



# UNIVERSITÀ DEGLI STUDI DI PALERMO

Dipartimento di Scienze Economiche, Aziendali e Statistiche

Master annuale di secondo livello in

**Data Science and Big Data Analytics**

**Analisi e pulizia di un Database su BigQuery**

Tesi di:

Ludovica Tomaselli

Relatore:

Prof. Marcello Chiodi

Tutor aziendale:

Dott.ssa Francesca Motisi

---

Anno Accademico 2020/2021

# Tavola dei Contenuti

<b><i>Prefazione.....</i></b>	<b><i>3</i></b>
<b><i>Introduzione.....</i></b>	<b><i>5</i></b>
<b><i>1. Google Cloud Platform.....</i></b>	<b><i>6</i></b>
1.1 <i>Cos'è Google Cloud Platform .....</i>	<i>6</i>
1.2 <i>BigQuery .....</i>	<i>7</i>
1.3 <i>Data Studio .....</i>	<i>9</i>
1.4 <i>BigQuery Geo Viz .....</i>	<i>11</i>
<b><i>2. Il mio tirocinio.....</i></b>	<b><i>12</i></b>
2.1 <i>Il database .....</i>	<i>12</i>
2.2 <i>Spiegazione del database .....</i>	<i>14</i>
2.3 <i>Obiettivo .....</i>	<i>15</i>
2.4 <i>Analisi conoscitiva .....</i>	<i>16</i>
2.5 <i>Analisi dati mancanti .....</i>	<i>21</i>
2.6 <i>Analisi numero scansioni .....</i>	<i>28</i>
2.7 <i>Il problema dell'UNNEST.....</i>	<i>34</i>
2.8 <i>Pulizia dei dati .....</i>	<i>38</i>
2.9 <i>Mappare i dati .....</i>	<i>48</i>
<b><i>3. Conclusioni e Sviluppi Futuri .....</i></b>	<b><i>49</i></b>
<b><i>Bibliografia .....</i></b>	<b><i>52</i></b>

# Prefazione

Questa prefazione ha lo scopo di offrire, a chi leggerà questo elaborato, una contestualizzazione all'interno del mio percorso formativo e personale.

L'ambito da cui provengo non potrebbe essere più lontano da quello del master nel cui contesto si sviluppa questa tesi.

Sono laureata in lettere, specificamente nell'ambito cinematografico. Ho fatto la sceneggiatrice, la producer, la video editor, la youtuber, persino la startupper, e poi, grazie a quello che era nato come un hobby, mi sono ritrovata a lavorare come grafica. Ma come sono potuta arrivare ad interessarmi alla Data Science e ai Big Data?

C'è stato un insieme di motivazioni che mi hanno spinto in questa direzione, non ultima la difficoltà a trovare lavoro nel mio ambito di provenienza e la necessità di trovare qualcosa che potesse darmi un maggiore senso di stabilità.

Non sono mai stata brava in matematica, “i numeri” mi hanno sempre confuso per la loro estrema astrazione, ma tutto ciò che riguardava l'uso dei computer mi ha sempre affascinato. Da ragazzina avevo anche imparato per conto mio ad utilizzare l'HTML, e mi sono spesso divertita ad analizzare il codice di pagine web, a “giocarci” come con un puzzle... Nulla di serio, chiaramente, ma comunque è ciò che ha costituito i miei primi passi nel mondo dell'informatica, un mondo che trovo molto attraente ma forse un po' troppo vasto per qualcuno che, come me, non ha basi nell'ambito, né l'età per ripartire da zero.

Stavo valutando possibili ambiti più specifici all'interno di questo settore in cui focalizzare le mie energie, quando mi sono imbattuta in un libro che ha immediatamente catturato la mia attenzione: un libro sul Data storytelling; ho iniziato a sfogiarlo, e mi sono resa conto che tutte le tecniche che venivano descritte al suo interno per uno storytelling efficace, io le conoscevo già!

Questo libro ha dato una luce diversa alla matematica, d'improvviso i numeri non erano più noiose e sterili immagini, ma protagonisti di storie che andavano interpretate, tradotte e raccontate. È qualcosa che ho trovato incredibilmente affascinante, e c'era in più il vantaggio di poter portare nel mio nuovo ambito qualcosa del mio vecchio ambito, invece di scartarlo e ricominciare da zero. Era, per dirla con Massimo Troisi, la mia possibilità di ricominciare da tre.

Se il libro mi ha fatto scoprire quest'ambito, il tirocinio è senz'altro stato ciò che mi ha fatto veramente appassionare: avere la possibilità di mettere mano ad un vero database e fare una vera analisi con conseguenze reali è stato incredibilmente soddisfacente. A questo si aggiunge il valore

del potermi confrontare con gente che lavora nell'ambito e che mi ha aiutato e supportato nel mio percorso; è stato impagabile. Dulcis in fundo, ho anche provato la soddisfazione di essere riuscita a lavorare con un database alquanto problematico che perfino i miei colleghi avevano infine etichettato come "inutile", ma da cui, contro ogni aspettative, sono riuscita a ricavare dei buoni risultati, scoprendo, insieme alla fatica, il piacere del dialogo con i dati, della ricerca di dar loro un senso, della scoperta delle imperfezioni; del lavoro, dal sapore artigianale, della pulitura dei dati per far emergere la "storia" presente nei dati ripuliti da errori ed interferenze.

Sono contenta di aggiungere che la mia esperienza con questo progetto non si è ancora conclusa e che il mio tirocinio presso Cloudtec proseguirà anche fuori dal master.

*Last but not least*, vorrei ringraziare il prof. Chiodi per avermi incoraggiato ad iscrivermi a questo master, che è stata un'esperienza intensa e certamente non facile ma piena di soddisfazioni, per avermi seguito come mio relatore e per avermi incoraggiato a non darmi vinta davanti alle complicazioni di questo database.

Ringrazio tutti i docenti che ci hanno seguito e supportato nel master.

