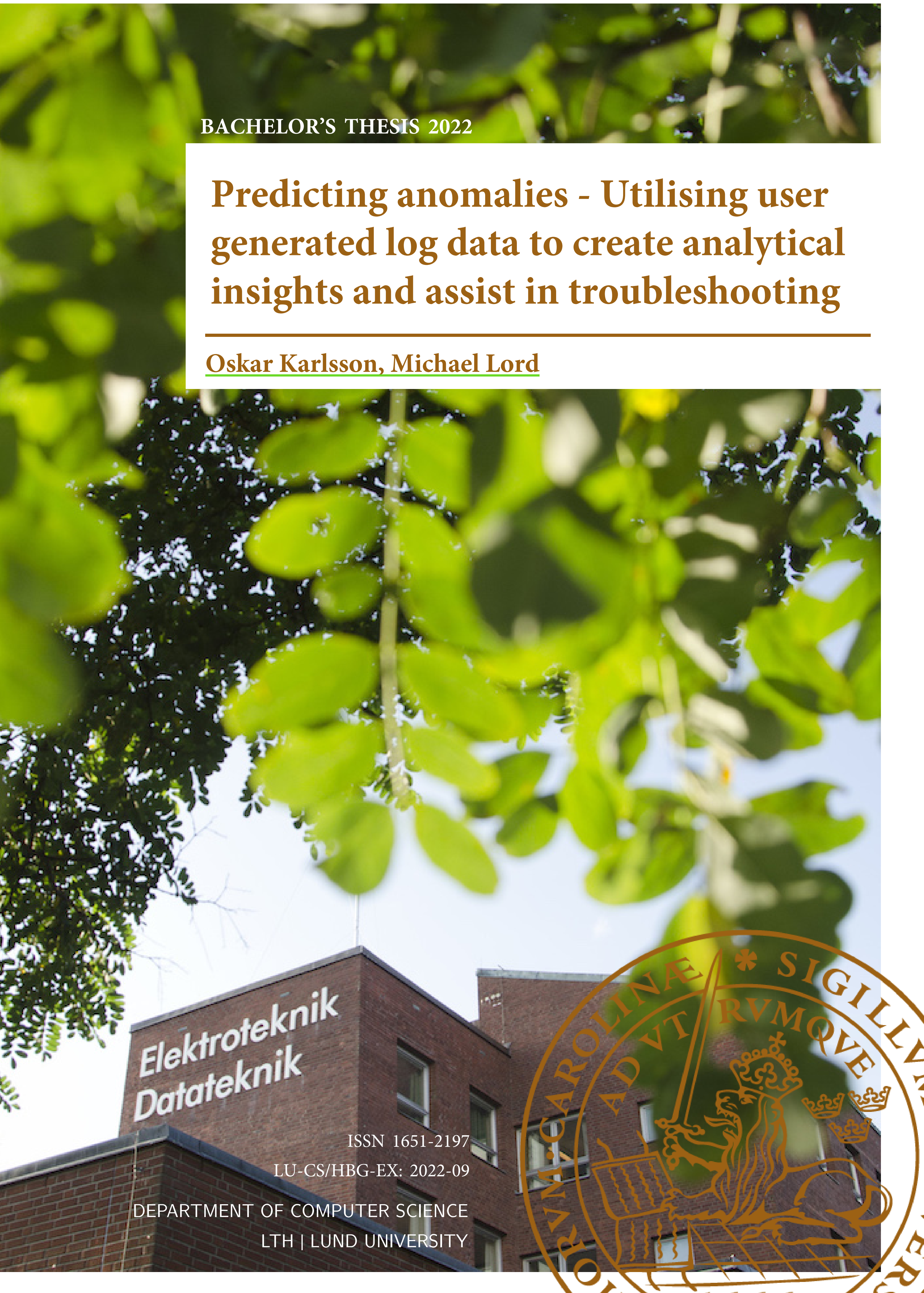


BACHELOR'S THESIS 2022

Predicting anomalies - Utilising user generated log data to create analytical insights and assist in troubleshooting

Oskar Karlsson, Michael Lord

ISSN 1651-2197
LU-CS/HBG-EX: 2022-09

DEPARTMENT OF COMPUTER SCIENCE
LTH | LUND UNIVERSITY

Predicting anomalies

Utilising user generated log data to create analytical insights and assist in troubleshooting

LUND UNIVERSITY
Campus Helsingborg

LTH School of Engineering at Campus Helsingborg
Department of Computer Science

Bachelor thesis:
Oskar Karlsson
Michael Lord

Abstract

A lot of log data is generated from products these days but it is not always used for analytical purposes. If that log data could be analysed and used for detecting when abnormal behaviour is occurring it could provide an efficient way to notice problems in the product and also assist in troubleshooting said problems. Log data generated by users of the online shopping cart at IKEA IT AB were analysed to create an anomaly detection prototype on the Google Cloud Platform. This prototype alerted when user activity was noted to be outside thresholds that were calculated with standard deviation on the hourly web user's cart usage in Country one. It also present relevant data for troubleshooting in a dashboard created in Google Data Studio if abnormal behaviour was found. The resulting prototype is a proof of concept that looked at a subset of log data but still proves the possibility of determining abnormal system behaviour from user generated data. The usefulness of such monitoring for companies in many fields can be large but future development of faster and more reliable models and alerting are needed.

LTH School of Engineering
Lund University
Box 882
SE-251 08 Helsingborg
Sweden

LTH Ingenjörshögskolan vid Campus Helsingborg
Lunds universitet
Box 882
251 08 Helsingborg

Printed in Sweden
Lunds universitet
Lund 2022

Keywords

Anomaly detection
BigQuery
Cloud computing
Google Cloud Platform
Troubleshooting
Web analytics

Sammanfattning

Mycket loggdata genereras från produkter nuförtiden men används inte alltid för analytiska ändamål. Om loggdata kunde analyseras och användas för att upptäcka onormalt beteende i ett system så kan det vara ett effektivt sätt att upptäcka problem i produkten och även hjälpa till med att felsöka dessa problem. Loggdata som genererats från användare av online-varukorgen på IKEA IT AB analyserades för att skapa en prototyp på Google Cloud Platform som upptäcker anomalier. Prototypen larmade när användaraktivitet noterades vara utanför tröskelvärdena som beräknades med standardavvikelse på den timvisa kundvagnsanvändningen i Tyskland. Om onormalt beetende upptäcktes presenterade prototypen även relevant data för felsökning i en instrumentpanel skapad i Google Data Studio. Den resulterande prototypen var bara ett proof of concept som tittade på en delmängd av loggdata men ändå bevisar möjligheten att fastställa onormalt systembeteende från användargenererad data. Nyttan av sådan övervakning kan vara stor för företag inom många områden men framtida utveckling av snabbare och mer pålitliga modeller och larm behövs.

Nyckelord

BigQuery
Datormoln
Detektering av anomali
Felsökning
Google Cloud Platform
Webbanalys

Forewords

In front of you lies the thesis ”Predicting anomalies”, the basis of which is a prototype utilizing user generated log data to create analytical insights and assist in troubleshooting. This thesis was written to fulfill the graduation requirements of the Computer Science and Engineering Bachelor’s program at Lunds Tekniska Högskola (LTH).

This project was undertaken at the request of IKEA IT AB where, together with our supervisor Ronny Roos, we formulated the goals and purpose of the thesis. The project has been difficult but very educational and many of the experiences gained during development will be of great value in our future career.

We want to thank Ronny Roos, Milad Yarahmadi and the cart team at IKEA IT AB for the opportunity of doing this thesis. The experience has been positive throughout thanks to the great expertise and welcoming attitude that we were met with from the first day. Furthermore, we wish to thank our supervisor and examiner, Christian Nyberg and Christin Lindholm, from LTH for the support and preparation given both during the project and during the program. We also want to thank family, friends and loved ones that have been supportive and encouraging throughout the course of the project.

We hope you enjoy reading.

Oskar Karlsson & Michael Lord

Helsingborg, June 7, 2022

Contents

Chapter 1

Introduction

This chapter explains the way this thesis came to. By going through the underlying problems, ideas and thoughts it will lead to the conclusion of what is to be examined.

1.1 Background

Customers all around the world uses IKEA IT AB's internet store to purchase items they want by adding them to the site shopping cart. This cart changes depending on the users actions such as removing or adding more items. Every change to the cart leaves a footprint event in a log table that is stored by IKEA IT AB for 30 days. IKEA IT AB, the company that is mainly responsible for IKEA's IT, calculates that they usually experience up to 300 log entries per second during high traffic. IKEA IT AB mainly uses GCP to run and host their web services.

The user generated log data are not used for any data analyses. IKEA IT AB are therefore interested in analysing and visualising this data to potentially discover errors or abnormal trends in the usage of the site shopping cart. An example of a potential error could be coupon codes that have been wrongly implemented or problems with the general usage of the cart. Discovering errors and trends in an early stage would enable earlier troubleshooting before any regular error reports are received from customers, other departments or automatic monitoring at IKEA IT AB.

The analysis will be visualised using *Google Data Studio*. A possible solution would be to use these graphs and analytical data to generate alerts when defined thresholds are exceeded. These thresholds could be upper and lower boundaries that encapsulate normal operation. E.g. if the number of log data during an interval drastically rise, the threshold would be exceeded and an alert would be generated notifying of abnormal operation.

Alerts will be used to notify IKEA IT AB about abnormal use which could indicate anomalies. Indication of anomalies can then be used to facilitate troubleshooting. More precisely, the visualised graphs will be a tool to clearly convey changes and behaviour patterns to assist in troubleshooting abnormal user activity.
In summary, IKEA IT AB hopes to be able to identify errors and visualise trends with the prototype that will be developed during this project.

Furthermore, the data requests to the GCP dataset must be done in an efficient way to not incur large cloud fees. Usage of too much system resources could lead to costs that would make it unfeasible to implement the system in a production environment.

This bachelor thesis is done in collaboration with IKEA IT AB.

1.2 Purpose

The purpose of this thesis is to investigate if a proof of concept can be created that analyses user generated log data to find anomalies and if the same data can be used to assist troubleshooting.

1.3 Goals

This thesis will investigate how to efficiently read large amounts of user generated log data from a dataset and if it's possible to visualise that data and generate alerts to notify of anomalies. If this is possible, then a prototype will be made and evaluated in a simulated environment. This prototype will include a dashboard that visualises relevant data to aid troubleshooting. Thereafter, the prototype will be improved to lessen it's cloud costs. The expected result of this prototype is to be informed of potential problems through user generated log data and thus allowing for improved troubleshooting.

1.4 Problems

The following questions are to be answered in this dissertation:

1. What data can be extracted from the logs?
2. How can the project be run cost-efficiently?
3. How to implement the thresholds to reduce, or increase the amount of alerts?
4. What data to visualise for assisting in troubleshooting?
5. Which mathematical models can be used to perform relevant data analysis?
6. What user generated log data are of interest for finding potential anomalies?

1.5 Motivation

A previous error in a coupon that made the whole cart free was one of the reasons for this thesis. That error increased the user activity to such a degree that the systems in turn slowly went down, both which caused a large loss of profit for IKEA IT AB. Therefore, this type of anomaly detection that look at user activity for anomaly detection was of interest.

On the end of the authors of this thesis, the motivation was partly working with a large company such as IKEA IT AB and also the usage of this type of detection. The possibility to work with extremely large amounts of data, a large technical infrastructure and the expertise of many technical fields that a company of IKEA IT ABs size offered was of interest. Furthermore, the actual use of such a system would be useful for many areas and companies. To have hands on experience and knowledge of this type of system would probably be valuable for future ambitions.

1.6 Limitations

The focus of the project will be mainly on the functionality side. Limitations on what scope to analyse will be made to fit the time span of the project.

Chapter 2

Technical background

This chapter describes the programs and other technicalities that form the basis of this thesis. Basic knowledge in computer science is assumed. However, concepts, programs and structures that are beyond basic knowledge, but relevant for the understanding of this thesis, are further explained here.

2.1 Google Cloud Platform

Google Cloud Platform, also commonly referred to as GCP, provides a cluster of cloud-based services hosted by Google on the same infrastructure as their own Google applications. These services vary largely in use but because they all exist on the Google structure, Google protects your data from fraudulent use, abuse and scam [19].

2.1.1 Google App Engine

Google app engine is a platform for developing and hosting web applications. It supports several popular languages such as Go, PHP, Java, Python and more. A core feature of app engine is its ability to fully take care of provisioning of servers and scaling [4].

2.1.2 Google BigQuery

BigQuery is an enterprise data warehouse with features to help manage and analyze data [5]. It can handle any sizes of data from bytes to petabytes [6]. SQL is the language used to make queries against the BigQuery database tables. Client libraries are also available which unlocks the possibility to transform and manage data with common programming languages such as Python and Java, as well as BigQuery's REST API and RPC API [5]. One thing to note is that BigQuery tables are append only. This means that updating or deleting data can't be done without truncating and recreating the entire table [22].

2.1.3 Google Data Studio

Google Data Studio is a tool that helps visualise data through interactive dashboards and reports [17]. With over 600 connectors Data Studio can access and visualise a wide variety of data [16]. It also has collaboration and sharing functions which makes it possible to collaborate in real-time as well as share reports to teams and/or individuals.

2.1.4 Google Cloud Billing

Google Cloud Billing is the service that handles payment to Google for using their Cloud services. By using a Cloud Billing account it's possible to define who pays for a given set of Google Cloud resources [7]. There's also functionality to get an overview of cost history, current cost trends, and forecasted costs with reports available in the Google Cloud console [8]. Several different types of reports are available depending on what billing data analysis is needed.

2.1.5 Google Cloud Functions

Cloud Functions makes it possible to run code in the cloud with no servers or containers to manage with no costs for when the function is idle [10]. Cloud functions can be written in Node.js, Python, Go, Java, .NET, Ruby and PHP programming languages [9]. There's two types of cloud functions: HTTP functions and event-driven functions. HTTP functions are invoked from standard HTTP requests such as GET, PULL, POST, DELETE and OPTIONS. Event-driven functions handles event from Cloud infrastructure. These events can be things such as messages on a pub/sub topic or changes in a Cloud Storage bucket [9].

2.1.6 Google Cloud Logging

Cloud Logging gives a user access to fully managed, real-time log management with storage, search, analysis and alerting. With the included Log Explorer it is possible to browse and view logs through filters and SQL-queries . It is also possible to route logs to databases in the current or other projects with various applicable filters [11].

2.1.7 Google Cloud Monitoring

Cloud Monitoring collects metrics, events and metadata from services such as Google Cloud, Amazon Web Services, hosted uptime probes, and application instrumentation. This data is then used to generate insights via dashboards, charts and alerts [12].

2.1.8 Google Cloud Scheduler

Cloud Scheduler is a managed enterprise-grade cron job scheduler. It allows for scheduling of almost any job on the GCP such as big data jobs, cloud infrastructure operations, and more. Cloud Scheduler also supports Cloud Pub/Sub which allows jobs to trigger Compute Engine, Google Kubernetes Engine and Cloud Functions [13].

2.1.9 Google Cloud Storage

Cloud Storage is a service for storing objects in Google Cloud. It can be used to store and retrieve any amount of data at any time. It can be used for a wide variety of scenarios such as serving website content, storing data for archival and disaster recovery or distributing large files to users via direct download [14]. Objects in Cloud Storage are are stored in containers called buckets [15]. All buckets are associated with a GCP project.

2.2 Microsoft Visio

Visio is an application included in the Microsoft 365 subscription service and is used for creating diagrams and flowcharts [25]. It contains features to create many different types of diagrams as well as tools to collaborate in real-time.

2.3 Microsoft Teams

Microsoft's calendar integrated communication platform. Built from the Microsoft 365 groups, Microsoft Graph and more, it hoist the same enterprise-level security, compliance and manageability as the rest of Microsoft's 365 products [26]. It is a platform designed for scheduling and holding live meetings. The integrated calendar and connection to Microsoft Outlook allows for easy scheduling of meetings as it keeps track of participants available time slots, emails the participants, and even suggests vacant meeting times.

2.4 Pycharm

PyCharm is an integrated development environment developed by JetBrains. It provides coding navigation and code editing for languages such as Python, JavaScript, CoffeeScript, TypeScript and more. The built in developer tools support debuggning, testing and database communication.

Many of the existing web development frameworks is supported in PyCharm, amongst them flask, Django and Google App Engine [21].

2.5 Python

Python is the third most used programming language in the world and the language most wish to work with as of 2021 [27]. It's an object-oriented, interpreted and high-level programming language. It uses dynamic semantics, dynamic typing and dynamic binding. These attributes, together with the high readability, makes the code very attractive for many purposes and counts as a language with low maintenance cost. The support for, and existence of, many packages and modules strongly encourages code re-use [3].

2.6 Slack

Slack is a communication tool that focuses mainly on different types of text communication. A magnitude of functions exist for the purpose of allowing efficient conversation between the right participants. Channels for those in the same team, project or section of a company. Furthermore, thread based chatting helps keep the overall flow clean and easy to follow while it also supports direct messaging. Applications of many sorts exist and can be created to increase the functionality and flexibility of the program to better fit the users needs [29].

2.7 SQL

Structured Query Language (SQL) has been the standard for communicating and manipulating databases since the late 1980s. Many versions of this language has been developed since then such as MySQL and PostgreSQL, however they all usually support the major commands from SQL [30]. An important function that is used in this thesis is the Window function. The window function in SQL takes a single or multiple rows and returns a single value from each of them. This allows for flexibility in other functions that go over multiple rows, as the window can be used to specify which rows to include [20].

Chapter 3

Method and analysis

This chapter details the steps, processes and reasons behind the decisions that were taken to get the project from an idea to reality.

3.1 Introduction

To structure the work and help planning the project, the first choice was to discuss what development method was to be used. After a discussion with the supervisors of the thesis, scrum was chosen. The possibility of re-valuating the process and making changes in direction after each scrum made that method fitting the thesis as the implementation and choices for the thesis would be determined by what information was found during the work [28].

Communication with IKEA IT AB was mainly held through daily scrum stand-up meetings. These were mostly held online via *Microsoft Teams* due to the ongoing Covid situation. Since most of the communication was done online, a digital forum was created in *Slack* for the project to allow efficient dialogue. Each sprint was set up to start with a meeting where the goals for that sprint were decided. At the end of each sprint there would be a demo of what had been accomplished during the sprint for the team and then a retrospective meeting.

Lastly, the work was planned in five phases - five sprints (see Figure 3.1. Before the first sprint started a preliminary plan of what was to be expected in each phase was constructed (see Figure 3.2). It was quickly noticed that instead of following that plan to a tee, using the retrospective and iterative nature of scrum to make future decisions was better. It allowed for the flexibility of spending more time in certain areas if that was thought to be beneficial and also tweak the future plan when more knowledge had been gained from previous sprints.

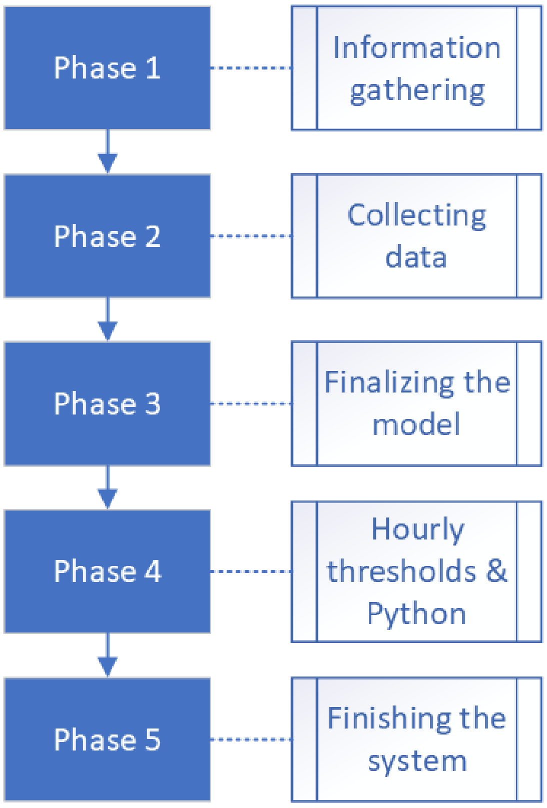
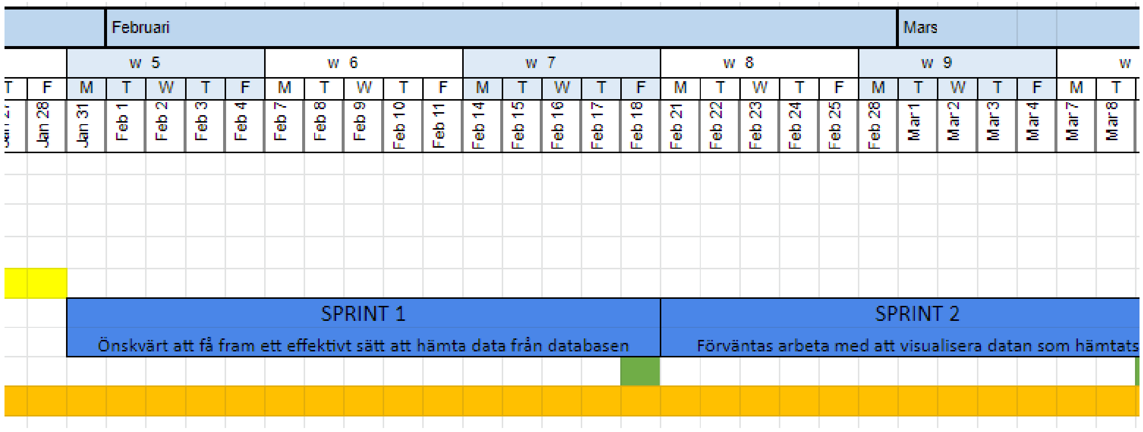


Figure 3.1: Overview of the project phases.



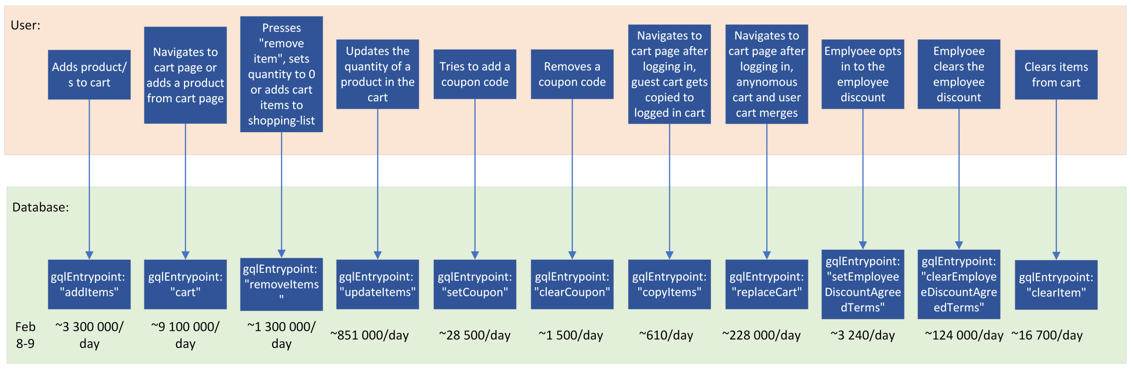
			Februari																				Mars				
		w 5					w 6					w 7					w 8					w 9					w
T	F	M	T	W	T	F	M	T	W	T	F	M	T	W	T	F	M	T	W	T	F	M	T	W	T	F	M	T
[illegible]	Jan 28	Jan 31	Feb 1	Feb 2	Feb 3	Feb 4	Feb 7	Feb 8	Feb 9	Feb 10	Feb 11	Feb 14	Feb 15	Feb 16	Feb 17	Feb 18	Feb 21	Feb 22	Feb 23	Feb 24	Feb 25	Feb 28	Mar 1	Mar 2	Mar 3	Mar 4	Mar 7	Mar 8
		SPRINT 1 Önskvärt att få fram ett effektivt sätt att hämta data från databasen															SPRINT 2 Förväntas arbeta med att visualisera datan som hämtats											

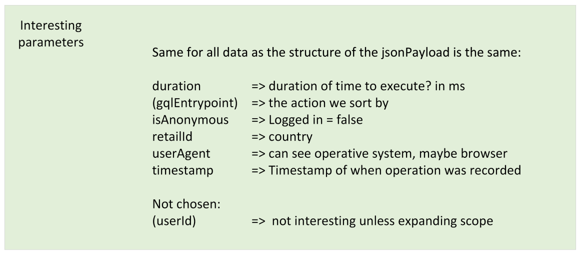
Figure 3.2: An excerpt from the preliminary plan spreadsheet.

