# Analysis of Avg. Time Response to Customer API Request

Aditya Dande*

February 13th, 2023

Hochschule Fresenius University of Applied Sciences

Faculty of Economics & Media

International Business School


Industrial Engineering & International Management

Cologne Campus

Matriculation Number: 400325270

Email: dande.aditya@studs.hs-fresenius.de

Word count: 5043

*https://github.com/adityadande/DataSciene

**abstract**

This Paper focuses on an introduction of API and how it is used, details of different types of APIs, why it became necessity for companies to adapt to digitization how DHL uses its API, someone with no knowledge of tracking their shipment to understand the process, what is active tracing, API Request made by the customers and/or employees of DHL. It discusses the files that are coming in Excel sheet every month, storing them, gathering all data for a year in one Excel and representing it with the help of Graph. Discussing in depth about coding that was needed to get these outputs Also here the paper discusses about users analysis about their usage results to get more data and find out users the with most requests to take appropriate actions and making sure that Servers are not taking lot of load. This paper also discusses the challenges that I faced which some I was able to find answers and some are still causing an issue. All these challenges are divided in sections to get more better understanding.

# Contents

# 1 Introduction

Digital supply chains predict and suggest actions, whereas traditional supply chains plan and respond. While fully digital supply chains operate in real time, are dynamic, and can adapt to changing conditions, traditional supply chains are largely static and operate on rules based on historical transactional inputs, lag times notwithstanding. Digital supply chains are networks as opposed to the linear nature of traditional supply chains. Digital supply chains use information from both OT (operational technology) and IT (information technology) systems, as opposed to traditional supply chains, which frequently rely on standalone systems. Digital supply chains balance profit and service levels rather than optimizing by node, shipment, or order. In a conventional supply chain, it can be difficult to identify potential issues and foresee their consequences. The majority of businesses must routinely assess the supply chains of their most important suppliers based on SKUs, for instance, and manually compile a list of the actions that would need to be taken to maintain production in the event that any of those links failed. In contrast, a digital supply chain can allow businesses to anticipate problems and act pro-actively without the need for time-consuming preplanning. Logistics management, sourcing and procurement, and conversion are all included in the planning and management of the supply chain. Involved parties such as suppliers, middlemen, outside service providers, and clients are coordinated with and worked with in this process. Above all, human decision-making is based on machine inputs in traditional supply chains, whereas in digital supply chains, machines are in charge of decision-making with human oversight. In logistical companies, the transportation plays an important role for the productivity and profitability. Hence for companies it is important to know the location of deliveries like where it is for every hour, Estimated Time Required to reach the

delivery. DHL Freight is among leading players in Europe. So its important for them to provide updates about deliveries whenever customer demands for it. But thousands of customers sends thousands of request per day then it becomes really difficult for company's system to send response on time. It is important for companies to know which requests are most frequently been asked in order to find alternative solutions for that specific request. Customers must go to https://activetracing.dhl.com/DatPublic/datSelection.do and enter their tracking number in order to track their package. or alternatively for customers can register via https://activetracing.dhl.com/DatPublic/login.do. For domestic or international tracking, your shipment will be given a tracking number or ID, which is a mix of numbers and potentially letters. The tracking number or ID can typically be provided by the shipper or online store. The tracking number or ID is frequently included in the confirmation email or shipment tracking message you receive after placing an order at an online store. After acquiring the Track and Trace ID, tracking events often surface 24 to 48 hours later. In general, a tracking event will show up once the package has arrived at the DHL site. DHL recommends clients to get in touch with their carrier if they do not have a tracking number.The shipment tracking systems of the particular business unit in charge of the shipment, however, may operate if you have additional shipping reference numbers (for example: DHL Express or DHL Freight).

This project uses R to effectively identify the data that leads to these problems and provides an overview. When large companies like Audi, BMW, and others order cargo from DHL Freight to be transported from one state to another or from one country in Europe to another, it takes longer via road. They therefore wish to continue tracking the location of their supplies. They ask DHL Freight via API where the shipment is at the moment. DHL replies with information about the location of the truck's actual terminal. Similar to this, numerous clients are

capable of sending thousands of requests in a single day. Which questions do customers frequently ask, and how long does it typically take for a client to receive an answer, according to the product manager of shipment tracing? The DHL IT team sends an Excel sheet with information about these requests once a month. Therefore, my project intends to streamline the process of these calculations, analyze the data to provide an overview of DHL's Web API usage, and do away with the need for manual work in each month's Excel sheet.

# 2 Theory

## 2.1 Introduction of 'MyAct' System

'MyAct' is a product of DHL which is managed by DHL Freight in bonn. It is an Active tracing system which let all customers to find out where their shipment currently located at. Anyone can visit to website and register. This system is designed for B2B shipments so its not available for domestic parcel services. Once user register according to which country that user belongs to, the country admin will get an email to grant access to this user and assign this user to its account in the system. After the user is assigned, the user is able to login and check the data on website. The other which most users prefers is after getting credentials they use this data to send a API request and get all the shipment data on their server. So They will have all this data available on there server as 'MyAct' system deletes history data after 6 months.

