UNIVERSITÀ DEGLI STUDI DI PALERMO

Dipartimento di Scienze Economiche, Aziendali e Statistiche

Master annuale di secondo livello in

Data Science and Big Data Analytics

Data Analysis and Data Cleaning on a BigQuery Database

Tesi di:
Ludovica Tomaselli

Relatore:
Prof. Marcello Chiodi
Tutor aziendale:
Dott.ssa Francesca Motisi

Anno Accademico 2020/2021

Contents Table

Introduction

This master's thesis is the result of the work I carried out at Cloudtec, an innovative startup based in Palermo that specializes in cloud-based solutions leveraging the power of artificial intelligence to innovate processes and services.

This thesis addresses several issues related to the usability of data contained in the database of a private postal services company.

To this end, Google Cloud Platform services were used, specifically BigQuery, Data Studio, and BigQuery Geo Viz, which are respectively used for data analysis, presentation, and geolocation.

The company's goal was to use the data we had to track the movements of mail carriers and their deliveries. However, the database presented some issues, including the presence of repeated data and anomalous frequencies regarding deliveries per mail carrier.

My work therefore began with the study of Google Cloud Platform services, with particular reference to those related to the project at hand. The work then focused first on evaluating the usability of the database for the proposed purposes and, secondly, on cleaning the data and removing "dirty" data.

The following work is structured into three chapters: the first chapter briefly discusses the nature and use of Google Cloud Platform services; the second chapter describes the procedures performed on the database, the analyses carried out, the difficulties and critical issues encountered, and the solutions found that make the database usable; finally, in the third chapter, I developed the conclusions and identified some possible lines of development for the continuation of the project.

Capitolo 1

Google Cloud Platform (GCP)

My internship focused on the analysis of a database using some of the tools offered by Google Cloud Platform, in particular BigQuery, Data Studio, and BigQuery Geo Viz. Part of my internship was dedicated to studying these services to become familiar with them and learn how to work with them. Specifically, I took a course on BigQuery in collaboration between Coursera and Google Cloud Training, which focused on the main aspects of its usage, and then a course on Udemy about Data Studio. In this chapter, I will briefly explain what Google Cloud Platform is and the three tools that I used.

1.1 What is Google Cloud Platform

Google Cloud Platform (GCP) is a collection of cloud computing services introduced by Google on April 7, 2008. GCP uses the same infrastructure as other Google products such as Google Search, YouTube, Google Drive, etc. The list of services offered is quite long and is divided into various categories that include, but are not limited to, AI and Machine Learning, API Management, Data Analytics, Databases, Developer Tools, etc. Here are some of the most relevant examples of the services offered:

- Compute Engine: Virtual machines running on Google data centers
- Anthos: Platform for modernizing existing apps and creating new ones
- Google Kubernetes Engine: Managed environment for running containerized apps
- Cloud Storage: Secure, durable, and scalable object storage
- BigQuery: Enterprise data warehouse

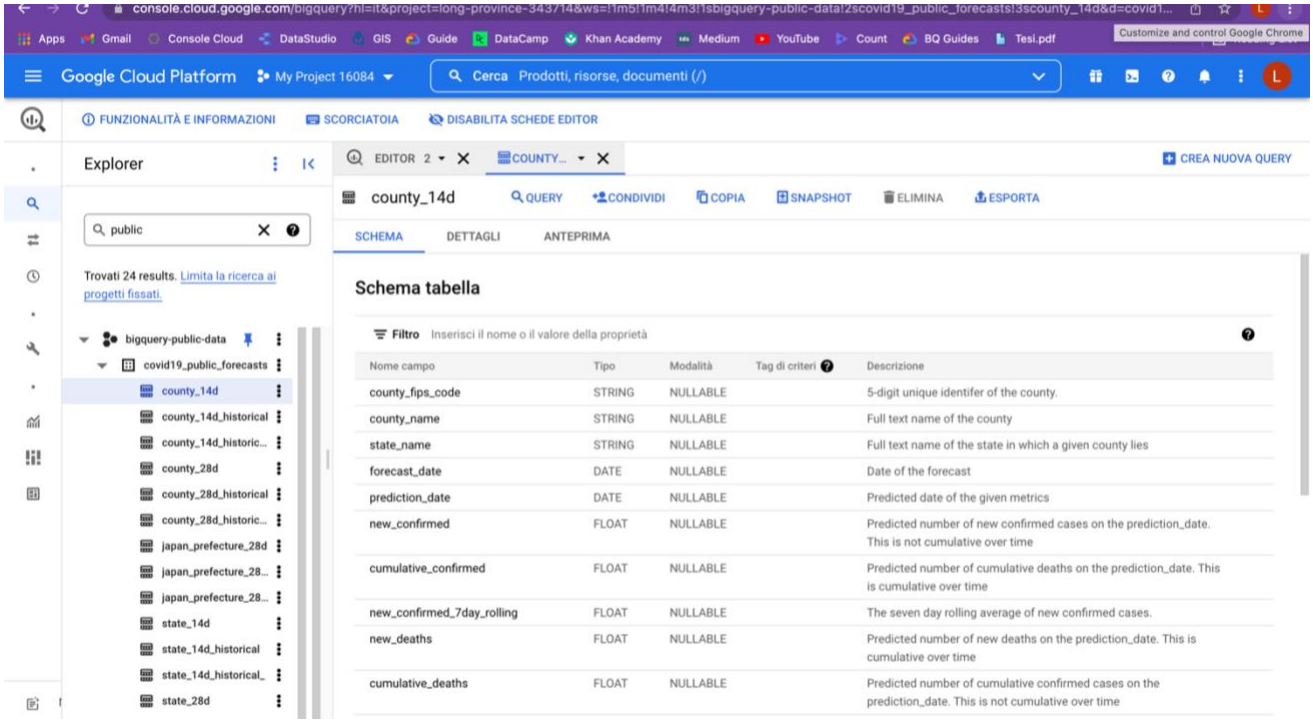
The range of services offered is so wide and covers so many areas that it would be impossible and even unnecessary to know them all. We will now focus on those relevant to this project.

1.2 BigQuery

BigQuery was made available on GCP starting in November 2011. It is a service that allows for interactive analysis of large data sets. BigQuery makes it easy to interact with databases uploaded to Google Storage and, since 2016, support for Google Sheets has also been added.

Incredibly fast, BigQuery is able to perform selection or grouping queries on billions of data in just seconds. Highly scalable and serverless, it makes managing and analyzing data much simpler and more direct, without the need to worry about managing infrastructure. Additionally, it makes it possible to share datasets arbitrarily with individual users or groups, or even providing public access.

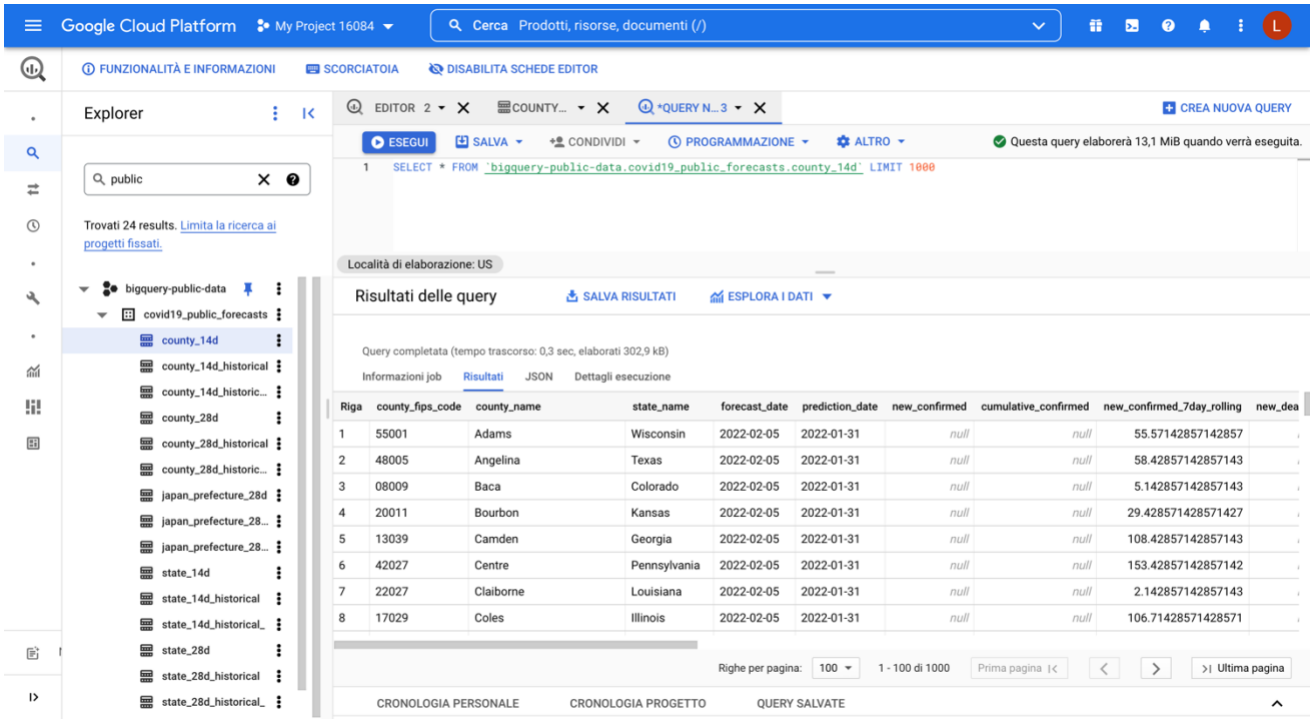
1.2.1 BigQuery Interface – Schema



BigQuery includes methods that allow to create, populate, and delete tables, as well as executing queries on them, and it's also possible to import external data into it.
By clicking on the desired table shown in the left menu, you can view its information. As shown in the image, the first thing displayed is the schema, and if desired, you can also check the details and preview of the table.

To query a database in BigQuery, simply create a new Query, enter the code in the appropriate space, and give the "Run" command. Initially, BigQuery used a non-standard SQL dialect known as BigQuery SQL, but with the release of BigQuery 2.0, support for standard SQL was added, and the BigQuery SQL dialect was renamed as Legacy SQL. BigQuery still supports both dialects, but it recommends the use of standard SQL.
I can anticipate that in my internship, I used the latter.

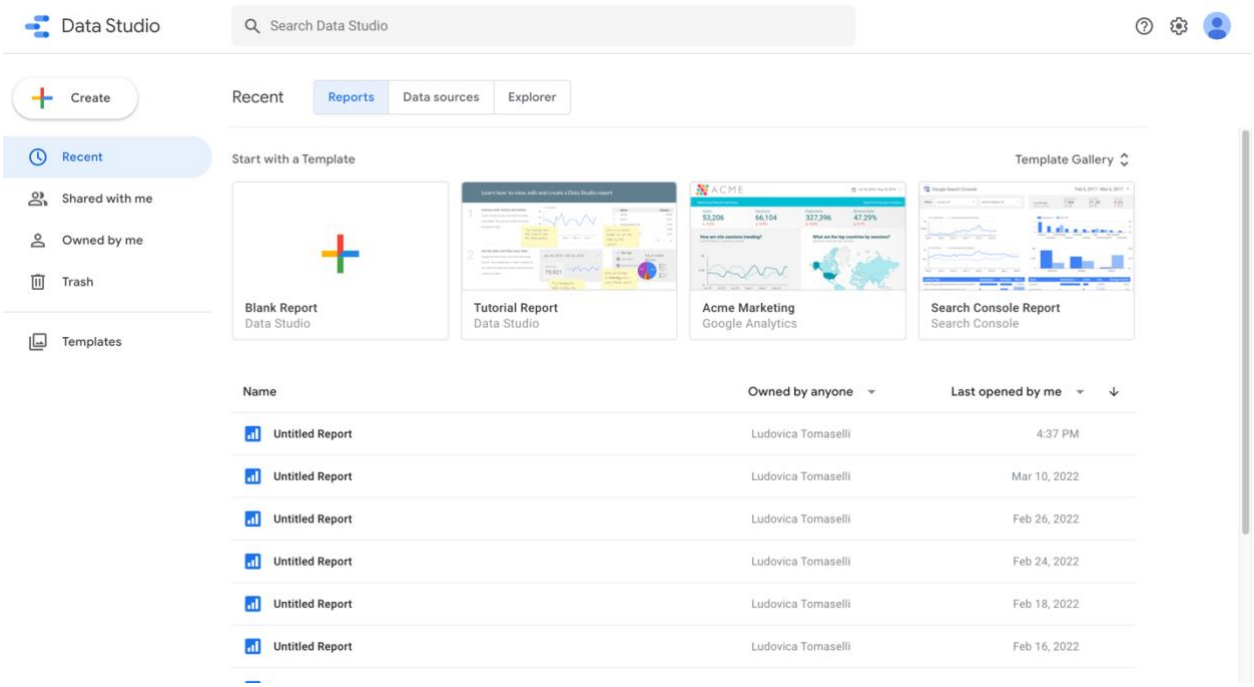
1.2.2 BigQuery Interface – Query



1.3 Data Studio

Google Data Studio is a free data visualization tool that allows users to quickly create clear and visually appealing presentations. Launched by Google in 2016, it is known for being easy to use, customize, and share. Another strength of Data Studio is its ability to connect to a wide range of data sources through the use of connectors, which come in three types: Google connectors, Partners connectors, and Open Source connectors. Google and Partners connectors are pre-integrated into Data Studio, while Open Source connectors are owned by third parties and require integrations to import data.

