries with three fields: `project_name`, `webhook`, and `threshold`. It allows the user to define a custom webhook for a specific project, so alerts for that project will be sent only to that webhook and not to the `default_webhook`. It also allows the user to set a threshold for a project, which specifies the maximum amount in `dollars` that can be spent on the project in a day. In this case, the *inter-quantile* algorithm will not be used, and the alert will be sent immediately if the day's spend exceeds the defined threshold. This feature is useful for users who have a good understanding of their business and want to set specific thresholds for individual projects.
For example, consider the following configuration:

```
project_webhook_threshold = [{
    project_name = "stime-preprod"
    webhook = "https://slack.stime.com/hooks"
    },{
    project_name = "dns-staging"
    threshold = 200
    }]
```

In this configuration, the *"stime-preprod"* project will have anomalies detected using the *inter-quantile* algorithm, and alerts will be sent to *"https://slack.stime.com/hooks"*. On the other hand, for the *"dns-staging"* project, alerts will be sent to the default webhook if the day's spend exceeds *200 dollars*. This feature is useful for users who have a good understanding of their business and want to set specific thresholds or custom webhooks for individual projects.

Finally, the last feature is defined when specific projects need to be ignored, the `projects_to_ignore` variable can be defined as a list of project names. For example, if the project *"3dns-dev"* needs to be ignored, the variable can be defined as follows:

```
projects_to_ignore = ["3dns-dev"]
```

Now that we have a better understanding of the main parameters used to run our API, let's take a closer look at how our cloud function leverages these parameters to send alerts.

4.2.1.3 Alerteam approach

Alerteam is designed to track the costs of active projects. By active projects, we mean those with data for the previous day in the billing data. It is important to note that *Alerteam* cannot operate in real-time due to a delay in the export of billing data to BigQuery tables by the Billing API. Therefore, we wait until the next day to ensure all exported data is available for the previous day. *Alerteam* performs a daily check of yesterday's costs for all active projects and sends an alert if the cost is higher than expected. To do so, it uses a hybrid method between InterQuantile Range (IQR) and pre-defined thresholds to determine if an alert should be sent. The figure 4.14 shows the execution flowchart of the cloud function `alerteam.py`:

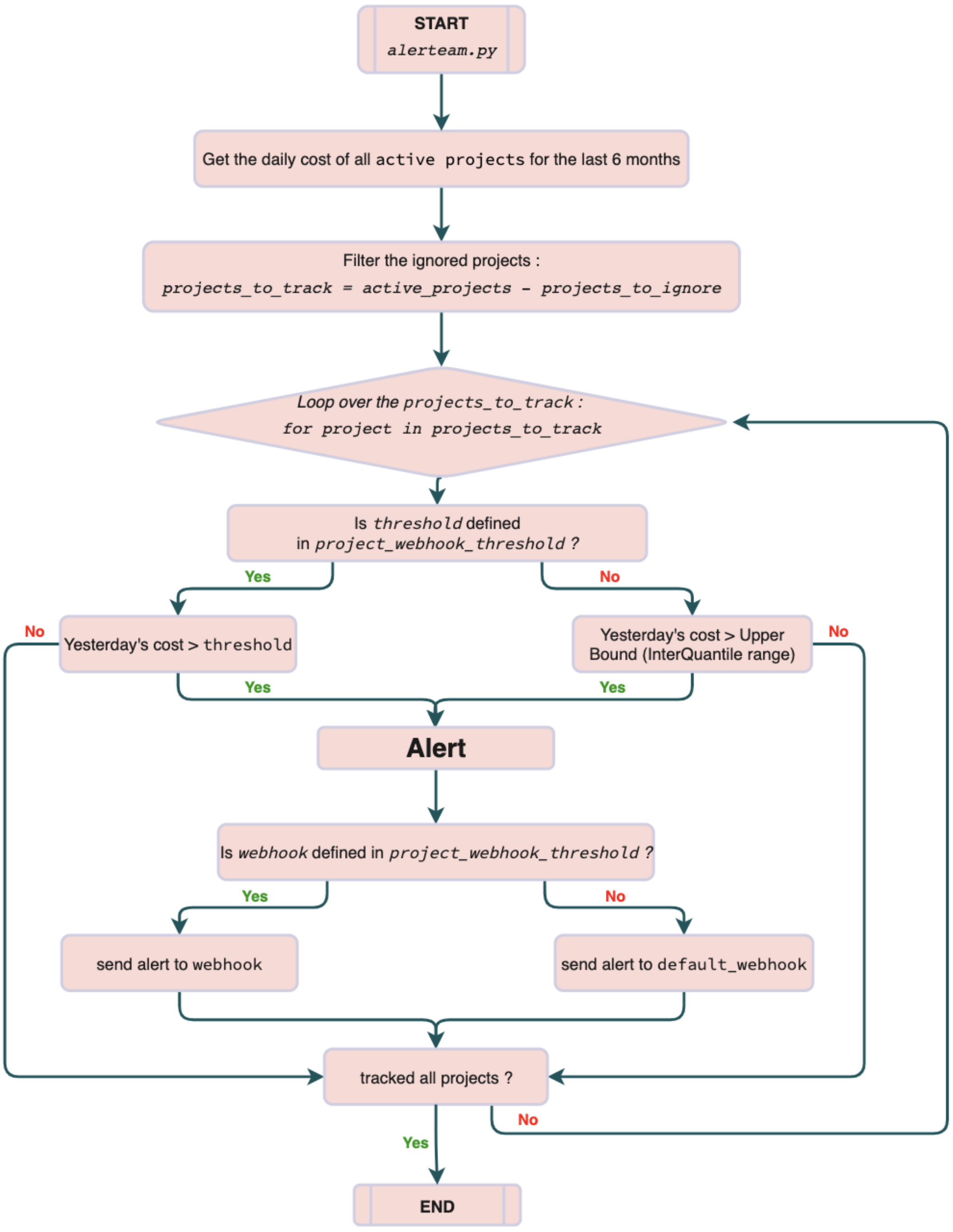

START
alerteam.py
Get the daily cost of all active projects for the last 6 months
Filter the ignored projects :
projects_to_track = active_projects - projects_to_ignore
Loop over the projects_to_track :
for project in projects_to_track
Is threshold defined in project_webhook_threshold ?
Yes
No
No
Yesterday's cost > threshold
Yesterday's cost > Upper Bound (InterQuantile range)
No
Yes
Yes
Alert
Is webhook defined in project_webhook_threshold ?
Yes
No
send alert to webhook
send alert to default_webhook
tracked all projects ?
No
Yes
END

Figure 4.14: Alerteam Cloud Function Execution Flow

Before deploying *Alerteam* with Terraform, there are some prerequisites that need to be met.

4.2.1.4 Prerequisites for Deployment

The first prerequisite is to enable the required APIs.

1. **Enable APIs :**

All the resource creation, update and deletion with *Terraform* are through a set of API calls. APIs in Google Cloud Platform (GCP) are disabled by default, so we need to have the following APIs enabled :

- **Compute Engine API**
- **Cloud Function API**
- **Cloud Scheduler API**

Once the required APIs have been enabled, the next step is to assign the necessary roles and permissions to the created service account in 4.2.1.2.

2. **Service account permissions :**

Terraform uses the created service account to deploy the API. However, we need to assign this service account the necessary roles in GCP. These roles define what the service account can and cannot do in the project. It is important to ensure that the service account has the correct roles assigned to it to avoid any issues during the deployment process. Therefore, we need to grant the created service account the following roles:

- **Cloud Functions Admin:** to create cloud functions
- **Cloud Functions Developer:** to deploy, update, and delete functions
- **Cloud Functions Service Agent:** Gives Cloud Functions access to managed resources (Google Storage Buckets for example).
- **Cloud Scheduler Admin:** to schedule the execution of the cloud function
- **BigQuery Data Viewer** and **BigQuery User:** to access and query the exported billing data in BigQuery.

Having covered the prerequisites for deployment, we will now move on to the process of deploying the API using *Terraform*.

4.2.1.5 Deployment with *Terraform*

The main resources we need to deploy *Alerteam* are:

1. Bucket resource
2. Bucket object resource which holds our code
3. Cloud function *alerteam.py*
4. Cloud scheduler to trigger our cloud function in a daily schedule

In *Terraform*, the resources that need to be created are specified in a file named *main.tf*. This means that the creation of the bucket, bucket object, cloud function and the cloud scheduler will be defined in the *main.tf* file. First, we will create a bucket resource.