## 2.2 Overview of API & Data

API stands for 'Application Programming Interface' which Servers uses to communicate with each other, in simple terms it manages warehouse data whenever

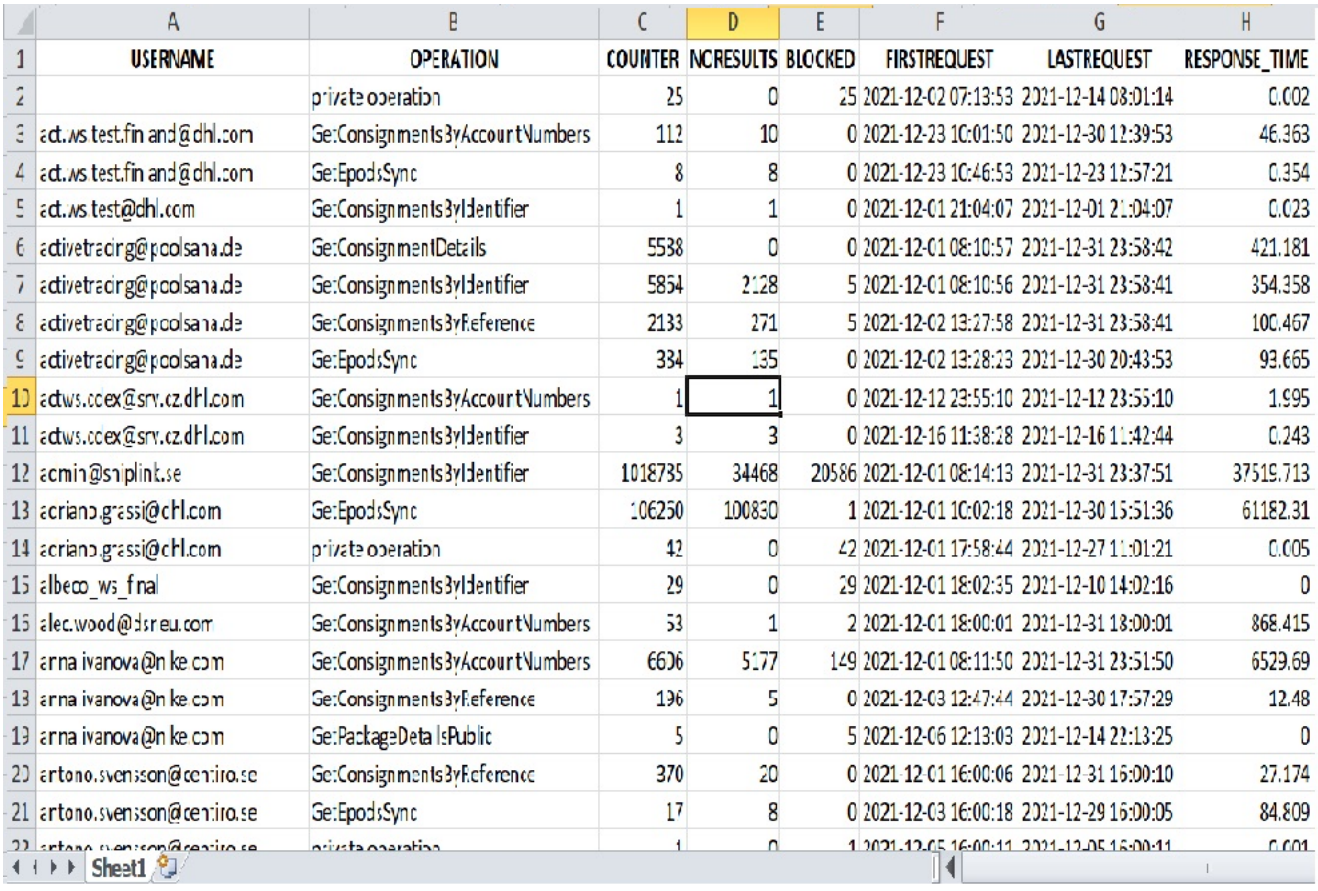
any sales happens. APIs works with receiving requests and sending responses principle. the scope of API in this paper is just to understand what data actually means in excel and not to actually send request or response. Using a set of definitions and protocols, APIs are techniques that let two software components communicate with one another. whenever anyone place an order online via seller websites e.g. Amazon then warehouse needs to subtract those shoes that are booked and tell amazon how many shoes are left. these two way communication is done with the help of API. There are two ways request and respond can be send REST APIs and SOAP APIs. SOAP stands for Simple Object Access Protocol. It uses XML format. Extensible Markup Language (XML) is a markup language and file format for storing, transmitting, and reconstructing data and REST stands for Representational State Transfer. It uses HTTP (The Hypertext Transfer Protocol) format. there are other types of APIs but most commonly and also in this case these two are used.