3.2 Phase One - Information gathering

This phase involved building an understanding of the GCP environment that IKEA IT AB is using and planning the implementation of a solution. The first step was reading relevant documentation regarding GCP to get a basic understanding of the workings of *Google Cloud Platform*. A GCP project was setup to isolate the cost, data and analysis for easier management. This project will be called cartops-analytics from here on. The GCP project used by IKEA IT AB that is relevant for this thesis will be referred to as SP.

A vital part in the beginning was clarifying how the live website user usage was reflected in the SP-logs. To gain this understanding and getting practical GCP knowledge, testing was done on the live website to track what logs were generated in *Cloud Logging* when using different cart actions. Furthermore, other parameters that were also provided from the logs was investigated to ascertain what could be used for troubleshooting or finding anomalies. This was done in conjunction with the IKEA IT AB cart-team consisting of seven employees with roles ranging from junior to senior developers. A summary of each action and the potentially most useful parameters were constructed in *Microsoft Visio* to aid future decision-making (see Figure 3.3).


User:
Adds product/s to cart
Navigates to cart page or adds a product from cart page
Presses "remove item", sets quantity to 0 or adds cart items to shopping-list
Updates the quantity of a product in the cart
Tries to add a coupon code
Removes a coupon code
Navigates to cart page after logging in, guest cart gets copied to logged in cart
Navigates to cart page after logging in, anynomous cart and user cart merges
Emplyoee opts in to the employee discount
Emplyoee clears the employee discount
Clears items from cart
Database:
gqlEntrypoint: "addItems"
gqlEntrypoint: "cart"
gqlEntrypoint: "removeItems"
gqlEntrypoint: "updateItems"
gqlEntrypoint: "setCoupon"
gqlEntrypoint: "clearCoupon"
gqlEntrypoint: "copyItems"
gqlEntrypoint: "replaceCart"
gqlEntrypoint: "setEmployeeDiscountAgreedTerms"
gqlEntrypoint: "clearEmployeeDiscountAgreedTerms"
gqlEntrypoint: "clearItem"
Feb 8-9
~3 300 000/ day
~9 100 000/ day
~1 300 000/ day
~851 000/day
~28 500/day
~1 500/day
~610/day
~228 000/day
~3 240/day
~124 000/day
~16 700/day


Interesting parameters

Same for all data as the structure of the jsonPayload is the same:

duration	=> duration of time to execute? in ms
(gqlEntrypoint)	=> the action we sort by
isAnonymous	=> Logged in = false
retailId	=> country
userAgent	=> can see operative system, maybe browser
timestamp	=> Timestamp of when operation was recorded

Not chosen:

(userId)	=> not interesting unless expanding scope

Figure 3.3: Describes what actions correlate to what gqlEntrypoint in the logs. Also lists the potentially most useful parameters.

As each action in the cart resulted in different log entries, a decision had to be made on which singular action to continue the work on. In order to evaluate which one would be of most use for IKEA IT AB, a risk analysis was made in *Microsoft Visio* with the parameters probability and consequence in mind. The definition of each term and where each cart action were placed were concluded in figure 3.4 and stood as the basis for which cart actions that were analysed further in the next phase.

At this part of the work, a better understanding of the project as a whole made it possible for a rough road-map to be created (see Figure 3.5). This allowed for a better overlook of what steps were needed to be completed to reach completion and to make sure no steps in the progress were missing. A road-map was also a useful tool to get an overlook of how the project was coming along in future phases of the work.

A flowchart was constructed to display what and where data would flow, what processes would be done in which part of the system and also how each part of the system would work together.

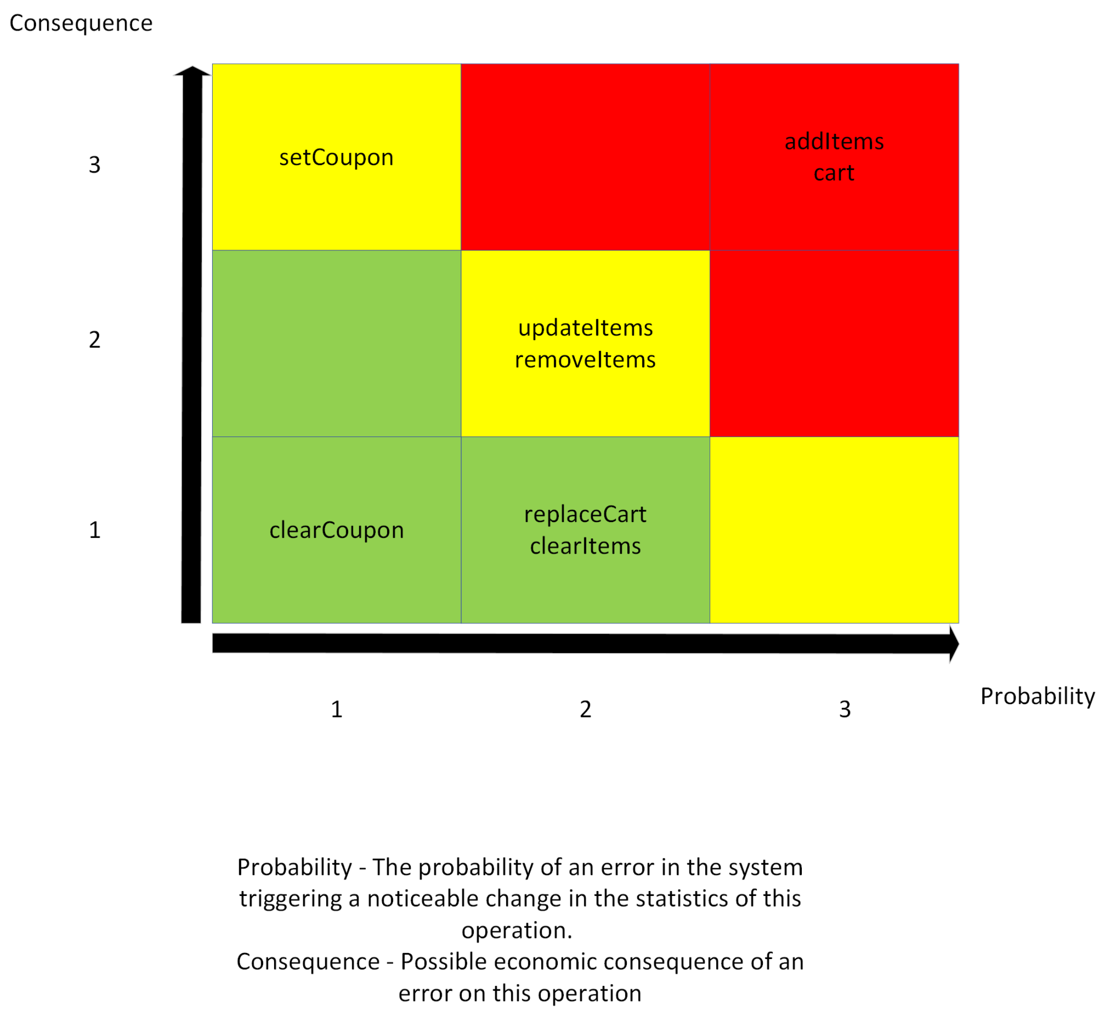


Figure 3.4: Risk analysis on each cart action with the probability and consequence of an error.

This was done to deepen the understanding of each part of the system in its final stage. As seen in figure 3.6, the parts done in the earlier stages of this phase come together here to give a clearer view of more exactly what data and structures are to be used.

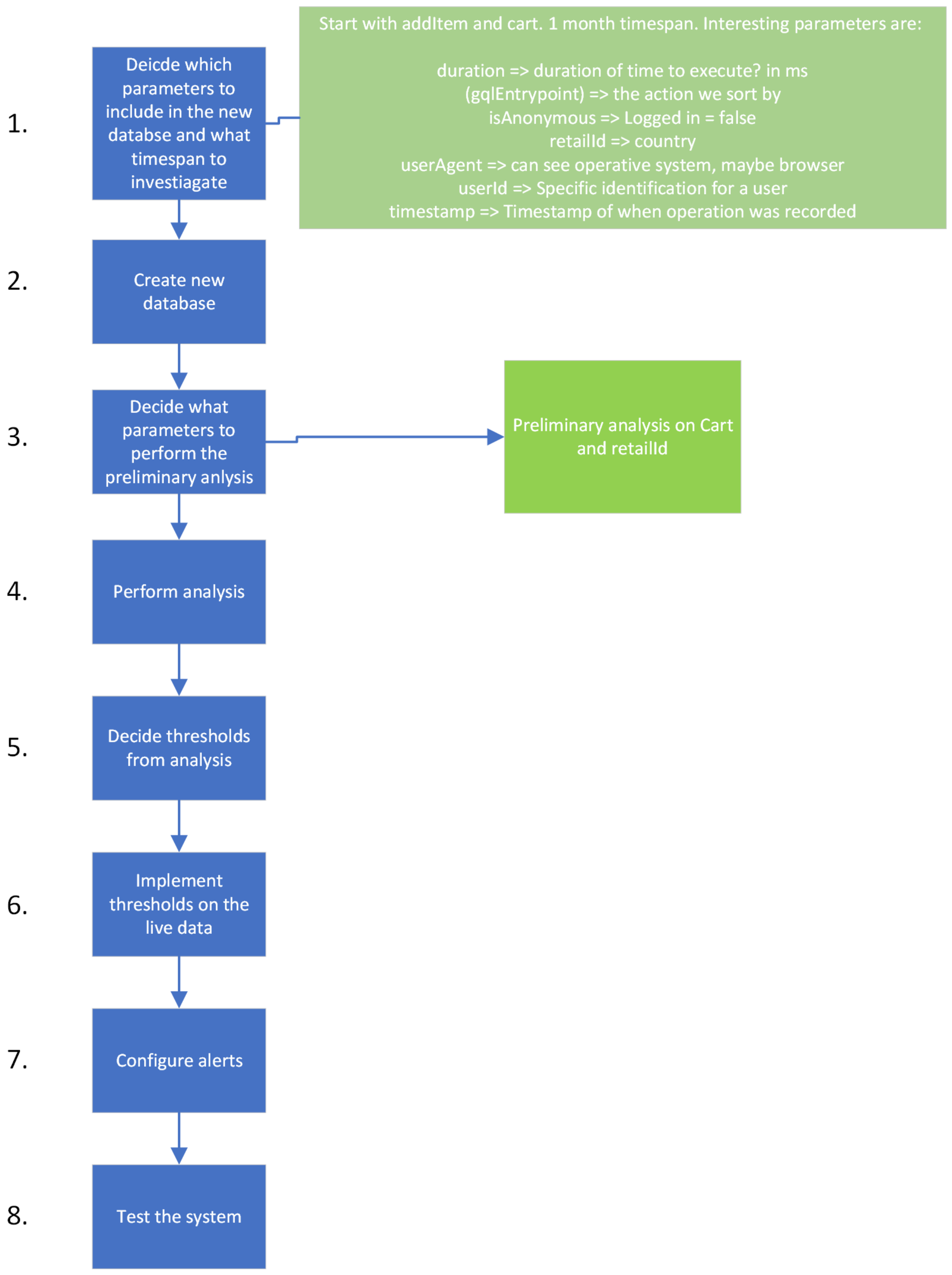
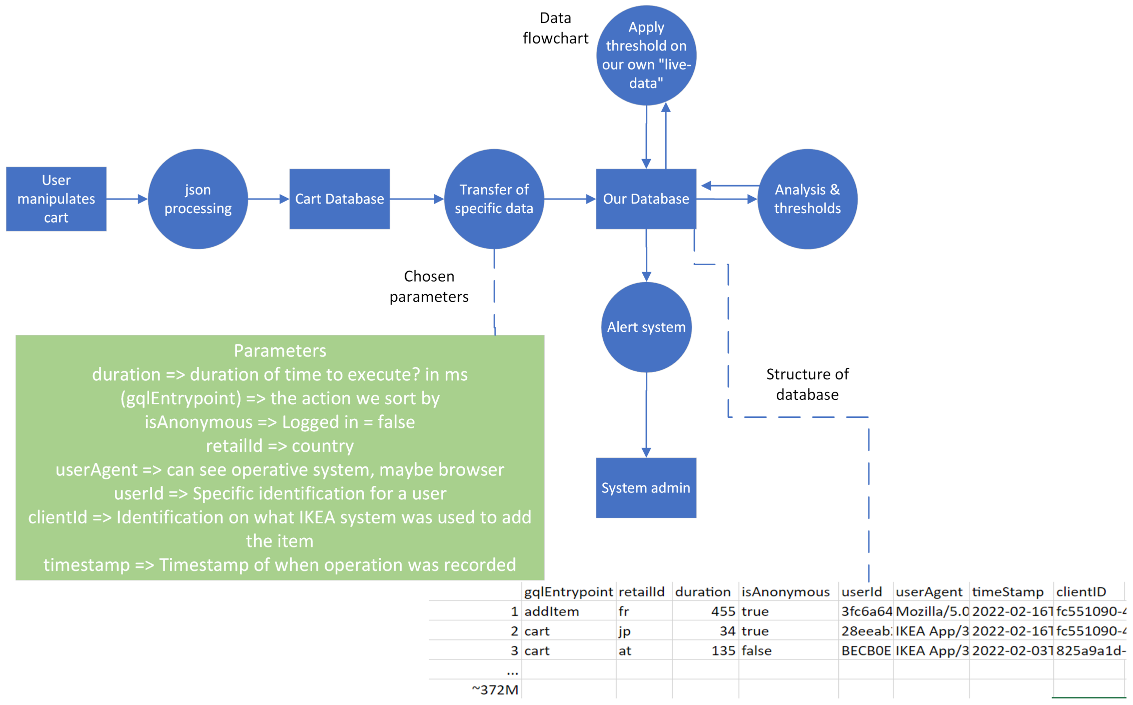

1.
Deicde which parameters to include in the new databse and what timespan to investiagate
Start with addItem and cart. 1 month timespan. Interesting parameters are:
duration => duration of time to execute? in ms
(gqlEntrypoint) => the action we sort by
isAnonymous => Logged in = false
retailId => country
userAgent => can see operative system, maybe browser
userId => Specific identification for a user
timestamp => Timestamp of when operation was recorded
2.
Create new database
3.
Decide what parameters to perform the preliminary anlysis
Preliminary analysis on Cart and retailId
4.
Perform analysis
5.
Decide thresholds from analysis
6.
Implement thresholds on the live data
7.
Configure alerts
8.
Test the system

Figure 3.5: Road map showing the preliminary planned steps of the project.



Parameters
duration => duration of time to execute? in ms
(gqlEntrypoint) => the action we sort by
isAnonymous => Logged in = false
retailId => country
userAgent => can see operative system, maybe browser
userId => Specific identification for a user
clientId => Identification on what IKEA system was used to add the item
timestamp => Timestamp of when operation was recorded

	gqlEntrypoint	retailId	duration	isAnonymous	userId	userAgent	timeStamp	clientID
1	addItem	fr	455	true	3fc6a64	Mozilla/5.0	2022-02-16T	fc551090-4
2	cart	jp	34	true	28eeab:	IKEA App/3	2022-02-16T	fc551090-4
3	cart	at	135	false	BECB0E	IKEA App/3	2022-02-03T	825a9a1d-
...								
~372M								

Figure 3.6: Data flow chart on the flow, structure and processes.

3.3 Phase Two - Collecting data

In phase two a dataset was created. This dataset was then used to setup a sink that would automatically build a copy of the raw logs contained in the SP project and store it in a table. The raw logs table were then used to build another table that was formatted according to our datamodel, seen in 3.1, that was created in phase one.

Table 3.1: Datamodel for the structured logs table

	gqlEntrypoint	retailId	duration	isAnonymous	userId	userAgent	clientId	timestamp
1	addItem	fr	455	true	3fc6a...	Mozilla/5.0...	fc551...	2022-02-16T09...
2	cart	jp	34	true	28eea...	IKEA App/3.1...	fc551...	2022-02-16T09...
3	cart	at	135	false	BECB0...	IKEA App/3.9...	825a9...	2022-02-03T12...

BigQuery queries were manually used to create, update and to insert new data into this table as it became available. Towards the end of phase two there was about five days worth of formatted data inserted into the table. This data was then analysed in *Google Data Studio* to determine what parameters were relevant for creating the thresholds and alarms in later phases of the project.

Analysis of the data was done to make sure a qualified choice was made on what model to proceed with in the thesis work. Graphs were made of country distribution and gqlEntrypoints distribution of the whole cart usage. These graphs were further dimensioned by every minute, hourly, and daily. Figure 3.7 illustrates both daily trends, weekly trends and the quantity difference in entries between gqlEntrypoints. The gqlEntrypoints "cart" and "addItems" were chosen for deeper analysis because of their large amounts of data, making them easier to analyse [24].

Analysis on the "addItems" and "cart" entries were done by looking at the market split over countries, total amount of entries per day, how many of the entries were created by members logged onto the IKEA IT AB platform and userAgent distribution. A decision to exclude Country two from all future data was reached because of the anomaly shown in figure 3.8. This anomaly was caused by the large influx of shopping due to the Ukraine crisis and IKEA IT ABs decision to seize operation in Country two.

To create the first prototype a single statistical model had to be used as the foundation for the threshold implementation. After a discussion with the team, the gqlEntrypoint "cart" was chosen as the focus for this model. The reasoning behind this choice was the larger data size in "cart" compared to "addItems" and also that "cart" is further downstream in the usage flow. Therefore more errors would be noticeable in "cart". Interest in the parameters of country and clientId arose from this discussion and were thus analysed further in the next phase before going ahead with the first prototype.

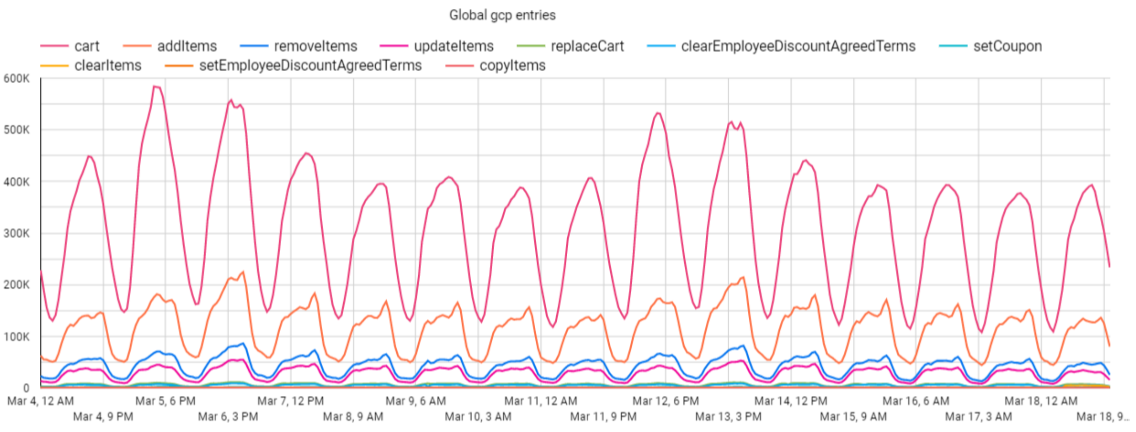


Figure 3.7: Distribution of gqlEntrypoints on the global market per hour over two weeks.

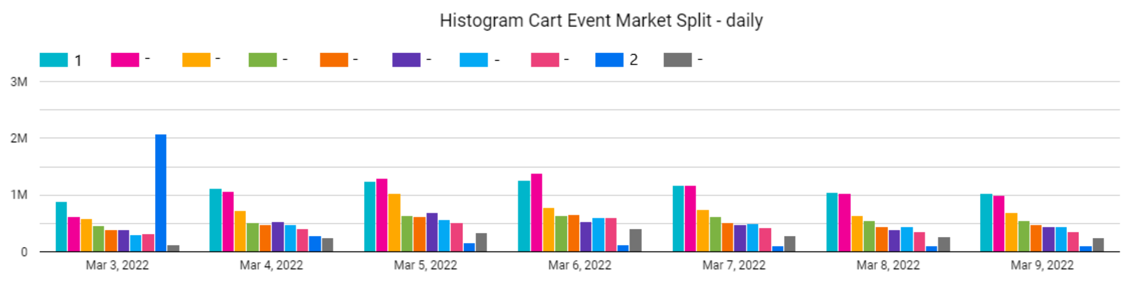


Figure 3.8: Daily market split of cart gqlEntrypoints with the Country two anomaly included.

3.4 Phase Three - Finalizing the first model

During phase three much effort was made to finalize the statistical model that would become the foundation for the first implementation of the alert thresholds. Analysis of models based on retailId and clientId was done to finalize the choice of model by analysing country composition and data entries per country over time (see Figure 3.9 & 3.10).

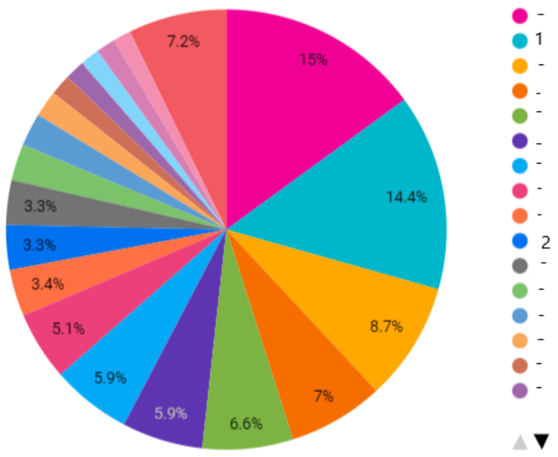


Figure 3.9: Shows the percentage of raw data per country

By looking at the alignment of the graphs in Figure 3.10 it was determined that most of the data is synchronized with an EU timezone. Therefore, an EU country was decided to be preferred as the first model to simplify for future expansion to more countries due to matching time zones. With this in mind, Country one was chosen as the model because of its large amount of data[24] coupled with being an EU country.

With the country decided, analysis continued with clientId based on the response received from the team at the end of phase two (see Section 3.3). Utilizing graphs that depicted the most used clientIds globally and in Country one (see Figure 3.11 & **??**), six clientIds were identified as the most common with five of these six also being the most common in Country one. After discussing these findings with the technical expert at IKEA IT AB, it was concluded that the first model would not be divided by clientIds to ease the production of a single initial prototype. Additionally, having five-six models per country would incur additional costs and management to keep all the models in production.