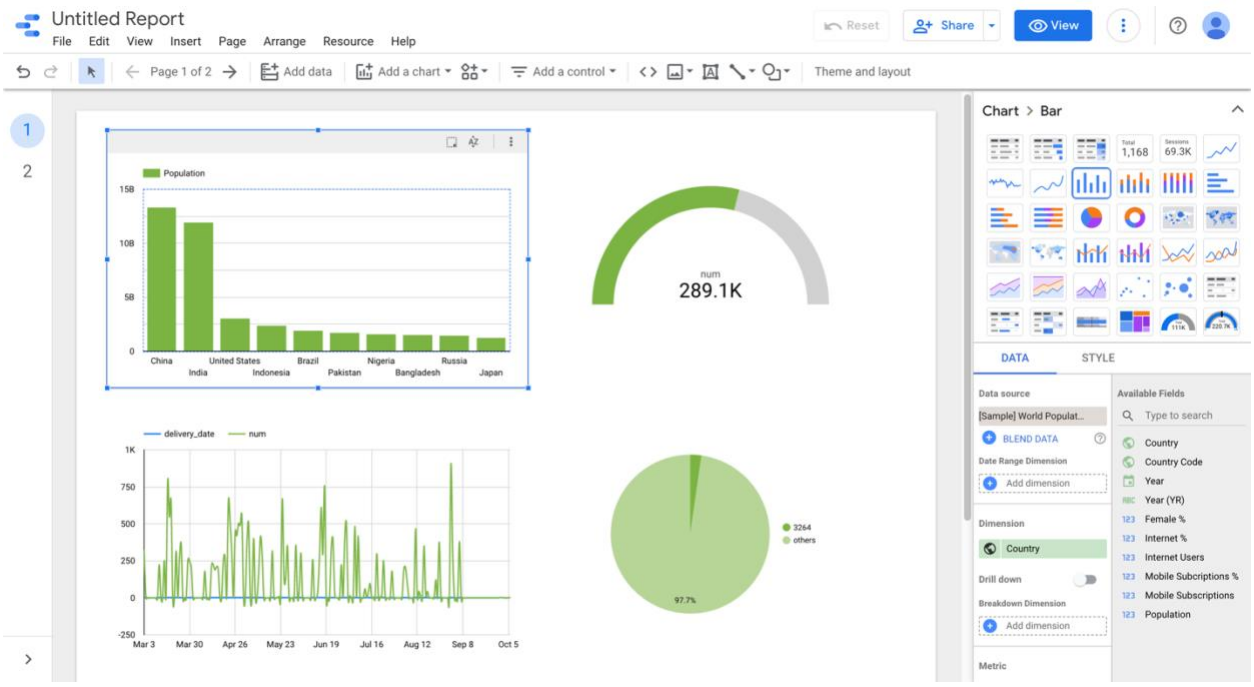
1.3.2 Homepage Data Studio



Once a data source (e.g. BigQuery, Google Sheets, Google Analytics, or even .csv or .xls documents, etc.) is connected to Data Studio, the data is automatically imported into the tool and transformed into the metrics and dimensions that you want to represent in the report. Moreover, the created dashboards are dynamic (obviously if it is not a manually uploaded document), which means that the data inside them updates automatically.

The Data Studio interface is very intuitive and easy to use and works in a way similar to PowerPoint: to create a new chart, simply click on the appropriate icon, decide what type of chart you want to create (and you can change the type later), place it on the page and modify it as desired.

1.3.2 Interface Data Studio



To load the data, simply click on the "Data Source" section and follow the procedure to import or connect the data. In the same section, further down, you can specify all the parameters that you want to display in the chart, and the "Style" section allows you to customize the graphics. There is no limit to the number of charts that can be inserted on a single page, it is possible to create multi-page documents, and you can write text, create shapes, insert images, in short, everything that can be useful to create efficient, clear and, why not? Visually appealing reports.

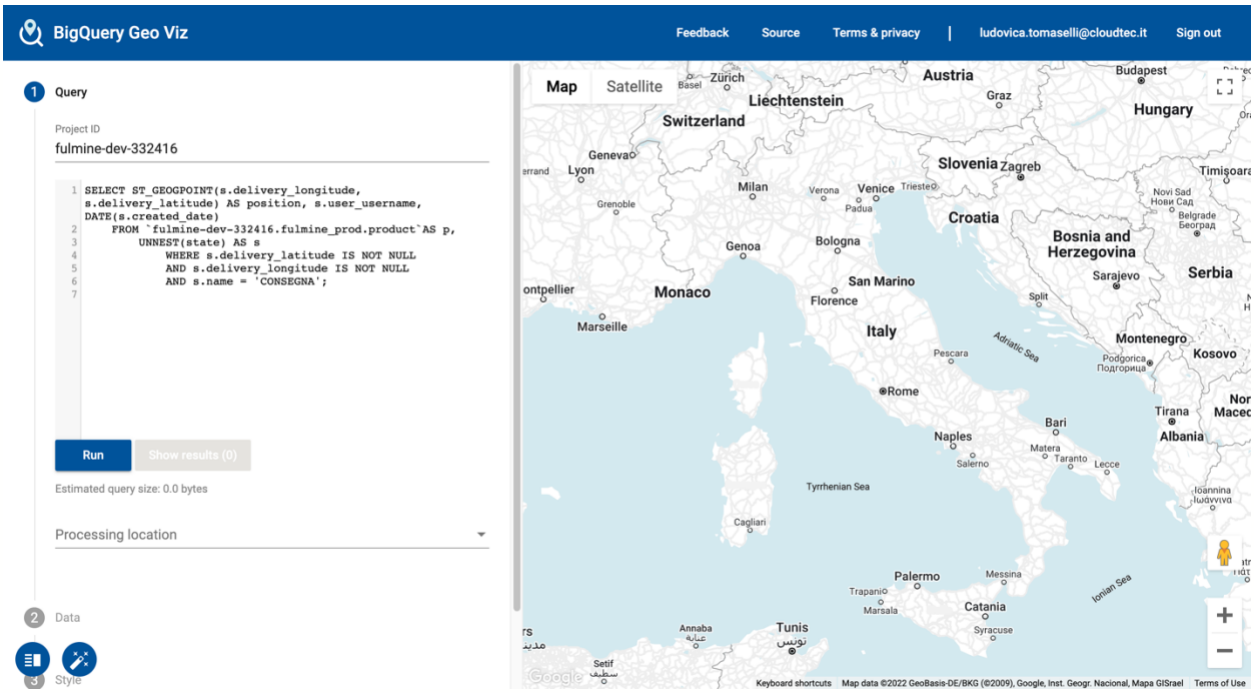
It is very important to emphasize that the documents created in this way can be downloaded as a PDF or shared with third parties.

I also want to point out that all the charts shown in the second chapter were created using Data Studio.

1.4 BigQuery Geo Viz

BigQuery Geo Viz is the tool I have used the least among the three, but at the same time, it is also the simplest and most straightforward. As the name suggests, it is directly connected to BigQuery and uses Google Maps APIs to visualize geospatial data. It is not a complete geospatial analysis tool and has many limitations, but it is still a useful tool for visualizing the results of a geospatial analysis query on a map, one query at a time.

17.672.87 Interface BigQuery Geo Viz



Capitolo 2

My internship

I carried out my internship at Cloudtec, a young Innovative Startup and Google Cloud partner based in Palermo, Italy, specializing in cloud migration, artificial intelligence, and data analysis. I would like to emphasize that my internship with them is currently ongoing, and this project is not to be considered concluded.
My work focused on the database of a private postal company, whose identity I will not reveal for privacy reasons. The database, from the outset, proved to be quite problematic and had a strong presence of dirty data, to the point that I had almost lost hope of being able to extract any useful information. However, thanks to the encouragement of my supervisor, I persisted in my analysis and discovered that by eliminating the relatively small percentage of out-of-scale data, the rest was perfectly usable.
In this chapter, I will illustrate the various stages of my journey.

2.1 The database

I was not able to determine the exact date of the creation of the database, as the person who designed it is no longer part of the team, but the first entry dates back to December 14, 2020. It was created and is still managed with MySQL8, but to avoid overloading the server with an excessive amount of data, every three months they are deleted locally. A copy is kept on BigQuery in Google Cloud Platform.

There have been some issues with this database, the main one (among those known) being data duplication attributable to the app. Several patches have been released to fix the problem, the latest dating back to November 2021.

Below, you can see the structure of the database.

2.4.5 Database Structure

ID	INTEGER	NULLABLE
CREATED_DATE	DATETIME	NULLABLE
BARCODE	STRING	**REQUIRED**
AGENCY_ID	INTEGER	**REQUIRED**
AGENCY_NAME	STRING	NULLABLE
NOTE	STRING	NULLABLE
RECIPIENT_TYPE	STRING	NULLABLE
DOCUMENT_TYPE	STRING	NULLABLE
DOCUMENT_NUMBER	STRING	NULLABLE
PRODUCT_TYPE	STRING	**REQUIRED**
↓ **STATE**	RECORD	REPEATED
NAME	STRING	**REQUIRED**
SEGNCOD	STRING	NULLABLE
USER_ID	INTEGER	NULLABLE
USER_USERNAME	STRING	**REQUIRED**
CREATED_DATE	DATETIME	NULLABLE
DELIVERY_SEND_STATE	STRING	NULLABLE
DELIVERY_CREATED_DATE	DATETIME	NULLABLE
DELIVERY_LATITUDE	FLOAT	NULLABLE
DELIVERY_LONGITUDE	FLOAT	NULLABLE

2.2 Explanation of the database

This is an individual table with a partition inside it. The schema was designed in this way to easily track the entire journey of each individual parcel, from acceptance to delivery.
Below are the individual fields:

2.4.5 Explanation of each field

ID	Identification code for each received parcel, less reliable than the barcode as it can be missing.
CREATED_DATE	Date of receipt of the parcel
BARCODE	Refers to the barcode of the letter and is the primary identification code
AGENCY_ID	Identification code of the agency that accepted the letter
AGENCY_NAME	The name of the agency that accepted the letter or package
NOTE	Possible notes.
RECIPIENT_TYPE	Role of the recipient of the letter (e.g. recipient, relative, delegate, organization, etc.)
DOCUMENT_TYPE	Type of document shown at the receipt (if necessary)
DOCUMENT_NUMBER	Code of the document shown at the receipt (if necessary)
PRODUCT_TYPE	Type of letter (SDOC, parcel, registered)

16.748.077 results

While the main table contains general data related to the letter, the STATE table identifies its movements. Each row records the acceptance and, subsequently, the various delivery attempts.