For example, the software system of the warehouse company contains daily number of shoes boxes they have in stock. Although many Transport Management Systems (TMS) are still independently developed internally, there are standard industry integration s that facilitate data and information sharing. Because internal and proprietary systems aren't typically designed to exchange data, this poses a problem. Especially globally, where big data can be overwhelming for businesses with manual processes, this poses a logistical challenge. APIs serve as a bridge between systems, enabling communication between companies and their customers. Applications that are unable to communicate with one another use APIs as a bridge. MuleSoft claims that visualizing APIs as a bilingual waitstaff in a restaurant is a simple way to understand how they work. A cashier, chefs, and customers are present. Since they are all autonomous from one another, waitstaff is necessary to ensure efficient communication between them all. APIs produce

automated workflows that largely do away with the requirement for manual entry. APIs can be customized to produce unique user experiences that meet customer expectations. The user experience is improved by APIs' easier ability to embed content on any website. As organizations and processes change, APIs can easily be updated. Alternate for APIs is EDI (Electronic Data Interchange), There are differences between their applications, even though EDIs and APIs both have the same goal of exchanging data, Only data in formats like ANSI, EDIFACT, TRADACOMS, VDA, XML, or UBL can be transmitted via EDI. Unlike EDIs, which must change their data format in order to transmit information, APIs can transfer data between systems without doing so. Due to their high implementation costs and implementation times, EDIs may not be available to small to medium-sized businesses, whereas APIs are typically quick and inexpensive. Because APIs are simpler to implement, maintain, and update, EDIs are gradually becoming obsolete and will eventually be replaced by them.

Inefficiencies caused by manual labor are a significant expense in the freight industry. Let's say you have a team member who manually books and tracks orders for an hour each day. When a customer places an order, you can automate these steps and receive real-time shipment updates sent directly to your servers by using the appropriate freight shipping API. More than five hours per week and more than 250 hours per year are saved as a result. Additionally, you'll be able to offer customers a smoother and more dependable experience, increasing retention. Freight shipment management won't require a full team. You only need one person to handle it. You can immediately increase the volume you ship without scaling your team or increasing your support expenses. Everyone else is free to concentrate on mission-critical, high-value tasks for your company. The automatic delivery of data makes it easier to quote, book, track, and validate. At each step, minimize or do away with human error. There will be no more misclassifying freight or typing

the incorrect address. no more choke points. To protect your margins, every step becomes more seamless. Be aware of where your data is. Reduce the amount of time spent searching emails, looking through various dashboards, and moving around the office. Consider a freight shipping API as the nervous system for your supply chain. Your account representatives, logistics teams, and warehouse staff all have access to the same information in one location at all times. The management can monitor and manage costs. Effective experience management is possible for everyone.



| | A | B | C | D | E | F | G | H |
|---|---|---|---|---|---|---|---|---|
| 1 | USERNAME | OPERATION | COUNTER | NORESULTS | BLOCKED | FIRSTREQUEST | LASTREQUEST | RESPONSE_TIME |
| 2 | | private operation | 25 | 0 | 25 | 2021-12-02 07:13:53 | 2021-12-14 08:01:14 | 0.002 |
| 3 | act.ws test.finland@dhl.com | Ge:Consignments3yAccountNumbers | 112 | 10 | 0 | 2021-12-23 10:01:50 | 2021-12-30 12:39:53 | 46.363 |
| 4 | act.ws test.finland@dhl.com | Ge:EpodsSync | 8 | 8 | 0 | 2021-12-23 10:46:53 | 2021-12-23 12:57:21 | 0.354 |
| 5 | act.ws test@dhl.com | Ge:Consignments3yIdentifier | 1 | 1 | 0 | 2021-12-01 21:04:07 | 2021-12-01 21:04:07 | 0.023 |
| 6 | activetracing@pcolsana.de | Ge:ConsignmentDetails | 5538 | 0 | 0 | 2021-12-01 08:10:57 | 2021-12-31 23:58:42 | 421.181 |
| 7 | activetracing@pcolsana.de | Ge:Consignments3yIdentifier | 5854 | 2128 | 5 | 2021-12-01 08:10:56 | 2021-12-31 23:58:41 | 354.358 |
| 8 | activetracing@pcolsana.de | Ge:Consignments3yReference | 2133 | 271 | 5 | 2021-12-02 13:27:58 | 2021-12-31 23:58:41 | 100.467 |
| 9 | activetracing@pcolsana.de | Ge:EpodsSync | 334 | 135 | 0 | 2021-12-02 13:28:23 | 2021-12-30 20:43:53 | 93.665 |
| 10 | adws.ccex@srv.cz.dhl.com | Ge:Consignments3yAccountNumbers | 1 | 1 | 0 | 2021-12-12 23:55:10 | 2021-12-12 23:55:10 | 1.995 |
| 11 | adws.ccex@srv.cz.dhl.com | Ge:Consignments3yIdentifier | 3 | 3 | 0 | 2021-12-16 11:38:28 | 2021-12-16 11:42:44 | 0.243 |
| 12 | admin@sniplink.se | Ge:Consignments3yIdentifier | 1018735 | 34468 | 20586 | 2021-12-01 08:14:13 | 2021-12-31 23:37:51 | 37519.713 |
| 13 | adriano.grassi@dhl.com | Ge:EpodsSync | 106250 | 100830 | 1 | 2021-12-01 10:02:18 | 2021-12-30 15:51:36 | 61182.31 |
| 14 | adriano.grassi@dhl.com | private operation | 42 | 0 | 42 | 2021-12-01 17:58:44 | 2021-12-27 11:01:21 | 0.005 |
| 15 | albeco_ws_final | Ge:Consignments3yIdentifier | 29 | 0 | 29 | 2021-12-01 18:02:35 | 2021-12-10 14:02:16 | 0 |
| 16 | alec.wood@dsreu.com | Ge:Consignments3yAccountNumbers | 53 | 1 | 2 | 2021-12-01 18:00:01 | 2021-12-31 18:00:01 | 868.415 |
| 17 | anna.ivanova@nike.com | Ge:Consignments3yAccountNumbers | 6606 | 5177 | 149 | 2021-12-01 08:11:50 | 2021-12-31 23:51:50 | 6529.69 |
| 18 | anna.ivanova@nike.com | Ge:Consignments3yReference | 196 | 5 | 0 | 2021-12-03 12:47:44 | 2021-12-30 17:57:29 | 12.48 |
| 19 | anna.ivanova@nike.com | Ge:PackageDetailsPublic | 5 | 0 | 5 | 2021-12-06 12:13:03 | 2021-12-14 22:13:25 | 0 |
| 20 | antonio.svensson@cemiro.se | Ge:Consignments3yReference | 370 | 20 | 0 | 2021-12-01 16:00:06 | 2021-12-31 16:00:10 | 27.174 |
| 21 | antonio.svensson@cemiro.se | Ge:EpodsSync | 17 | 8 | 0 | 2021-12-03 16:00:18 | 2021-12-29 16:00:05 | 84.809 |
| 22 | antonio.svensson@cemiro.se | private operation | 1 | 0 | 1 | 2021-12-05 16:00:11 | 2021-12-05 16:00:11 | 0.001 |