Ringrazio Francesco Bianco, che si è interessato al mio profilo e mi ha offerto la possibilità di svolgere il tirocinio presso Cloudtec, e tutto il team di Cloudtec che mi ha accolto caldamente. In particolare, ringrazio sentitamente Francesca Motisi, il mio tutor aziendale, che mi ha guidato e supportato durante il mio percorso, e Marco Barcellona, che ha pazientemente risposto a tutte le mie numerose domande sul database.

# Introduzione

Questa tesi di master è frutto del lavoro svolto presso l'azienda Cloudtec, una giovane startup innovativa di Palermo che si occupa di realizzare soluzioni sul cloud che sfruttano la potenza dell'intelligenza artificiale per innovare processi e servizi.

Nella tesi si sono affrontati alcuni problemi relativi all'utilizzabilità dei dati contenuti nel database di un'azienda privata di servizi postali. A questo scopo si sono utilizzati i servizi di Google Cloud Platform, specificamente BigQuery, Data Studio e BigQuery Geo Viz che servono rispettivamente per l'analisi di dati, la loro presentazione e la loro geolocalizzazione.

L'esigenza dell'azienda era quella di utilizzare i dati contenuti nel database per tracciare i movimenti dei postini e le loro consegne. Il database presentava, tuttavia, alcune problematiche tra cui spiccano la presenza di dati ripetuti e di frequenze anomale riguardanti le consegne per postino.

Il lavoro è pertanto iniziato con lo studio dei servizi di Google Cloud Platform con particolare riferimento a quelli inerenti al progetto in questione e si è successivamente rivolto innanzi tutto alla valutazione dell'utilizzabilità del database per le finalità proposte e, in un secondo momento, alla pulizia dei dati ed alla eliminazione dei dati "sporchi".

Il lavoro che segue è strutturato in tre capitoli: nel primo capitolo si affronta brevemente la natura e l'utilizzo dei servizi di Google Cloud Platform; nel secondo capitolo vengono descritte le procedure eseguite sul database, le analisi effettuate, le difficoltà e criticità riscontrate e le soluzioni trovate che rendono il database utilizzabile; nel terzo capitolo, infine, ho sviluppato le conclusioni e individuato alcune possibili linee di sviluppo per il prosieguo del progetto.

# Capitolo 1

## Google Cloud Platform (GCP)

Il mio tirocinio si è incentrato sull'analisi di un database attraverso alcuni degli strumenti offerti da Google Cloud Platform, in particolare BigQuery, Data Studio e BigQuery Geo Viz. Parte del mio tirocinio si è incentrata sullo studio di questi servizi per prendere familiarità e imparare a lavorarci. In particolare, ho seguito un corso su BigQuery in collaborazione tra Coursera e Google Cloud Training che si è focalizzato sugli aspetti principali del suo uso utilizzo, e poi un corso su Udemy riguardo Data Studio.

In questo capitolo spiegherò brevemente cosa sono Google Cloud Platform e i tre strumenti da me utilizzati.

### 1.1 Cos'è Google Cloud Platform

Google Cloud Platform (GCP) è un'insieme di servizi di cloud computing, introdotto da Google il 7 aprile 2008. GCP si serve della stessa infrastruttura usata dal resto dei prodotti di Google (i.e. il motore di ricerca Google, Youtube, Google Drive, etc). La lista dei servizi offerti è di lunghezza non indifferente, e si divide in varie categorie che includono, ma non si limitano a, AI e Machine Learning, Gestione delle API, Analisi di Dati, Database, Strumenti per sviluppatori, etc. Riporto di seguito alcuni degli esempi più rilevanti tra i servizi offerti:

- Compute Engine: Macchine virtuali in esecuzione nei data center di Google
- Anthos: Piattaforma per la modernizzazione delle app esistenti e la creazione di nuove
- Google Kubernetes Engine: Ambiente gestito per l'esecuzione di app containerizzate
- Cloud Storage: Archiviazione di oggetti sicura, durevole e scalabile
- BigQuery: Data Warehouse per aziende

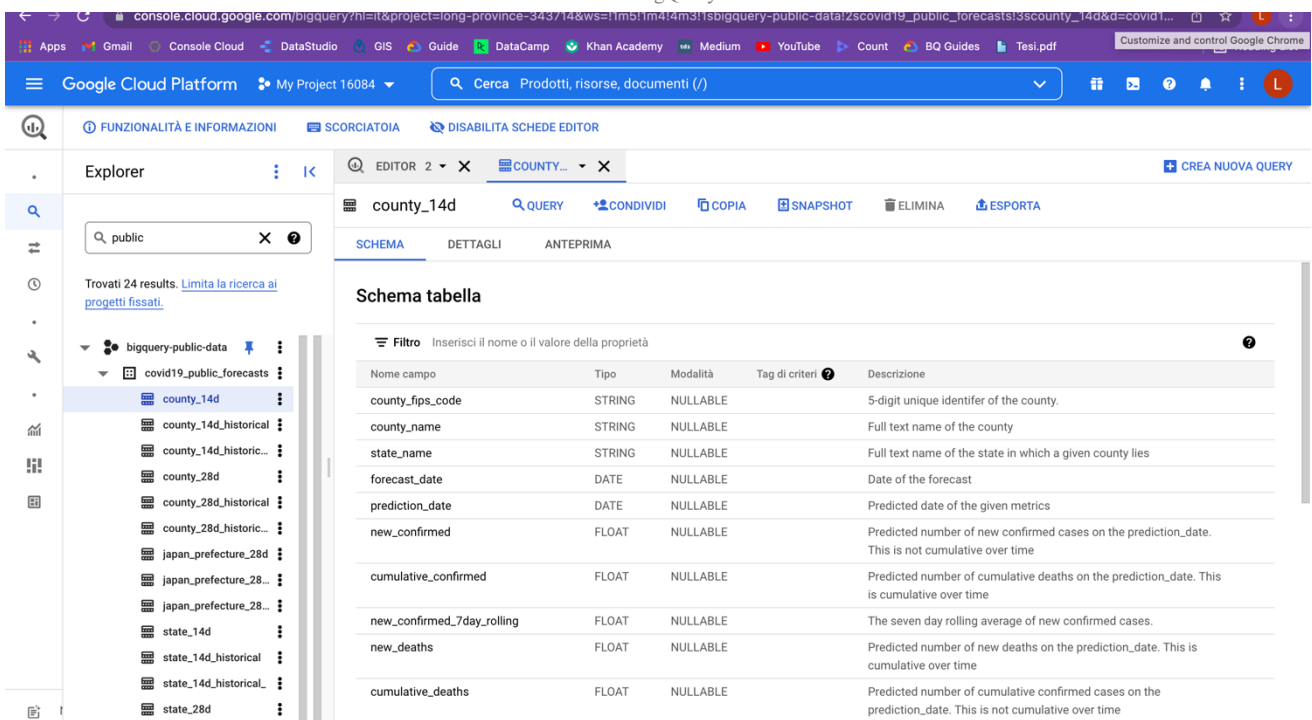
La gamma di servizi offerta è talmente ampia e copre talmente tanti ambiti che sarebbe impossibile e perfino inutile conoscerli tutti. Ci focalizzeremo, adesso, su quelli rilevanti per questo progetto.