2.2.2 Exposure of individual fields of the database in the nested table

↓ STATE	
NAME	Identifies whether the letter is in acceptance or delivery
SEGNCOD	Identifies the delivery method or the reason for a failed delivery
USER_ID	Identification code of the postal worker
USER_USERNAME	Identifying name of the postal worker
CREATED_DATE	Date of the movement indicated in the row (acceptance or delivery attempt)
DELIVERY_SEND_STATE	Identifies whether the letter has been sent or if an error occurred
DELIVERY_CREATED_DATE	Date on which the letter was delivered
DELIVERY_LATITUDE	The two areas, latitude and longitude, refer to the position of the delivery.
DELIVERY_LONGITUDE	

17.672.087 results

2.3 Goal

The purpose of my analysis was to verify if the database, which had not been used or properly reviewed so far, was actually usable and if the data contained in it was reliable. In particular, I was interested in understanding if it would have been possible to track the movements of the postmen. Below I will illustrate the phases of my analysis.

2.4 Familiarization analysis

To begin, I made some queries to understand the content of the database and be able to approach it better. I started by searching for the usernames of all the postmen and counting them.

In the following examples, I will show the code I wrote and the first 10 results returned by BigQuery for illustrative purposes (except in cases where the result is less than 10). The results are always in descending order (unless specified otherwise), so the first results are always the most extreme. I also specify that all the names of the postmen have been replaced with fictional names to protect the company's privacy.

```
SELECT COUNT(DISTINCT s.user_username)
  FROM `database-dev-332416.database_prod.product`AS p,
    UNNEST(state) AS s;
```

2.4.1 Total number of postmen

Number of Postmen
1261

```
SELECT DISTINCT state.user_username,
  FROM `database-dev-332416.database_prod.product`,
    UNNEST(state) AS state
  ORDER BY state.user_username;
```

2.4.2 All postmen

	Postmen
1	Alpha
2	Beta
3	Gamma
4	Delta
5	Epsilon
6	Zeta
7	Eta
8	Theta
9	Iota

10	Kappa

1.261 results

Next, I verified all the days on which scans were made, which resulted in 361 days of activity.

```
SELECT DISTINCT(DATE(s.delivery_created_date)) AS giorno_lavorativo
  FROM `database-dev-332416.database_prod.product`AS p,
    UNNEST(state) AS s
  ORDER BY day;
```

2.4.3 All recorded dates

	Recorded dates
1	*null*
2	2020-12-14
3	2020-12-15
4	2020-12-16
5	2020-12-17
6	2020-12-18
7	2020-12-19
8	2020-12-20
9	2020-12-21
10	2020-12-22

361 results

I then verified the total number of scans performed by each delivery postman, showing the highest results first.

```
SELECT DISTINCT state.user_username as postini, COUNT(id) as scansioni
  FROM `database-dev-332416.database_prod.product`,
    UNNEST(state) AS state
  GROUP BY postini
  ORDER BY scansioni DESC;
```

2.4.4 Total scans for each postman

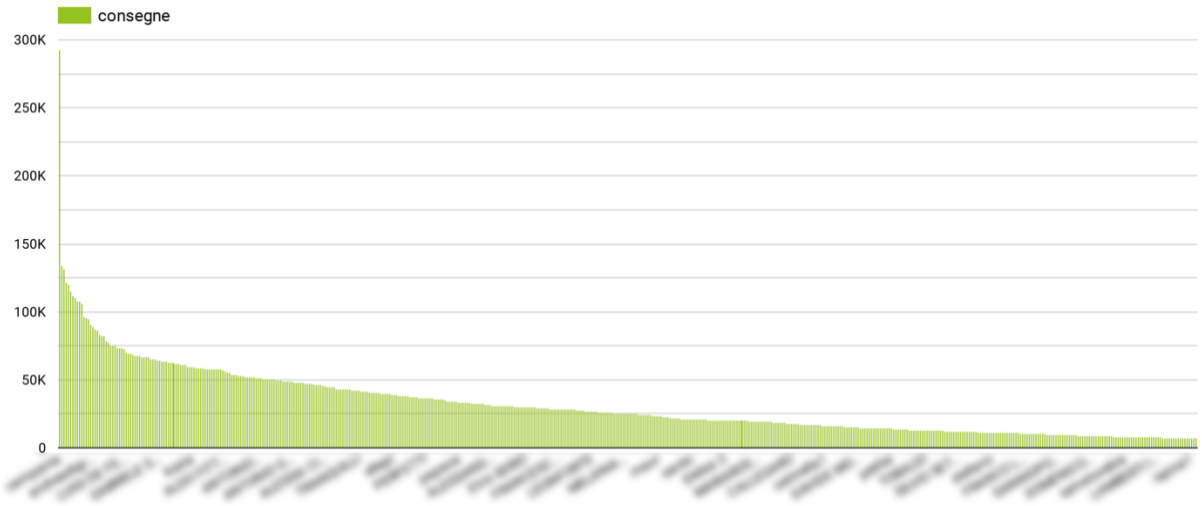
	Postman	Scans
1	lambda	292527
2	mi	134384
3	NI	131233
4	XI	122095
5	OMICRON	119974

6	Pi	115596
7	RHO	111744
8	Sigma	110612
9	Tau	107780
10	Upsilon	107565

1.216 results

The result I obtained was surprising, it was definitely excessive. I decided to view it in Data Studio to better understand, through visual impact, how much data was involved in this volume. Fortunately, the number of scans that were so high, while not insignificant, seemed to occupy only a relatively small part of the total.

2.4.5 Total scans for each postman – Histogram



Based on this data, I decided to calculate the average total deliveries per postman, which I found quite reasonable.

```
SELECT AVG(consegne) as Media_consegne, VARIANCE(consegne) as Varianza
FROM(
SELECT DISTINCT state.user_username as postini, COUNT(id) as consegne
  FROM `database-dev-332416.database_prod.product`,
    UNNEST(state) AS state
  GROUP BY postini
  ORDER BY consegne DESC);
```

2.4.6 Average deliveries for postman

Average scans	Variance
14532.96	5.05

I wanted to also verify the working days for each individual postman, and in this case as well, I employed Data Studio to visualize the results.

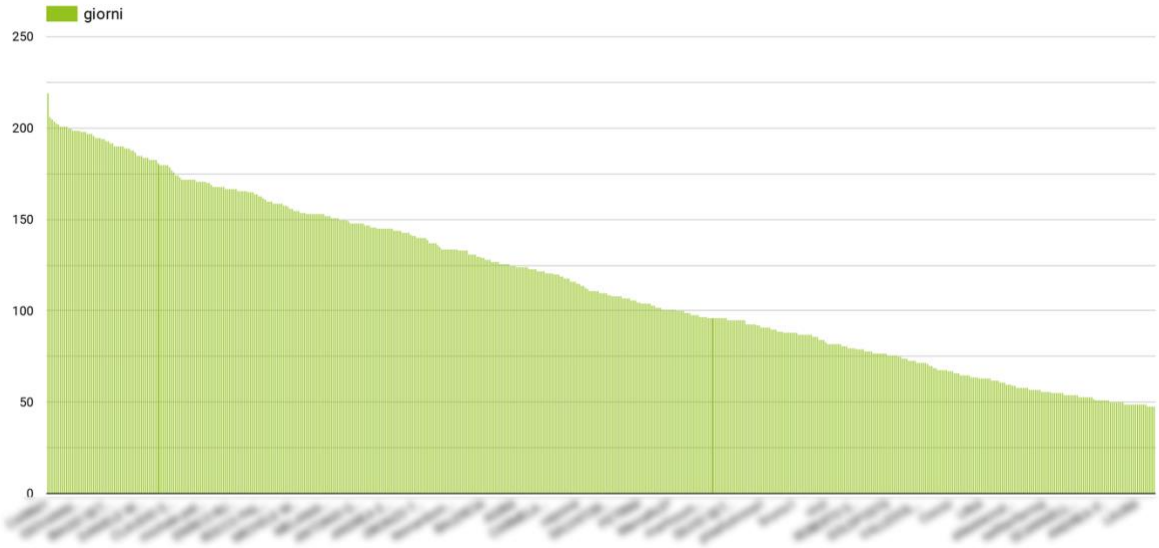
```
SELECT R.postini, COUNT(DISTINCT R.giorni) as giorni_lavorativi
FROM
(SELECT DISTINCT state.user_username as postini, DATE(state.created_date) as giorni
   FROM `database-dev-332416.database_prod.product`,
      UNNEST(state) AS state
   GROUP BY postini, giorni
   ORDER BY giorni DESC) as R
GROUP BY R.postini
ORDER BY giorni DESC;
```

2.4.7 Total working days for each postman

	Postman	Working days
1	Gamma	219
2	Epsilon	206
3	Theta	205
4	Omicron	204
5	Omega	203
6	Iota	202
7	Phi	201
8	Psi	201
9	Lambda	201
10	Xi	201

1.216 results

2.4.8 Total working days for each postman - Histogram


giorni
250
200
150
100
50
0

I then searched for the total daily scans. Again, the number was too high, and even from the histogram, it can be seen that the absurdly high data is too much.

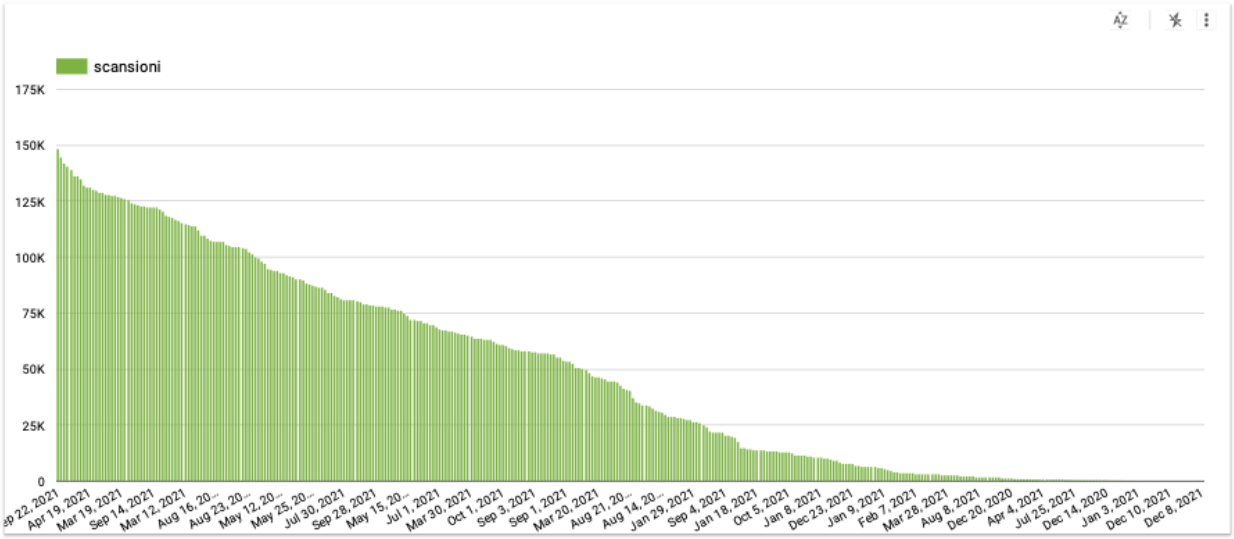
```
SELECT DISTINCT(DATE(s.delivery_created_date)) AS giorno, COUNT(DISTINCT id) AS scansioni
  FROM `database-dev-332416.database_prod.product`AS p,
    UNNEST(state) AS s
  GROUP BY giorno
  ORDER BY scansioni DESC;
```

2.4.9 Total daily scans

	Day	Scans
1	2021-09-22	148509
2	2021-03-16	144516
3	2021-05-17	142131
4	2021-05-18	140502
5	2021-06-17	139348
6	2021-06-16	136236
7	2021-07-20	136164
8	2021-03-18	135026
9	2021-09-21	132063
10	2021-05-19	131396

361 results

2.4.10 Total daily scans - Histogram



I continued and decided to investigate further the nature of the problem. While it was clear that there were some dirty data, there was still a possibility to clean them up.