Sheet1

The data consist of 8 columns “USERNAME, OPERATION, COUNTER, NORESULTS, BLOCKED, FIRSTREQUEST, LASTREQUEST, RESPONSE_TIME”. Username consist of all users Email address which are registered in DHL sys-

tem from all over the world, Operation explains which type of API request users are sending, there are 10 types of requests that are: GetConsignmentByReferencePublic, GetConsignmentDetails, GetConsignmentDetailsPublic, GetConsignmentsByAccountNumbers, GetConsignmentsByIdentifier, GetConsignmentsByIdentifierPublic, GetConsignmentsByReference, GetEpodsSync, GetLatestEventByIdentifiers, GetPackageDetailsPublic, private operation. Main rypes are 5 which are:

1. **GetConsignmentsByIdentifier:** can be used if the DHL Order-Code, Shipment- ID, or Package-IDis known.
2. **GetConsignmentsByReference:** can be used if an exact Customer Reference String of a shipment is known.
3. **GetConsignmentsByAccountNumbers:** returns all shipments/events, based on Customer DHL Account Number(s), can be used with date/time-parameters.
4. **GetLatestEventByIdentifiers:** returns Latest Status & Location for multiple Shipment IDs (not Customer References), no Shipment Details / Event History.
5. **GetEpodsSync:** returns ePods (electronic Proof of Delivery) (& some other documents possible) of shipments immediately (synchronous), suitable for up to 10 Shipment-IDs, per Request.

private operation are restricted which we are not authorized to know but other operations are self explanatory. Operation which are ending with 'Public' are used by public with no password or another security measure hence they are able to see limited data compare to other operations. Counter shows number of times that user requested every month. one customer can request different types of operation hence we can see one user multiple times in excel. No Result shows

that when user sends a request and they did not got any response in return. In this country admin of that user needs to be notified. Blocked shows user did not sent a proper request, it could be the case of syntax error, null values, bad value etc. first request and last request shows the data and time of those request which are out of the scope of this project and lastly the Response time which shows response time required by server in seconds in a month per user.

## 2.3 Coding in R

Before importing the files in R first I needed to change name of excel to remove all the special characters and spaces then store all those excel files in one folder. The full code which is needed to replicate the result is available in appendix at last. first I installed and loaded all the required libraries, set a working directory then I got the all files in R which ends with xls. as it will get files but I cant still access the data. hence I used the function to get the data and only required columns of excel files, at last combined all the data with the help of do call function as mentioned below:

```
# Import those files in R
webUsageExcelsLists <- list.files(pattern = ".xls")


# Get the data of these Excels Files
webUsageDataLists = lapply(webUsageExcelsLists, function(i){
  x = read_excel(i, sheet=1)
  x <- x[,c(2,4,5,6,7,10)]
  x$file = i
  x
})
```

```
# Binding the data from all the Excels
allUsageData = do.call("rbind.data.frame",webUsageDataLists)
```

Now I calculated the average response time with simple division of response time per counter rounded result up to 4 decimal points and created the excel file in local folder.

```
#AVG. RESPONSE TIME for all available files
allUsageData$AVG_RESPONSE_TIME <- round(
                        allUsageData$RESPONSE_TIME  /
                        allUsageData$COUNTER, digits = 4)


# Create an Excel
write_xlsx(allUsageData,"D:/FRESENIUS/SEMESTER 2/Data Science/
            Data/more data/allAPIResponse.xlsx")
```