## 1.2 BigQuery

BigQuery è stato messo a disposizione su GCP a partire da novembre 2011, si tratta di un servizio che permette un'analisi interattiva di grandi set di dati. BigQuery consente di interagire con facilità con i database caricati su Google Storage e, dal 2016, è stato aggiunto anche il supporto per Google Sheets.

Incredibilmente rapido, BigQuery riesce ad eseguire query di selezione o di raggruppamento su miliardi di dati in pochi secondi; altamente scalabile e serverless, rende la gestione e l'analisi di dati molto più semplice e diretta, senza bisogno di preoccuparsi di gestire l'infrastruttura. In più rende possibile condividere i dataset arbitrariamente con singoli utenti o gruppi, o perfino dando accesso pubblico.

### 1.2.1 Interfaccia BigQuery – Schema



The screenshot shows the Google Cloud Platform BigQuery interface. The left sidebar displays the Explorer view with a search for 'public' and a list of datasets under 'bigquery-public-data', including 'covid19\_public\_forecasts' and 'county\_14d'. The main panel shows the 'county\_14d' table selected, with the 'SCHEMA' tab active. The schema table lists the following fields:

Nome campo	Tipo	Modalità	Tag di criteri	Descrizione
county_fips_code	STRING	NULLABLE		5-digit unique identifier of the county.
county_name	STRING	NULLABLE		Full text name of the county
state_name	STRING	NULLABLE		Full text name of the state in which a given county lies
forecast_date	DATE	NULLABLE		Date of the forecast
prediction_date	DATE	NULLABLE		Predicted date of the given metrics
new_confirmed	FLOAT	NULLABLE		Predicted number of new confirmed cases on the prediction_date. This is not cumulative over time
cumulative_confirmed	FLOAT	NULLABLE		Predicted number of cumulative deaths on the prediction_date. This is cumulative over time
new_confirmed_7day_rolling	FLOAT	NULLABLE		The seven day rolling average of new confirmed cases.
new_deaths	FLOAT	NULLABLE		Predicted number of new deaths on the prediction_date. This is cumulative over time
cumulative_deaths	FLOAT	NULLABLE		Predicted number of cumulative confirmed cases on the prediction_date. This is not cumulative over time

BigQuery contiene metodi che permettono sia di creare, popolare e cancellare tabelle, sia di eseguire query sopra di esse, è inoltre possibile importare dati esterni al suo interno.

Cliccando sulla tabella desiderata, mostrata nel menu a sinistra, è possibile visualizzarne le informazioni: come mostrato nell'immagine, la prima cosa che viene mostrata è lo schema, e volendo si possono controllare anche i dettagli e l'anteprima della tabella.

Per interrogare un database su BigQuery, basta creare una nuova Query, inserire il codice nello spazio apposito e dare il comando “Esegui”. Inizialmente, BigQuery utilizzava un dialetto SQL non standard conosciuto come BigQuery SQL, ma con il lancio di BigQuery 2.0 è stato rilasciato il supporto per SQL standard, e il dialetto BigQuery SQL è stato rinominato come Legacy SQL. BigQuery supporta tutt’ora entrambi i dialetti, ma consiglia l’utilizzo di SQL standard. Posso anticipare che, nel mio tirocinio, ho utilizzato quest’ultimo.

### 1.2.2 Interfaccia BigQuery – Query

The screenshot displays the Google Cloud Platform BigQuery interface. The top navigation bar includes the Google Cloud Platform logo, the project name 'My Project 16084', a search bar, and various utility icons. The main interface is divided into three sections: Explorer, Editor, and Results.

**Explorer:** Shows a tree view of the project's data assets. The 'bigquery-public-data' folder is expanded, showing 'covid19\_public\_forecasts' and 'county\_14d'.

**Editor:** Displays the SQL query being executed: `SELECT * FROM `bigquery-public-data.covid19_public_forecasts.county_14d` LIMIT 1000`. The query is saved as 'QUERY N...3'.

**Results:** Shows the execution status and the resulting data table. The query completed successfully in 0.3 seconds, processing 302.9 kB of data. The results table has 10 columns: 'Riga', 'county\_fips\_code', 'county\_name', 'state\_name', 'forecast\_date', 'prediction\_date', 'new\_confirmed', 'cumulative\_confirmed', 'new\_confirmed\_7day\_rolling', and 'new\_dea'.