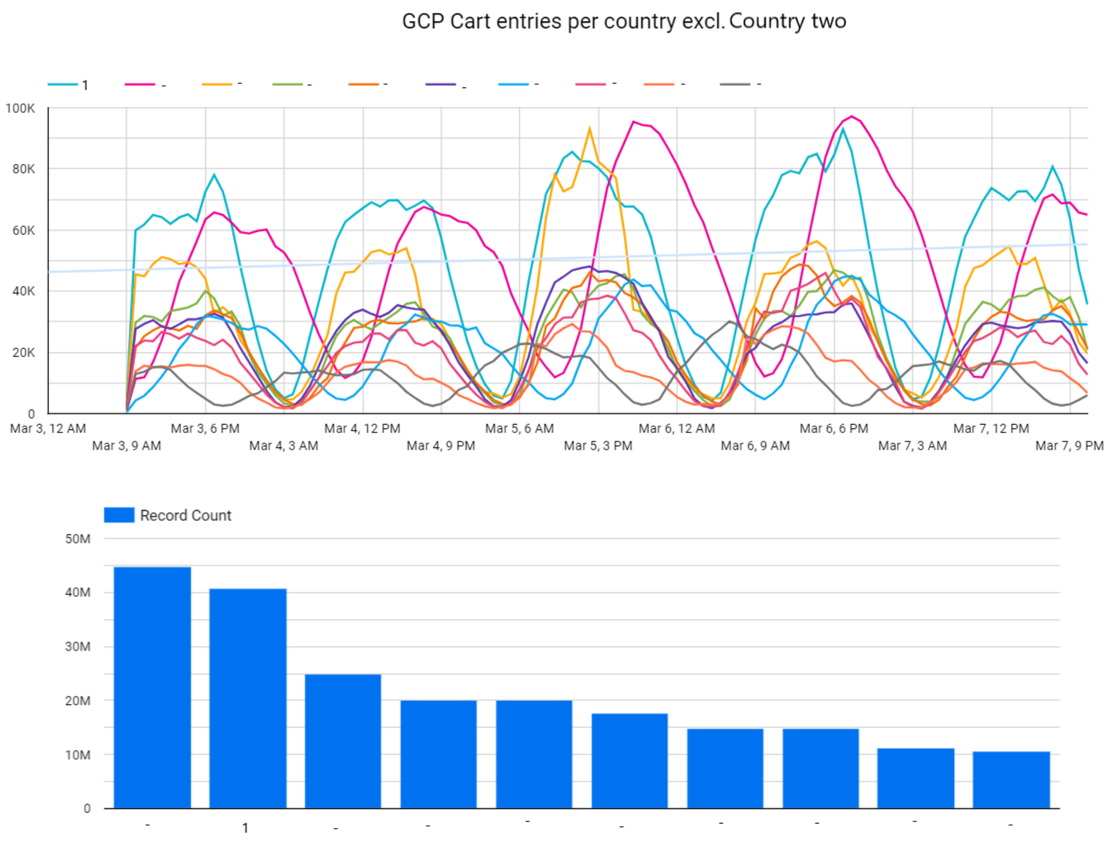


Figure 3.10: Displays log-entries of type cart per country excluding Country two.

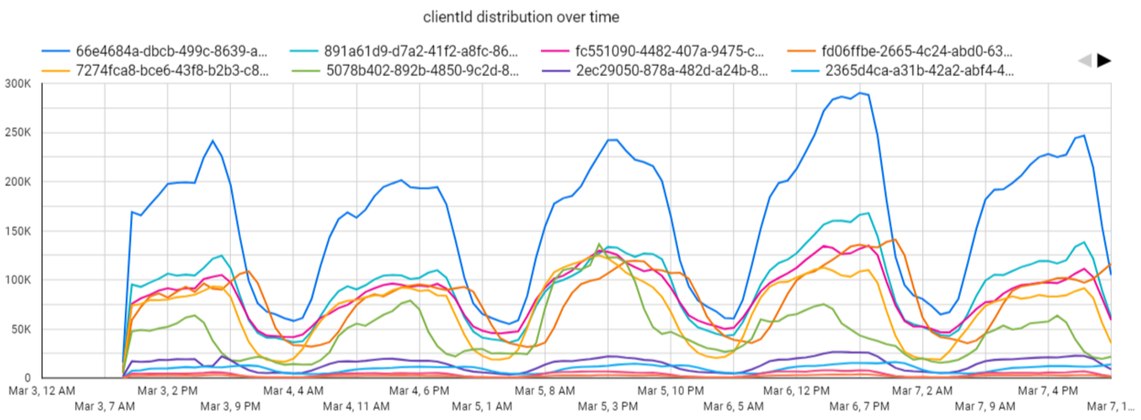


Figure 3.11: clientId distribution over time globally.

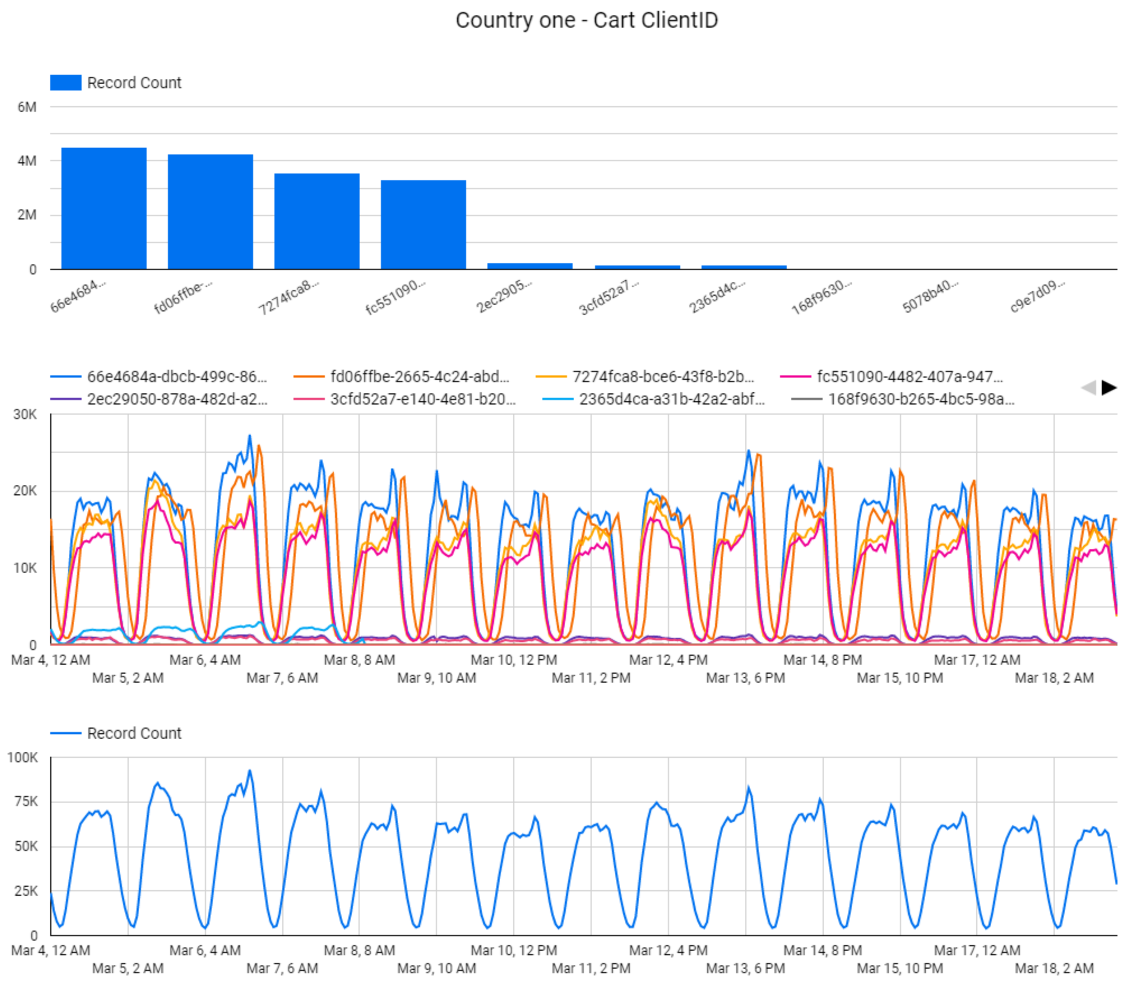


Figure 3.12: clientId distribution over time in Country one.

Once Country one was decided as a starting point together with gqlEntrypoint ”cart”, which was decided in phase two, a first prototype was created (See Figure 3.13). This prototype worked on a daily time frame and shows the implementation of thresholds on the collected data. By calculating the standard deviation and mean based on seven previous days of data the thresholds in the prototype was created by utilizing equation 3.1 and 3.2. UT represents the upper threshold whereas LT represents the lower threshold.

$$UT = \bar{x} + 4 * \sigma \tag{3.1}$$

$$LT = \bar{x} - 4 * \sigma \tag{3.2}$$

where:

$\bar{x}$ = mean
σ = standard deviation
UT = upper threshold
LT = lower threshold

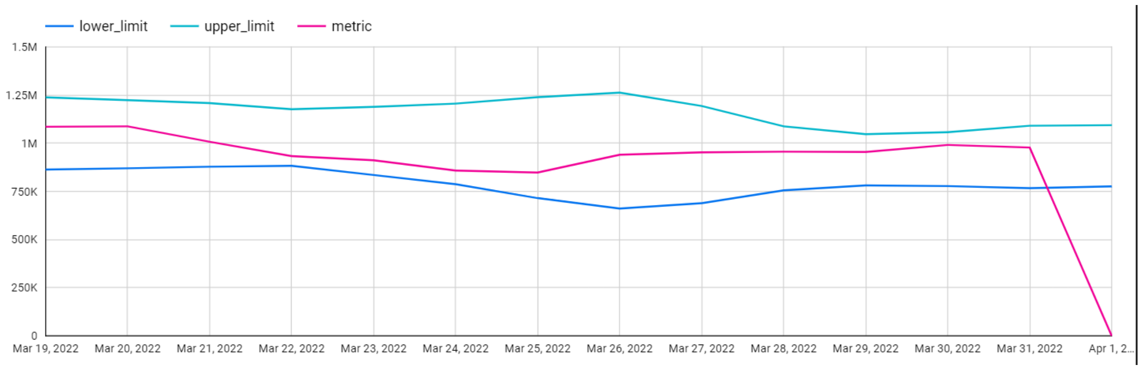


Figure 3.13: First prototype of threshold implementation. The drop of the metric in the last day is because that metric has yet to be summarised.

This first prototype of the threshold implementation was shown to the team. Discussion regarding grouping the data-points for the mean and standard deviation by weekday instead of week was brought into consideration for phase four.

3.5 Phase Four - Hourly thresholds & Python

The fourth sprint started with continued work on the prototype that was created at the end of phase three (see Section 3.4, figure 3.13). The next step was to create a prototype that used hourly data instead of daily data. From the analysis done in phase one, it was established that there was variation in the data on an hourly time-frame that was due to daily usage trends (see Figure 3.7). Therefore, each hour was compared to the same hour in previous days to account for these reoccurring trends. Two preliminary models were created for this purpose by using scheduled queries. The first model only looked at the previous four identical weekdays (see Figure 3.14), while the second model looked at the seven days preceding the chosen day (see Figure 3.15).

The choice to base one of the models on the same weekdays was due to a problem with using normal distribution for anomaly detection with data that has seasonal patterns [2]. Data with seasonal patterns is not perfectly normally distributed which means that the model needs to be flexible to not trigger false alerts all the time.

Confirmation of these seasonal/weekly patterns can be seen in figure 3.15 as the thresholds breaks on the 2nd of April is not seen in figure 3.14. This shows a weekly pattern where usage increases on Saturdays. Tweaking the number of sigmas for the standard deviation did little difference for

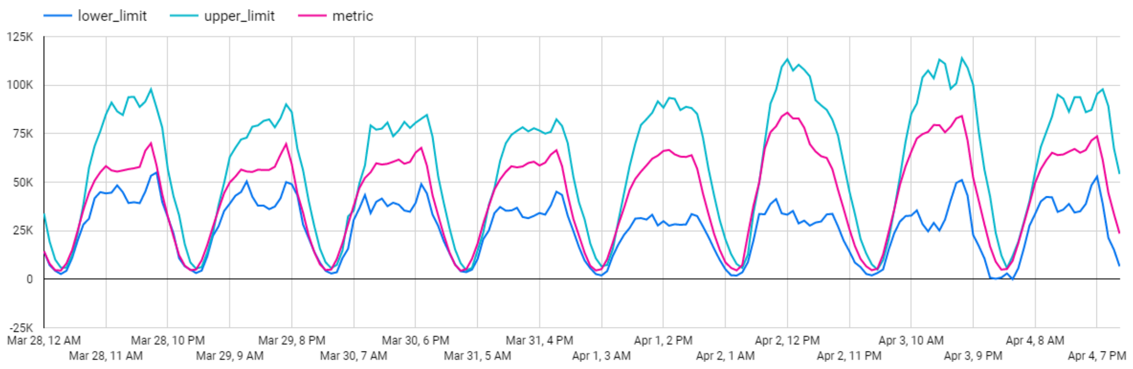


Figure 3.14: Prototype of hourly thresholds based on previous four identical weekdays.

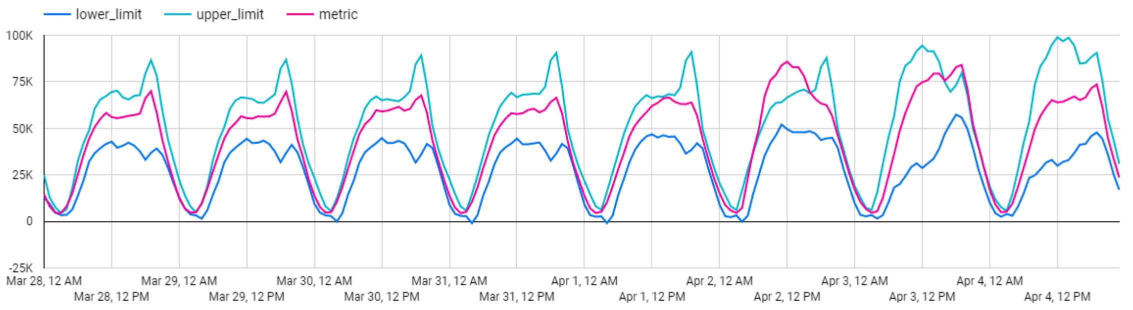


Figure 3.15: Prototype of hourly thresholds based on the seven preceding days data.

the second model to handle this type of pattern. Therefore, a decision was made to discontinue the analysis of the second model, figure 3.15, and focus on the first model, figure 3.14.

A detailed analysis of the first model revealed that a substantial amount of days had threshold breaks in the mornings, between around 03:00 and 07:00 (see Figure 3.16). At a first glance this could point toward the model being faulty as there are no signs of abnormal behaviour during those hours. However, further reflection lead to the realisation of Country one having changed from wintertime to summertime on March 27th at 02:00. The forward change of one hour skewered the whole data-set causing the data before that date and after to have an offset of one hour. While it did not cause a problem during the daytime where the daily variance was large enough to handle this change, the night on the other hand that were more stable and showed less variance were more fragile to this change. Knowing that this problem was one existing solely during the times around daylight saving made it possible to ignore these threshold breaks for the sake of further analysis.

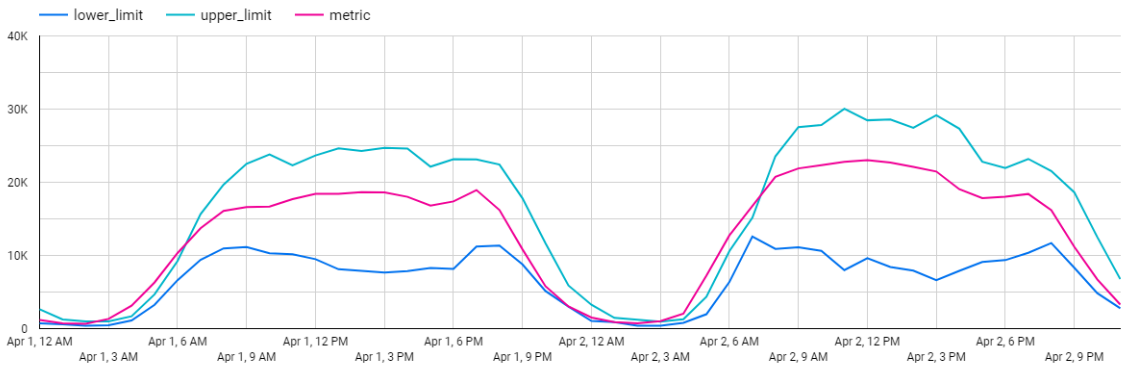


Figure 3.16: Threshold breaks in the mornings due to summertime.

The daylight saving problem was discussed with the team and technical expert before deciding that no effort would be put in looking for a solution but instead prioritize the main goal of the project. Since the goal was not to put the system into production, rather to prove a concept, it was prioritised to create a model to observe the overall behaviour of the thresholds instead of a production-perfect model.

With prototypes created on daily and hourly granularity, both were presented to the team to understand which would be of most use for them. Discussion with the team made it clear that a fast reaction to anomalies was important. Therefore, the hourly model was chosen to be continued on. However, the team clarified that the choice of not focusing on any clientId would introduce a lot of noise to the model. Instead selecting a singular clientId would make the model more accurate. The clientId representing web users was recommended and ended up being the one chosen for the model (see Figure 3.17).

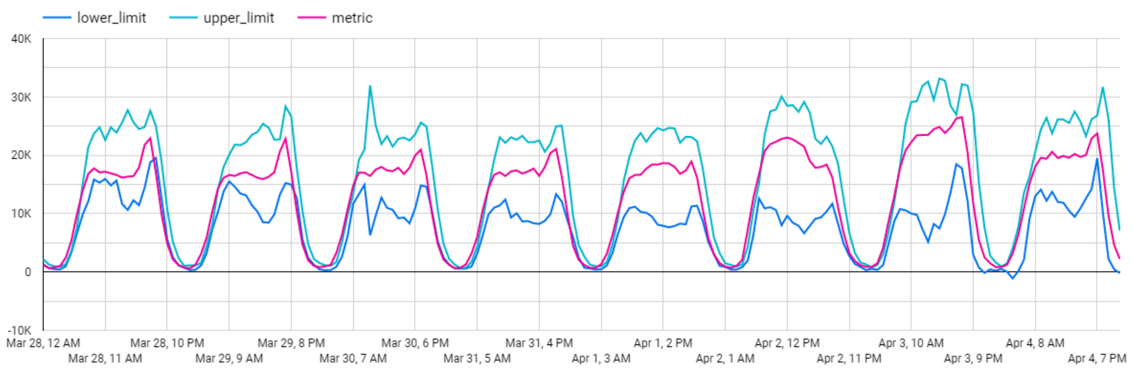


Figure 3.17: Prototype of hourly thresholds based on both four previous identical weekdays that looks only at the web user clientId.

As the model had now been decided, the next step would be to set up the alerting system. The plan was to create a simple logic on whether the observed metric was between the calculated upper and lower thresholds. Then, if the metric had broken either of those thresholds, a message would be sent to the team informing of a broken threshold. Looking into different possibilities led to investigating the *Google Cloud Monitoring* if that would be suitable for the alerting. However, the difficulty of getting the necessary values from a *BigQuery* table into the *Cloud Monitoring* system halted this idea. Further scouring of the internet found that *Google App Engine* had all the necessities to accomplish the alerting task.

Python was chosen as the language for the alerting programming because of its vast usage, popularity and the interest in gaining experience in a new coding language. As IDE, *PyCharm* was the choice as it supports both *Google App Engine*, Flask and comes with Database tools, making it a good fit for this project.

To read from the *BigQuery* tables it was necessary to connect to a service account on GCP that had read permission on the *BigQuery* dataset. When deploying an app from the *Google App Engine* it automatically connects to the service account that is created when enabling *Google App Engine* for that GCP project. However, when deploying the code from *PyCharm* a service account key was instead needed to manually connect to this service account to gain access to the *BigQuery* dataset.

After connecting to the dataset with *PyCharm* the threshold breaking logic was coded using the web framework flask. It was planned to use *Google App Engine* to send the alerting messages as it supports sending emails. But to actually allow the app to have the permission to send emails was a difficult task and the phase ended while still investigating how to accomplish this.

A demo was held for the team and thoughts surfaced about giving some flexibility in choices for the user regarding the threshold breaks and alerts. Examples of ideas that were talked about was adding consecutive threshold breaks to filter out some of the falsly broken thresholds and also adding static values to the thresholds during certain times when the standard deviations were small, like during the nights.

3.6 Phase Five - Finishing the system

Phase five was the last sprint of the thesis. It started by discussing *Google App Engine* with the team. During this discussion it was decided that *Google App Engine* was too complex for the purpose of the thesis. As the supervisor at IKEA IT AB put it: ”it’s like shooting flies with a cannon”. Instead it was recommended that the script should be built and deployed as a simpler *Google Cloud Function.*

The *Python* script created in *PyCharm* at the end of phase 4 could simply be updated and re-used since both *Cloud Functions* and *Google App Engine* support *Python.* One of the important revisions of the code was the removal of e-mail functionality since sending e-mails is part of the *Google App Engine* API and not available in *Cloud Functions.* As some kind of communication channel was needed to send out the alerts, *Slack* seemed the obvious choice due to it’s regular day to day use within the company. A *Slack* message could be sent to a suitable channel with a simple webhook using an already existing *Slack* application set-up by the team for sending notifications to their own channels. With this webhook implemented the script was programmatically tested and then deployed.

The *Cloud Functions* subscribed to a pub/sub topic which meant that the *Cloud Function* runs when a message is sent to the pub-sub topic. A scheduling job in *Google Cloud Scheduler* was then created to send a message to the Pub/Sub topic every hour at minute five. The scheduling at minute 5 were chosen to avoid execution overlap with the scheduled query that runs every hour at minute zero.

Once the script was deployed a second script was created with the purpose of only sending a message in *Slack* when consecutive broken thresholds were noted. This was achieved by modifying the *Cloud Function* with a new *Python* Class. This class added the functionality of saving and reading a text file stored in *Google Cloud Storage.* Once a threshold is noted the script reads the txt file and if it returns true (meaning the previous hour also had a threshold break) a message is sent to *Slack.* Afterwards no matter if a message was sent or not the line in the text file is replaced with ”True”. If a threshold has not been breached the text in the txt file is instead replaced with ”False” (see Appendix 9.2 for the code).

The model at this point in the project set the upper and lower limits without considering what the numbers actually meant. The implications of this behaviour was that negative lower limits were allowed (see 4th of April in Figure 3.17 as an example). For this lower limit to be broken negative amount of cart requests would have to be measured which is impossible. Therefore, a fix was added to the model where the lower limit would not go below zero making it always possible for the lower threshold to be broken.

During the second week of phase 5 the *Cloud Billing* for the project was revisited due to increasing costs. It was noted that costs rose significantly after the implementation of the hourly scheduled queries during phase 4 (see Figure 3.18). After some investigation it was discovered that the culprit for the increasing costs was the table structure of the raw log data. When the sink for the raw data was created in phase 2 the setting for exporting a partitioned table was not selected. This resulted in a sharded table instead of a partitioned table. Sharded tables adds query overhead and affects query performance which meant that the scheduled query didn’t filter on dates properly and therefore processed the whole raw log table each time the query was run [18]. This resulted in the sharp linear increase of cumulative costs between mid April and the start of May that can be seen in figure 3.18. The proper solution to this problem would be to convert and start using a partitioned table for the raw logs. However, implementing a large change to the raw data table this late into the project was determined to be too big of a risk. Instead the sharded table format was kept and the WHERE clause was modified to force the query to manually filter the unused dates. This reduced the scheduled query costs from 17$ per day to 2$ per day (see Figure. 3.19).