Here I wanted to find out the all counters and required time for them according to the operations. Hence I aggregated counter by adding them with respect to operations, similar with response time. Now I got two objects with operations in common, so I created the dataset with the help of data frame of all these three values. The problem is now I have to again divide response time to counter as previously calculated avg response time aggregated with sum for operations will not give accurate data. After that I got perfect excel with dataset to calculate and interpret the data

```
#COUNTER vs OPERATION
usageCouterOperationBased <- aggregate(COUNTER ~ OPERATION,
                                       allUsageData, sum)
```

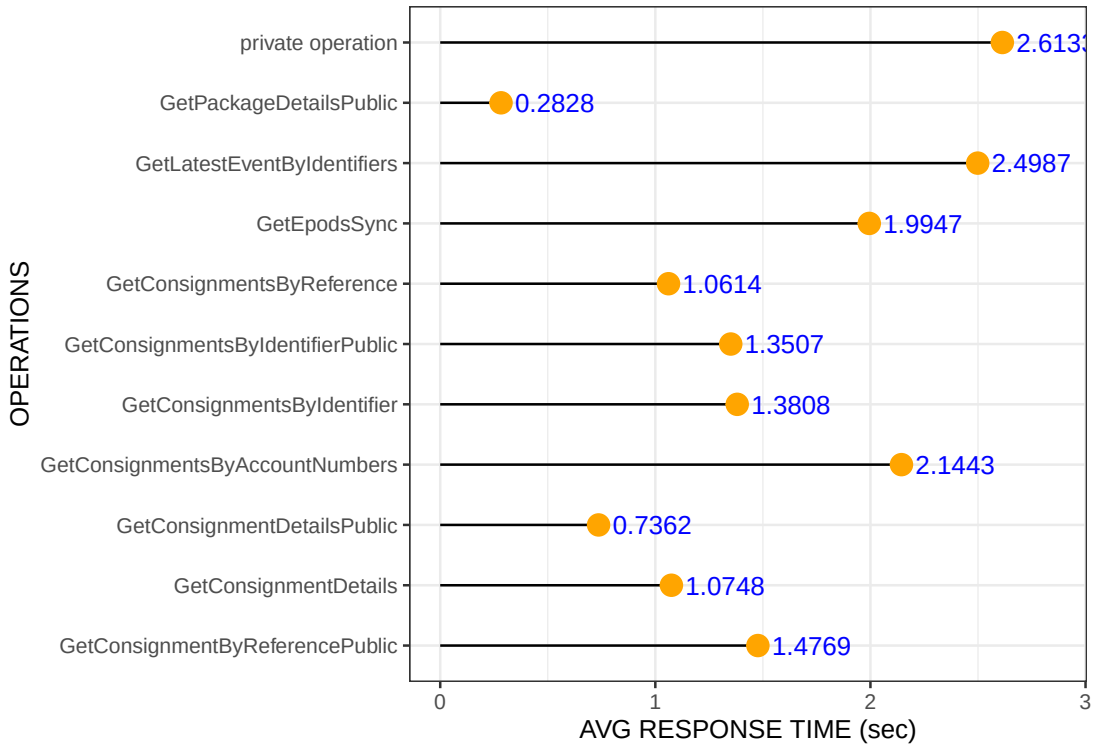
```
#RESPONSE TIME vs OPERATION
usageResponseTimeOperationBased <- aggregate(RESPONSE_TIME ~
                                  OPERATION, allUsageData, sum)

# Collect all the data in one file
data2 <- data.frame(OPERATION<-
                      usageResponseTimeOperationBased$OPERATION,
            COUNTER=usageCouterOperationBased$COUNTER,
            RESPONSE_TIME=
              usageResponseTimeOperationBased$RESPONSE_TIME
)
#AVG. RESPONSE TIME vs OPERATION
data2$AVG_RESPONSE_TIME <- round(data2$RESPONSE_TIME  /
                                   data2$COUNTER,
                                 digits = 4)
data2

# Create an Excel
write_xlsx(data2,"D:/FRESENIUS/SEMESTER 2/Data Science/Data/
           more data/yearBaseAvgAPIResponse.xlsx")
```

I have now the final data about how much time it required on a average for each operations with counter. This data is useful and hence needed to be plotted on the graph, so first I created the excel sheet of this data and stored it on my github profile (here I uploaded the file so to make it easily accessible from anywhere) the using getURL function I got the raw version of excel and converted in to an object. I choose the lollipop graph as it makes more sense to show time and end

at point. I showed avg. response time digits up to 5 at the end of each point and offset with value 0.25 so the values wont mix up with graph end points.




Graph 1: Avg. Response Time For Year 2022

The graph above gives clear view of most requested operations. But business requirement is also to know which are the users who are requesting these requests. The user details like if that user is DHL employee or DHL customer, Third party vendor or administrator who is looking over DHL active tracing system. First I needed to find who are the users sent the most number of requests so I got the excel sheet from github grouped the data by username summarized by counter additions then ordered by this sum of counter in descending and used the function head to show the top 6 results. Similarly for the no responses,no results and no blocked requests of the users. In username we can able to see country or name of the company that user works for if its not DHL employee. I wanted to represent this user analysis in one place hence I used kable to align side by side, added comma