Riga	county_fips_code	county_name	state_name	forecast_date	prediction_date	new_confirmed	cumulative_confirmed	new_confirmed_7day_rolling	new_dea
1	55001	Adams	Wisconsin	2022-02-05	2022-01-31	null	null	55.57142857142857	
2	48005	Angelina	Texas	2022-02-05	2022-01-31	null	null	58.42857142857143	
3	08009	Baca	Colorado	2022-02-05	2022-01-31	null	null	5.142857142857143	
4	20011	Bourbon	Kansas	2022-02-05	2022-01-31	null	null	29.428571428571427	
5	13039	Camden	Georgia	2022-02-05	2022-01-31	null	null	108.42857142857143	
6	42027	Centre	Pennsylvania	2022-02-05	2022-01-31	null	null	153.42857142857142	
7	22027	Claiborne	Louisiana	2022-02-05	2022-01-31	null	null	2.142857142857143	
8	17029	Coles	Illinois	2022-02-05	2022-01-31	null	null	106.71428571428571	

The bottom of the interface shows pagination controls (1 - 100 di 1000) and tabs for 'CRONOLOGIA PERSONALE', 'CRONOLOGIA PROGETTO', and 'QUERY SALVATE'.



## 1.3 Data Studio

Google Data Studio è uno strumento gratuito di visualizzazione di dati che permette di creare rapidamente presentazioni chiare e dalla grafica apprezzabile. Lanciato da Google nel 2016, ha la caratteristica di essere facile da usare, da personalizzare e da condividere. Un altro punto di forza di Data Studio sta nel fatto che si collega a tantissime origini dati attraverso la presenza di connettori; ce ne sono di tre tipi: Google connectors, Partners connectors e Open Source connectors. Google connectors e Partners connectors sono pre-integrati in Data Studio, mentre gli Open Source connectors sono di proprietà di terze parti e hanno bisogno di integrazioni per poter procedere all'importazione.

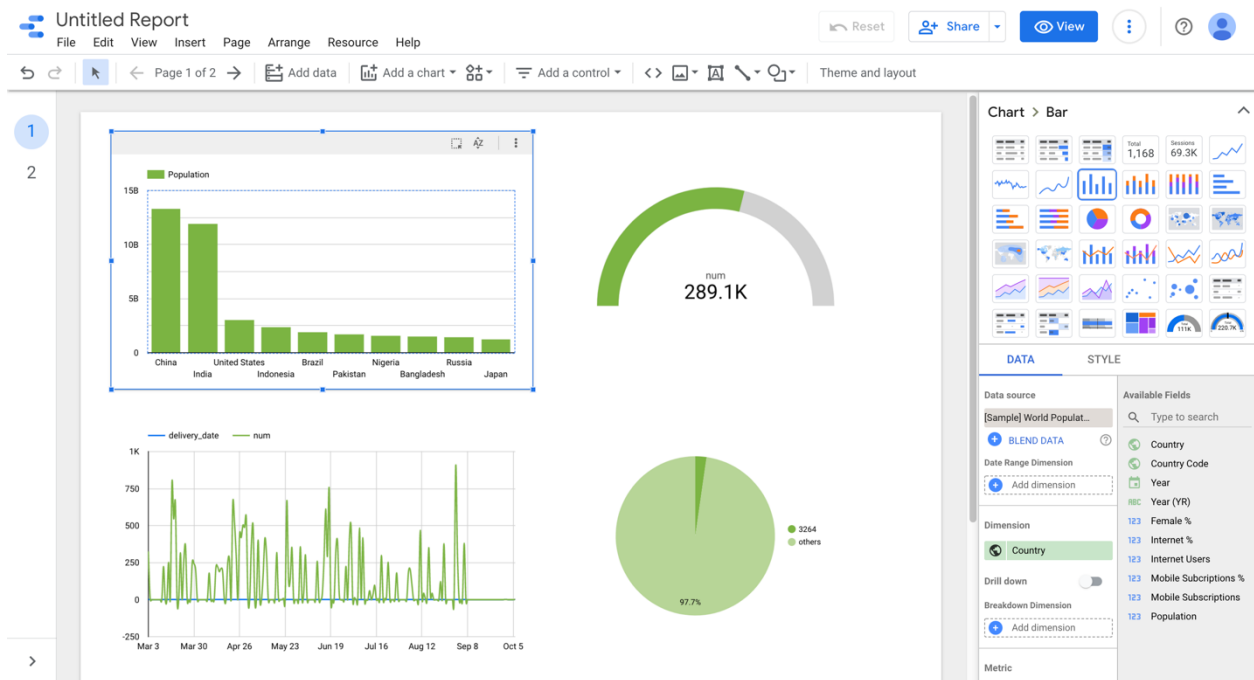
### 1.3.2 Homepage Data Studio

Name	Owned by anyone	Last opened by me
Untitled Report	Ludovica Tomaselli	4:37 PM
Untitled Report	Ludovica Tomaselli	Mar 10, 2022
Untitled Report	Ludovica Tomaselli	Feb 26, 2022
Untitled Report	Ludovica Tomaselli	Feb 24, 2022
Untitled Report	Ludovica Tomaselli	Feb 18, 2022
Untitled Report	Ludovica Tomaselli	Feb 16, 2022

Una volta che un'origine dati (i.e. Bigquery, Google Sheets, google analytics, o anche documenti .csv o .xls, etc.) è collegata a Data Studio, I dati vengono importati automaticamente nel tool e trasformati nelle metriche e dimensioni che si vogliono rappresentare all'interno del report. Per di più, le dashboard create sono dinamiche, (ovviamente se non si tratta di un documento caricato manualmente), cioè I dati al loro interno si aggiornano in automatico.

L'interfaccia di Data Studio è molto intuitiva e di facile utilizzo e funziona in modo molto simile a powerpoint: per creare un nuovo grafico, basta cliccare sull'apposita icona, decidere che tipo di grafico si vuole realizzare (contando che la tipologia può anche essere modificata in un secondo momento), posizionarlo nella pagina e modificarlo a piacimento.

### 1.3.2 Interfaccia Data Studio



Per caricare i dati, sarà sufficiente cliccare nello spazio apposito “Data Source” e seguire il procedimento per importare o collegare i dati. Nello stesso spazio, più in basso, si potranno dare tutti i parametri che si vogliono mostrare nel grafico, e la sezione “Style” permette di personalizzarne la grafica.