I also checked the last day on which scans were recorded.

```
SELECT DISTINCT(DATE(s.delivery_created_date)) as ultima scansione
FROM `database-dev-332416.database_prod.product`AS p,
      UNNEST(state) AS s
   ORDER BY ultima scansione DESC
   LIMIT 1;
```

2.4.11 Last scan

Last scan
2021-12-31

However, I must mention that, following further analysis, I noticed that the months of November and December are unfortunately to be considered almost missing, since the number of scans is limited to one per day sporadically, which leads me to believe that they are errors.

2.5 Missing data analysis

At this point, having a general idea of the data I was dealing with, I wanted to verify the percentage of missing data, in particular those related to latitude and longitude. I started by working on a specific mail carrier on a specific day to first verify the validity of the query.
As a first step, I constructed subqueries with CASE WHEN in which I first counted all values related to a single worker and a single randomly chosen date, then counted all missing values and finally all present values. I then returned the results of these values and calculated the percentage.

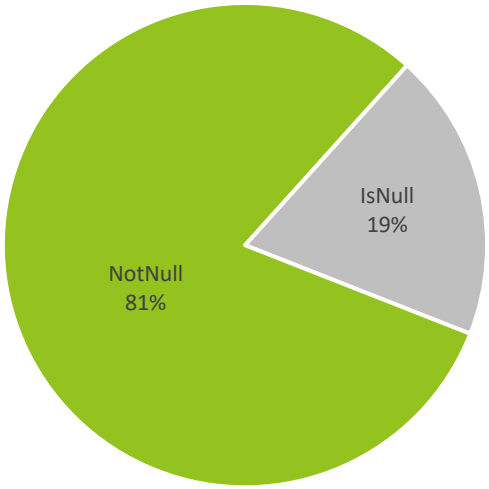
```
SELECT total, NotNull, IsNull, ROUND(IsNull * 100.0 / total,2) AS Null_percent, ROUND(NotNull * 100.0 /
total,2) AS NotNull_percent
FROM (SELECT
      (SELECT SUM
            (CASE WHEN s.user_username = 'Phi'
                  AND s.created_date BETWEEN '2021-09-01T00:00:00.000'
                  AND '2021-10-10T00:00:00.000'
                  THEN 1 ELSE 0 END) AS deliveries
            FROM `database-dev-332416.database_prod.product`as p,
            UNNEST(state) AS s)
```

```
    AS total,

                                                                    continua

(SELECT SUM
        (CASE WHEN s.user_username = 'Phi'
            AND s.created_date BETWEEN '2021-09-01T00:00:00.000'
            AND '2021-10-10T00:00:00.000'
            AND s.delivery_latitude IS NOT NULL
            AND s.delivery_longitude IS NOT NULL
            THEN 1 ELSE 0 END) AS deliveries
        FROM `database-dev-332416.database_prod.product`as p,
        UNNEST(state) AS s)
    AS NotNull,

    (SELECT SUM
        (CASE WHEN s.user_username = 'Phi'
            AND s.created_date BETWEEN '2021-09-01T00:00:00.000'
            AND '2021-10-10T00:00:00.000'
            AND s.delivery_latitude IS NULL
            AND s.delivery_longitude IS NULL
            THEN 1 ELSE 0 END) AS deliveries
        FROM `database-dev-332416.database_prod.product`as p,
        UNNEST(state) AS s)
    AS IsNull);
```

As formulated, the query returned the following result:

2.4.5 Comparison and percentage of present and missing data on latitude and longitude for a mail carrier

total	NotNull	IsNull	Null_percent	NotNull_percent
171	138	33	19.3	80.7



At this point, I proceeded to apply the query to the entire database.

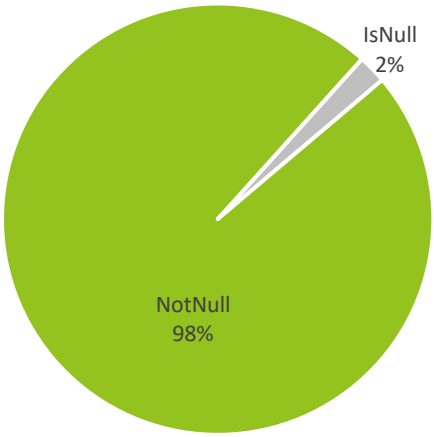
```
SELECT total, NotNull, IsNull, ROUND(IsNull * 100.0 / total,2) AS Null_percent, ROUND(NotNull * 100.0 /
total,2) AS NotNull_percent
FROM (SELECT
      (SELECT COUNT(*) AS deliveries
            FROM `database-dev-332416.database_prod.product`as p,
            UNNEST(state) AS s)
            AS total,

      (SELECT SUM
            (CASE WHEN s.delivery_latitude IS NOT NULL
             AND s.delivery_longitude IS NOT NULL
                  THEN 1 ELSE 0 END) AS deliveries
            FROM `database-dev-332416.database_prod.product`as p,
            UNNEST(state) AS s)
            AS NotNull,

      (SELECT SUM
            (CASE WHEN s.delivery_latitude IS NULL
             AND s.delivery_longitude IS NULL
                  THEN 1 ELSE 0 END) AS deliveries
            FROM `database-dev-332416.database_prod.product`as p,
```

```
    UNNEST(state) AS s)
    AS IsNull);
```

The results obtained, shown below, were very positive, with a very low presence of missing data.

2.5.2 Comparison and percentage of present and missing data on latitude and longitude

total	NotNull	IsNull	Null_percent	NotNull_percent
17672087	17290203	381884	2.16	97.84



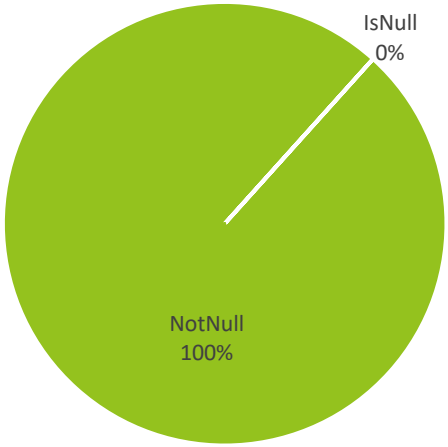
I repeated the query with other fields among those most relevant to my analysis to also verify the percentage of missing data there, and I obtained excellent results again, which approached 100% of present data.
Below, I report the results on the "created_date" field:

```
SELECT total, NotNull, IsNull, ROUND(IsNull * 100.0 / total,2) AS Null_percent, ROUND(NotNull * 100.0 /
total,2) AS NotNull_percent
FROM (SELECT
    (SELECT COUNT(*) AS deliveries
        FROM `database-dev-332416.database_prod.product`as p,
        UNNEST(state) AS s)
        AS total,

    (SELECT SUM
        (CASE WHEN p.created_date IS NOT NULL
            THEN 1 ELSE 0 END) AS deliveries
```

```
        FROM `database-dev-332416.database_prod.product`as p,
        UNNEST(state) AS s)
        AS NotNull,

    (SELECT SUM
        (CASE WHEN p.created_date IS NULL
            THEN 1 ELSE 0 END) AS deliveries
        FROM `database-dev-332416.database_prod.product`as p,
        UNNEST(state) AS s)
        AS IsNull);
```

2.5.3 Comparison and percentage of present and missing data on created_date

total	NotNull	IsNull	Null_percent	NotNull_percent
17672087	17672086	1	0.0	100.0



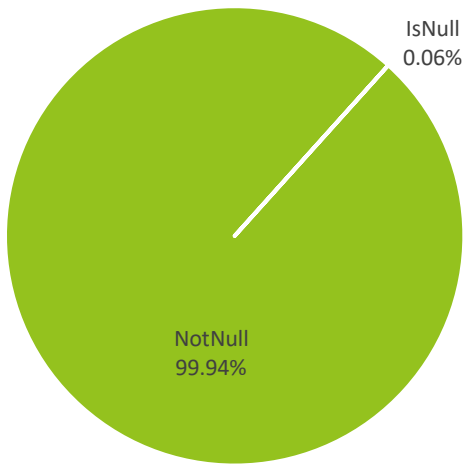
The results on the state.created_date field:

```
SELECT total, NotNull, IsNull, ROUND(IsNull * 100.0 / total,2) AS Null_percent, ROUND(NotNull * 100.0 /
total,2) AS NotNull_percent
FROM (SELECT
    (SELECT COUNT(*) AS deliveries
        FROM `database-dev-332416.database_prod.product`as p,
        UNNEST(state) AS s)
        AS total,
```

```
(SELECT SUM
    (CASE WHEN s.created_date IS NOT NULL
        THEN 1 ELSE 0 END) AS deliveries
    FROM `database-dev-332416.database_prod.product`as p,
    UNNEST(state) AS s)
    AS NotNull,

(SELECT SUM
    (CASE WHEN s.created_date IS NULL
        THEN 1 ELSE 0 END) AS deliveries
    FROM `database-dev-332416.database_prod.product`as p,
    UNNEST(state) AS s)
    AS IsNull);
```

2.5.4 Comparison and percentage of present and missing data on state.created_date

total	NotNull	IsNull	Null_percent	NotNull_percent
17672087	17662120	9967	0.06	99.94

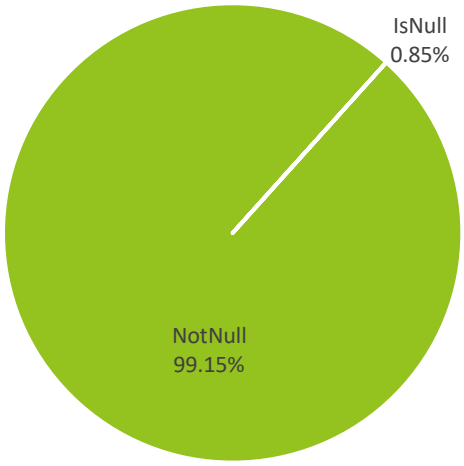


The results on the state.delivery_send_state field:

```
SELECT total, NotNull, IsNull, ROUND(IsNull * 100.0 / total,2) AS Null_percent, ROUND(NotNull * 100.0 /
total,2) AS NotNull_percent
FROM (SELECT
    (SELECT COUNT(*) AS deliveries
        FROM `database-dev-332416.database_prod.product`as p,
        UNNEST(state) AS s)
        AS total,

    (SELECT SUM
        (CASE WHEN s.delivery_send_state IS NOT NULL
            THEN 1 ELSE 0 END) AS deliveries
        FROM `database-dev-332416.database_prod.product`as p,
        UNNEST(state) AS s)
        AS NotNull,

    (SELECT SUM
        (CASE WHEN s.delivery_send_state IS NULL
            THEN 1 ELSE 0 END) AS deliveries
        FROM `database-dev-332416.database_prod.product`as p,
        UNNEST(state) AS s)
        AS IsNull);
```

2.5.5 Comparison and percentage of present and missing data on state.delivery_send_state

total	NotNull	IsNull	Null_percent	NotNull_percent
17672087	17522727	149360	0.85	99.15



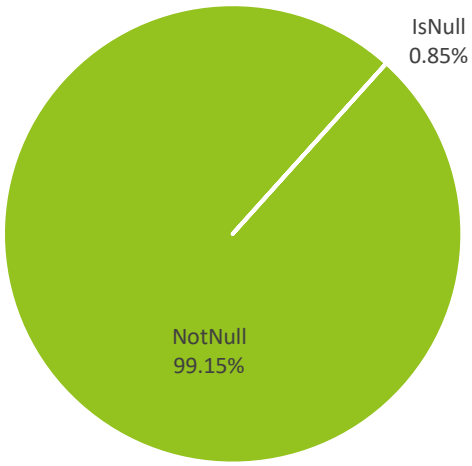
Lastly, the results on the state.delivery_created_date field:

```
SELECT total, NotNull, IsNull, ROUND(IsNull * 100.0 / total,2) AS Null_percent, ROUND(NotNull * 100.0 /
total,2) AS NotNull_percent
FROM (SELECT
    (SELECT COUNT(*) AS deliveries
        FROM `database-dev-332416.database_prod.product`as p,
        UNNEST(state) AS s)
        AS total,

    (SELECT SUM
        (CASE WHEN s.delivery_created_date IS NOT NULL
            THEN 1 ELSE 0 END) AS deliveries
        FROM `database-dev-332416.database_prod.product`as p,
        UNNEST(state) AS s)
        AS NotNull,

    (SELECT SUM
        (CASE WHEN s.delivery_created_date IS NULL
            THEN 1 ELSE 0 END) AS deliveries
        FROM `database-dev-332416.database_prod.product`as p,
```

```
UNNEST(state) AS s)
AS IsNull);
```