to the numbers, kept values digits till 10, I wanted to get all four observation in one kable list but the output was showing all four side by side and not two by two square, hence I was used two lists of kable.

```
# Users Analysis


library(RCurl)
x <- getURL("https://raw.githubusercontent.com/adityadande/
            DataSciene/main/allAPIResponse.csv")
apiData <- read.csv(text = x)


# Username vs Counter
agg_tbl <- apiData %>% dplyr::group_by(USERNAME) %>%
  dplyr::summarise(sum(COUNTER))
agg_tbl <- agg_tbl[order(-agg_tbl$`sum(COUNTER)`), ]
iris2 <- head(agg_tbl)


# Username vs No response
agg_tb2 <- apiData %>% dplyr::group_by(USERNAME) %>%
  dplyr::summarise(sum(NORESULTS))
agg_tb2 <- agg_tb2[order(-agg_tb2$`sum(NORESULTS)`), ]
iris3 <- head(agg_tb2)


knitr::kable(
  list(
    iris2, iris3
  ), align = "lccrr", caption = "Users with the most API request
  & result",
```

```
  digits = 10, format.args = list(big.mark = ",",

# NORESULT / COUNTER
apiData$NORESULT_AVG <- apiData$NORESULTS  / apiData$COUNTER
agg_tb3 <- apiData[,c(1,2,3,4,8)]
agg_tb3 <- agg_tb3 %>% dplyr::group_by(USERNAME) %>%
  dplyr::summarise(sum(NORESULT_AVG))

iris4 <- head(agg_tb3[order(-agg_tb3$`sum(NORESULT_AVG)`),])

# BLOCKED / COUNTER
apiData$BLOCKED_AVG <- apiData$BLOCKED  / apiData$COUNTER
agg_tb4 <- apiData[,c(1,2,3,5,9)]
agg_tb4 <- agg_tb4 %>% dplyr::group_by(USERNAME) %>%
  dplyr::summarise(sum(BLOCKED_AVG))

iris5 <- head(agg_tb4[order(-agg_tb4$`sum(BLOCKED_AVG)`),])

knitr::kable(
  list(
    iris5, iris4
  ), align = "lccrr", caption = "Avg. Most NO RESULT & BLOCKED
  per COUNTER For each Customer Analysis",
  digits = 10, format.args = list(big.mark = ",",))
```

# 3 Challenges & Their Solutions

## 3.1 R Syntax

Since anyone can add new features to R, the resulting code for various R packages is frequently a bit of a mess. For instance, the two blocks of code below accomplish essentially the same task while utilizing vastly different syntaxs. Since anyone can add to it however they please, this kind of consistency is common in R, but there is no way to avoid it.

## 3.2 Managing the data

The first challenge I faced is how to extract all the data in one file without disturbing the format, finding the **"do.call()"** function was time consuming. After that organizing data on basis of Operation with group by was becoming difficult as it was not showing the result what I was expecting hence I had find alternative way i.e. aggregate function. but the limitation of this function that other columns were got ignored. Hence I had to use that function twice with counter and response time then again find avg. time create data with all these four columns. During creating report removing from indent was difficult and still proving hard to rid off. In report conclusion managing table side by side is hard and still couldn't able find solution. Tables inn conclusion were mis-aligned two table were at centered align and two were right aligned, then going back to lecture notes (from github R markdown template.Rmd) I found out the reason for it and corrected it.

## 3.3 Graph

For the graph I wanted to create bar graph of counter and operations and then blue line going through it which would represent avg. response time but I couldn't able to accomplished it. then I just went bar graph but the names of operations were large and hence they were intersecting with each other. Initially I thought to tilt all operations names by 45 degrees. This was good solution but not beautiful. Then I faced problem with names of X axis and Y-axis but this was less time consuming to find solution hence I went R graphs website to find good alternative for bar graph. I found the lollipop graph which was perfect and beautiful but here I faced major issues, I wanted to represent data in descending order and I tried all possible solution out there on internet, if looked at code closely we can see that I put desc() function but it doesn't work and replicate on graph.Also I had little problem with finding how to put actual values of graph at the end of each plot and then managing distance of them to end point of each plot.

## 3.4 Presentation Slides

For creating presentation using Markdown, I had to face problem of warnings with plots showing images & managing image sizes presentation. At beginning I was creating presentation in PPT output and hence I was creating new PPT file and showing the output then I switch to slidy presentation so I had to struggle to manage size and positions of images. Also managing and showing these images in was issue, finding columns to place images in 2X2 was time consuming. Another problem was warning were showing whenever I was using library(dplyr) and if i didnt use it then using arrange() was not possible, it was showing error couldn't find this function,alternatively I used dplyr::arrange() and it solved the issue. Same issue I had to face with using kable functionality, with kable I wanted to

give separate captions to all tables,I tried way showed on internet but with no success, so I gave general title with one title for two tables. while using orderby() function with was getting a lot of difficult to get result in descending order, hence I used minus sign to make it work.

### 3.5 Github

Installing git and configurating it with R studio was challenging and learning push and pull operations as well. Getting the excels access from R was bigger issue as whenever I was trying to view raw file it was asking me to download that file. After searching for a while I found out that I have to upload CSV type file and not xlsx type file. Then I converted my Excel files of dataset into CSV Format and uploaded on my github profile. Also finding getURL() took some time. After I uploaded all the excel files and images on github, I changed my codes to make it accessible from internet without downloading or changing any code that was little difficult. Another issues I was having with PULL request from github. I read through available material but it was difficult for me to understand. Hence I went to prof. Huber for help, Prof. helped me understand whole process of how to work with github with this process, how to use command prompt, how to retrieve data in local PC, how to push data in github online profile, how to fork repositories etc.