1. **Bucket**

The bucket gets only a name, we add the following code in *main.tf*:

```
resource "google_storage_bucket" "bucket" {
  name = "alerteam_billing_alerts"
}
```

After that we have created the Cloud Storage bucket, the next step is to create an object within the bucket to store the code for our Cloud Function.

2. **Bucket Object**

A cloud function has the possibility to load a zip folder containing the code to run. Having a Python application running by Cloud Function, we need to put the *requirements.txt*[1] and a Python file *alerteam.py* in a zip folder named *alerting-api-code.zip*. Now, we will create the bucket object that contains *alerting-api-code.zip*:

```
resource "google_storage_bucket_object" "cloud-function-archive" {
  name   = "alerting-api-code.zip"
  bucket = google_storage_bucket.bucket.name
  #relative path to your alerting-api-code.zip file
  source = "./alerting-api-code.zip"
}
```

We have created the Cloud Storage object to store our code. With this resource in place, we can proceed to create the Cloud Function.

[1]a text file that lists the required Python modules for the function. which can then be installed automatically by the Cloud Function's runtime environment.

3. **Cloud function**

Now, it is time for our Cloud function to run *alerteam.py*, the following Terraform code deploys a Google Cloud Function. The code contains several parameters that configure the function's runtime environment, trigger settings, and access to resources:

```
resource "google_cloudfunctions_function" "function" {
  name        = "alerteam"
  runtime     = "python39"
  available_memory_mb = 256
  source_archive_bucket = google_storage_bucket.bucket.name
  source_archive_object =
      google_storage_bucket_object.cloud-function-archive.name
  trigger_http = true
  entry_point = "alerteam"
  environment_variables = local.env_vars
}
```

Here is an overview of the different parameters :

- `name`: specifies the name of the Cloud Function.
- `runtime`: specifies the runtime environment that the function will use. In this case, the function is written in Python 3.9.
- `available_memory_mb`: specifies the amount of memory (in MB) that is allocated to the function when it runs.
- `source_archive_bucket` and `source_archive_object`: specify the Cloud Storage bucket and object where the function's deployment package is stored. The deployment package *alerting-api-code.zip* contains all the code and dependencies required for the function to run.
- `trigger_http`: specifies that the function will be triggered by an HTTP request.
- `entry_point`: specifies the name of the function that will be invoked when the Cloud Function is triggered.
- `environment_variables`: sets the environment variables that will be available to the function at runtime. In this case, the `local.env_vars` variable is used to pass environment variables to the function. With reference to 4.2.1.2, `local.env_vars` is defined as:

```
locals {
  env_vars = {
    "billing_project_id" =
        var.params_deployment_project.project_name
    "BQ_BILLING_DATA_PATH" =
        var.params_deployment_project.BQ_DATA_PATH
    "DEFAULT_WEBHOOK" = var.default_webhook
    "project_webhook_threshold" = var.project_webhook_threshold)
    "projects_to_ignore" = var.projects_to_ignore
  }
}
```

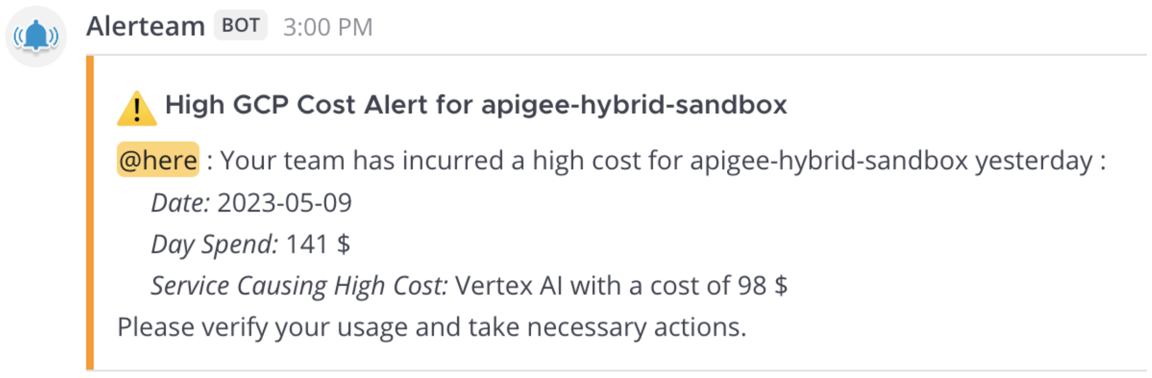
After creating the Cloud Function, the next step is to create a Cloud Scheduler job that will trigger an HTTP call to the function at regular intervals.