2.5.6 Comparison and percentage of present and missing data on state.delivery_created_date

total	NotNull	IsNull	Null_percent	NotNull_percent
17672087	17522727	149360	0.85	99.15



2.6 Analisys on scans

Despite a large number of seemingly dirty data, the database had at least a very low percentage of missing data, especially in the fields most relevant to me.
At this point, I decided to verify the daily scans performed by each mail carrier. The results obtained were predictably absurdly high: the highest peaks almost reached 20,000 daily scans, a ridiculously high number. But even excluding the dozens of results that high, there were still hundreds of daily scans above 600 (the maximum indicative number declared by the company itself).

```
SELECT DISTINCT s.user_username as postino,count(DISTINCT barcode) as scansioni,
DATE(s.delivery_created_date) as giorno
  FROM `database-dev-332416.database_prod.product`as p,
```

```
    UNNEST(state) AS s
    WHERE s.name = 'CONSEGNA' and s.delivery_send_state = 'SENT'
GROUP BY postino, giorno
ORDER BY scansioni DESC;
```

2.6.1 Daily scans per mail carrier in descending order

	Postino	Scansioni	Giorno
1	lambda	18921	2021-03-15
2	lambda	17739	2021-03-16
3	lambda	15772	2021-03-19
4	lambda	14293	2021-03-17
5	lambda	13953	2021-03-23
6	lambda	13923	2021-03-18
7	NI	13558	2021-09-27
8	lambda	12238	2021-03-22
9	lambda	11935	2021-03-12
10	lambda	11608	2021-03-24

62572 results

Since these high results were mainly related to an earlier period (mostly March and April 2020), I wanted to verify if the situation improved more recently by repeating the query but limiting it first to everything that was delivered from September to today, and then everything that was delivered from November to today, to verify if the patch released had improved the situation.

```
SELECT DISTINCT s.user_username as postino,count(DISTINCT barcode) as scansioni,
DATE(s.delivery_created_date) as giorno
   FROM `database-dev-332416.database_prod.product`as p,
      UNNEST(state) AS s
      WHERE s.name = 'CONSEGNA' and s.delivery_send_state = 'SENT'
```

```
    AND date(s.created_date) >= '2021-09-01'
GROUP BY postino, giorno
ORDER BY scansioni DESC;
```

2.6.2 Daily scans per mail carrier in descending order after September

	Postino	Scansioni	Giorno
1	NI	9435	2021-09-27
2	ZETA	4309	2021-09-22
3	ETA	4205	2021-09-13
4	THETA	3887	2021-09-14
5	IOTA	3877	2021-09-17
6	kappa	3790	2021-09-29
7	Ksi	3735	2021-09-22
8	OMICRON	3732	2021-09-15
9	XI	3700	2021-09-13
10	PI	3637	2021-09-12

8370 results

The situation improved from September onwards, but not as much as I had hoped.

Even more surprising was the result obtained in the following query, referring to the period from November onwards, in which, as I had mentioned in the previous paragraph, the numbers are drastically reduced. It's worth noting that only 41 rows were returned.

```
SELECT DISTINCT s.user_username as postino,count(DISTINCT barcode) as scansioni,
DATE(s.delivery_created_date) as giorno
  FROM `database-dev-332416.database_prod.product`as p,
    UNNEST(state) AS s
```

```
    WHERE s.name = 'CONSEGNA' and s.delivery_send_state = 'SENT'
    AND date(s.created_date) >= '2021-11-01'
GROUP BY postino, giorno
ORDER BY scansioni DESC;
```

2.6.3 Daily scans per mail carrier in descending order after November

	Postino	Scansioni	Giorno
1	CHI	5	2021-11-11
2	PSI	2	2021-12-03
3	OMEGA	2	2021-12-20
4	ALPHA	1	2021-11-03
5	BETA	1	2021-11-11
6	OMEGA	1	2021-11-16
7	GAMMA	1	2021-11-22
8	DELTA	1	2021-11-23
9	EPSILON	1	2021-11-25
10	ALPHA	1	2021-11-25

41 results

To better understand the issue, I decided to delve further into the details by selecting some of the mail carriers with the highest number of scans.
For a start, I wanted to see if all their daily scans were as high and I wanted to view them from the most recent to the oldest.

```
SELECT date(s.delivery_created_date) as giorno, count(DISTINCT id) as scansioni,
    FROM `database-dev-332416.database_prod.product`AS p,
        UNNEST(state) AS s
        where s.user_username = 'lambda'
    GROUP BY giorno
    ORDER BY giorno DESC;
```

2.6.4 Daily scans of the postman lambda

	Giorno	Scansioni
1	2021-04-24	1962
2	2021-04-23	1168
3	2021-04-22	4015
4	2021-04-21	4650
5	2021-04-20	3377
6	2021-04-19	4742
7	2021-04-17	563
8	2021-04-16	3254
9	2021-04-15	9654
10	2021-04-14	5174

45 results

```
SELECT date(s.delivery_created_date) as giorno, count(DISTINCT id) as scansioni,
  FROM `database-dev-332416.database_prod.product`AS p,
    UNNEST(state) AS s
    where s.user_username = 'NI'
  GROUP BY giorno
  ORDER BY giorno DESC;
```

2.6.5 Daily scans of the postman Ni

	Giorno	Scansioni
1	2021-12-31	3
2	2021-12-28	6
3	2021-11-25	500
4	2021-11-24	1
5	2021-11-22	4
6	2021-11-21	498
7	2021-11-20	5
8	2021-11-19	1
9	2021-10-24	500
10	2021-10-21	1500

47 results

I also decided to check the weekly and monthly scans for each mail carrier, in case the scans were all uploaded in one day but performed over multiple days. However, the result clearly shows that this is not the case, and that the excessively high scan numbers are maintained for every day of the week. Even in the analysis over a month, the numbers only slightly decrease. However, in the latter case, once the first excessive results are removed, the others seem plausible.

```
SELECT EXTRACT(ISOWEEK FROM s.delivery_created_date) as settimana, count(DISTINCT id) as scansioni, s.user_username as postino
  FROM `database-dev-332416.database_prod.product`AS p,
    UNNEST(state) AS s
    where s.name = 'CONSEGNA' and s.delivery_send_state = 'SENT'
    GROUP BY settimana, postino
    ORDER BY scansioni DESC;
```

2.6.6 Weekly scans per mailman

	Settimana	Scansioni	Postino
1	11	87008	lambda
2	12	54448	lambda
3	10	46363	lambda
4	13	37359	lambda
5	15	25216	lambda
6	16	19913	lambda
7	14	15545	lambda
8	39	13558	NI
9	24	12484	XI
10	16	12207	XI

18209 results

```
SELECT EXTRACT(MONTH FROM s.delivery_created_date) as mese, count(DISTINCT id) as scansioni, s.user_username as user
  FROM `database-dev-332416.database_prod.product`AS p,
    UNNEST(state) AS s
```

```
where s.name = 'CONSEGNA' and s.delivery_send_state = 'SENT'
GROUP BY mese, user
ORDER BY scansioni DESC;
```

2.6.7 Monthly scans per mailman

	Postino	Scansioni	Giorno
1	3	208120	lambda
2	4	78572	lambda
3	5	29839	XI
4	6	27958	XI
5	4	25688	XI
6	3	24991	mi
7	9	23843	THETA
8	3	20728	Upsilon
9	2	20677	mi
10	8	20483	THETA

5909 results

Next, I wanted to inspect in detail the scans made based on a single date and a single username.

```
SELECT  *
  FROM `database-dev-332416.database_prod.product`AS p,
    UNNEST(state) AS s
    WHERE s.user_username = 'NI'
    AND DATE(s.delivery_created_date) = '2021-10-21'
  ORDER BY p.created_date DESC;
```

Below I report only a part of the result, since the query returned all the fields of the database and I wouldn't be able to show them all here.

2.6.8 Expanded daily scans of the postman Ni

	id	barcode	State.created_date	State.user_username
1	16508184	FGENL47129061002245 20210927	2021-09-20T09:49:17	Ni
			2021-09-20T09:49:17	Ni
			2021-09-20T09:49:17	Ni
			2021-09-20T09:49:17	Ni
2	16508184	FGENL47129061002245 20210927	2021-09-20T09:49:17	Ni
			2021-09-20T09:49:17	lambda
			2021-09-20T09:49:17	phi
			2021-09-20T09:49:17	Ni
3	16508181	FGENL47129061002244 20210927	2021-09-20T09:49:16	lambda
			2021-09-20T09:49:16	Ni

3000 results

The query as formulated extracted all the barcodes connected to the Ni postman, but also all the movements of these barcodes, even when another postman was involved. For this reason, I also wanted to verify how often the name of the Ni postman was displayed.

```
SELECT COUNT(s.user_username) as frequenza_postino
  FROM `database-dev-332416.database_prod.product`AS p,
    UNNEST (state) AS s
    WHERE s.delivery_created_date = '2021-05-02T20:39:02.460000'
    AND  s.user_username = 'NI';
```

2.6.9 Postman frequency in the result

Postman frequency
874

I repeated this check for various combinations of postmen and dates, and I noticed the presence of duplicate data among the results. This made me suspicious of the real cause of these high numbers, and I searched for the cause of these duplications, which led me to the UNNEST problem.

2.7 The UNNEST problem

As seen so far, to access the content of the nested folder 'state' I used the UNNEST(state) formula: in fact, the UNNEST command was the only method I was able to use to access the content of the nested table. However, the problem with UNNEST is that it creates a cross join with the main

table, thus duplicating the data. This duplication could falsify the results and therefore be the cause of the excessive scan numbers that I have found so far.
How to overcome this problem? I spent a lot of time searching for a possible solution.