## 4 Conclusion

The final result came from this project helped analyse the users usage of API requests distributed mostly equal to all operations and most requested operation of all is public operation means API requests to DHL server are preferred without

any credentials and limited information. But users from country liken Portugal and Sweden are among the most API requests. They are sending relatively large numbers of API Requests, where a large % of these API Requests are not returning any results, and therefore where the Customer API solution is heavily contributing to the increasingly high-demand on the Active Tracing API, but where the Customer may not be benefiting from this.

Although I consider that many of DHL API Customer's usage appears to be rather "poor usage", below short summary of what I consider to be the 2 worst cases AcT API implementations where I have identified what I describe above:

1. **Country PL – 1 Customer login**

**poczta@inf-art.pl** is sending total 6.8 Million Requests per month (this is 25% of total myACT API Requests), where 86% do not return results

2. **Country SE – 1 Customer login**

**gustaf.angback@trademax.se** is sending total 3.5 Million Requests per month (this is 13% of total myACT API Requests), where 72% do not return results

| USERNAME | SUM(COUNTER) |
|---|---|
| public | 1,896,781,185 |
| **gustaf.angback@trademax.se** | **29,494,173** |
| cz_api_actfre@dhl.com | 25,766,010 |
| epoddell_p | 22,496,090 |
| **poczta@inf-art.pl** | **17,497,202** |
| epodsamsung_p | 11,199,993 |

Table 1: Users with most API requests for year 2022

| USERNAME | SUM(NORESULTS) |
|---|---|
| public | 1,670,115,319 |
| epoddell_p | 18,790,654 |
| **gustaf.angback@trademax.se** | **18,357,996** |
| **poczta@inf-art.pl** | **12,353,224** |
| j.gustafsson@veddesta-distribu | 10,400,639 |
| epodsamsung_p | 9,786,744 |

Table 2: Users with most No results of API requests for year 2022

| USERNAME | SUM(NORESULT_AVG) |
|---|---|
| public | 25.22147 |
| **gustaf.angback@trademax.se** | **19.84390** |
| erik@gardenstore.se | 18.72757 |
| j.gustafsson@veddesta-distribu | 18.54682 |
| epodabb_p | 17.53943 |
| epodabbspa_p | 17.35120 |

Table 3: Users with most avg. no results of API requests for year 2022

| USERNAME | SUM(BLOCKED_AVG) |
|---|---|
| cz_api_actfre@dhl.com | 10.88137 |
| manuel.murcia@saloodo.com | 10.00021 |
| customerportal_np | 10.00007 |
| eic_np | 10.00003 |
| dcg-prod | 10.00003 |

| USERNAME | SUM(BLOCKED_AVG) |
|---|---|
| bas@mshop.se | 10.00001 |

Table 4: Users with most avg. blocked API requests for year 2022

# 5 Future scope

With regarding business requirement, is it does not hold much of an importance as now DHL IT team sending calculated avg. response time for each month. But for yearly response time calculations we can further go in depth and also by calculating for each month (it is not preferred to be done via R soft as it would more convenient with MS excel) and create graph for every month. By doing this we can get analytically data as on which month API usages fluctuated the most. Having information about username, counters and their login history I can do linear regression for predicting which users will the API the most in future and how much counter will increase and can any other alternative way given to those particular customers and also increase number of servers and increase RAM of CPU. The big weakness in future scope that as IT started to send avg response time monthly since September 2022, dataset excel afterwards are different compare to previous excels the columns will get affected so I have to change a code little bit but that is insignificant to the final outcome hence I just added files till September 2022.