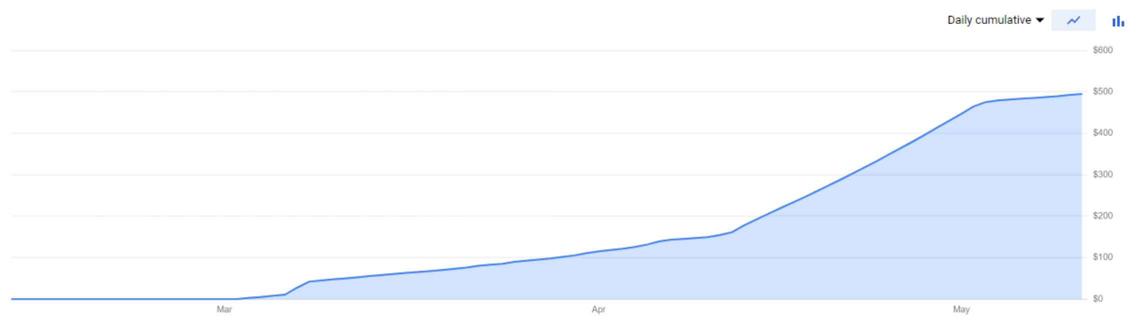


Figure 3.18: Cloud Billing daily cumulative costs.

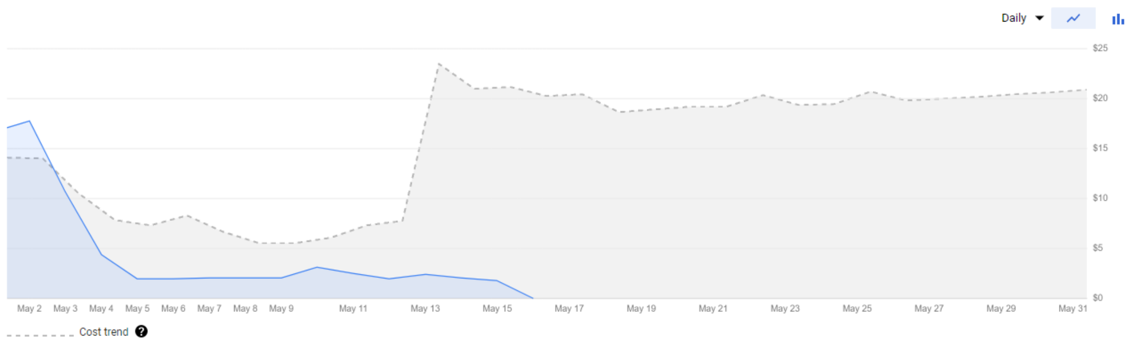


Figure 3.19: *Cloud Billing* daily costs for the month of May

Broken thresholds had occasionally been reported in the *Slack* channel since the activation of the alerting scripts. When investigating the time of the broken thresholds against the threshold graph (see Figure 3.17) the alerts happened at times when one of the threshold limits had in fact been broken, thus confirming that the scripts were fully operational. To provide more information regarding the broken thresholds a dashboard was created. The purpose of this dashboard was to help the team have a single location where more data was shown to help troubleshoot and find the cause of a broken threshold.

A first draft of the dashboard was made by adding any and all graphs regarding data that could potentially be useful for the team. It was also experimented on which time frame each graph should be on to only show the relevant information. Then, presenting this to the team allowed for their feedback regarding which graphs were useful, not necessary and what would actually help in their troubleshooting work. The final result was a three page dashboard (see Figures 4.2, 4.3, 4.4).

When discussing with the team, ideas of more dashboards for other teams and purposes were thrown around. An interesting idea that was presented was a specific dashboard for the product owner, where information regarding costs, products and other information related to the broken threshold would be presented. But this was never implemented since the project was approaching its deadline.

Since the dashboard for the alerts had now been created, the final step of the system had been completed. Testing now had to be done to ensure that the system worked as intended. Some working functionality had already been confirmed since alerts had been recorded between the initiation of the *Cloud Function* and the completion of the dashboard. However, controlled testing had yet to be done. To simulate a problem in the system that would result in users not being able to use the cart functionality for a time period, the sink that sent data from the SP project was disabled for 38 minutes. Eight of these minutes were before a change in hour, and 30 minutes were after the hour had changed. This meant that 30 minutes of downtime were simulated for the latter hour. This resulted in the average use going below the lower limit, thus triggering a breakout. The system gave an alert and the dashboard showed the relevant data, thus confirming the system working as intended.

Simulation of an increased usage was discussed but was decided to not be done due to lack of time and higher complexity. However, calculations on how much increase in activity would trigger a breakout on the upper limit was done to give an idea of how the system would react to an increase in activity.

The last part of the phase was a demo and then a meeting with the technical expert where the whole prototype and all decisions were reviewed to wrap up the project. Information surrounding IKEA IT AB costs, detection systems already in place and down-times were talked about to give a better understanding of how the prototype would fit into the existing architecture.

3.7 Information evaluation

When gathering sources and looking through references for this thesis, a few factors were used to determine their credibility. Documentation that were written by the creator or provider of the services were mostly used since the companies behind the products keep the most updated and relevant information regarding the implementation of their products. When taking sources from other sites, the affiliation and type of site were considered. Sites reporting on technical matters was prioritized due to their expertise in the technical field and therefore high chance of providing accurate information. Lastly, new material and information received were always considered with the context and knowledge that was already possessed to make sure it was not contradictory to that.

Chapter 4

Results

The thesis resulted in four separate parts all running on the *Google Cloud Platform* that forms the finished product and the test of the whole prototype. These resulting parts and results will be further discussed in this chapter.

4.1 Datasets

The *BigQuery* dataset RawLogs and its underlying tables form the backbone of the project. This is where the raw logs are stored as well as the tables that stores the calculated average, upper and lower thresholds (see Figure. 4.1). The sink that transfers the logs from the SP GCP project targets this dataset and end up in the cart_(*) table. All tables except the cart_(*) tables are partitioned by date. The cart_(*) table is where the raw logs are stored and is the only table that is sharded as mentioned in Section 3.6. Several of the tables works as filters for the raw logs. This is due to the raw logs containing many columns that were decided during analysis to not be relevant for the project. Relevant tables that are used in the finished product are:

1. **cart_(*)**
   Stores the raw logs. The * represents the shard number. Figure 4.1 shows cart_(76) which contains the data from the 76th day since starting importing the logs.

2. **cartops_analysis_hourly**
   Relevant columns from the previous hour in the cart_(*) table are stored here temporarily to reduce query overhead.

3. **cartops_analysis_collection_hourly**
   Every hour the columns from **cartops_analysis_hourly** are inserted into this table for long-term storage.

4. **hourly_thresholds_weekday_clientId**
   The metric from the previous hour is stored here as well as the upper/lower thresholds and standard deviation.

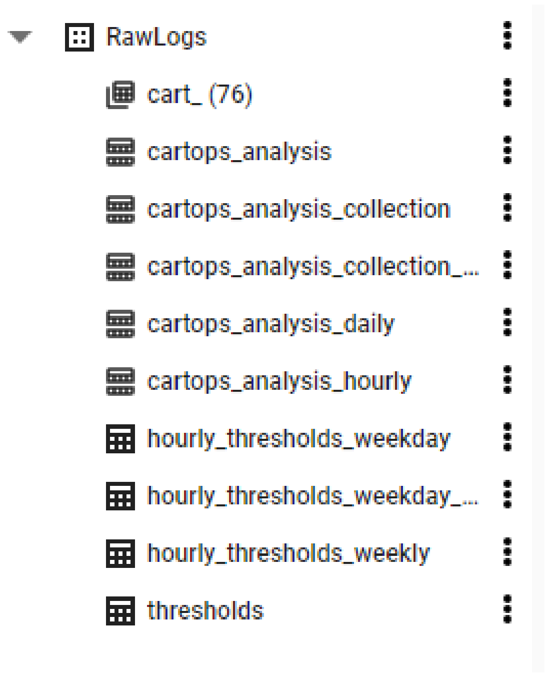

RawLogs
cart_ (76)
cartops_analysis
cartops_analysis_collection
cartops_analysis_collection_...
cartops_analysis_daily
cartops_analysis_hourly
hourly_thresholds_weekday
hourly_thresholds_weekday_...
hourly_thresholds_weekly
thresholds

Figure 4.1: The *BigQuery* dataset RawLogs and its underlying tables, including tables not needed for the final prototype.

4.2 Scheduled Query

The scheduled query is responsible for filling the previously mentioned tables with data from the raw logs in 1 hour intervals. They are also responsible for calculating the standard deviation as well as the lower and upper threshold which are then stored in the hourly_thresholds_weekday_clientid table. The standard deviation is calculated using the metric of four earlier data points in **cartops_analysis_collection_hourly** where the hour and weekday are identical to the previous hour. To reduce query costs and performance the scheduled queries are also responsible for filtering out the columns that were not included in the decided datamodel (see Figure. 4.1). The scheduled query is what makes up most of the costs of the project. The full query syntax for the hourly scheduled query and the unused daily scheduled query can be found in the appendix, chapter 9 section 9.1.

Table 4.1: Datamodel for the structured logs table

	gqlEntrypoint	retailId	duration	isAnonymous	userId	userAgent	clientId	timestamp
1	addItem	fr	455	true	3fc6a...	Mozilla/5.0...	fc551...	2022-02-16T09...
2	cart	jp	34	true	28eea...	IKEA App/3.1...	fc551...	2022-02-16T09...
3	cart	at	135	false	BECB0...	IKEA App/3.9...	825a9...	2022-02-03T12...

4.3 Cloud Functions

Two *Cloud Functions* written in *Python* are used in the finished product, **threshold_alerting** and **threshold_alerting_consecutive**. **threshold_alerting** sends a message in a *Slack* if the metric in the last row from **hourly_thresholds_weekday_clientId**, i.e. the metric from the last hour, is higher or lower than the stored thresholds. The message notifies of a broken threshold and contains a link to the dashboard that presents more detailed figures and information. **threshold_alerting_consecutive** works similarly to **threshold_alerting** but sends a message only after two thresholds has been broken consecutively. This is achieved by storing the previous result in *Cloud Storage*.

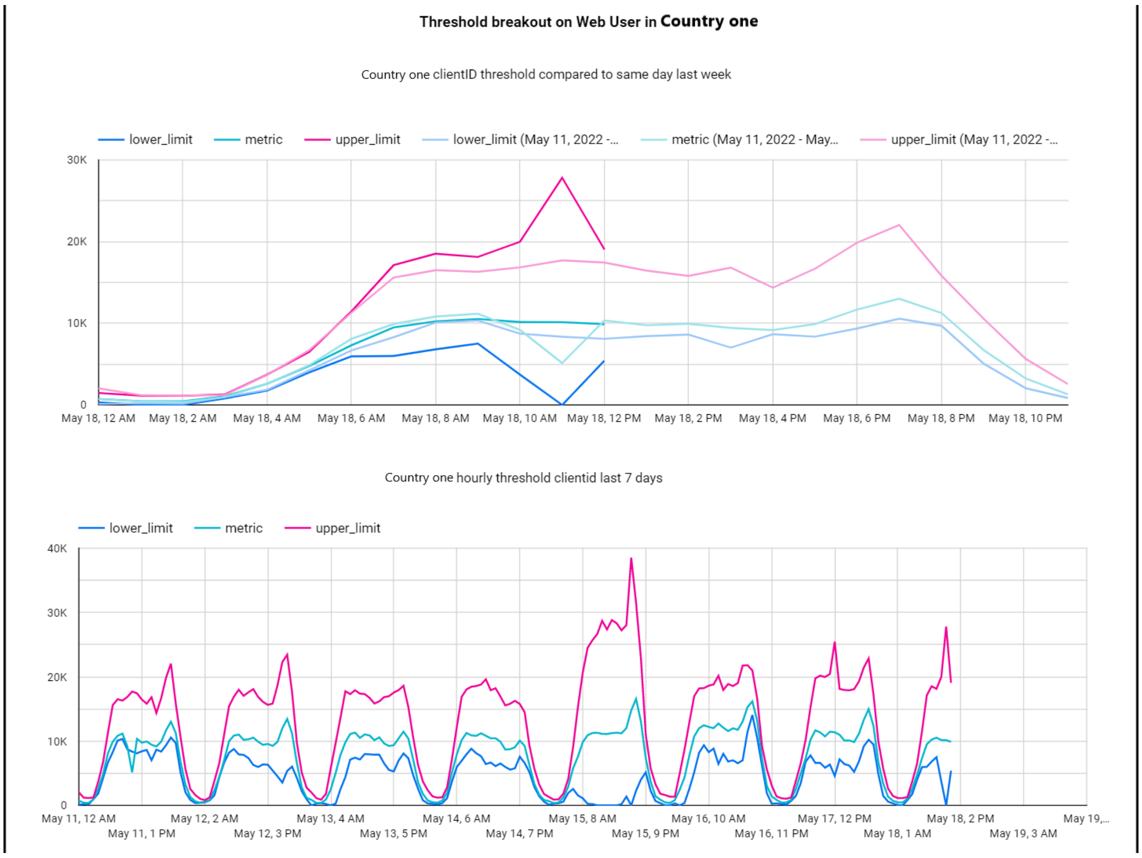


Figure 4.2: Page one of the dashboard.

4.4 Dashboard

The dashboard contains four pages. The first page contains graphs showing the metric and thresholds of the current day compared to the same weekday a week before. It also contains an overview of the metrics and thresholds for the whole week (see Figure. 4.2). The second page contains three graphs where **retailid**, **clientid** and **gqlentrypoint** can be controlled to get a detailed view of how the metrics for specific countries, clientIds and/or gqlEntrypoints look like (see Figure. 4.3). The third page contains two graphs (see Figure. 4.4). The first graph shows the average duration of actions aggregated by hour. This graph is also controllable and can be filtered on **retailid**, **clientid** and **gqlentrypoint**. The second graph on this page shows entries in the raw logs where gqlEntrypoint is NULL.

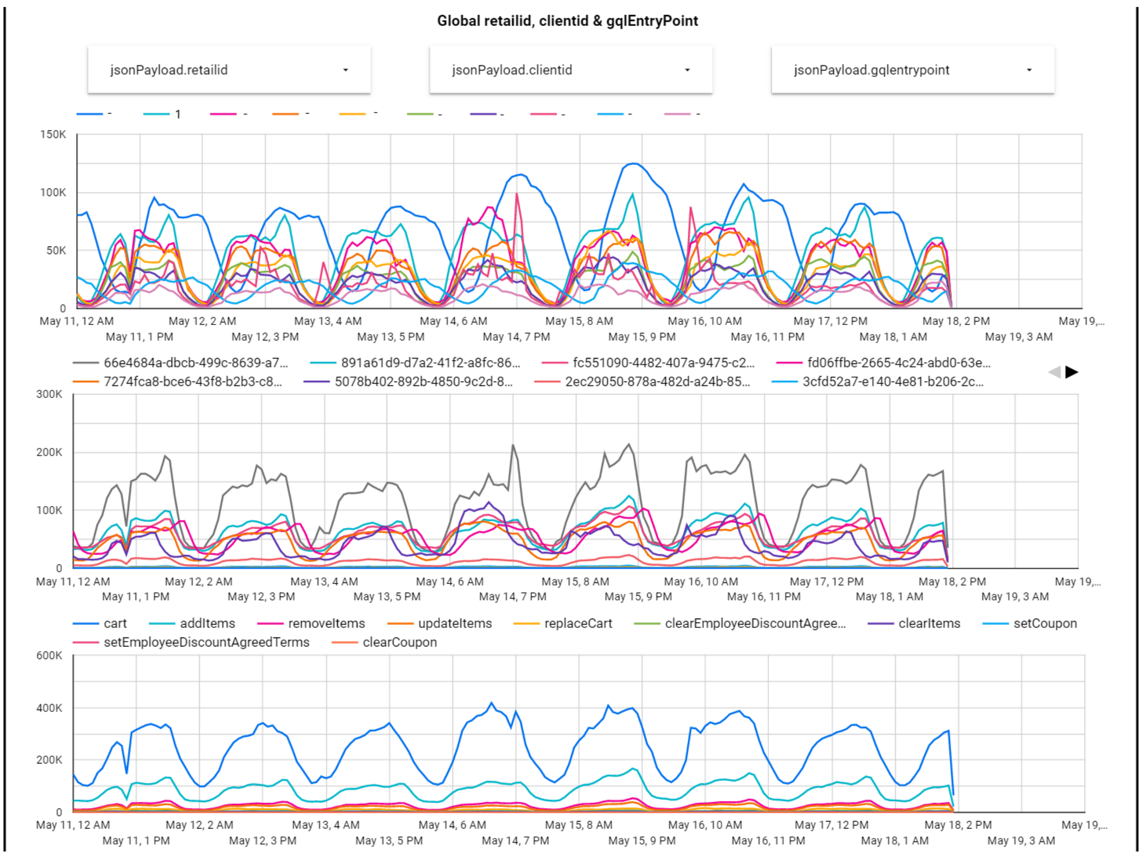


Figure 4.3: Page two of the dashboard.

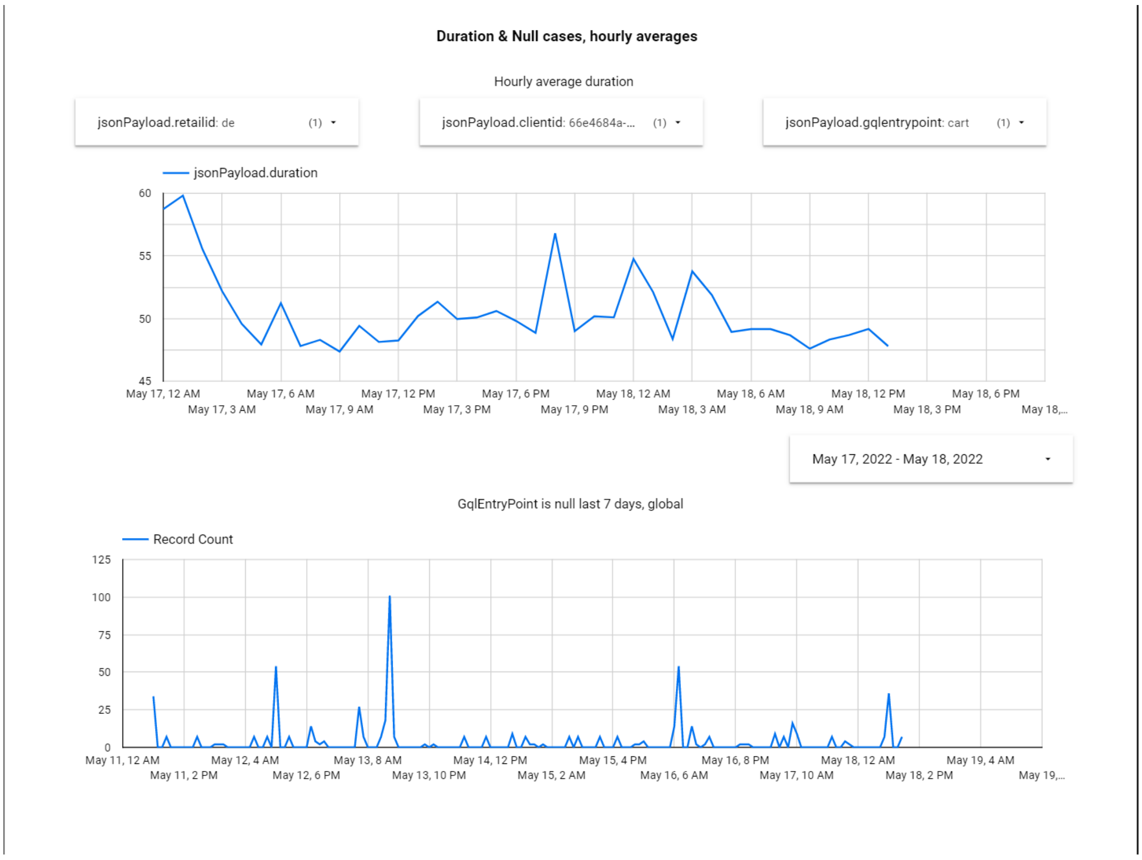


Figure 4.4: Page three of the dashboard.

4.5 Testing

The testing gave eight minutes of downtime for the 10 am to 11 am time frame and 30 minutes of downtime for the 11 am to 12 am time frame (see Figure 4.5).
Compared to the same time and day the previous week the 10 am average had dropped by 18.3 percent ($\bar{x}$ = 9201, $\bar{X}$ = 11266, P% = 81.7) while the 11 am average dropped by 54.6 percent ($\bar{x}$ = 5130, $\bar{X}$ = 11300, P% = 45.4).

$$P\% = (\bar{x}/\bar{X}) * 100 \tag{4.1}$$

$P\%$ = percentage change
$\bar{x}$ = average usage of that hour
$\bar{X}$ = average usage on the previous identical hour and weekday

Assuming equal usage the whole hour by using equation 4.2, the 10 am assumed average would be 10617 ($\bar{x}$ = 9201, ma = 52, nda = 10617) resulting in a decrease of 13.3 percent ($\bar{x}$ = 9201, $\bar{X}$ = 10617, P% = 86.7). In the same way the 11 am assumed average would be 10260 ($\bar{x}$ = 5130, ma = 30, nda = 10260) and a user decrease of 50 percent ($\bar{x}$ = 5130, $\bar{X}$ = 10260, P% = 50).

$$nda = \bar{x}/ma * 60 \tag{4.2}$$

nda = average if same amount of usage is assumed for the whole hour
$\bar{x}$ = average usage of that hour
ma = minutes of activity thus far in that hour

The 11 am to 12 am test broke the lower limit and thus triggered an alert, resulting in a *Slack* message informing of the brokent threshold and a link to the dashboard.