My first attempt was to use CONCAT to connect the day and the ID, thus creating a combined identification key that would allow individual values to repeat, but not the combination. As can be seen from the results, this method was not effective, and furthermore, I realized that, even in the best-case scenario, it would still exclude all possible second delivery attempts made in a day from the search, thus returning a falsified result.

```
SELECT COUNT(concon) as scansioni, giorno, postino
FROM (
   SELECT DISTINCT(CONCAT(giorno, barcode)) as concon, giorno, barcode, postino
   FROM (
      SELECT barcode, date(s.created_date) as giorno, s.user_username as postino
      FROM `database-dev-332416.database_prod.product`as p,
      UNNEST(state) AS s
      WHERE s.name = 'CONSEGNA'
   ) as test
) as res
group by giorno, postino
order by scansioni DESC;
```

2.4.5 Daily scans per postman with Concat

	Scans	Day	Mailman
1	21224	2021-03-16	lambda
2	18210	2021-03-15	lambda
3	15780	2021-03-18	lambda
4	14437	2021-03-17	lambda
5	12571	2021-03-22	lambda
6	12379	2021-03-23	lambda
7	12370	2021-03-12	lambda
8	11281	2021-03-19	lambda
9	11065	2021-05-15	XI
10	10579	2021-03-24	lambda

73464 results

The most successful attempt consisted of a simpler and more effective solution: I created a subquery where I wanted to see state.created_date and state.user_username, grouping them by created_date and user_username. Then, in the main query, I wanted to see state.created_date and then do a count on the same data: this, in fact, since it is mandatory and indicates every movement of the letter, can easily be used to track the number of scans.

```
SELECT DATE(res.created_date) AS giorno , COUNT(res.created_date) AS scansioni, res.user_username
AS postino
FROM
(
   SELECT  s.created_date, s.user_username
   FROM `database-dev-332416.database_prod.product`AS p,
   UNNEST(state) AS s
   WHERE s.name = 'CONSEGNA'
   GROUP by s.created_date, s.user_username
) AS res

GROUP BY giorno, postino
ORDER BY scansioni DESC;
```

2.7.2 Daily scans per postman with a subquery

	Day	Scans	Mailman
1	2021-03-16	13097	lambda
2	2021-03-15	12311	lambda
3	2021-03-18	10607	lambda
4	2021-03-17	10096	lambda
5	2021-03-23	9554	lambda
6	2021-03-22	9232	lambda
7	2021-03-12	9113	lambda
8	2021-05-15	8957	XI
9	2021-03-19	8349	lambda
10	2021-03-24	8258	lambda

73464 results

Unfortunately, the results continue to be unsatisfactory, but they show an improvement starting from the month of September.

```
SELECT DATE(res.created_date) AS giorno , COUNT(res.created_date) AS scansioni, res.user_username
AS postino
FROM
(
   SELECT  s.created_date, s.user_username
   FROM `database-dev-332416.database_prod.product`AS p,
   UNNEST(state) AS s
   WHERE s.name = 'CONSEGNA' AND s.created_date >= '2021-09-01'
   GROUP by s.created_date, s.user_username
) AS res

GROUP BY giorno, postino
ORDER BY scansioni DESC;
```

2.7.3 Daily scans per postman with a subquery, filtering results after september

	Day	Scans	Mailman
1	2021-09-23	3674	XI
2	2021-09-15	3187	OMICRON
3	2021-09-13	2824	XI
4	2021-09-23	2660	RHO
5	2021-09-14	2430	THETA
6	2021-09-21	2191	ZETA
7	2021-10-01	2108	TAU
8	2021-09-06	2098	XI
9	2021-09-08	2084	Upsilon
10	2021-09-06	2068	PSI

9728 results

Once again, however, the results for the period after the app patch are too low.

```
SELECT DATE(res.created_date) AS giorno , COUNT(res.created_date) AS scansioni, res.user_username
AS postino
FROM
(
   SELECT  s.created_date, s.user_username
   FROM `database-dev-332416.database_prod.product`AS p,
   UNNEST(state) AS s
   WHERE s.name = 'CONSEGNA' AND s.created_date >= '2021-11-01'
   GROUP by s.created_date, s.user_username
) AS res

GROUP BY giorno, postino
ORDER BY scansioni DESC;
```

2.7.4 Daily scans per postman with a subquery, filtering results after november

	Day	Scans	Mailman
1	2021-11-11	5	CHI
2	2021-12-10	2	OMEGA
3	2021-11-04	1	Alpha
4	2021-11-01	1	ALPHA
5	2021-11-02	1	BETA
6	2021-11-08	1	OMEGA
7	2021-11-22	1	GAMMA
8	2021-11-17	1	DELTA
9	2021-11-20	1	EPSILON
10	2021-11-18	1	ALPHA

40 results

2.8 Data cleaning

I was becoming more and more convinced at this point that the problem was with the data itself and that any result I could obtain with it would still be an unreliable result. However, spurred on by my advisor, I decided not to give up and make one last attempt in my analysis: I decided to calculate the quantity and percentage of days on which it appears that a single delivery person has made a number of scans between 10 and 600, and those that are outside this range.

```
SELECT totale, Sopra_600, Sotto_10 , Tra_10_e_600, ROUND(Sopra_600 * 100.0 / totale,2) AS
Sopra_600_percent, ROUND(Sotto_10 * 100.0 / totale,2) AS Sotto_10_percent, ROUND(Tra_10_e_600 *
100.0 / totale,2) AS Tra_10_e_600_percent
FROM (SELECT
    (
    SELECT COUNT(test.scansioni)
      FROM
      (SELECT giorno, scansioni, postino
      FROM
      (SELECT DATE(res.created_date) AS giorno , COUNT(res.created_date) AS scansioni,
res.user_username AS postino
      FROM
      (  SELECT  s.created_date, s.user_username
        FROM `database-dev-332416.database_prod.product`AS p,
        UNNEST(state) AS s
        WHERE s.name = 'CONSEGNA'
        GROUP by s.created_date, s.user_username
      ) AS res
      GROUP BY giorno, postino
      ORDER BY scansioni DESC)) as test
    )
        AS totale,

    (
    SELECT COUNT(test.scansioni)
      FROM
      (SELECT giorno, scansioni, postino
      FROM
      (SELECT DATE(res.created_date) AS giorno , COUNT(res.created_date) AS scansioni,
res.user_username AS postino
```

```
        FROM
        (   SELECT  s.created_date, s.user_username
            FROM `database-dev-332416.database_prod.product`AS p,
            UNNEST(state) AS s
            WHERE s.name = 'CONSEGNA'
            GROUP by s.created_date, s.user_username
        ) AS res
        GROUP BY giorno, postino
        ORDER BY scansioni DESC)
        WHERE scansioni >= 600) as test
    )
        AS Sopra_600,
```

continua

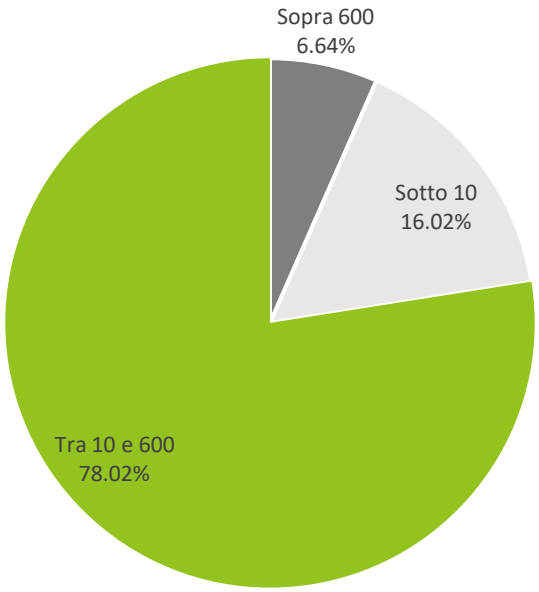
```
    (
    SELECT COUNT(test.scansioni)
        FROM
        (SELECT giorno, scansioni, postino
        FROM
        (SELECT DATE(res.created_date) AS giorno , COUNT(res.created_date) AS scansioni,
res.user_username AS postino
        FROM
        (   SELECT  s.created_date, s.user_username
            FROM `database-dev-332416.database_prod.product`AS p,
            UNNEST(state) AS s
            WHERE s.name = 'CONSEGNA'
            GROUP by s.created_date, s.user_username
        ) AS res
        GROUP BY giorno, postino
        ORDER BY scansioni DESC)
        WHERE scansioni <= 10) as test
    )
        AS Sotto_10,
```

```
(
SELECT COUNT(test.scansioni)
  FROM
  (SELECT giorno, scansioni, postino
  FROM
  (SELECT DATE(res.created_date) AS giorno , COUNT(res.created_date) AS scansioni,
res.user_username AS postino
  FROM
  (  SELECT  s.created_date, s.user_username
    FROM `database-dev-332416.database_prod.product`AS p,
    UNNEST(state) AS s
    WHERE s.name = 'CONSEGNA'
    GROUP by s.created_date, s.user_username
  ) AS res
  GROUP BY giorno, postino
  ORDER BY scansioni DESC)
  WHERE scansioni BETWEEN 10 AND 600) as test
  )
    AS Tra_10_e_600);
```

2.4.5 Comparison and percentage of daily scans for a mail carrier between 10 and 600, and those that fall outside this range

Totale	Sopra_600	Sotto_10	Tra_10_e_600
73464	4876	11766	57316

Sopra_600_percent	Sotto_10_percent	Tra_10_e_600_percent
6.64	16.02	78.02

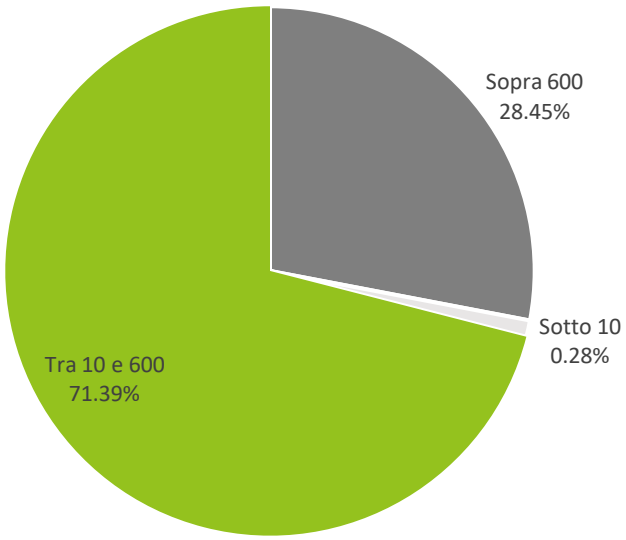


The results obtained in this way, using the subquery previously created, give a more positive picture than I would have imagined at this point. I repeated the query by doing a SUM instead of a COUNT, to see not only how many times the out-of-scale data was present, but also how much it amounted to, and once again the result I obtained pleasantly surprised me: despite the number of unusable data being not insignificant, it still constitutes the smaller part of the database, leaving a good 70% of useful data.

2.8.2 Comparison and percentage of out-of-scale scans.

Totale	Sopra_600	Sotto_10	Tra_10_e_600
14930960	4247380	41123	10659787

Sopra_600_percent	Sotto_10_percent	Tra_10_e_600_percent
28.45	0.28	71.39



For completeness, I decided to also analyze individual workers to track the general trend of their scans. I selected 5 of them, choosing among those with the highest overall number of scans and those who were active after September.

As can be seen in the first example reported (image 2.7.6), the results for the lambda mail carrier are significantly disproportionate, with an average that consistently remains well above two thousand daily scans. However, for the following four mail carriers, the situation improves: there are still some peaks of excessive scans, but they are more limited and the average seems to revolve around a more feasible range of results.

It should be noted that while the scans for the first mail carrier only refer to the months of March and April 2021, the other mail carriers were active in more recent periods.

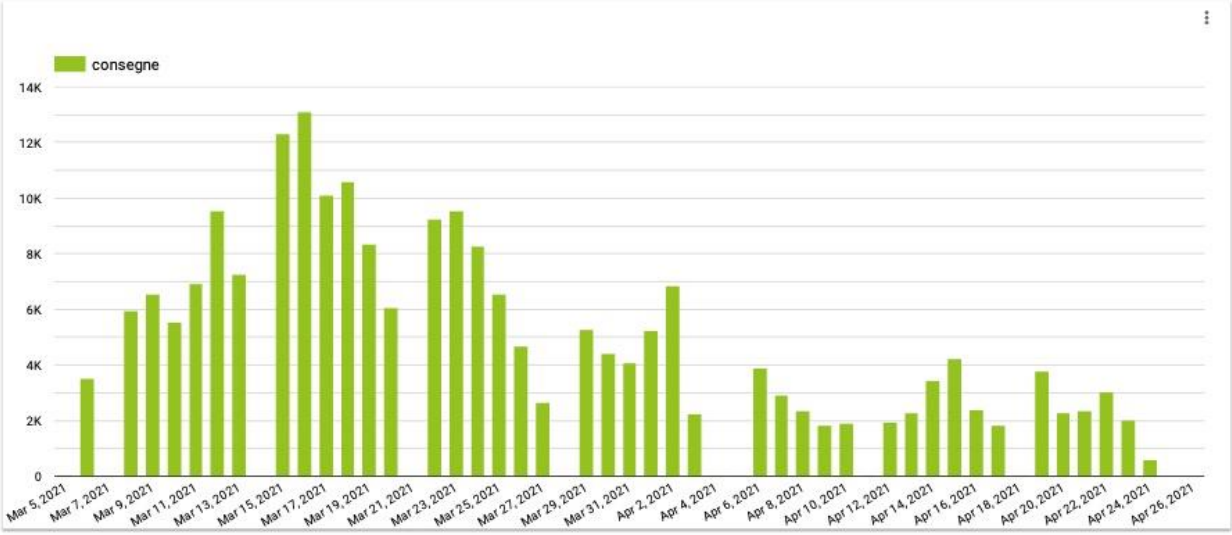
```
SELECT DATE(res.created_date) AS giorno , COUNT(res.created_date) AS scansioni, res.user_username
AS postino
FROM
(
   SELECT  s.created_date, s.user_username
   FROM `database-dev-332416.database_prod.product`AS p,
   UNNEST(state) AS s
   WHERE s.user_username = 'lambda'
```

```
    GROUP by s.created_date, s.user_username
) AS res

GROUP BY giorno, postino
ORDER BY scansioni DESC;
```