4. **Cloud scheduler**

We will configure the Cloud Scheduler job to run at *15:00 AM* every day in the Paris timezone using the `schedule` parameter. We will also specify the `http_target` parameter with the *uri* parameter set to the `https_trigger_url` for our Cloud Function. This will trigger an *HTTP* call to our Cloud Function at the specified schedule. The Cloud Scheduler job will be created using the following configuration:

```
resource "google_cloud_scheduler_job" "billing-alerts" {
  name             = "billingAlerts"
  schedule         = "0 15 * * *"
  time_zone        = "Europe/Paris"
  http_target {
    uri   = google_cloudfunctions_function.function.https_trigger_url
  }
}
```

With the *main.tf* and *terraform.tfvars* files in hand, the next step is to execute the command `terraform apply` to deploy *Alerteam*. Once deployed, the system will send alert messages when necessary. An example of the alert message sent by *Alerteam* can be seen in the following figure 4.15 :


Alerteam BOT 3:00 PM

⚠️ High GCP Cost Alert for apigee-hybrid-sandbox

@here : Your team has incurred a high cost for apigee-hybrid-sandbox yesterday :

Date: 2023-05-09

Day Spend: 141 $

Service Causing High Cost: Vertex AI with a cost of 98 $

Please verify your usage and take necessary actions.

Figure 4.15: Alert message sent by *Alerteam*

After successfully deploying the *Alerteam* API for our client *Stime*, the next step is to deploy our forecasting Prophet models.

4.2.2 Forecasting API

The forecasting API involves enhancing an existing dashboard at *Stime* by deploying our previously developed Prophet models to incorporate cost trends. First, let us delve into an overview of the existing dashboard.

4.2.2.1 Existing dashboard overview

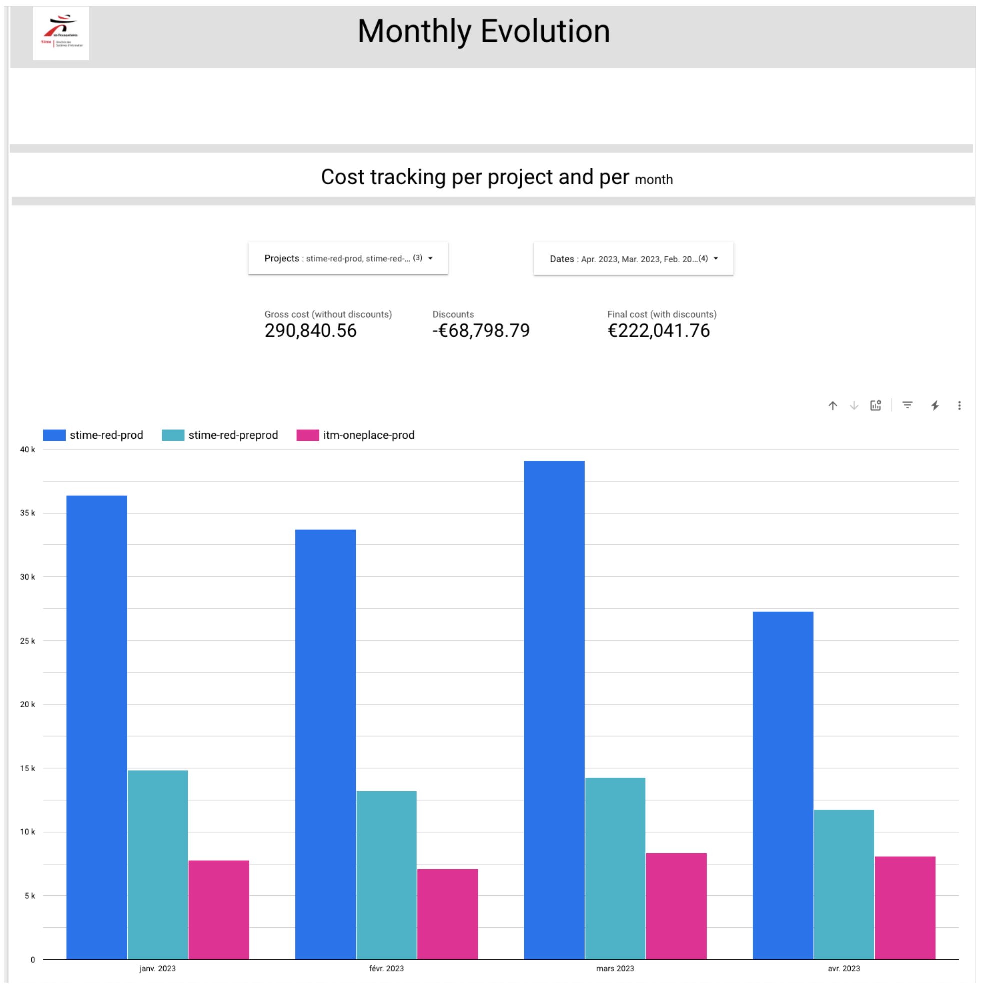
A cost tracking dashboard has already been developed for the client *Stime* using *Looker studio*. This dashboard provides valuable insights into cost monitoring and analysis. It consists of three pages, each serving a specific purpose:

- *Global cost tracking by project:* this page presents an overview of the overall cost for each project, allowing stakeholders to understand the cost distribution and identify projects with significant expenses.

- *Monthly cost tracking by project:* the second page focuses on monitoring the monthly evolution of costs for individual projects. It enables stakeholders to visualize and analyze cost trends, identifying any spikes or fluctuations in expenditure.

- *Cost tracking by GCP service:* the third page of the dashboard offers a breakdown of costs based on different Google Cloud Platform (GCP) services utilized. It provides valuable insights into the specific services contributing most to the overall costs.

Figure 4.16 illustrates the monthly cost evolution tracking for three projects: *stime-red-preprod*, *itm-oneplace-prod*, and *stime-red-prod*. It demonstrates how the dashboard visually represents the cost changes over time for each project.


Monthly Evolution

Cost tracking per project and per month

Projects : stime-red-prod, stime-red-... (3)

Dates : Apr. 2023, Mar. 2023, Feb. 20...(4)

Gross cost (without discounts)
290,840.56

Discounts
-€68,798.79

Final cost (with discounts)
€222,041.76

Figure 4.16: Existing dashboard

In the next sections, we will outline the proposed addition; the goal is adding a trends page to this existing dashboard to display estimated future cost trends. To achieve this, we plan to deploy forecasting models that will predict costs per project for the upcoming months. These predictions will then be used to populate the new page of the dashboard. This trends dashboard will provide stakeholders with valuable insights into the expected cost patterns, enabling them to make informed decisions and plan their budgets effectively. To build the trends page and enable cost forecasting within our dashboard, we will need to develop an API in the backend. This API will implement the Prophet model to generate accurate cost predictions before displaying them to the users. Now, we will dive into the architecture of this API.

4.2.2.2 Architecture

The proposed deployment architecture aims to integrate forecasts into the existing dashboard. The process involves six main steps:

1. **Exporting Billing Data to BigQuery:** GCP billing data is exported and stored in a BigQuery table for easy querying and analysis.

2. **Triggering the Cloud Function:** A Cloud Function is triggered on a scheduled basis using Cloud Scheduler. This function makes API calls to Vertex AI for the creation of machine learning models. Each project is associated with a Prophet model.

3. **Data preparation:** The Cloud Function prepares the required data (training and prediction data) to develop models.

4. **Model Development:** The Cloud Function interacts with Vertex AI to develop Prophet models for cost forecasting. These models use the daily cost data retrieved from BigQuery in step 3.

5. **Getting training data from BigQuery:** Vertex AI requires access to the daily cost data per project from BigQuery to train the Prophet models effectively.