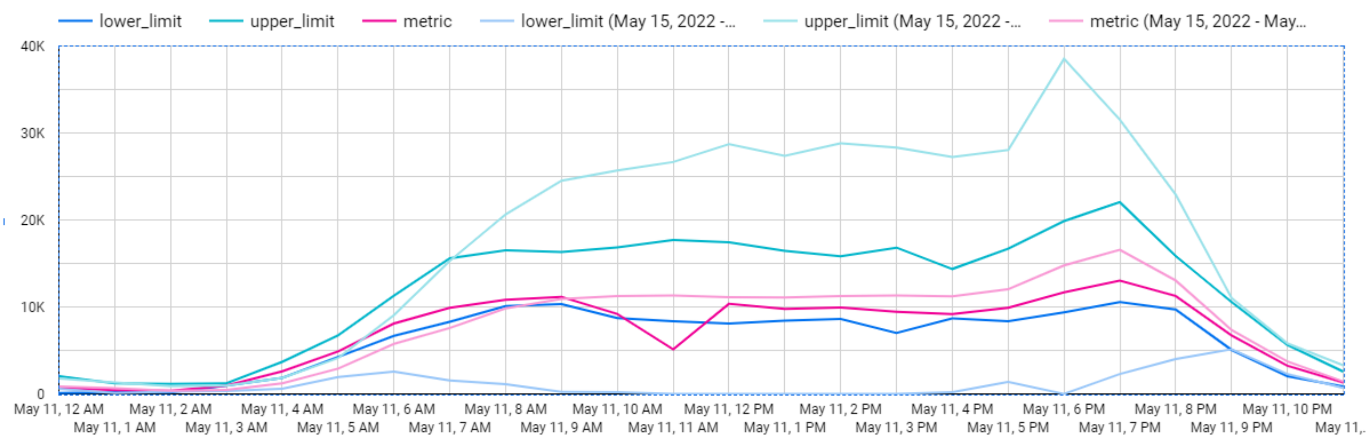
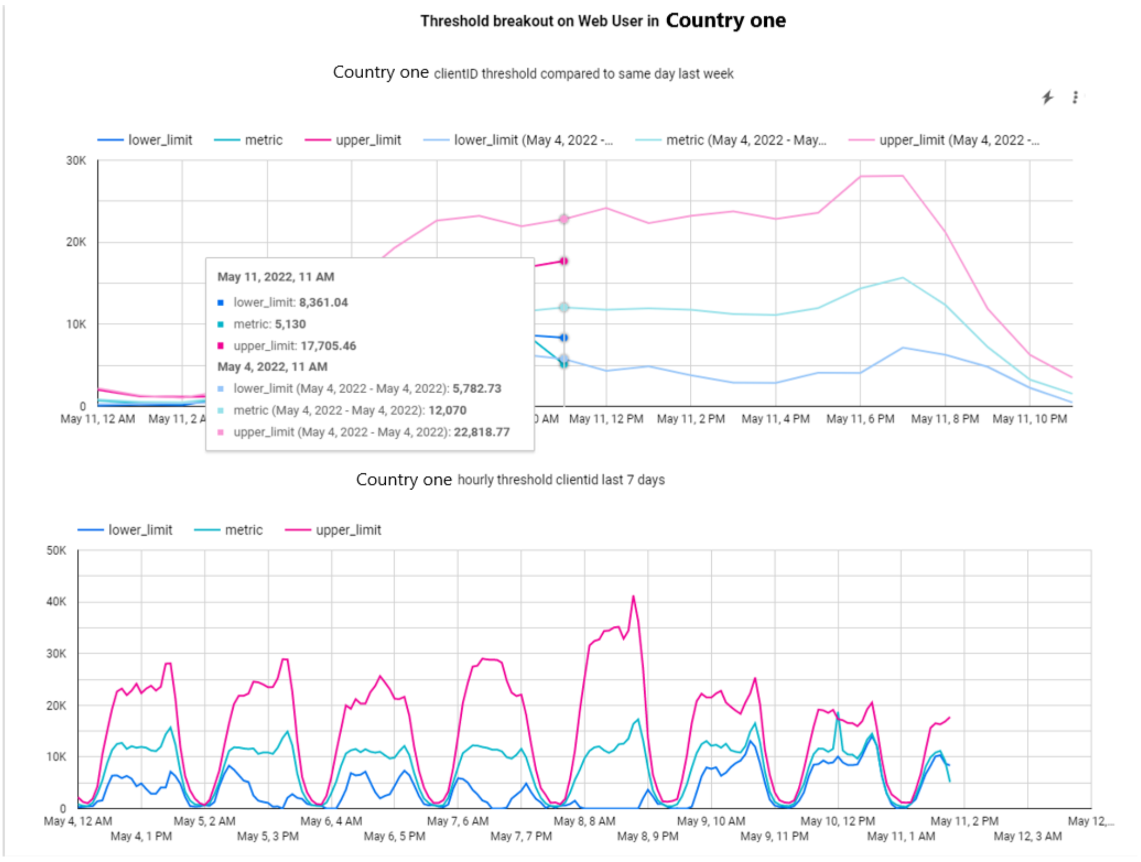


Figure 4.5: The prototype model on the simulated test day, taken from the dashboard. The thick lines are the metrics for that day while the transparent are a comparison to last weeks model.

4.5.1 Dashboard

As a result of the simulated threshold break the dashboard showed updated information. The decrease in average compared to previous week is visible on the first page and on the bottom figure the 11 am decrease is also visible (see Figure 4.6). Second page shows a decrease on a global scale on all factors (see Figure 4.7) while the third page shows an slight upward trend in the duration and nothing out of the ordinary in the null cases (see Figure 4.8).

The thresholds varies largely over the day and week to week. At nights the model is more sensitive as the standard deviations are smaller at those times. To trigger an alert it would on average not require more than a deviation of around 20 percent activity. On the other hand, during daytime the standard deviations show huge variance. There exist hours that need zero activity during the whole hour to cause an alert, same hours would require an increase of upwards 300 percent usage to cause an upper threshold to be broken.



May 11, 2022, 11 AM
- lower_limit: **8,361.04**
- metric: **5,130**
- upper_limit: **17,705.46**

May 4, 2022, 11 AM
- lower_limit (May 4, 2022 - May 4, 2022): **5,782.73**
- metric (May 4, 2022 - May 4, 2022): **12,070**
- upper_limit (May 4, 2022 - May 4, 2022): **22,818.77**

Figure 4.6: The first page of the dashboard at the hour of the simulated threshold break. Showing the current metric in strong colors at the top and last weeks metrics in the transparent colors. Bottom figure presents the whole weeks metrics.

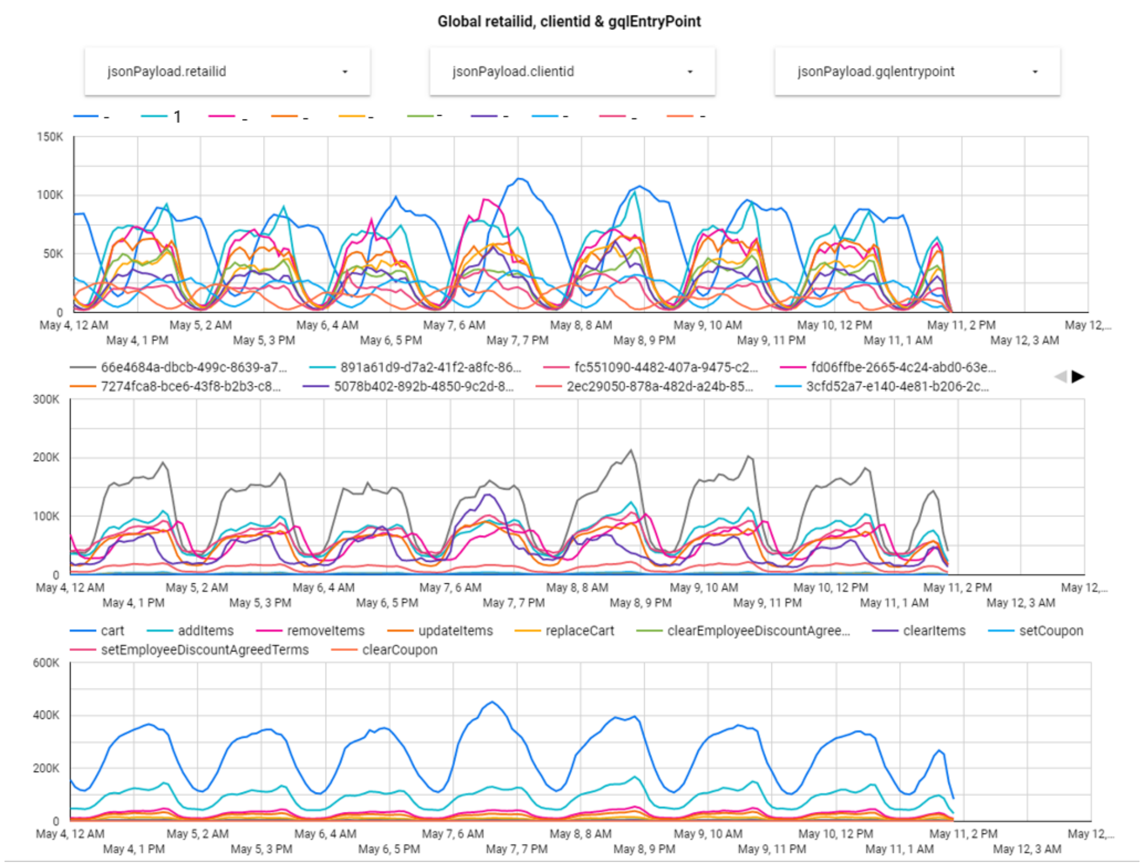


Figure 4.7: The second page of the dashboard at the hour of the simulated threshold break. Showing the global market on country-, clientId-, and gqlEntryPoint level.

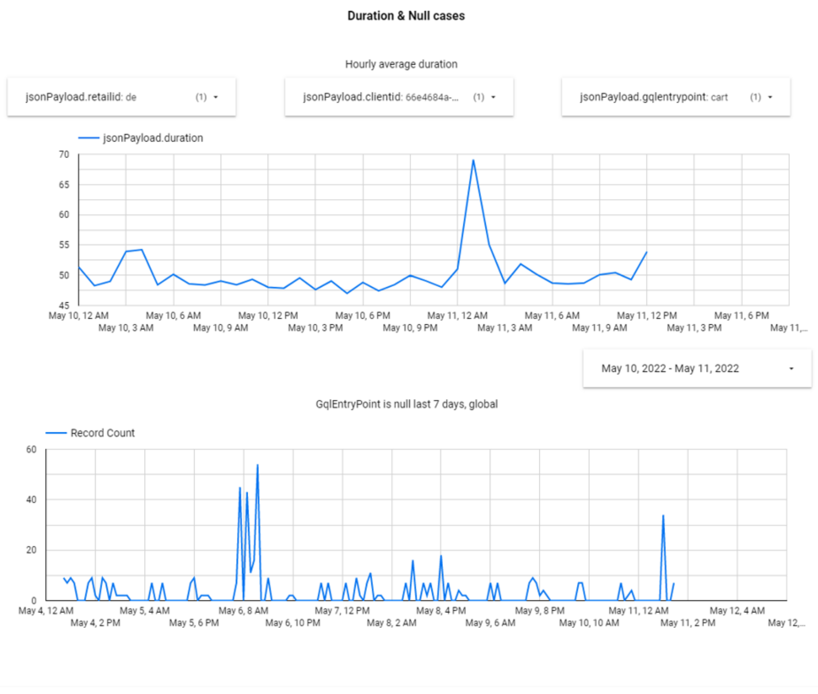


Figure 4.8: The third page of the dashboard at the hour of the simulated threshold break. The average duration for each hour can be seen at the top, with the possibility to change parameters. GqlEntryPoint null cases are seen at the bottom.

Chapter 5

Discussion

In this chapter thoughts and reflections that arose during development will be discussed further. The results, decisions, future prospects and ethical aspects will be discussed.

5.1 Reflection on results

The purpose of the thesis was to investigate the possibility of finding abnormal usage based on user generated log data and to make a prototype that generated alerts from said data. The built prototype uses log data every hour to determine the thresholds of what normal behaviour is and alerts when usage is outside said thresholds. The results tells us that the purpose of this thesis has been investigated and proven possible as the log data is demonstrated to work as the basis on which the threshold tables are created and anomalies determined from. Furthermore, the testing presented in the results confirms that the system detects anomalies.

With that said, while the prototype is a successful proof of concept, it has some obvious flaws in the anomaly detection. The amount of false alerts found in the prototype were significant, especially with the first script. However, even if the prototype does send false alerts it also succeeded in finding real anomalies as proven by the testing done. Furthermore, the dashboard could potentially provide great value for faster troubleshooting once a problem occurs. Thus very useful tools could be developed based on this prototype.

Looking at the running cost of this prototype, the initial costs were expensive at nearly 20 dollars per day. However, as mentioned in chapter 3 section 3.6, an optimization of the scheduled query reduced daily costs by almost 90% to about two dollars per day. On average that would conclude to about 60 dollars per month. As the system runs the storage cost will increase linearly but at a very slow rate, thus not considered as an increasing cost for this discussion. Just looking at Country one, the profit gained by IKEA IT AB from the online shopping is approximately 1 million euros per hour. In a context of this prototype, that means it would take 500 000 days (approx. 1370 years) for the prototype to reach the cost of a single hour of production. This implies that if the model prevents more than a total of one hour of down-time in production over a span of 1370 years it would be beneficial for IKEA IT AB. However, as the cost of even a minute is substantial (approx. 16.7k euros) the benefits of noticing anomalies even faster is definitely of interest to investigate further.

The amount of downtime needed to trigger a response in this prototype varies heavily depending on the time of day because of the large variation in the standard deviation. As an example, during the night a decrease of 20 percent in usage for an hour will usually trigger an alert. While during the day 100 percent downtime can be required for the whole hour to trigger an alert. Increase in usage also show the same variation between day and night, where night time is relatively sensitive while daytime can need up to 300 percent increase before triggering an alert. The implication of this is that the threshold model is incomplete as it occasionally needs extreme cases to trigger alerts. A complete lack of user activity for a whole hour would mean enormous losses to IKEA IT AB before an alert is triggered. The reason behind a need for these extreme cases is that the prototype only looks at the data once per hour and due to the incomplete threshold model that does not completely fit the real world data.

When comparing the amount of broken thresholds between the two implemented scripts a clear difference was noted. The use of consecutive threshold breaks removed ten of the eleven noted broken thresholds that had been observed since the setup was started. As no real anomaly had been found during this thesis work it can be assumed that all broken thresholds were false alerts. Meaning that nothing out of the ordinary was happening at the time and that the cause of an alert was just a flaw in the threshold model. Therefore, the use of consecutive threshold breaks could be said to remove ten of eleven false alerts. The usefulness of a system relies heavily on how accurate the alerts are, making the consecutive script highly relevant. The downside of using consecutive threshold breaks are the doubling of the reaction time for an alert since the system now requires two time-frames to spot any anomaly. An increase of detection time from one hour to two hours is an enormous difference when applied to a system where every offline minute means a huge economic loss. As a result of that, the usefulness of the consecutive script on an hourly time-frame would be worse than looking into a better model that would have the possibility to flag on each threshold break. Another way to improve the system to keep the consecutive script would be an increase in the update frequency. Decreasing the time between data updates and function execution would lessen the severity of doubling the time-frame and make the consecutive solution more viable. Furthermore, it is also possible to make the model require more than two consecutive broken thresholds to trigger. Resulting in an even lower possibility that it is in fact a falsely broken threshold [2].

Another problem with having large time-frames is in the case where the downtime or system failure falls not completely within an hour but is split over two consecutive hours in such a way that no alerts are triggered for any of the two hours. This would potentially cause certain cases to not be noticeable even if anomalies did happen. A model that takes into account the last hour's values and therefore notices a continuation of a downtrend could possibly find this type of behaviour. An increase in frequency of the query and function execution would also reduce the size of down times that would go unnoticed.

The many false alerts produced by this model is one of its biggest problems. As a user of the system it would cause distrust in a system if a majority of the alerts were false. The time spent looking for a potential problem when there is none is wasted time and when a problem does get spotted it might not be taken seriously because of all the previous false alerts. Therefore, for future work a priority should be in minimizing the amount of falsely broken thresholds.

As the model for the prototype only takes into account the last 4 weeks it will always miss the annual trends. This means that the activity effects of e.g. Black Friday and Christmas day will always be flagged as anomalies. It is not necessarily negative to be aware of these anomalies happening, but given the size of IKEA IT AB they are fully aware of these events beforehand and therefore these notifications could be seen as excessive.

There are theoretical cases where even the current prototype would be useful. For example, in the event of an outage or failure of some sort during a time where it is hard to get a hold of staff. Calculations suggests that a response time of two hours could be possible. In the event of such a case the time it would take for the current version of the prototype to give an alert could be useful to quicker noticing a problem and the information provided in the dashboard could help in finding the source of the problem. Another useful scenario would be in a situation where the system appears to work normally while there in reality is an error in the system. One such example that was given by IKEA IT AB were the implementation of faulty coupon codes. Such an error would potentially create a large influx of user data with people wanting to take advantage of the faulty codes. Internally everything would look normal until the large user influx would start creating other problems such as higher response times etc. The system that was created could potentially notice such an error before a collapse or decline in performance of any of the involved systems.

5.2 Reflection on decisions

Many discussion and choices had to be made during the development of the system and investigation of data. A big decision that was made early was the exclusion of Country two from all models. Figure 3.8 shows the spike of Country two's usage. The spike was deemed unnatural and an anomaly because of the lack of usage in Country two thereafter and also because it was known that the stores would close in Country two soon. Meaning, Country two's population would most likely start stockpiling and that was indeed the case proven by the figure.

This speaks to the importance of digesting and processing data before making decisions during a project. Gaining a proper understanding of what the data represents and how the surroundings affect the system is highly valuable to make the most qualified decision during development. Which in turn, makes a large impact on the direction, scope and quality of the final product.

In our case, the final model only looks at Country one and therefore it was not directly affected. However, say the model was instead looking at an European scale and the model was made during a time when Country two was shut down. The unpredictability of how the model would react to Country two opening up again could cause problems.

The choice of which parameter to base the threshold model on would have a huge impact on when the prototype would notice anomalies. Therefore, a large portion of the thesis was spent analysing data and getting an understanding of what the data actually represented and meant. Ideas of looking at the whole of Europe was discussed and also to look at the android, IOS and web users instead of just the web users. However, when looking at multiple countries or multiple operating systems problems appear because of the larger scope. If a small country would shut down completely it might not be noticed and similarly if major problems were to occur for a single operative system it might not be noticed due to the larger scope. As a result of that the decision was made to single out a very specific usage and focus on that, in turn reducing the noise in the model. A loss of that decision is the alerting for all other problems except the web users in Country one. If more time existed for the thesis a suggestion was to implement the same prototype on other countries or clientIds to get a bigger coverage of the active market.

In phase 4 the decision between using identical weekdays and just days prior was also an important one (see Figure 3.14 and Figure 3.15 respectively). Previous investigation of data had shown strong indications of trends that varied depending on weekday. Comparing the two previously mentioned graphs showed that the model taking into account weekdays had less falsely broken thresholds. Seen specifically at 2nd of April in the figures. This was a strong indicator that the data used had biases for weekdays and had to be taken into account.

Testing of the prototype only happened once during the thesis work. However, the alerts were confirmed to work earlier as normal usage occasionally caused falsely broken thresholds. Even if math can be used to calculate when the prototype would alert during different user activity it is not possible to calculate how the prototype would react to different parts of the system failing or behaving abnormally. Therefore, further testing of those areas would be beneficial in telling how well this type of monitoring would handle different types of outages or user trends. Another area that could have had more testing is how well the dashboard actually assists in troubleshooting. By setting up a fake case and asking the team to use the prototype to come to a conclusion of the problem could be a way to investigate the usefulness of the dashboard.

5.3 Future development

One of the most prominent improvements for this model would be increasing the frequency of execution. This increase would directly make the reaction time faster and therefore any abnormal behaviour would be noticed more quickly. As a result, the troubleshooting process can start earlier and in turn save more money as the problem would hopefully be resolved faster. The downside would be an increased running cost as the scripts and queries would be executed more often. As the data used for executing more frequently would be the same, a speculation of linear increase in costs has been made. That would mean to check for anomalies every five minute would then cost twelve times more and land on 24 dollars per hour. Still an insignificant number compared to the cost of each offline minute.

A problem that would surface when increasing the frequency is the fluctuation of the threshold model on a smaller time-frame. When investigating data, the minute to minute variation could be enormous because of random simultaneous occurring usage. When looking at a larger time-frame the fluctuation mostly disappeared. The models based on an hourly time-frame had far less fluctuation that was extreme but instead mostly followed trends. Therefore, it can be assumed that five minutes would also suffer somewhat from the same fluctuation problem. To counteract this, consecutive threshold breaks could be used as it has been shown to lessen the amount of false threshold breaks. The increase of response time would still be a lot faster than the current prototype.

However, the better solution is revisiting the threshold model completely. The technical expert suggested machine learning to be the most efficient way of handling the thresholds. This solution would most likely allow for a lot faster alerting times while also lessening the falsely broken thresholds significantly at the same time. Prediction of the cost for a machine learning model has not been done in this thesis but can be speculated to be minute compared to the cost of even a single minute of system downtime at IKEA IT AB.

The fields and areas in which this concept could be applied is also enormous. The prototype simply looked at one specific tiny area of the cart usage, whereas by changing which parameters trigger an alert it could most likely be applied to the whole cart and actually to any part of a system that processes user data. As most products created have an end user, the necessary user data can be collected and analysed to quickly find problems in the usage. Thereby providing a better customer support with faster troubleshooting, more up time as anomalies can probably be noticed before a whole system goes down and thus saving money for the company in question.

As mentioned earlier the dashboard purely looks into data relevant for the troubleshooting in this prototype. However, given the type of data that is processed it is probable to analyse patterns and trends in user behaviour related to the company business. Greater understanding in what way the system monitored is used is of enormous use in fine tuning or shaping the product to better fit the business model that is aimed for. An example is to gather user activity during a down-time or a decline in performance to calculate the economical loss and change in user behaviour to have grounds for decisions surrounding areas that might need improvement in the company or ways to direct user behaviour when the product is not working correctly.

5.4 Ethical Aspects

There is one ethical aspect connected to the project that warrants discussion. This is the ethics regarding the collection and storing of user data in this project. While the collection and storage of data could be regarded as unethical, especially if done without proper consent from the users, the collection done in this project can be seen as ethically viable. This is due to the collected log data containing no real identifiers that makes it possible to connect the data collected to any specific individual. However, the raw logs does contain an userID column that could possibly be used with information from other departments to build a profile on the user. Nevertheless it can be argued that in a production environment it's not viable to save the raw logs indefinitely which means the raw logs containing the userID column would be deleted after a set amount of time. Also IKEA IT AB are very open about their data collection to users which means that collecting data can be seen as ethically viable [1]. In summary, as long as the scheduled queries aren't modified to start including the userID and IKEA IT AB keeps its open integrity policy the collection of data in the project should be ethical.

Chapter 6

Conclusion

The first step in creating a system that alerts when user behaviour is following an abnormal trend was to look into what user generated log data was stored by IKEA IT AB. By analysing this data a conclusion could be made of which parameters would be most useful for getting an overview of the user activity. Using *BigQuery* to save the chosen data, a threshold model was made looking specifically at the cart activity and more exact the web users in Country one. This model was based on that standard deviation was applicable on this data and that the pattern held for weekday based trends on an hourly level. A prediction of what could be called normal behaviour was therefore determined through this threshold model. Alerts were implemented that would notify when user activity fell outside previously mentioned normal behaviour, this notification was sent through *Slack* containing a link to the dashboard created for the purpose of assisting in troubleshooting.

This prototype fulfilled all the criteria for the purpose of this thesis, making a proof of concept if user generated logs could be used for tracking abnormal behaviour and if the same data could help troubleshooting. The following questions were answered during this thesis:

1. What data can be extracted from the logs?
   A summary of the data fields determined to be of use for the purpose of this thesis is seen in figure 3.1
2. How can the project be run cost-efficiently?
   By keeping the storage to only the required data for detection and troubleshooting and by making sure the queries are executed efficiently on partitioned tables the costs can be kept minimal.
3. How to implement the thresholds to reduce, or increase the amount of alerts?
   As the model for thresholds was based on standard deviation, changing sigma or time-frame of the model would affect the amounts of alerts sent. Other models could be investigated further to allow for other ways to modify the alert accuracy and frequency.
4. What data to visualise for assisting in troubleshooting?
   By suggesting many different graphs to the team for the purpose of troubleshooting it was possible to pinpoint what data was believed to be most useful to present in the dashboard. However, no testing was done to confirm the usefulness of it in production.
5. Which mathematical models can be used to perform relevant data analysis?
   Investigating different mathematical models and talking with the technical expert landed in the model of choice being standard deviation model based on normal distribution. Other mathematical models could potentially solve the occasional false alerts produced. The best choice for model would most likely be a machine learning one.
6. What user generated log data are of interest for finding potential anomalies?
   By first gaining an understanding of how the real life actions were mirrored in the log data and then visualising and analysing that data it was possible to summarise what user activity could be seen in the log data. That data was then used for determining what would be the best to use for noticing anomalies together with feedback from the team.

Although the prototype does find anomalies, the current model is lacking due to extreme cases sometimes being needed to trigger an alert, many falsely broken thresholds and a large time-frame needed before an alert is sent. However, the most important conclusion from this thesis is the potential of how useful a system like this can be. The cost from using this prototype at the time-frame of once per hour is around 2$ per day. If the prototype prevents more than 1 hour of downtime in 1370 years, it would make up for it's running cost.