# 6 Appendix

```
install.packages("magrittr")
install.packages("dplyr")
install.packages("gganimate")
install.packages("pivottabler")
install.packages("kniter")
install.packages("rmarkdown")
install.packages("raster")
install.packages("rgdal")
install.packages("tibble")
install.packages("tidyverse")
install.packages(c("readxl","writexl"))
install.packages("png")
install.packages("grid")
install.packages("distill")
install.packages("tinytex")



tinytex::install_tinytex()


rm(list = ls())
library(readxl)
library(writexl)
library(tidyr)
library(ggplot2)
library(tibble)
```

```r
library(dplyr)
library(tidyverse)
library(xlsx)
library(magrittr)

#Set directory to where the files are stored
setwd("D:/FRESENIUS/SEMESTER 2/Data Science/Data")
getwd()



# Import those files in R
webUsageExcelsLists <- list.files(pattern = ".xls")
webUsageExcelsLists

# Get the data of these Excels Files
webUsageDataLists = lapply(webUsageExcelsLists, function(i){
  # choose sheet number of that excel file
  x = read_excel(i, sheet=1)
  # choose columns of that excel file
  x <- x[,c(2,4,5,6,7,10)]
  x$file = i
  x
})
# Now we can access all the data from excel individually by:
webUsageDataLists[[5]]
```

```r
# Binding the data from all the Excels
allUsageData = do.call("rbind.data.frame",webUsageDataLists)

#AVG. RESPONSE TIME for all available files
allUsageData$AVG_RESPONSE_TIME <- round(
  allUsageData$RESPONSE_TIME  / allUsageData$COUNTER,
  digits = 4)

# Create an Excel
write_xlsx(allUsageData,"D:/FRESENIUS/SEMESTER 2/Data Science
           /Data/more data/allAPIResponse.xlsx")

View(allUsageData)

data1 <- data.frame(OPERATION = allUsageData$OPERATION,
                    COUNTER = allUsageData$COUNTER,
                    RESPONSE_TIME = allUsageData$RESPONSE_TIME
)

#COUNTER vs OPERATION
usageCouterOperationBased <- aggregate(COUNTER ~ OPERATION,
                                       allUsageData, sum)

#RESPONSE TIME vs OPERATION
usageResponseTimeOperationBased <- aggregate(RESPONSE_TIME
                           ~ OPERATION, allUsageData, sum)
```

```
# Collect all the data in one file
data2 <- data.frame(OPERATION <-
                      usageResponseTimeOperationBased$OPERATION,
                    COUNTER =
                      usageCouterOperationBased$COUNTER,
                    RESPONSE_TIME =
                      usageResponseTimeOperationBased$RESPONSE_TIME
)
#AVG. RESPONSE TIME vs OPERATION
data2$AVG_RESPONSE_TIME <- round(data2$RESPONSE_TIME  /
                           data2$COUNTER, digits = 4)
data2


Create an Excel
write_xlsx(data2,"D:/FRESENIUS/SEMESTER 2/Data Science/
           Data/more data/yearBaseAvgAPIResponse.xlsx")


# Plot the Graphs
library(RCurl)
x <- getURL("https://raw.githubusercontent.com/adityadande/
            DataSciene/main/yearBaseAvgAPIResponse.csv")
graphData <- read.csv(text = x)


graphData %>%
  dplyr::arrange(graphData, desc(graphData$AVG_RESPONSE_TIME)) %>%
  dplyr::mutate(name=
  factor(graphData$OPERATION....usageResponseTimeOperationBased.OPERATION,
```

```
        levels=
graphData$OPERATION....usageResponseTimeOperationBased.OPERATION)) %>%
  ggplot( aes(x=
    graphData$OPERATION....usageResponseTimeOperationBased.OPERATION,
    y=graphData$AVG_RESPONSE_TIME)) +
  geom_segment( aes(xend=
  graphData$OPERATION....usageResponseTimeOperationBased.OPERATION, yend=0)) +
  geom_point( size=4, color="orange") +
  coord_flip() +
  theme_bw() +
  xlab("OPERATIONS") +
  ylab("AVG RESPONSE TIME (sec)")  +
  geom_text(aes(label = signif(graphData$AVG_RESPONSE_TIME, digits = 5))
        , nudge_y = 0.1, color="blue")


# Users Analysis


library(RCurl)
x <- getURL("https://raw.githubusercontent.com/adityadande/
      DataSciene/main/allAPIResponse.csv")
apiData <- read.csv(text = x)


# Username vs Counter
agg_tbl <- apiData %>% dplyr::group_by(USERNAME) %>%
  dplyr::summarise(sum(COUNTER))
agg_tbl <- agg_tbl[order(-agg_tbl$`sum(COUNTER)`), ]
iris2 <- head(agg_tbl)
```

```r
# Username vs No response
agg_tb2 <- apiData %>% dplyr::group_by(USERNAME) %>%
  dplyr::summarise(sum(NORESULTS))
agg_tb2 <- agg_tb2[order(-agg_tb2$`sum(NORESULTS)`), ]
iris3 <- head(agg_tb2)

knitr::kable(
  list(
    iris2, iris3
  ), align = "lccrr",
  caption = "Users with the most API request & result",
  digits = 10, format.args = list(big.mark = ",",
  scientific = FALSE), booktabs = TRUE, valign = 't')
# NORESULT / COUNTER
apiData$NORESULT_AVG <- apiData$NORESULTS
/ apiData$COUNTER
agg_tb3 <- apiData[,c(1,2,3,4,8)]
agg_tb3 <- agg_tb3 %>% dplyr::group_by(USERNAME)
%>% dplyr::summarise(sum(NORESULT_AVG))

iris4 <- head(agg_tb3[order(-agg_tb3$`sum(NORESULT_AVG)`),])

# BLOCKED / COUNTER
apiData$BLOCKED_AVG <- apiData$BLOCKED  / apiData$COUNTER
agg_tb4 <- apiData[,c(1,2,3,5,9)]
agg_tb4 <- agg_tb4 %>% dplyr::group_by(USERNAME) %>%
```

```
  dplyr::summarise(sum(BLOCKED_AVG))


iris5 <- head(agg_tb4[order(-agg_tb4$`sum(BLOCKED_AVG)`),])


knitr::kable(
  list(
    iris5, iris4
  ), align = "lccrr",
  caption = "Avg. Most NO RESULT & BLOCKED per COUNTER For each Customer Analysis",
  digits = 10, format.args =
    list(big.mark = ",",scientific = FALSE), booktabs = TRUE, valign = 't')
```