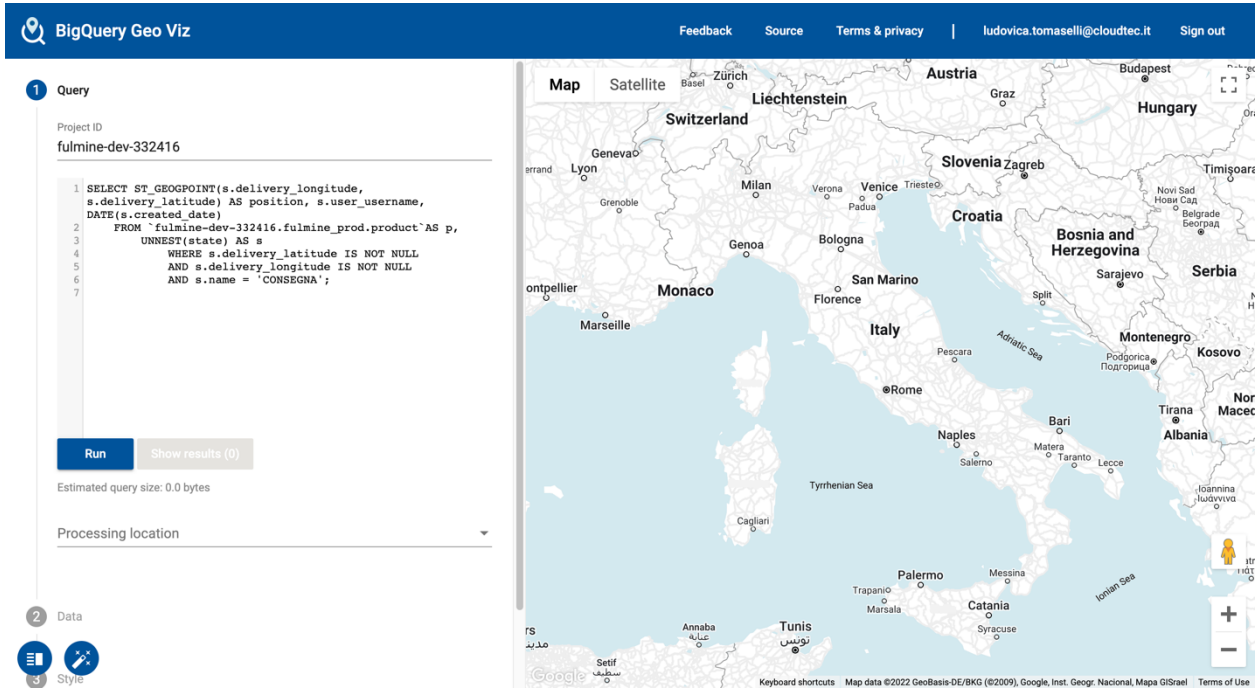
Non c'è limite al numero di grafici che si possono inserire in una sola pagina, è possibile creare documenti di più pagine, c'è la possibilità di scrivere, creare forme, inserire immagini, insomma tutto ciò che può servire per creare report efficienti, chiari e, perché no? Anche di bell'aspetto. Molto importante sottolineare che i documenti così realizzati possono essere scaricati come pdf o condivisi con terzi.

Voglio inoltre sottolineare che tutti i grafici riportati nel secondo capitolo sono stati realizzati utilizzando Data Studio.

## 1.4 BigQuery Geo Viz

BigQuery Geo Viz è, tra i tre strumenti, quello che ho utilizzato di meno, ma è, al tempo stesso, anche il più semplice e diretto. Come si evince dal nome, è direttamente collegato a BigQuery e sfrutta le API di Google Maps per visualizzare dati geospaziali. Non si tratta di uno strumento di analisi geospaziale completo e ha molte limitazioni, ma resta comunque un utile supporto per visualizzare i risultati di una query di analisi geospaziale su una mappa, una query alla volta.

17.672.87 Interfaccia BigQuery Geo Viz



# Capitolo 2

## Il mio tirocinio

Ho svolto il mio tirocinio presso l'azienda Cloudtec di Palermo, una giovane Startup Innovativa partner di Google Cloud che si occupa di migrazione cloud, intelligenza artificiale e analisi dei dati. Voglio specificare che il mio tirocinio presso di loro sta correntemente continuando, e questo progetto non è da considerarsi concluso.

Il mio lavoro si è focalizzato sul database di un'azienda postale privata, la cui identità non rivelerò per ragioni di privacy. Il database in questione si è rivelato, sin da subito, piuttosto problematico e con una forte presenza di dati sporchi, al punto che avevo quasi perso le speranze di riuscire a ricavarne dei dati utili ma, grazie all'incoraggiamento del mio relatore, ho perseverato nella mia analisi e ho scoperto che, eliminando la relativamente piccola percentuale di dati fuori scala, il resto era perfettamente utilizzabile.

In questo capitolo, illustrerò le varie tappe del mio percorso.

### 2.1 Il database

Non mi è stato possibile risalire alla data esatta della creazione del database, in quanto la persona che lo ha progettato non fa più parte del team, ma la prima entry risale al 14 dicembre 2020. È stato creato e viene tutt'ora gestito con MySQL8 ma, per evitare di sovraccaricare il server con un'eccessiva mole di dati, ogni tre mesi questi vengono cancellati localmente. Ne viene conservata una copia su BigQuery in Google Cloud Platform.

Ci sono state alcune problematiche con questo database, quella principale (tra quelle note) consiste in una duplicazioni di dati attribuibile all'app. Sono state rilasciate varie patch per ovviare al problema, l'ultima risalente a novembre 2021.

Di seguito, riporto la struttura del database.

2.4.5 Struttura dal database

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

## 2.2 Spiegazione del database

Si tratta di un'unica tabella con all'interno una partizione.