The foremost way of improving the threshold model is with either increasing the frequency or by changing the mathematical model. The frequency change would result in faster detection of anomalies while model change would lessen the amount of false threshold breaks while increasing the accuracy and scope of anomalies found. The type of model most befitting this task would most likely be a machine learning model. These improvements would in all probability make the model more expensive to run but also detect anomalies faster and thus save more money.

The areas where this type of system could be applied and be of use is large. Any product that collects data from end users could in theory implement this system. The user of the system is also not limited to just assisting in troubleshooting when alerts has been triggered. Since other types of data can also be presented in the dashboard it could display information useful for the business perspective of the company too.

Chapter 7

Glossary

clientId Refers to the field in the JSON log file that defines what application, tool or platform that was used to perform the action that generated the log entry.. 19–22, 25, 30, 31, 39

cron Cron is a utility program that helps user to schedule tasks to repeat at a specific time [23]. A task scheduled in cron is called a cron job. Users can choose what task they want to automate and when to execute it. . 11

dataset A collection of data. Since this report mainly handles tabular data, a dataset corresponds to one or more database tables.. 8, 19, 25

GCP Acronym for Google Cloud Platform.. 8, 15, 25

gqlEntrypoint Refers to the field in the JSON log file that defines what action that was performed to generate the log entry.. 15, 19, 20, 23, 30, 31

IDE Acronym for integrated development environment. An IDE is a software environment that includes the basic tools required to write and test software. Examples include Microsoft Visual Studio, Eclipse and IntelliJ. . 25

partitioned table A partitioned table is a table that is divided into segments, called partitions. This makes it more efficient to query and manage the data.. 26, 41

pub/sub Pub/sub, also called publish–subscribe pattern, is a messaging pattern where the sender, a.k.a. publisher, doesn't program the message to be sent to any specific subscriber. Instead the publisher categorizes messages into classes without knowing if there are any subscribers. On the other end the receiver, a.k.a. subscriber, express interest in one or more classes and therefore only receive messages that are of interest without having specific knowledge about who is the publisher. . 26

retailId Refers to the field in the JSON log file that defines in what market, i.e. country, the action was performed to generate the log entry.. 19, 20, 30

scrum Scrum is a lightweight framework that helps people, teams and organizations generate value through adaptive solutions for complex problems [28]. . 13

sharded table A sharded table is a larger table that has been separated into multiple smaller tables.. 26

sink A sink or data sink generally refers to the destination of data flow. In this report sink refers to the feature of Cloud Logging that routes logs to a dataset.. 19, 26, 27

userAgent Refers to the field in the JSON log file that defines what device that was used to perform the action that generated the log entry.. 19, 30

web service A service offered by an electronic device to another electronic device that communicates via the internet. Possesses both a front-end and a back-end. . 8

webhook A method of augmenting or altering behavior of a web page or web application with custom callbacks.. 26

Chapter 8

References

[1] IKEA Svenska Försäljnings AB. *Integritetspolicy för IKEA kunder [Privacy policy for IKEA customers]*. URL: https://www.ikea.com/se/sv/customer-service/privacy-policy/integritetspolicy-pub019133cd (visited on 05/28/2022).

[2] Anomaly.io. *Anomaly Detection with the Normal Distribution*. URL: https://anomaly.io/anomaly-detection-normal-distribution/index.html (visited on 04/14/2022).

[3] Python Software Foundation. *What is Python? Executive Summary*. URL: https://www.python.org/doc/essays/blurb/ (visited on 05/09/2022).

[4] Google. *App Engine documentation*. URL: https://cloud.google.com/appengine/docs (visited on 05/03/2022).

[5] Google. *BigQuery documentation - What is BigQuery?* URL: https://cloud.google.com/bigquery/docs/introduction (visited on 05/03/2022).

[6] Google. *BigQuery Product Page*. URL: https://cloud.google.com/bigquery (visited on 05/03/2022).

[7] Google. *Cloud Billing Documentation*. URL: https://cloud.google.com/billing/docs (visited on 05/27/2022).

[8] Google. *Cloud Billing Documentation - Cloud Billing Reports*. URL: https://cloud.google.com/billing/docs/reports (visited on 05/27/2022).

[9] Google. *Cloud Functions documentation - Writing Cloud Functions*. URL: https://cloud.google.com/functions/docs/writing (visited on 05/05/2022).

[10] Google. *Cloud Functions Product Page*. URL: https://cloud.google.com/functions (visited on 05/03/2022).

[11] Google. *Cloud Logging Product Page*. URL: https://cloud.google.com/logging (visited on 05/10/2022).

[12] Google. *Cloud Monitoring documentation*. URL: https://cloud.google.com/monitoring/docs (visited on 05/09/2022).

[13] Google. *Cloud Scheduler Product Page*. URL: https://cloud.google.com/scheduler (visited on 05/19/2022).

[14] Google. *Cloud Storage documentation*. URL: https://cloud.google.com/storage/docs (visited on 05/27/2022).

[15] Google. *Cloud Storage documentation - What is Cloud Storage?* URL: https://cloud.google.com/storage/docs/introduction (visited on 05/27/2022).

[16] Google. *Google Data Studio Connect to Data*. URL: https://datastudio.google.com/data (visited on 05/09/2022).

[17] Google. *Google Data Studio Overview*. URL: https://datastudio.google.com/overview (visited on 05/09/2022).

[18] Google. *Introduction to partitioned tables*. URL: https://cloud.google.com/bigquery/docs/partitioned-tables (visited on 05/12/2022).

[19] Google. *Why Google Cloud*. URL: https://cloud.google.com/why-google-cloud (visited on 05/10/2022).

[20] The PostgreSQL Global Development Group. *PostgreSQL: Documentation 9.1: Windows Functions*. URL: https://www.postgresql.org/docs/9.1/tutorial-window.html (visited on 05/03/2022).

[21] JetBrains. *Features - PyCharm*. URL: https://www.jetbrains.com/pycharm/features/ (visited on 05/10/2022).

[22] Puneet Jindal. *Google BigQuery Architecture: The Comprehensive Guide*. 2022-01-07. URL: https://hevodata.com/blog/google-bigquery-data-warehouse/ (visited on 05/03/2022).

[23] Linas Levanas. *Cron Job: A Comprehensive Guide for Beginners 2022*. 2022-04-21. URL: https://www.hostinger.com/tutorials/cron-job (visited on 05/27/2022).

[24] Sarah Littler. *The Importance and Effect of Sample Size*. URL: https://select-statistics.co.uk/blog/importance-effect-sample-size/ (visited on 04/15/2022).

[25] Microsoft. *Flowchart Maker and Diagramming Software — Microsoft Visio*. URL: https://www.microsoft.com/en-ww/microsoft-365/visio/flowchart-software (visited on 05/09/2022).

[26] Microsoft. *Welcome to Microsoft Teams - Microsoft Teams — Microsoft Docs*. URL: https://docs.microsoft.com/en-us/microsoftteams/teams-overview (visited on 05/03/2022).

[27] Stack Overflow. *Stack Overflow Developer Survey 2021*. URL: https://insights.stackoverflow.com/survey/2021#most-popular-technologies-language (visited on 05/10/2022).

[28] Ken Schwaber and Jeff Sutherland. *The Scrum Guide*. URL: https://scrumguides.org/scrum-guide.html (visited on 04/14/2022).

[29] Slack. *Features — Slack*. URL: https://slack.com/features (visited on 05/10/2022).

[30] W3Schools. *SQL Introduction*. URL: https://www.w3schools.com/sql/sql_intro.asp (visited on 05/10/2022).

Chapter 9

Appendix

9.1 Scheduled Queries

Hourly Scheduled Query

```sql
#standardsql
create or replace table RawLogs.cartops_analysis_hourly
partition by DATE_TRUNC(timestamp, DAY)
as select
  jsonPayload.gqlEntrypoint
  , jsonPayload.retailId
  , jsonPayload.duration
  , jsonPayload.isAnonymous
  , jsonPayload.userId
  , jsonPayload.userAgent
  , jsonPayload.clientID
  , timestamp
from `RawLogs.cart_*`
where Timestamp_trunc(timestamp, hour) =
TIMESTAMP_SUB(Timestamp_trunc(CURRENT_TIMESTAMP(), hour), INTERVAL 1 HOUR)
AND jsonPayload.gqlEntrypoint = "cart"
AND jsonPayload.retailId = "de"
AND _table_suffix between FORMAT_DATE("%Y%m%d", DATE_SUB(CURRENT_DATE(),
INTERVAL 1 DAY)) AND FORMAT_DATE("%Y%m%d", CURRENT_DATE())
;


insert into `RawLogs.cartops_analysis_collection_hourly` (
select *
from `RawLogs.cartops_analysis_hourly`
)
;

INSERT into RawLogs.hourly_thresholds_weekday_clientId (metric, hourly_avg,
upper_limit, lower_limit, weekday, timestamp, std_dev)
SELECT (
    SELECT COUNT(gqlEntrypoint)
    FROM RawLogs.cartops_analysis_collection_hourly
    WHERE TIMESTAMP_TRUNC(timestamp, HOUR) =
    TIMESTAMP_SUB(TIMESTAMP_TRUNC(CURRENT_TIMESTAMP(), HOUR), interval 1 HOUR)
    # Change clientId based on preference
    AND clientId = "66e4684a-dbcb-499c-8639-a72fa50ac0c3"
    ) as metric,
AVG(
    metric
) as hourly_avg,
AVG(metric) + 4 *STDDEV_SAMP(metric) as upper_limit,
IF (AVG(metric) - 4 *STDDEV_SAMP(metric) < 0, 0, AVG(metric) - 4
*STDDEV_SAMP(metric) ) as lower_limit,
EXTRACT(DAYOFWEEK from TIMESTAMP_SUB(CURRENT_TIMESTAMP, interval 1 HOUR))
weekday,
TIMESTAMP_SUB(TIMESTAMP_TRUNC(CURRENT_TIMESTAMP(), HOUR), interval 1 HOUR) as
timestamp,
STDDEV_SAMP(
    metric
) as std_dev
from (
    SELECT COUNT(gqlEntrypoint) metric,
    TIMESTAMP_TRUNC(timestamp, HOUR) metric_hour,
    FROM `RawLogs.cartops_analysis_collection_hourly`
    WHERE EXTRACT(dayofweek from timestamp) = extract(dayofweek from
    TIMESTAMP_SUB(CURRENT_TIMESTAMP, interval 1 HOUR))
    and extract(hour from timestamp) = extract(hour from
    TIMESTAMP_SUB(CURRENT_TIMESTAMP, interval 1 HOUR))
    and timestamp between TIMESTAMP_SUB(CURRENT_TIMESTAMP(), INTERVAL 29 DAY)
    AND TIMESTAMP_SUB(CURRENT_TIMESTAMP(), interval 2 HOUR)
    # Change clientId based on preference
    AND clientId = "66e4684a-dbcb-499c-8639-a72fa50ac0c3"
    GROUP BY metric_hour
    )
;
```

Daily Scheduled Query

```
#standardsql
create or replace table RawLogs.cartops_analysis_daily
partition by DATE_TRUNC(timestamp, DAY)
as select
  jsonPayload.gqlEntrypoint
  , jsonPayload.retailId
  , jsonPayload.duration
  , jsonPayload.isAnonymous
  , jsonPayload.userId
  , jsonPayload.userAgent
  , jsonPayload.clientID
  , timestamp
from `RawLogs.cart_*`
where CAST(timestamp as DATE) = DATE_SUB(CURRENT_DATE(), INTERVAL 1 DAY)
AND jsonPayload.gqlEntrypoint = "cart"
AND jsonPayload.retailId = "de"
AND _table_suffix = FORMAT_DATE("%Y%m%d", DATE_SUB(CURRENT_DATE(), INTERVAL 1
DAY))
;

insert into `RawLogs.cartops_analysis_collection` (
select *
from `RawLogs.cartops_analysis_daily`
)
;
```

```sql

INSERT `RawLogs.thresholds` (metric, average, upper_limit, lower_limit, std_dev,
date)
SELECT (
    SELECT COUNT(gqlEntrypoint)
    FROM RawLogs.cartops_analysis_collection
    WHERE EXTRACT(DATE from timestamp) = CURRENT_DATE() - 1
    ) as metric,
AVG(
    metric
) as average,
AVG(metric) + 3 *STDDEV_SAMP(metric) as upper_limit,
AVG(metric) - 3 *STDDEV_SAMP(metric) as lower_limit,
STDDEV_SAMP(
    metric
) as std_dev,
CURRENT_DATE() - 1 as date
from (
    SELECT COUNT(gqlEntrypoint) metric,
    EXTRACT(DATE from timestamp) metric_date,
    FROM `RawLogs.cartops_analysis_collection`
    GROUP BY metric_date
    )
WHERE metric_date between DATE_SUB(CURRENT_DATE() - 2, INTERVAL 7 DAY) AND
CURRENT_DATE() - 2
;
```

9.2 Cloud Functions

Threshold Alert Cloud Function

main.py

```python
import base64
import check_threshold
import send_slack_message

def hello_pubsub(event, context):
    """Triggered from a message on a Cloud Pub/Sub topic.
    Args:
         event (dict): Event payload.
         context (google.cloud.functions.Context): Metadata for the event.
    """
    pubsub_message = base64.b64decode(event['data']).decode('utf-8')
    print(pubsub_message)

    result, timestamp = check_threshold.check_threshold()
    if result:
      send_slack_message.create_slack_message(timestamp)
```

check_threshold.py

```python
from google.cloud import bigquery

```

```python
def check_threshold():
    client = bigquery.Client()
    query_job = client.query(
        """
            select *
            from `RawLogs.hourly_thresholds_weekday_clientId`
            order by timestamp desc LIMIT 1
            """
    )

    results = query_job.result()

    for row in results:
        metric = format(row.metric)
        upper_limit = format(row.upper_limit)
        lower_limit = format(row.lower_limit)
        timestamp = format(row.timestamp)

    # If threshold in last hour has been broken, return true. Else false.
    if float(metric) > float(upper_limit) or float(metric) <=
    float(lower_limit):
        return True, timestamp
    else:
        return False, timestamp
```

send_slack_message.py

```python
import requests
import json

def create_slack_message(timestamp):
    payload = {"text": "A threshold has been broken at " + timestamp + "
    UTC!\nPlease check the
    <https://datastudio.google.com/reporting/dashboard_address_key|dashboard>
    for more information!"}
    webhook = 'https://hooks.slack.com/services/WEB_HOOK_KEY'
    send_slack_message(payload, webhook)


def send_slack_message(payload, webhook):
    return requests.post(webhook, json.dumps(payload))
```

requirements.txt

```
# Function dependencies, for example:
# package>=version
google-cloud-bigquery
```

Consecutive Threshold Alert Cloud Function

main.py

```python
import base64
import check_threshold
import send_slack_message
import read_write

bucket = 'test-bucket'
prev_result_blob = 'last_result.txt'

def main(event, context):
  result, timestamp = check_threshold.check_threshold()
  last_result = read_write.read_line(bucket, prev_result_blob)
  if result:
    if last_result == 'true':
      send_slack_message.create_slack_message(timestamp)
    read_write.write_line(bucket,
                          prev_result_blob,
                          "true")
  else:
    read_write.write_line(bucket,
                          prev_result_blob,
                          "false")
```

check_threshold.py

```python
from google.cloud import bigquery


def check_threshold():
    client = bigquery.Client()
    query_job = client.query(
        """
            select *
            from `RawLogs.hourly_thresholds_weekday_clientId`
            order by timestamp desc LIMIT 1
            """
    )

    results = query_job.result()

    for row in results:
        metric = format(row.metric)
        upper_limit = format(row.upper_limit)
        lower_limit = format(row.lower_limit)
        timestamp = format(row.timestamp)

    print(metric)
    print(upper_limit)
    print(lower_limit)


    # If threshold in last hour has been broken, return true. Else false.
    if float(metric) > float(upper_limit) or float(metric) <=
    float(lower_limit):
        return True, timestamp
    else:
        return False, timestamp
```

read_write.py

```
from google.cloud import storage


def write_line(bucket_name, destination_blob, text_line):
    client = storage.Client()
    bucket = client.get_bucket(bucket_name)
    blob = bucket.blob(destination_blob)
    blob.upload_from_string(text_line)


def read_line(bucket_name, destination_blob):
    client = storage.Client()
    bucket = client.bucket(bucket_name)
    blob = bucket.blob(destination_blob)
    with blob.open(mode='r') as f:
        return f.read()
```

send_slack_messages.py

```
import requests
import json


def create_slack_message(timestamp):
    payload = {"text": "A threshold has been broken consecutively at " +
    timestamp + " UTC!\nPlease check the
    <https://datastudio.google.com/reporting/dashboard_address_key|dashboard>
    for more information!"}
    webhook = 'https://hooks.slack.com/services/WEB_HOOK_KEY'
    send_slack_message(payload, webhook)


def send_slack_message(payload, webhook):
    return requests.post(webhook, json.dumps(payload))
```

requirements.txt

```
# Function dependencies, for example:
# package>=version
google-cloud-bigquery
google-cloud-storage
```

9.3 Threshold Prototypes

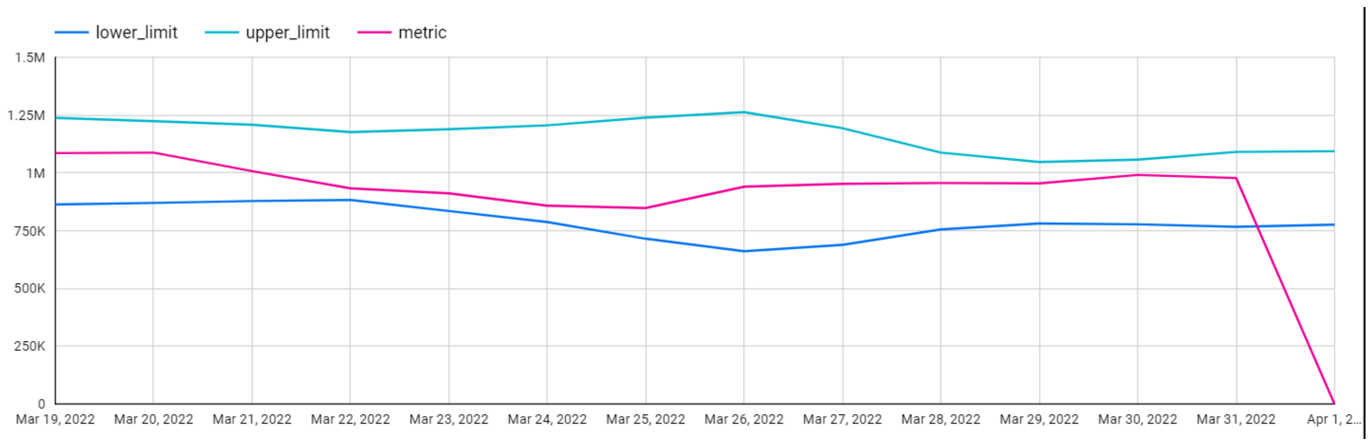


Figure 9.1: Daily Threshold Prototype

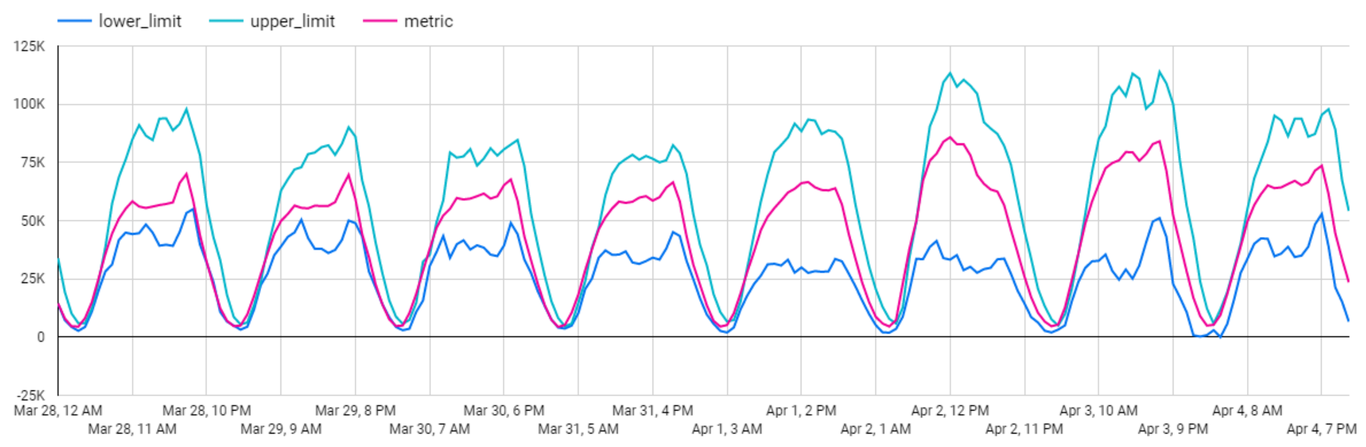


Figure 9.2: Hourly Threshold Prototype with standard deviation based on four previous matching weekdays.

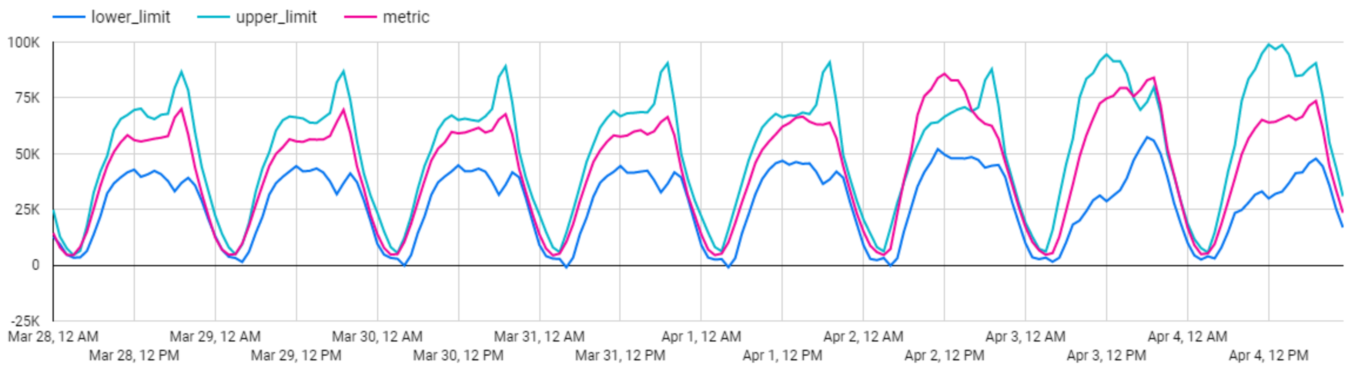


Figure 9.3: Hourly Threshold Prototype with standard deviation based on seven previous days.