6. **Storing Forecasts in BigQuery:** Once the Prophet models are created, the resulting forecasts are stored in BigQuery. This allows for easy access and retrieval for visualization purposes.

7. **Visualization in Looker Dashboard:** The Looker dashboard accesses the forecast tables in BigQuery and displays the predicted cost values in the newly created page.

Figure 4.17 shows a visual architecture of the Forecasting API:

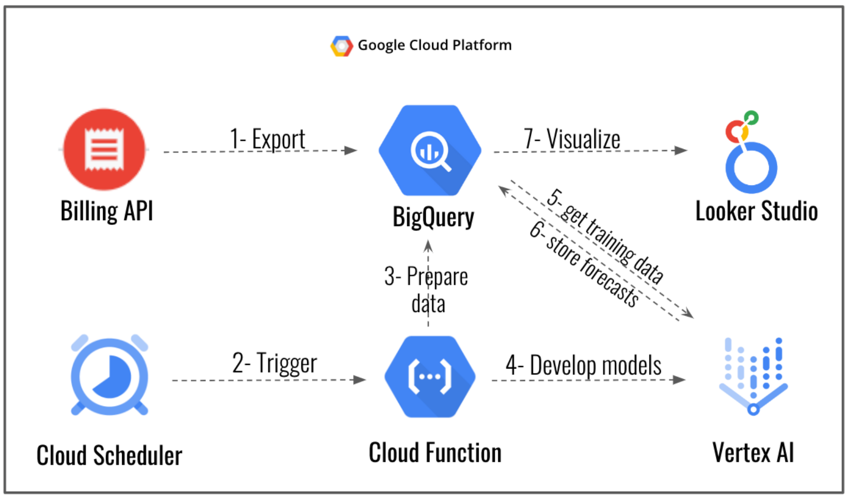


Figure 4.17: Forecasting API

Again, the deployment process is achieved using *Terraform*. We require enabling the same APIs as in the case of *Alerteam*, in addition to *Vertex AI API*. Again, similar to *Alerteam*, the deployment of the forecasting API requires the same main resources: Bucket, Bucket object, a cloud function and a cloud scheduler. These resources can be created using the same code provided for deploying Alerteam. This code can be reused and modified to accommodate the specific requirements of the forecasting API. Note that the creation of a new page in the existing dashboard to visualize the forecasts will be done manually and not with Terraform.
In contrast to *Alerteam*, the forecasting API is designed to be less configurable, as its primary purpose is to populate a dashboard with forecasts. Users of the dashboard do not need to concern themselves with the configuration aspects, except for three specific parameters that we will see in the next section.

4.2.2.3 Parameters

The first parameter is `forecast_horizon_days`:

1. `forecast_horizon_days`

The `forecast_horizon_days` variable determines the number of days for which the forecasting API will generate forecasts for each project. By default, this parameter is set to 30 days, providing a one-month projection of cost trends.

```
forecast_horizon_days = 30
```

The second parameter is `SERVICE_ACCOUNT`:

2. `SERVICE_ACCOUNT`

It specifies the corresponding credentials key JSON file of a service account. In analogy with the required permissions and roles for the service account used by *Alerteam*, we will need the same set of permissions for the service account of the forecasting API. However, we will add the *Vertex AI Admin* role. This role is used for developing and training forecasting models.

```
service_account_credentials = "../stime-key.json"
```

Finally, the last parameter is BILLING_EXPORT_BQ_DATA_PATH which is the path to the BigQuery table containing the billing data:

3. BILLING_EXPORT_BQ_DATA_PATH

```
BILLING_EXPORT_BQ_DATA_PATH =
    "stime_billing.stime_billing_export_v1_987654"
```

A crucial part of the forecasting API is preparing data for the development of Machine Learning models.