Lo schema è stato ideato in questo modo con l'obiettivo di tracciare con facilità l'intero percorso fatto da ogni singola lettera, dall'accettazione alla consegna.

Espongo di seguito i singoli campi:

2.4.5 Esposizione singoli campi del database

<b>ID</b>	Codice identificativo di ogni lettera ricevuta, meno affidabile del barcode dato che può essere mancante.
<b>CREATED_DATE</b>	Data di ricezione della lettera.
<b>BARCODE</b>	Si riferisce al barcode della lettera ed è il codice identificativo primario.
<b>AGENCY_ID</b>	Codice identificativo dell'agenzia che ha accettato la lettera.
<b>AGENCY_NAME</b>	Il nome dell'agenzia che ha accettato la lettera o pacco.
<b>NOTE</b>	Eventuali note.
<b>RECIPIENT_TYPE</b>	Ruolo di chi riceve la lettera (e.g. destinatario, parente, delegato, ente, etc).
<b>DOCUMENT_TYPE</b>	Tipo di documento mostrato alla ricezione (se necessario).
<b>DOCUMENT_NUMBER</b>	Codice di documento mostrato alla ricezione (se necessario).
<b>PRODUCT_TYPE</b>	Tipo di lettera (sdoc, parcella, raccomandata)

16.748.077 righe

Mentre la tabella principale contiene i dati generali riferiti alla lettera, la tabella STATE ne identifica i movimenti. Al suo interno si registrano, per ogni riga, l'accettazione e, a seguire, i diversi tentativi di consegna.

2.2.2 Esposizione singoli campi del database nella tabella annidata

↓ STATE	
NAME	Identifica se la lettera è in accettazione o in consegna
SEGNCOD	Identifica la modalità di consegna o la ragione di una mancata consegna
USER_ID	Codice identificativo del postino
USER_USERNAME	Nome identificativo del postino
CREATED_DATE	Data del movimento indicato nella riga (accettazione o tentativo di consegna)
DELIVERY_SEND_STATE	Identifica se la lettera è stata inviata o se si è verificato un errore
DELIVERY_CREATED_DATE	Data in cui la lettera è stata consegnata
DELIVERY_LATITUDE	I due ambiti, latitudine e longitudine, si riferiscono alla posizione della consegna
DELIVERY_LONGITUDE	

17.672.087 righe

## 2.3 Obiettivo

Lo scopo della mia analisi era di verificare se il database, che non era stato finora utilizzato né propriamente visionato, fosse effettivamente utilizzabile e se i dati contenuti in esso fossero affidabili. In particolare mi interessava comprendere se sarebbe stato possibile tracciare i movimenti dei postini.

Di seguito illustrerò le fasi della mia analisi.

## 2.4 Analisi conoscitiva

Per cominciare, ho fatto delle query per conoscere il contenuto del database e potermi quindi approcciare meglio. Ho iniziato con il ricercare gli username di tutti i postini e contarli.

Negli esempi a seguire, mostrerò il codice da me scritto e i primi 10 risultati restituiti da BigQuery a titolo esemplificativo (fatta eccezione per i casi in cui il risultato è inferiore a 10), i risultati sono sempre in ordine discendente (a meno che non specifichi diversamente), quindi i primi risultati sono sempre quelli più estremi. Specifico anche che tutti i nominativi dei postini sono stati sostituiti da nomi di fantasia, per tutela della privacy.

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

2.4.1 Numero totale postini

Totale postini
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 Elenco Postini

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

1.261 righe



A seguire ho fatto una verifica di tutti i giorni in cui sono state fatte scansioni, da cui sono risultati 361 giorni di attività.

```
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 Elenco giorni lavorativi

	Giorno lavorativo
1	<i>null</i>
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 righe

Ho quindi verificato il totale di scansioni fatte da ogni fattorino, mostrando per primi i risultati più alti

```
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;
```

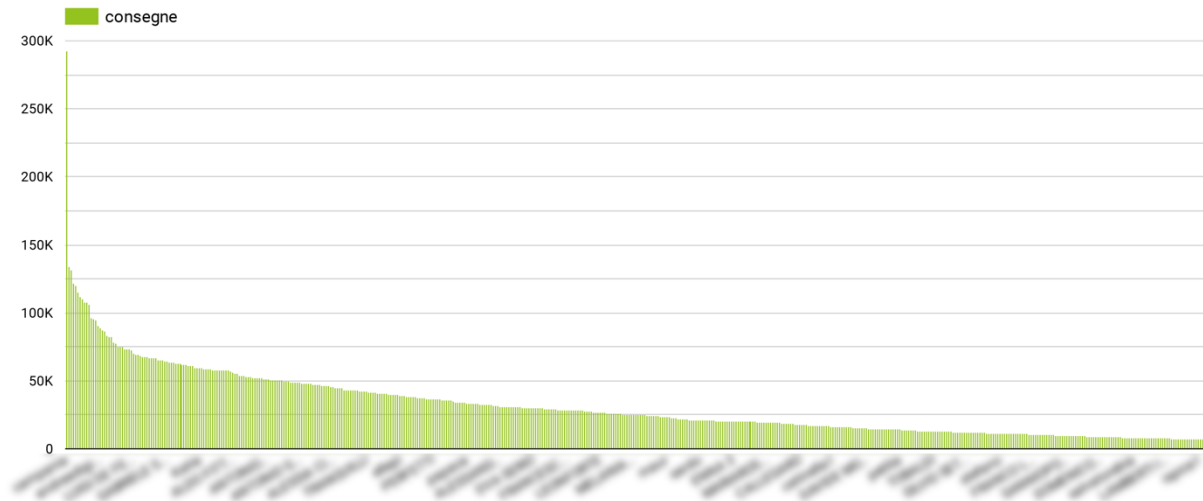
2.4.4 Totale scansioni per ogni postino

	Postini	Scansioni
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 righe

Il risultato così ottenuto mi ha sorpreso, era decisamente eccessivo. Ho deciso di visionarlo in Data Studio per comprendere meglio, tramite l'impatto visivo, a quanto ammontasse il numero di dati di questa portata. Fortunatamente il numero di scansioni così elevate, per quanto non indifferente, sembrava occupare solo una parte relativamente piccola del totale.

2.4.5 Totale scansioni per ogni postino – Istogramma



Basandomi su questi dati, ho deciso di calcolare la media di consegne totali per postino, una media che ho trovato abbastanza ragionevole.