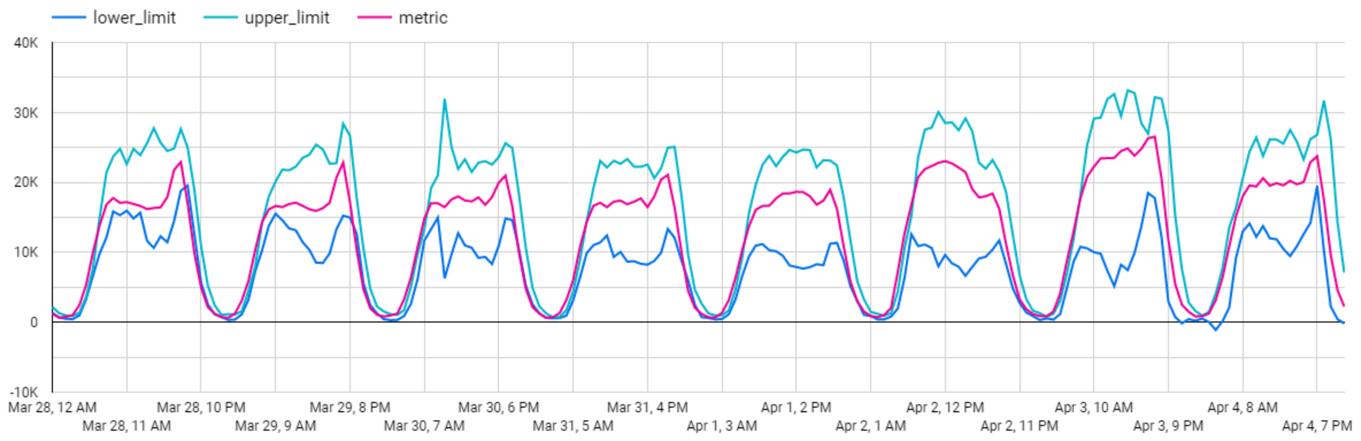


Figure 9.4: Hourly Threshold Prototype with standard deviation based on four previous matching weekdays and filtered on ”Web user” clientId

9.4 Data Studio Analysis

Global log data and country composition

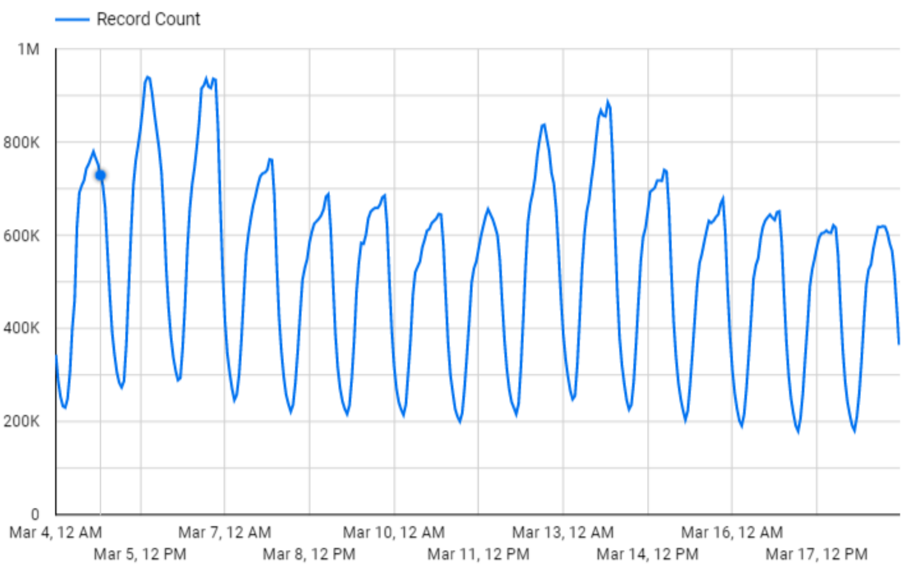


Figure 9.5: Global log data aggregated on hour.

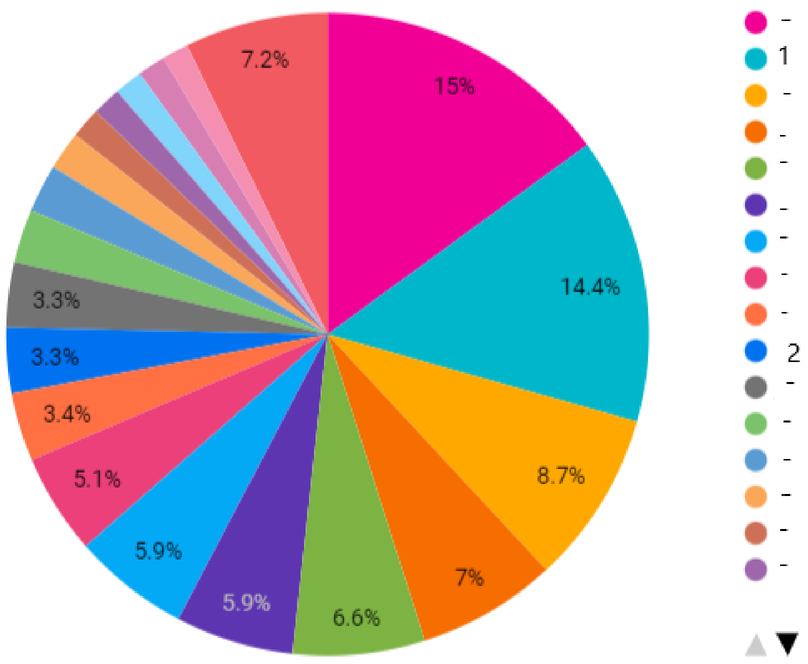


Figure 9.6: Country composition of log data.

gcpEntrypoint and duration aggregated on minute

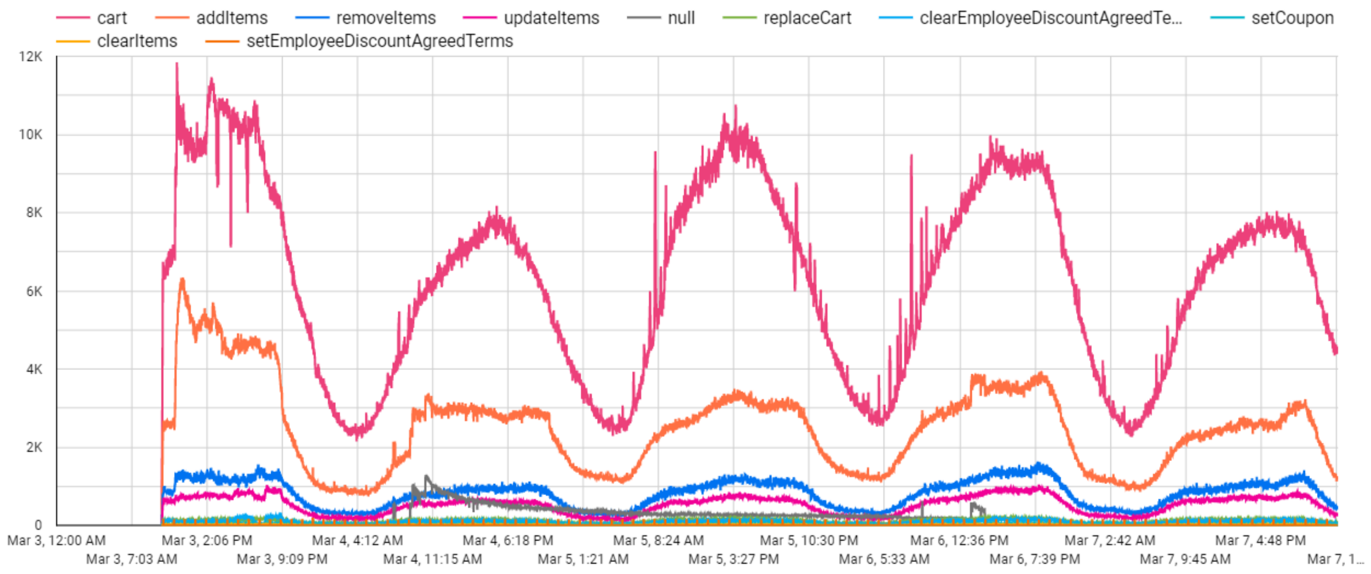


Figure 9.7: Global log data divided over gcpEntrypoint and aggregated on minute.

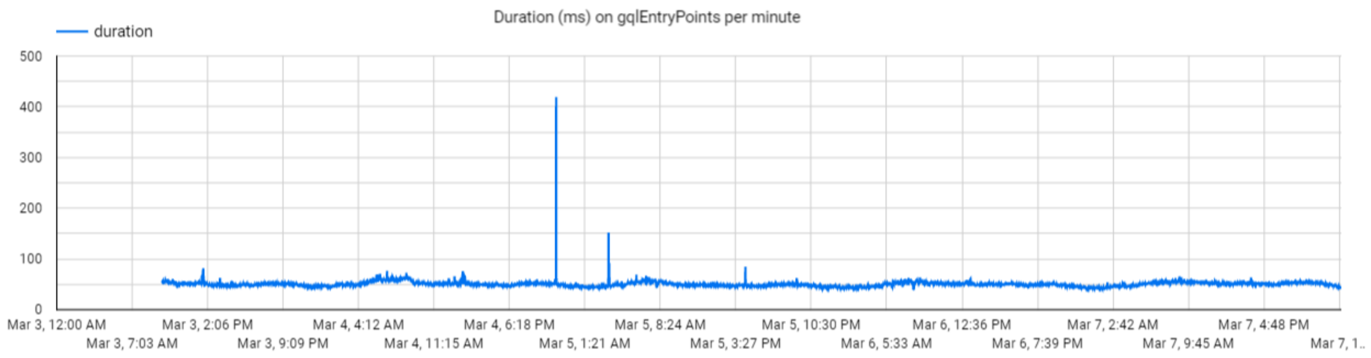


Figure 9.8: Duration on global log data aggregated on minute.

Average usage of cart operations(gqlEntrypoint) on each hour globally and in Country one.

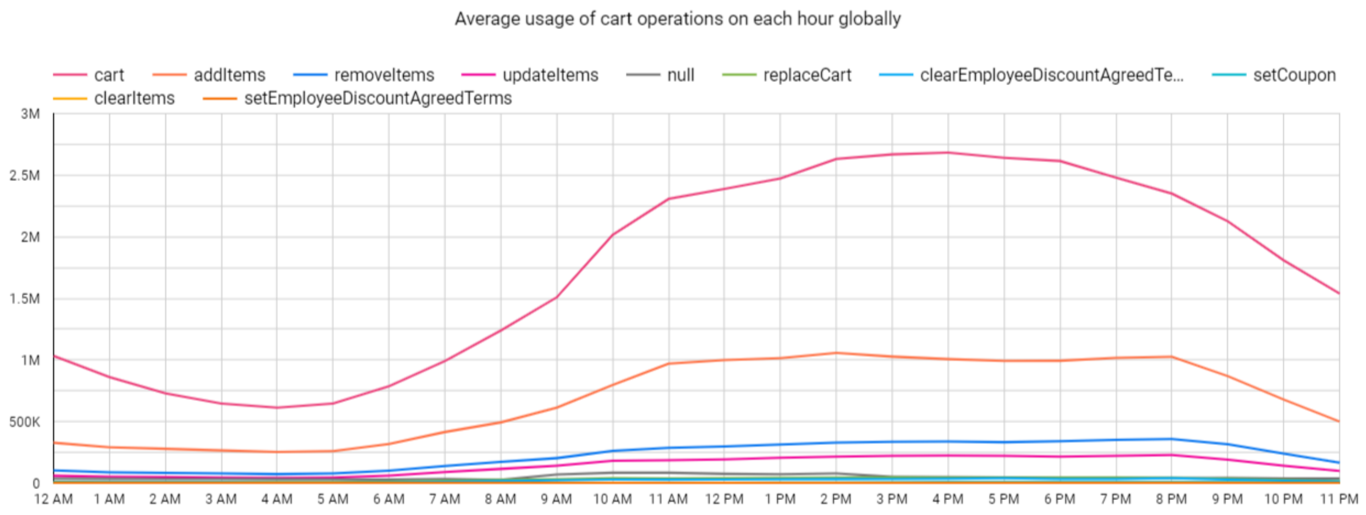


Figure 9.9: Average usage of cart operations on each hour globally.

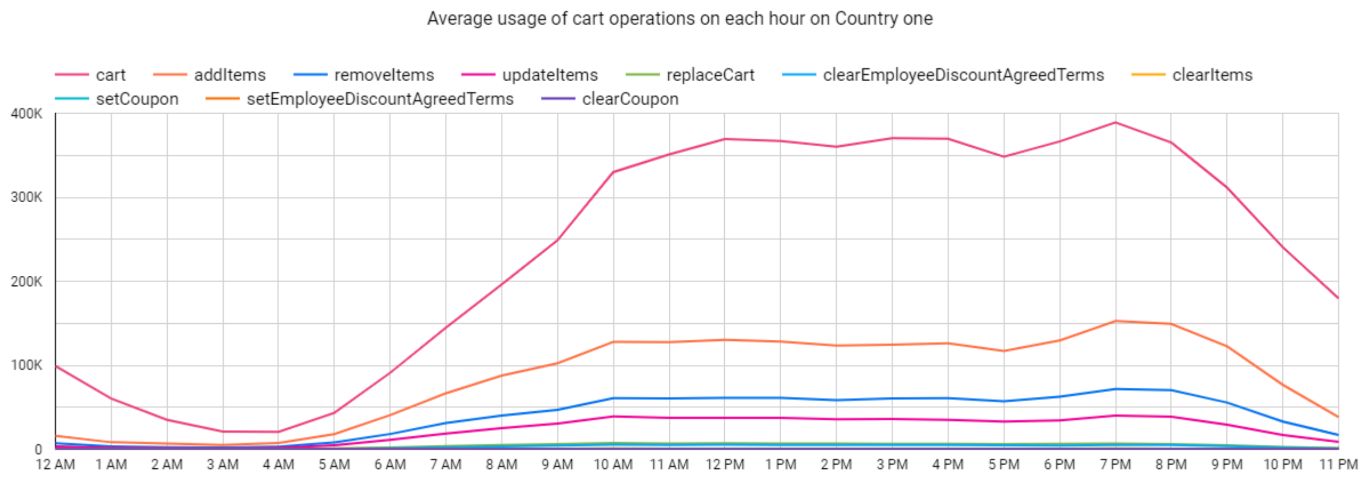


Figure 9.10: Average usage of cart operations on each hour in Country one.

Global and Country one log entries divided over gcpEntrypoint

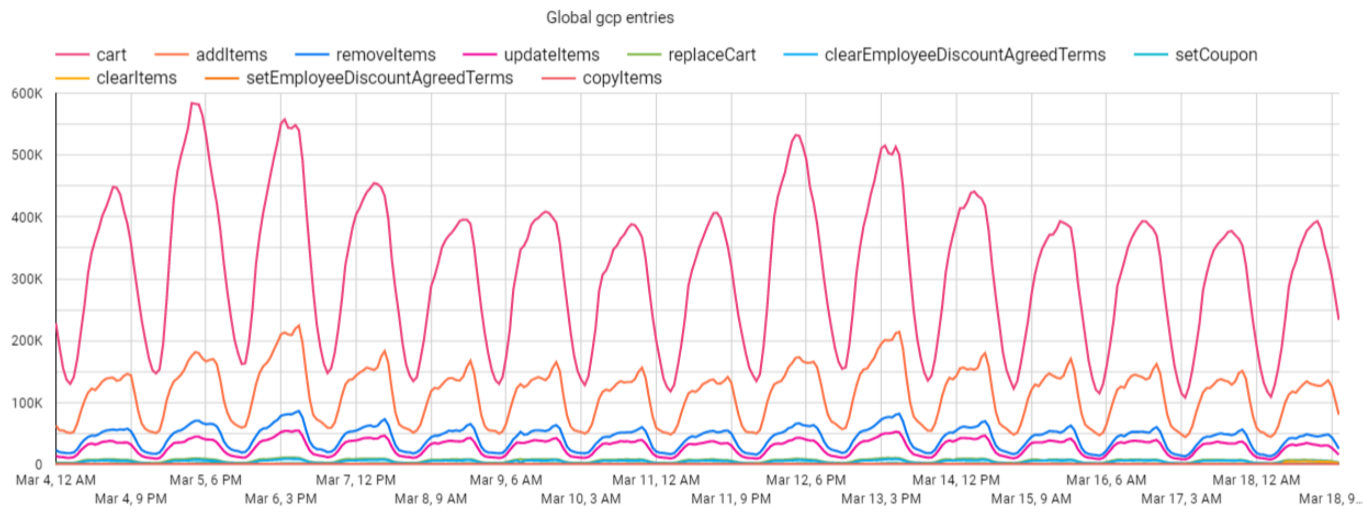


Figure 9.11

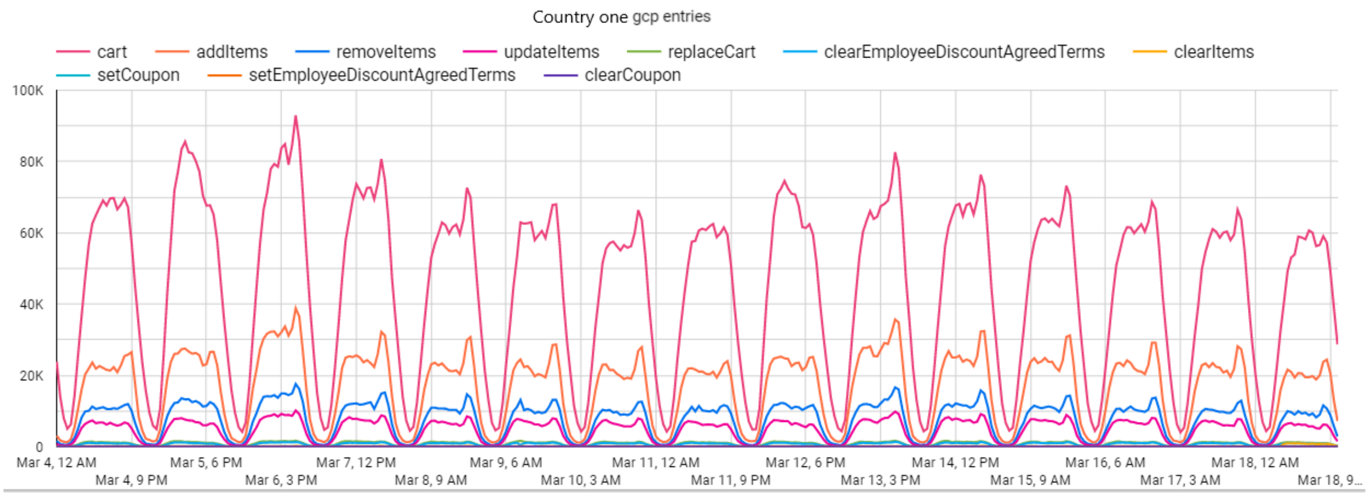


Figure 9.12

Number of addItems events, addItems split by market and logged in users aggregated by day.

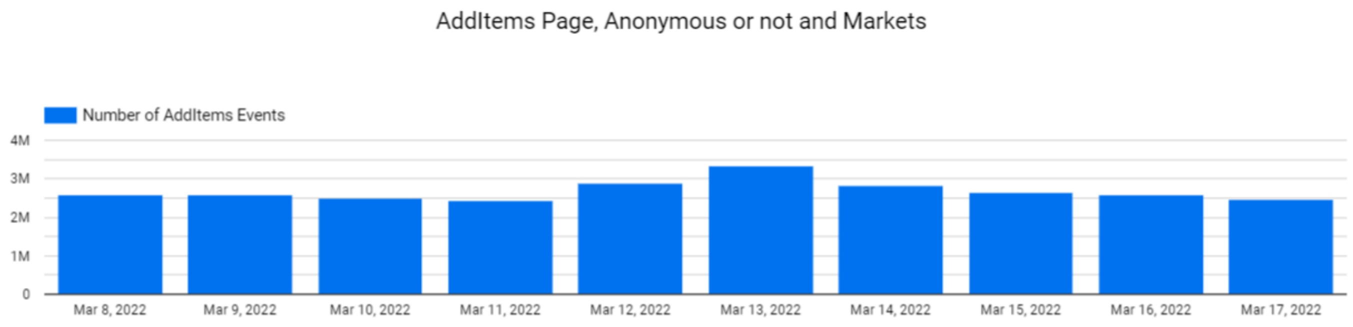


Figure 9.13

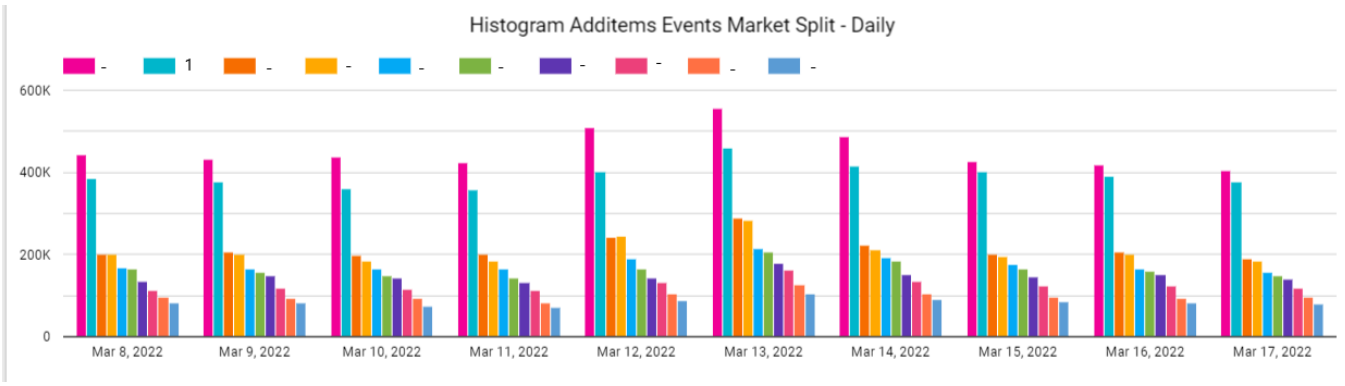


Figure 9.14

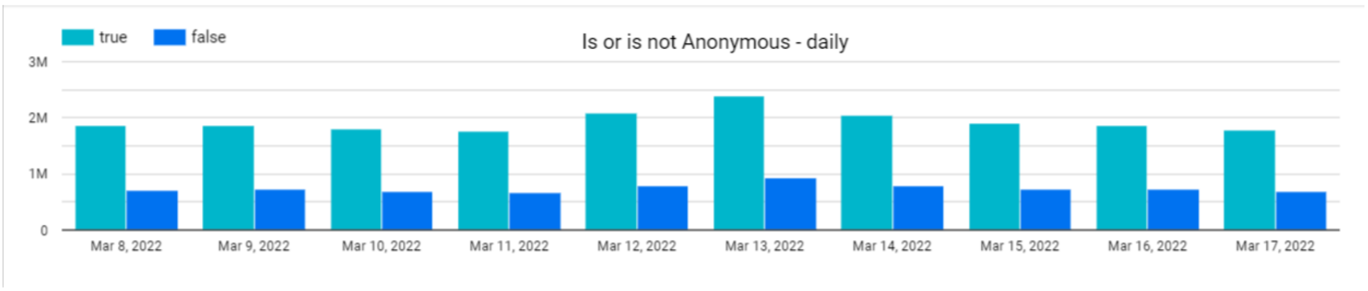


Figure 9.15

Number of addItems events, addItems split by market and logged in users aggregated by day excluding Country two.