4.2.2.4 Data preparation

Using the exported data located at BILLING_EXPORT_BQ_DATA_PATH, the forecasting API requires three fixed parameters that represent the table paths for model development by Vertex AI. To accommodate this, we create a dataset named *forecasting_api* within the deployment project *stime_billing*, which consists of three tables: train, pred, and *forecasts*. The DATA block defines the paths of these three tables : the path of the training data (TRAINING_DATA_BQ_PATH), input data for making predictions (PREDICTION_INPUT_BQ_PATH), and storing the forecast results in BigQuery (FORECASTS_DATASET_BQ_PATH).

```
DATA = {
    TRAINING_DATA_BQ_PATH = "stime_billing.forecasting_api.train"
    PREDICTION_INPUT_BQ_PATH = "stime_billing.forecasting_api.pred"
    FORECASTS_DATASET_BQ_PATH = "stime_billing.forecasting_api.forecasts"
}
```

To develop our models, we need to prepare the training data which is located at a BigQuery table TRAINING_DATA_BQ_PATH. It consists of a table containing the columns *Day*, *project_name*, and *cost*. This table holds six months of historical data for each unique *project_name*. Once the model is trained, we can make predictions using it. However, for the model to generate accurate forecasts, it requires input data. Thus, we create a second table, known as the prediction input table, located at PREDICTION_INPUT_BQ_PATH. This table adheres to a predefined structure set by *Vertex AI*, featuring a *Day* column for timestamps, an identifier for each time series (*project_name*), and `NULL` values in the *cost* column, which is the target for prediction.

The resulting forecasts generated by the models are stored in the `FORECASTS_DATASET_BQ_PATH` table, which is created by Vertex AI. This table contains the *Day* column for timestamps, a *project_name* column, and a *predicted_cost* column. The *predicted_cost* column represents the forecasted costs by Vertex AI for the designated forecast horizon.
Understanding how Vertex AI creates and uses these models for predictions is important.

4.2.2.5 Models development in Vertex AI

Google Cloud offers a pipeline for Prophet model training and a pipeline for Prophet batch prediction.Because Prophet models can only fit a single time series, the training pipeline uses a Vertex AI Custom Training Job to train multiple Prophet models in parallel. The number of models trained equals the number of unique values in the `time_series_identifier_column` parameter. The pipeline and the parameter values are defined by the following function:

```
(
    train_job_spec_path,
    train_parameter_values,
) = utils.get_prophet_train_pipeline_and_parameters(
    time_column="Day",
    time_series_identifier_column="project_name",
    target_column="cost",
    forecast_horizon=var.forecast_horizon_days,
    data_source_bigquery_table_path=var.DATA.TRAINING_DATA_BQ_PATH
)
```

The following sample code demonstrates how to run a Prophet model training pipeline:

```
job = aiplatform.PipelineJob(
    template_path=train_job_spec_path,
    parameter_values=train_parameter_values
)
job.run()
```

Like with the training job, the prediction pipeline is defined as:

```
(
    prediction_job_spec_path,
    prediction_parameter_values,
) = utils.get_prophet_prediction_pipeline_and_parameters(
    model_name=model,
    time_column="Day",
    time_series_identifier_column="project_name",
    target_column="cost",
    data_source_bigquery_table_path=var.DATA.PREDICTION_INPUT_BQ_PATH,
    bigquery_destination_uri=var.DATA.FORECASTS_DATASET_BQ_PATH
)
```

We need to specify two important parameters: `data_source_bigquery_table_path` and `bigquery_destination_uri`. The `data_source_bigquery_table_path` parameter refers to the BigQuery table path in a specific format. This table contains the data that will serve as input for the forecasting model. Additionally, the `bigquery_destination_uri` parameter specifies the desired destination dataset where the generated forecasts will be stored. We launch the prediction pipeline using the *job.run()* function, similar to how we executed the training pipeline. Now that we have understood the data preparation and model training processes, we will explore how these two tasks are orchestrated.

4.2.2.6 Forecasting process

The forecasting process, executed by the cloud function, follows a specific sequence of tasks that occur every `forecast_horizon_days`:

1. The training (`train`) and prediction (`pred`) tables are overwritten to ensure the latest data is used.

2. The models are trained, with each *project_name* having its own dedicated model.

3. Forecasts are generated for each project, looking `forecast_horizon_days` into the future.

4. The forecast results are overwritten in the specified table `FORECASTS_DATASET_BQ_PATH`.

To ensure the proper execution order of these tasks, the cloud function first performs task 1, overwriting the training and prediction tables. This ensures that the models have access to the most up-to-date data for training. Once the forecasts have been stored in the table specified by `FORECASTS_DATASET_BQ_PATH`, they can be visualized on a new page in the dashboard, this page will be created manually in Looker studio.
With the deployment of the forecasting API underway, we are successfully achieving the objectives of this thesis. Moving forward, the following section will provide a summary of my work during this internship and highlight the personal and professional contributions I have made.