2.8.3 Daily scans of the lambda mail carrier + Histogram

	Day	Scans
1	2021-03-16	13097
2	2021-03-15	12311
3	2021-03-18	10607
4	2021-03-17	10096
5	2021-03-23	9554
6	2021-03-12	9530
7	2021-03-22	9232
8	2021-03-19	8349
9	2021-03-24	8258
10	2021-03-13	7260

44 results



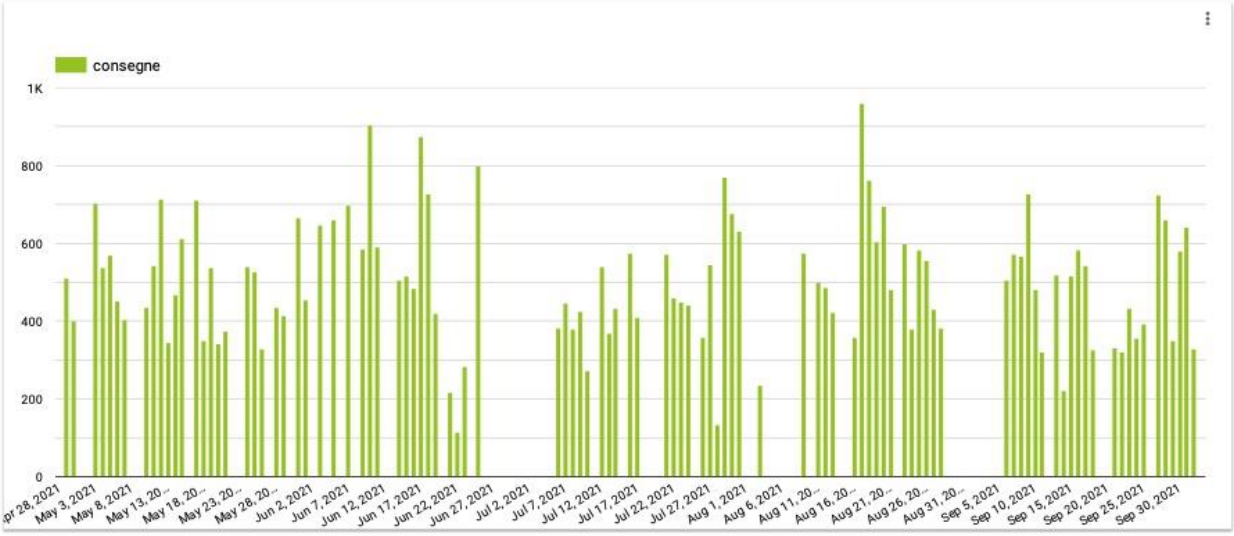

```
SELECT DATE(res.created_date) AS giorno , COUNT(res.created_date) AS scansioni, res.user_username
AS postino
FROM
(
```

```
    SELECT  s.created_date, s.user_username
    FROM `database-dev-332416.database_prod.product`AS p,
    UNNEST(state) AS s
    WHERE s.user_username = 'mi'
    GROUP by s.created_date, s.user_username
) AS res

GROUP BY giorno, postino
ORDER BY scansioni DESC;
```

2.8.4 Daily scans of the mi mail carrier + Histogram

	Day	Scans
1	2021-03-02	1395
2	2021-02-23	1306
3	2021-02-04	1262
4	2021-03-01	1221
5	2021-02-05	1177
6	2021-02-18	1172
7	2021-02-19	1136
8	2021-02-22	1071
9	2021-03-04	1068
10	2021-03-19	1066

166 results

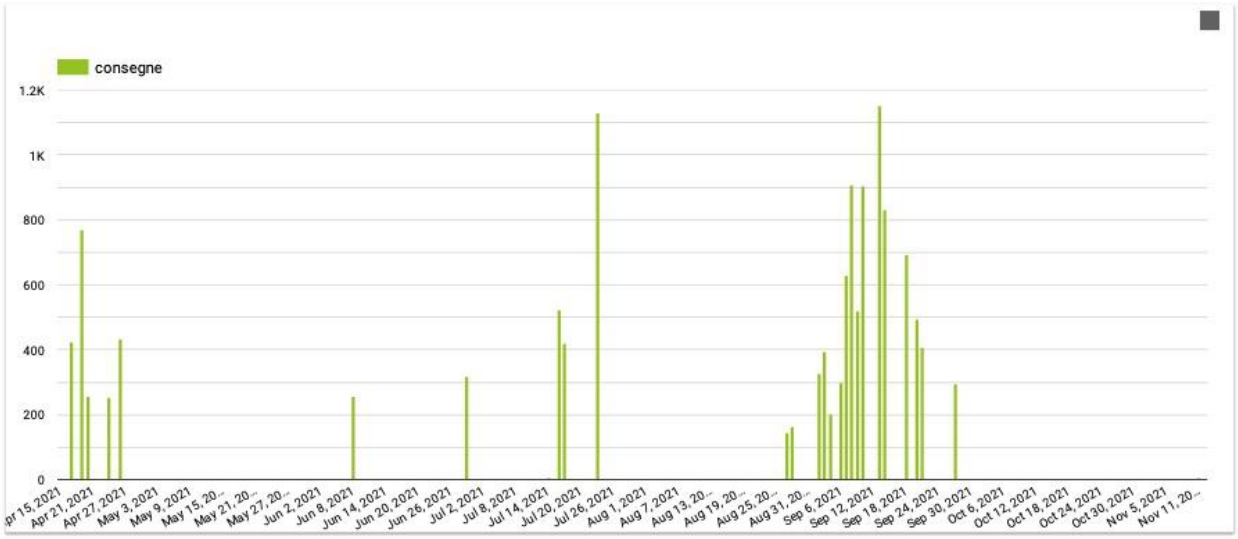


```
SELECT DATE(res.created_date) AS giorno , COUNT(res.created_date) AS scansioni, res.user_username
AS postino
FROM
(
   SELECT  s.created_date, s.user_username
   FROM `database-dev-332416.database_prod.product`AS p,
   UNNEST(state) AS s
   WHERE s.user_username = 'CHI'
   GROUP by s.created_date, s.user_username
) AS res

GROUP BY giorno, postino
ORDER BY scansioni DESC;
```

2.8.5 Daily scans of the chi mail carrier + Histogram

	Day	Scans
1	2021-09-13	1152
2	2021-07-23	1131
3	2021-09-08	907
4	2021-09-10	905
5	2021-09-14	830
6	2021-04-19	770
7	2021-09-18	692
8	2021-09-07	630
9	2021-07-16	524
10	2021-09-09	521

28 results



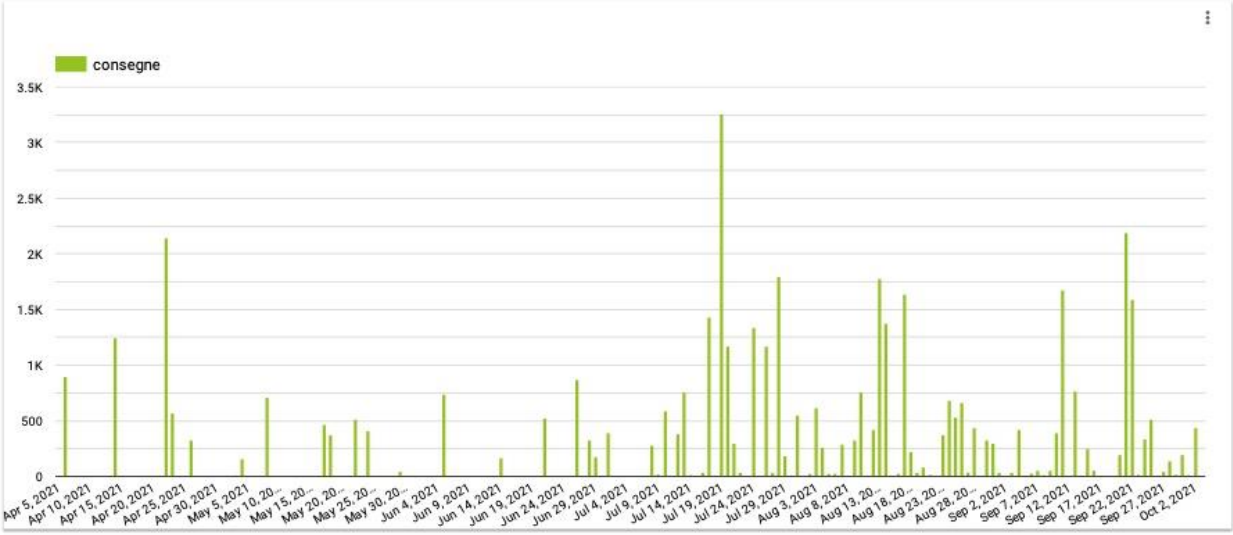

```
SELECT DATE(res.created_date) AS giorno , COUNT(res.created_date) AS scansioni, res.user_username
AS postino
FROM
(
   SELECT  s.created_date, s.user_username
   FROM `database-dev-332416.database_prod.product`AS p,
   UNNEST(state) AS s
   WHERE s.user_username = 'ZETA'
   GROUP by s.created_date, s.user_username
) AS res

GROUP BY giorno, postino
ORDER BY scansioni DESC;
```

2.8.6 Daily scans of the zeta mail carrier + Histogram

	Day	Scans
1	2021-07-19	3254
2	2021-09-21	2191
3	2021-04-22	2145
4	2021-07-28	1794
5	2021-08-13	1781
6	2021-09-11	1679
7	2021-08-17	1639

8	2021-09-22	1588
9	2021-07-17	1436
10	2021-08-14	1380

93 results



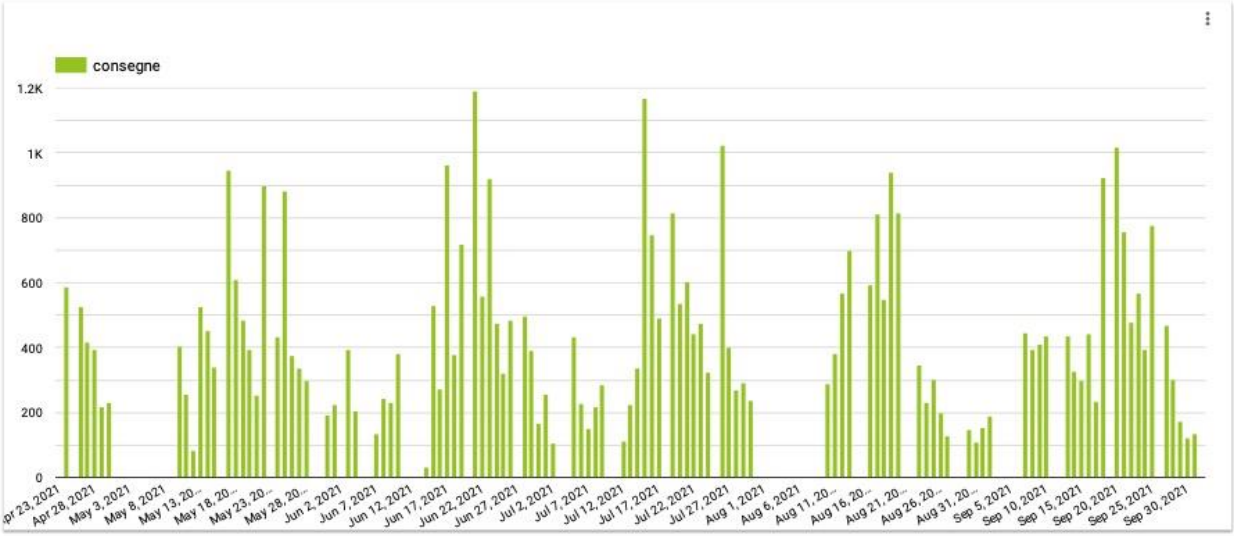

```
SELECT DATE(res.created_date) AS giorno , COUNT(res.created_date) AS scansioni, res.user_username
AS postino
FROM
(
   SELECT  s.created_date, s.user_username
   FROM `database-dev-332416.database_prod.product`AS p,
   UNNEST(state) AS s
   WHERE s.user_username = 'NI'
   GROUP by s.created_date, s.user_username
) AS res

GROUP BY giorno, postino
ORDER BY scansioni DESC;
```