```
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 Numero totale postini

Media consegne	Varianza
14532.96	5.05

Ho voluto anche verificare i giorni lavorativi per ogni singolo postino, e anche in questo caso ho visionato i dati su Data Studio

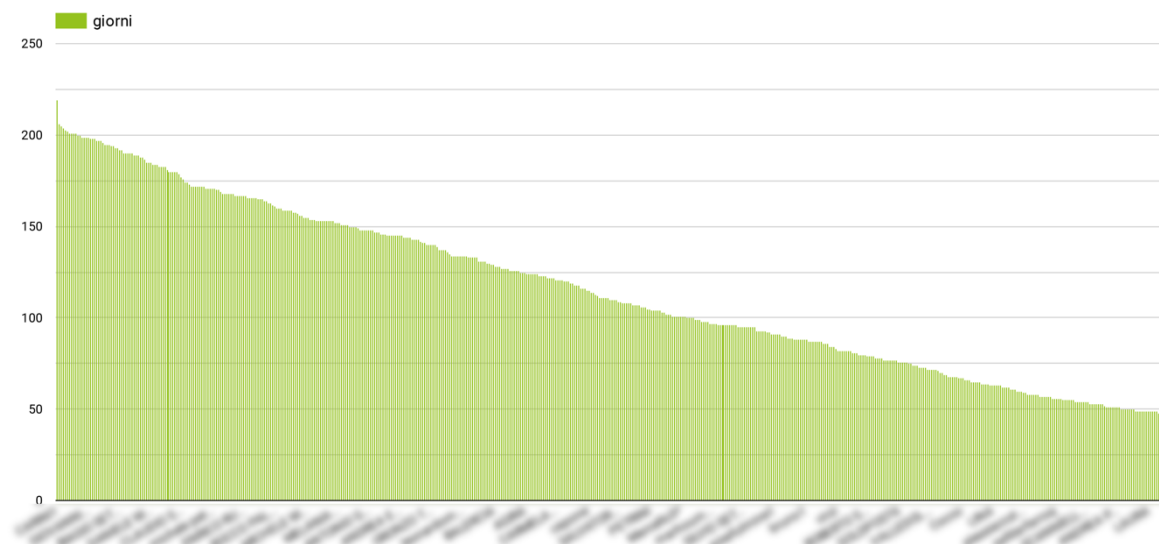
```
SELECT R.postini, COUNT(DISTINCT R.giorni) as giorni_lavorativi
FROM
(SELECT DISTINCT state.user_username as postini, DATE(state.created_date) as
giorni
FROM `database-dev-332416.database_prod.product`,
UNNEST(state) AS state
GROUP BY postini, giorni
ORDER BY giorni DESC) as R
GROUP BY R.postini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

2.4.7 Totale giorni lavorativi per ogni postino

	Postini	Giorni lavorativi
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 righe

2.4.8 Totale giorni lavorativi per ogni postino – Istogramma



Ho ricercato poi il totale di scansioni giornaliere. Nuovamente il numero è risultato troppo elevato, anche dall'istogramma si può vedere che i dati assurdamente alti sono troppi.

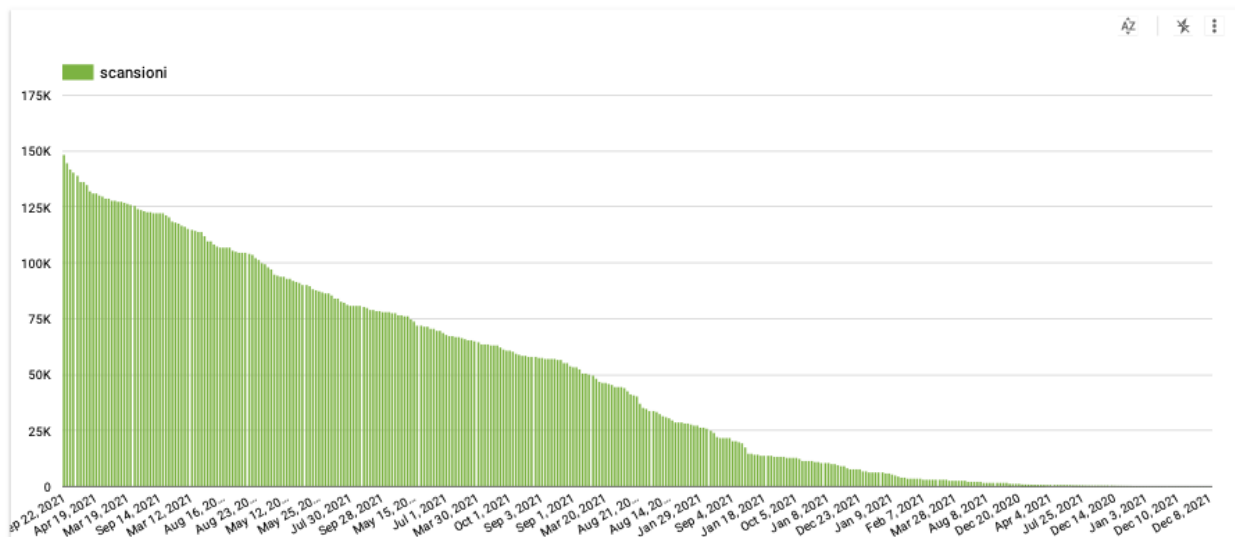
```
SELECT DISTINCT(DATE(s.delivery_created_date)) AS giorno, COUNT(DISTINCT id) AS
scansioni
FROM `database-dev-332416.database_prod.product` AS p,
UNNEST(state) AS s
GROUP BY giorno
ORDER BY scansioni DESC;
```

2.4.9 Totale scansioni giornalieri

	Giorno	Scansioni
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 righe

2.4.10 Totale scansioni per ogni postino



Ho comunque deciso di non fermarmi e cercare di approfondire la natura del problema: per quanto fosse chiara la presenza di dati sporchi, c'era ancora la possibilità di poterli ripulire.

Ho anche verificato l'ultimo giorno in cui si sono registrate scansioni.