Future work and Conclusion

In conclusion, this thesis has made significant contributions to the field of cost management in Google Cloud Platform (GCP). The successful development and deployment of the *Alerteam* API for cost monitoring and anomaly detection have provided *Neoxia*'s client, *Stime*, with a powerful tool to detect high costs and take prompt action. This has enabled *Stime* to optimize their cloud spending and ensure efficient resource allocation. The next step in the project is to deploy the forecasting API, which will enhance the existing dashboard by providing valuable insights into future cost trends. This will empower stakeholders to make informed decisions, proactively manage their budgets, and avoid potential budget overruns.

From a professional standpoint, this internship has been a remarkable learning experience. Working alongside talented professionals at *Neoxia* has exposed me to a wealth of knowledge and expertise, enabling me to build a strong professional network. On the technical front, I have gained proficiency in various GCP services and obtained the Professional Data Engineer certification, solidifying my understanding of cloud technologies. I have also developed essential coding and packaging skills to ensure maintainable and readable code. Additionally, I have gained hands-on experience in designing API architectures and deploying them using infrastructure-as-code tools like *Terraform*. Moreover, I had the opportunity to apply agile methodologies by utilizing tools like *Trello* for managing tickets and sprint planning. This internship has been a truly enriching experience both personally and professionally, providing valuable insights into the industry and equipping me with a diverse skill set.

Overall, this thesis has not only addressed the challenges of cost management in GCP but has also contributed to the growth of knowledge in the field. The implemented solutions and the lessons learned during this internship will have a lasting impact on the organization and its clients. This experience has been instrumental in shaping my career as a data scientist and engineer, and I am confident that the skills acquired will pave the way for future success in the industry.

Bibliography

[1] Bigquery documentation. "https://cloud.google.com/bigquery".

[2] Cloud billing api. "https://cloud.google.com/billing/docs/how-to/budget-api-overview".

[3] Cloud scheduler : Fully managed cron job service. "https://cloud.google.com/scheduler".

[4] Compute engine documentation. "https://cloud.google.com/compute/docs".

[5] Google cloud platform. "https://cloud.google.com".

[6] Hawkins, d.m.: Identification of outliers. springer netherlands (1980). "https://doi.org/10.1007/978-94-015-3994-4".

[7] Isolation forest. "https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.IsolationForest.html".

[8] Lof: identifying density-based local outliers. "https://dl.acm.org/doi/10.1145/335191.335388#sec-ref".

[9] Looker studio. "https://cloud.google.com/looker-studio".

[10] Partial autocorrelation function (pacf). "https://towardsdatascience.com/a-step-by-step-guide-to-calculating-\autocorrelation-and-partial-autocorrelation-8c4342b784e8".

[11] Prophet: Automatic forecasting procedure. "https://pypi.org/project/fbprophet/".

[12] Vertex ai. "https://cloud.google.com/vertex-ai".

[13] What are google cloud functions? "https://cloud.google.com/functions/docs/concepts/overview".

[14] Kriegel H. P. Ng R. T. BREUNIG, M. M. et J. (2000) SANDER : Taylor sj, letham b. 2017. forecasting at scale. `"https://doi.org/10.7287/peerj.preprints.3190v2"`.

[15] Shivam CHAUDHARY : towardsdatascience : Why “1.5” in iqr method of outlier detection?

[16] Cleveland W. S. McRae J. E. Terpenning I. J. CLEVELAND, R. B. : *STL: A seasonal-trend decomposition procedure based on loess.* Journal of Official Statistics, 1990.

[17] J. (1997) HOCHREITER, S. et Schmidhuber : Long short-term memory. neural computation, 9(8), 1735–1780.

[18] Athanasopoulos G. (2018) HYNDMAN, R.J. : Forecasting: principles and practice, 2nd edition, otexts: Melbourne, australia. otexts.com/fpp2.