2.8.7 Daily scans of the ni mail carrier + Histogram

	Day	Scans
1	2021-06-21	1191
2	2021-07-15	1169
3	2021-07-26	1024
4	2021-09-20	1016

5	2021-06-17	962
6	2021-05-17	948
7	2021-08-19	940
8	2021-09-18	924
9	2021-06-23	922
10	2021-05-22	898

111 results

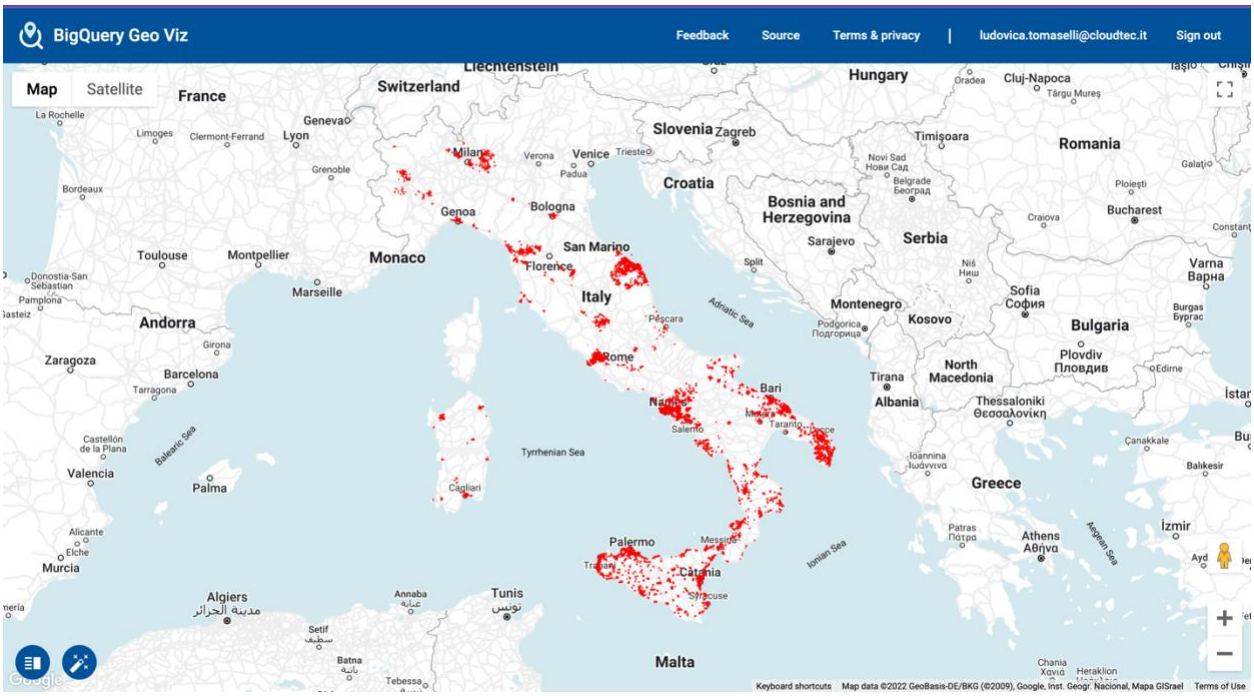


2.9 Mapping the data

Despite the difficulties, it would seem that the database data is, after all, usable and that the subquery workaround is effective in eliminating duplicates. At this point, I was also able to verify the geographic coordinates. First of all, I looked for a way to transform the latitude and longitude data, which BigQuery identified as FLOAT, into GEOGRAPHY data, that is, data that could be read as coordinates and analyzed on Geo Viz.

```
SELECT ST_GEOGPOINT(s.delivery_longitude, s.delivery_latitude) AS position
  FROM `database-dev-332416.database_prod.product`AS p,
    UNNEST(state) AS s
      WHERE s.delivery_latitude IS NOT NULL
      AND s.delivery_longitude IS NOT NULL
      AND s.name = 'CONSEGNA';
```

2.4.5 Mappatura di tutte le scansioni



Against all expectations, it will therefore be possible to work with this data, excluding from the analysis that approximate 30% that falls outside of the scale.

Chapter 3

Conclusions and Future Developments

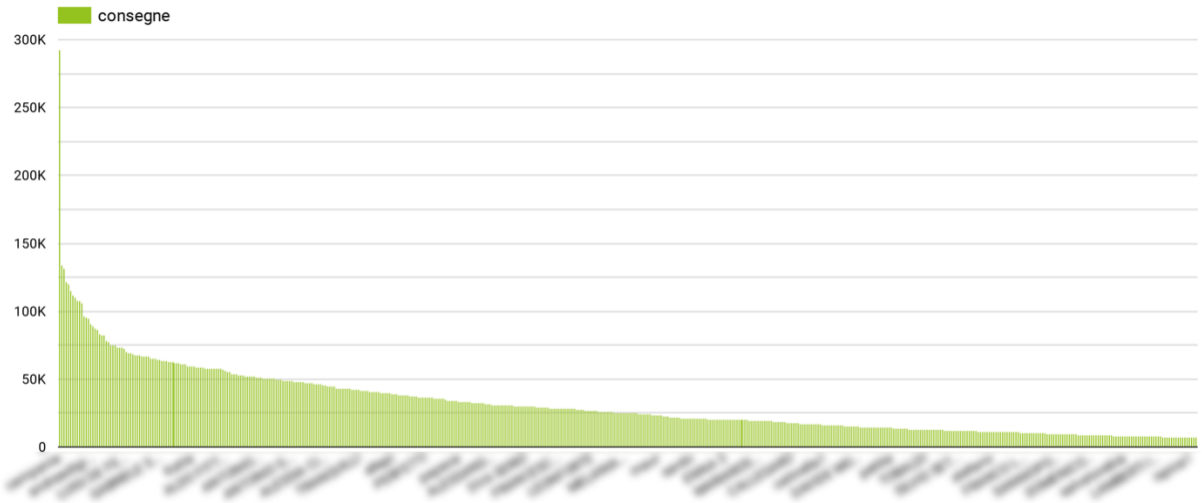
At the beginning of my internship, I was given a database of dubious quality, with the intention of discovering if the company could use the data it contained, even if only in part. It was the database of a postal company, and ideally the data was supposed to be used to track the movements of letters and mail carriers.

The database was uploaded to Google BigQuery and, to analyze it, I also used Google Data Studio and, to a lesser extent, Google BigQuery Geo Viz.

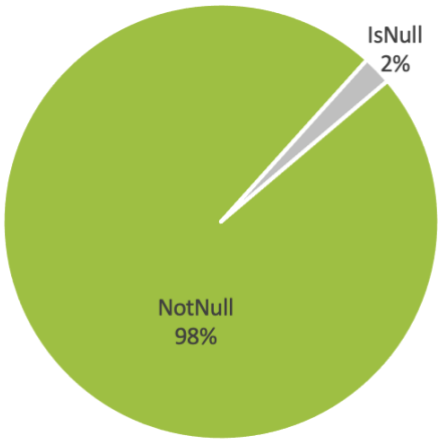
In the first chapter of this thesis, I briefly talked about these three services to give at least a general idea of what they are.

In the second chapter, I went into the details of my analysis: I started by showing the structure of the database and then moved on to illustrate the queries that allowed me to understand the data I was dealing with.
From the very beginning, I noticed the presence of anomalous data: the number of deliveries attributed to individual employees in just over a year reached frighteningly high peaks. Fortunately, however, these peaks occupied the smaller part of the database, as can be seen in the histogram created through Data Studio and reported below.

2.4.5 Total scans for each mail carrier - Histogram



Before delving into the nature of these peaks, I wanted to verify the presence of missing data. In particular, keeping in mind one of the ultimate objectives of this analysis, which was to discover if it would have been possible to track the movements of the mail carriers, I checked the percentage of missing data in the fields of latitude, longitude, delivery status, and delivery date.
The results obtained from this research were incredibly positive: latitude and longitude showed only 2% of missing data, and this was the highest number of missing data in this research, all the other fields were close to 0%.

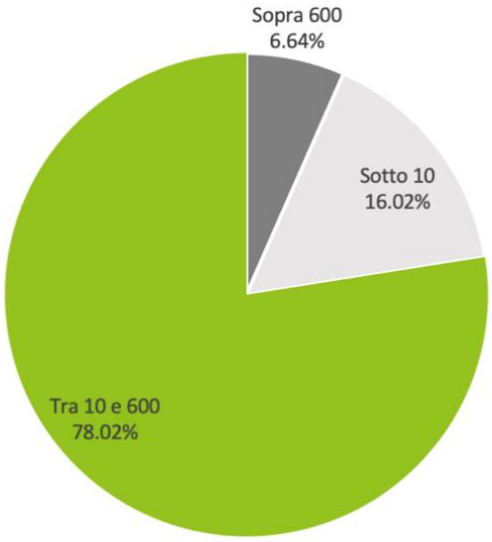


At this point, it was time to understand the cause of the excessively high results I had obtained: I queried the database multiple times to see the number of scans for each mail carrier divided by

day, week, and month. I individually analyzed some of the mail carriers, verifying the deliveries on individual days, and this made me notice the presence of duplicate data.
I researched the origin of these duplications and discovered a problem related to the 'UNNEST' command, a command used in BigQuery to read the contents of a nested folder but, unfortunately, creates a cross join that duplicates data.

I managed to find a formula to bypass this problem, using a subquery with a 'GROUP BY': this solved the problem of duplicate data. Unfortunately, despite some improvement, the peaks of excessive scans continued to be present and it was evident, at this point, that the problem was with the data itself.

Through Data Studio, however, I noticed that the excessive peaks were limited to only one part of the database. So, I wanted to investigate the amount of usable data, that is, what percentage of daily scans was between 10 and 600, an indicative number that I had obtained from the company itself.
The percentage of valid data turned out to be surprisingly high, amounting to as much as 78%, a remarkable result considering there was little hope of obtaining any usable data.



Thanks to my analysis, it will be possible to use these filtered data to carry out various analyses that will benefit the company: considering that these data had already been deemed unusable, I can conclude that this analysis has had a decidedly positive outcome.
The next goal will be to extract the routes of the mail carriers with the intention of improving them, and we will try to apply machine learning functions to calculate new routes that can optimize deliveries. It will be possible to verify delivery times, the most active areas, and evaluate the possible opening of new branches in these areas.
In addition, the growth trend can be analyzed and compared to the company's economic data to verify if the increase in revenue corresponds to an increase in activities

Bibliografy

Norman R. Draper, Harry Smith (1998)
Applied Regression Analysis,Wiley ed.

Rao, C Radhakrishna (1973)
Linear Statistical Inference and its Applications: Second Editon, Wiley ed.

P. McCullagh and J. A. Nelder, Generalized linear models, Monographs on Statistics and Applied Probability, Chapman & Hall, London, 1983.

Donald F. Morrison, Multivariate statistical methods, second ed., McGraw-Hill Book Co., New York, 1976, McGraw-Hill Series in Probability and Statistics.

Agresti, Alan (2021), Foundations of Statistics for Data Scientists: With R and Python, Chapman & Hall/CRC Texts in Statistical Science.