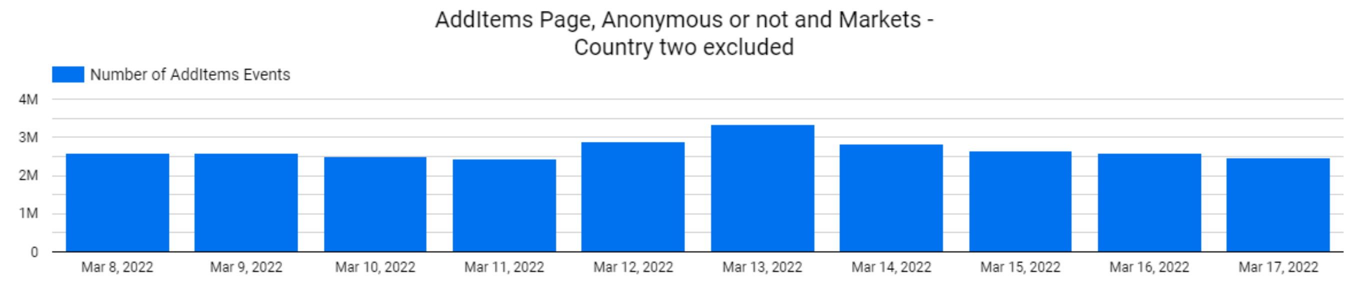


Figure 9.16

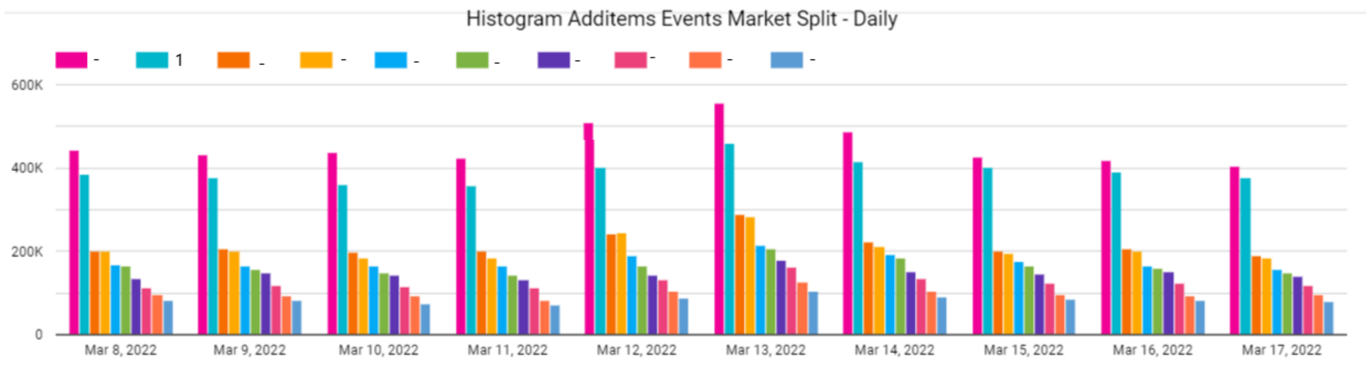


Figure 9.17

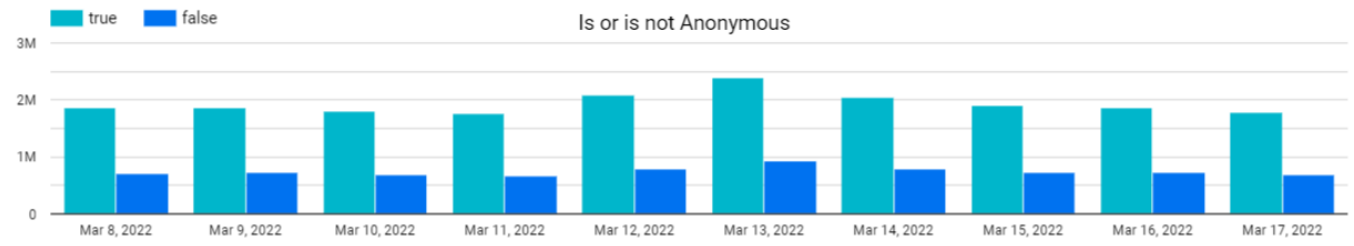


Figure 9.18

Record count of logs compared to userAgent and userAgent distribution globally except Country two.

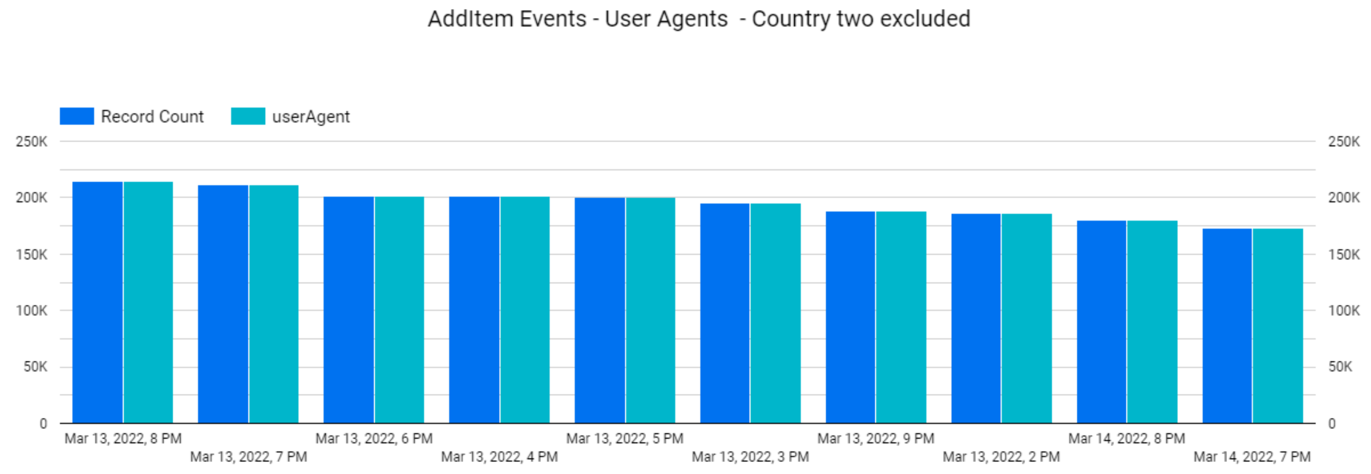


Figure 9.19

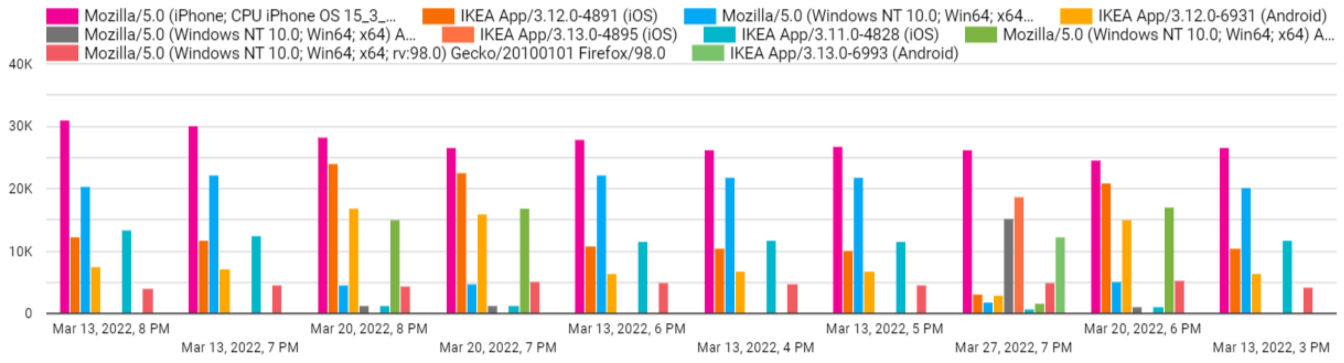


Figure 9.20: userAgent distribution globally except Country two.

Looking at the record count, market split, and logged in users on gcpEntrypoint - cart.

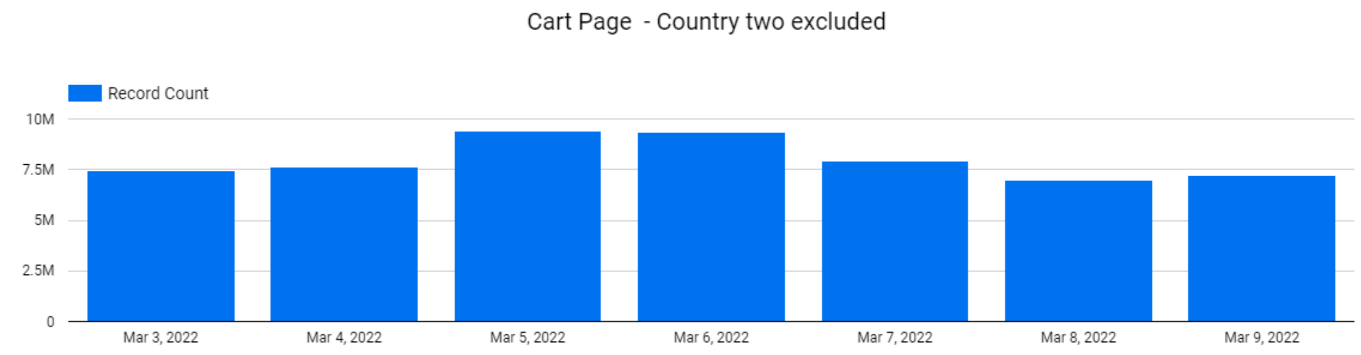


Figure 9.21

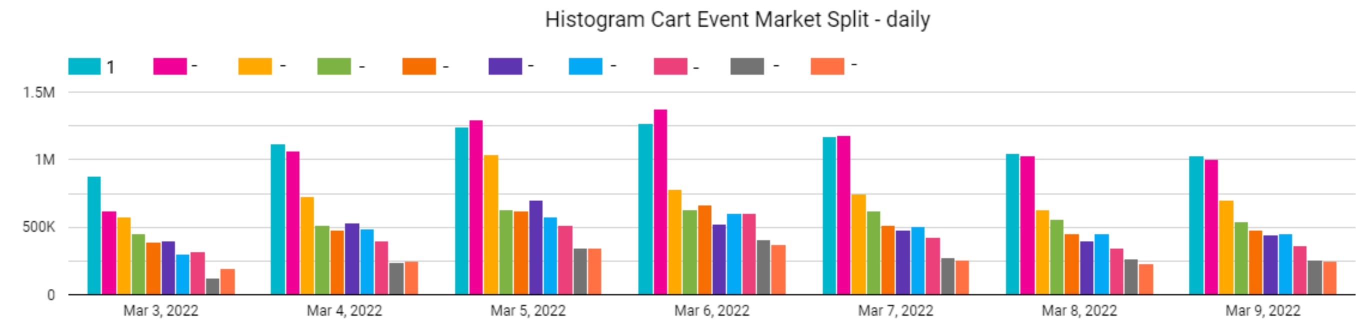


Figure 9.22

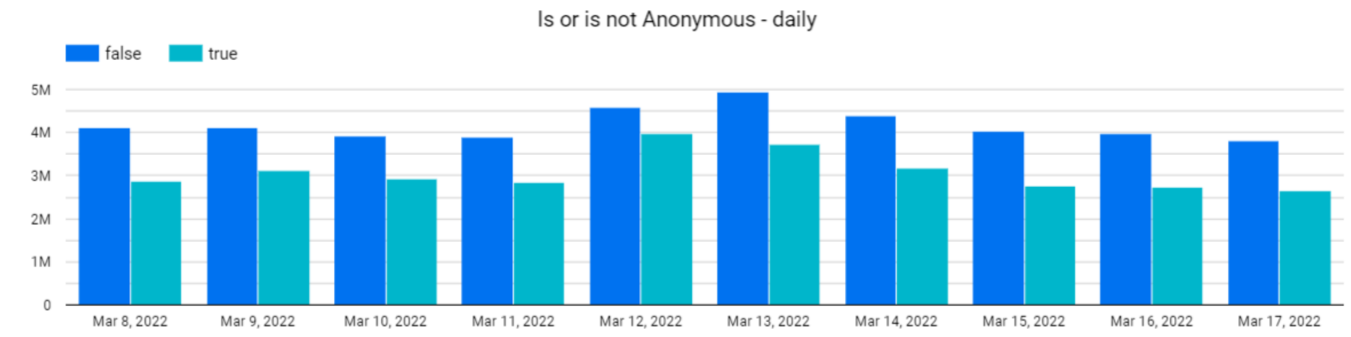


Figure 9.23

gqlEntrypoint - cart user agent distribution.

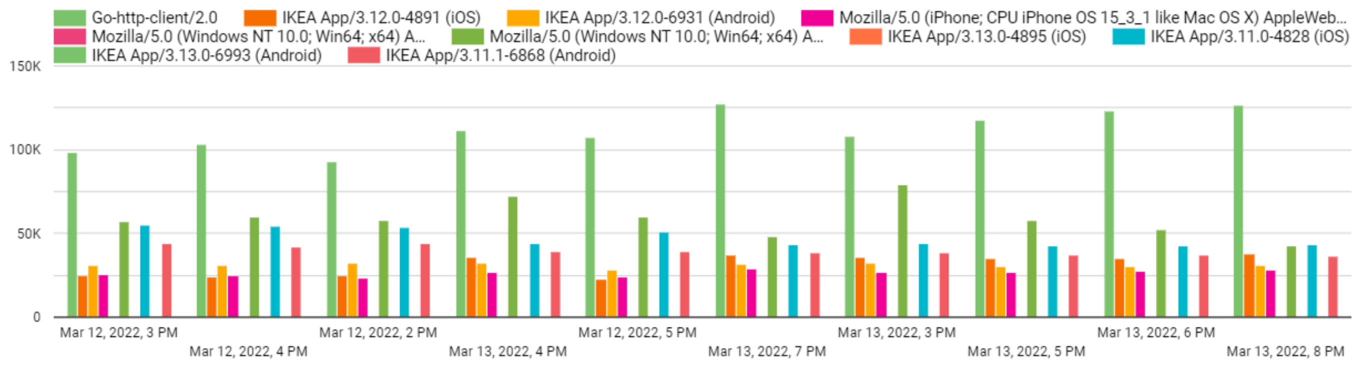


Figure 9.24

clientId distribution over time excluding Country two.

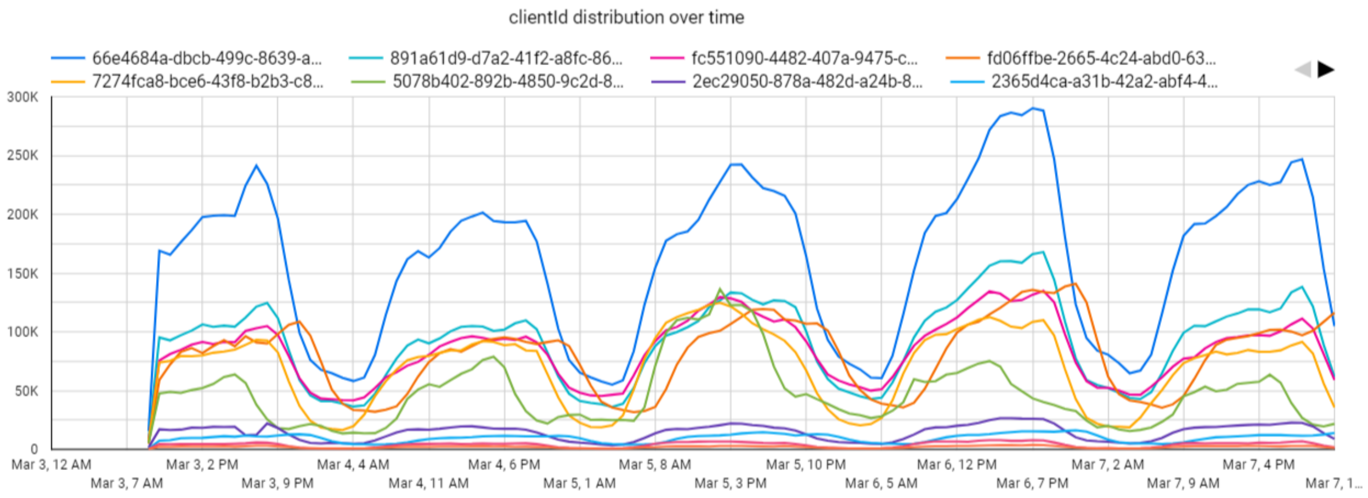


Figure 9.25

gqlEntrypoint - cart ClientID analysis in Country one

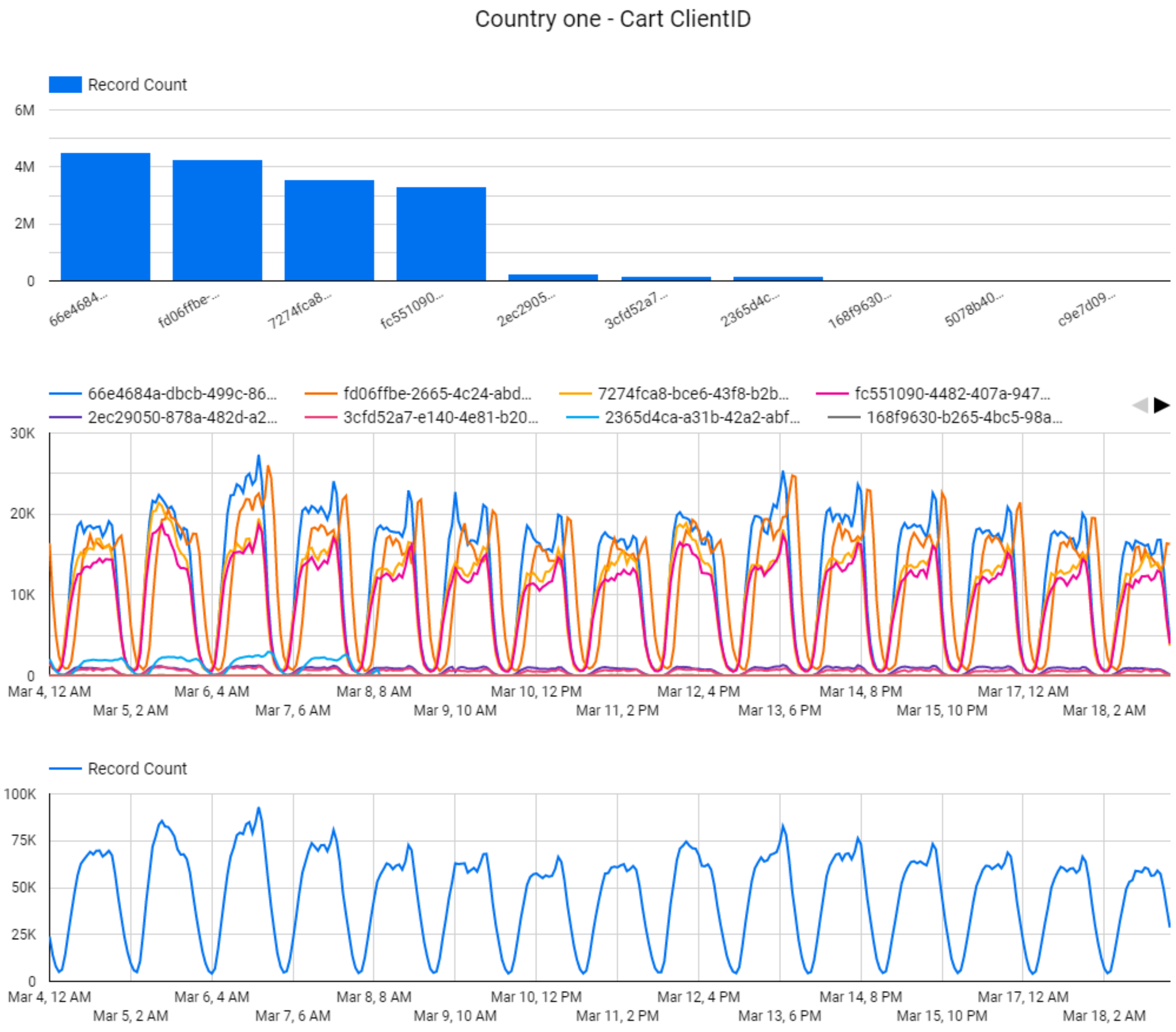
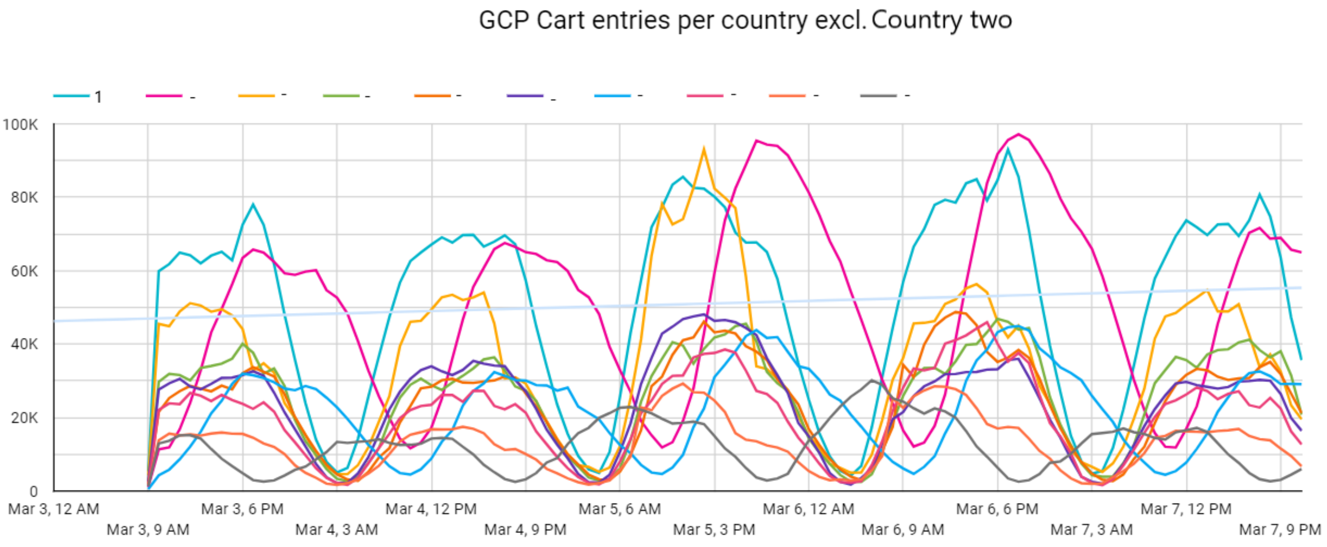


Figure 9.26

gqlEntrypoint - cart entries per country with Country two excluded.



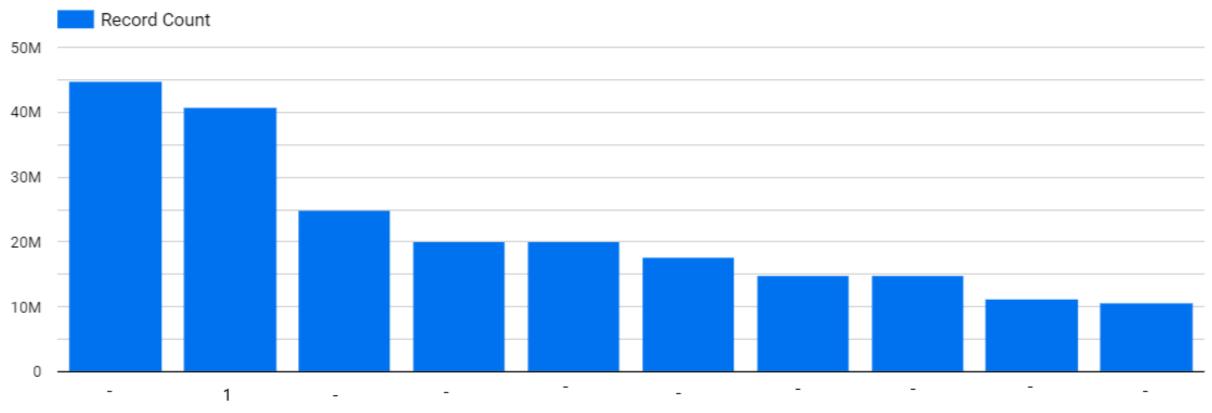


Figure 9.27