```
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 Ultima scansione

Ultima scansione

2021-12-31

Devo menzionare, però, che, a seguito di ulteriori analisi, ho notato che i mesi di novembre e dicembre sono, purtroppo, da considerarsi pressoché mancanti, dato che il numero di scansioni si limita saltuariamente ad una al giorno, il che mi da motivo di credere che si tratti di errori.

## 2.5 Analisi dati mancanti

Avendo a questo punto un'idea generale dei dati con cui avevo a che fare, ho voluto verificare la percentuale di dati mancanti, in particolare di quelli riferiti a latitudine e longitudine. Ho iniziato lavorando su uno specifico postino in uno specifico giorno per verificare come prima cosa la validità della query.

Come primo passo, ho costruito le subquery con CASE WHEN in cui ho proceduto prima a contare tutti i valori riferiti ad un solo lavoratore e ad una sola data scelti casualmente, poi a contare tutti i valori mancanti e infine tutti i valori presenti. Ho poi proceduto a restituire come risultato i valori così ottenuti e a calcolarne la percentuale.

```
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,

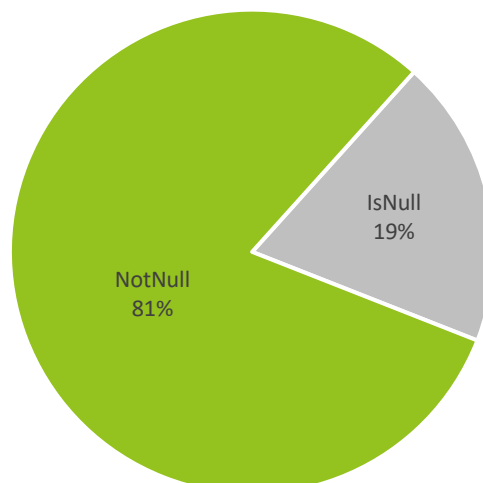
(SELECT SUM
  (CASE WHEN s.user_username = 'Phi'
    AND s.created_date BETWEEN '2021-09-01T00:00:00.000'
    AND '2021-10-10T00:00:00.000'
    AND s.delivery_latitude IS NULL
    AND s.delivery_longitude IS NULL
    THEN 1 ELSE 0 END) AS deliveries
FROM `database-dev-332416.database_prod.product` as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

```

Così formulata, la query mi ha restituito il seguente risultato:

2.4.5 Comparazione e percentuale dati presenti e dati mancanti su latitudine e longitudine per un postino

total	NotNull	IsNull	Null_percent	NotNull_percent
171	138	33	19.3	80.7



A questo punto, ho proceduto ad applicare la query all'intero 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,

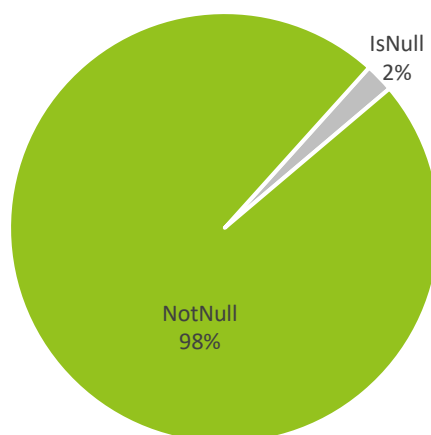
      (SELECT SUM
        (CASE WHEN s.delivery_latitude IS NOT NULL
          AND s.delivery_longitude IS NOT NULL
            THEN 1 ELSE 0 END) AS deliveries
        FROM `database-dev-332416.database_prod.product` as p,
        UNNEST(state) AS s)
      AS NotNull,

      (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);
```

I risultati ottenuti, riportati di seguito, sono stati molto positivi, con una bassissima presenza di dati mancanti.

2.5.2 Comparazione e percentuale dati presenti e dati mancanti su latitudine e longitudine

total	NotNull	IsNull	Null_percent	NotNull_percent
17672087	17290203	381884	2.16	97.84



Ho ripetuto la query con altri campi tra quelli più rilevanti ai fini della mia analisi per verificare anche lì la percentuale dei dati mancanti, ed ho ottenuto nuovamente degli ottimi risultati, che sfioravano il 100% di dati presenti.

A seguire, riporto i risultati sul campo `created_date`:

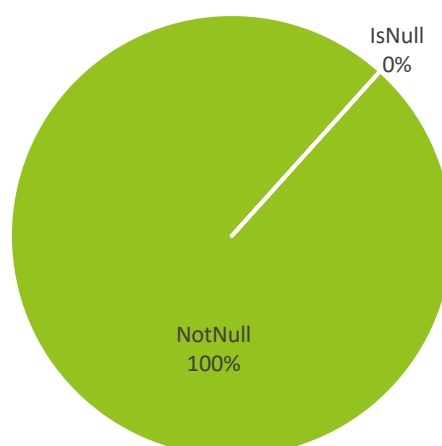
```
SELECT total, NotNull, IsNull, ROUND(IsNull * 100.0 / total,2) AS Null_percent,
ROUND(NotNull * 100.0 / total,2) AS NotNull_percent
FROM (SELECT
  (SELECT COUNT(*) AS deliveries
   FROM `database-dev-332416.database_prod.product` as p,
   UNNEST(state) AS s)
  AS total,

  (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 Comparazione e percentuale dati presenti e dati mancanti su `created_date`

total	NotNull	IsNull	Null_percent	NotNull_percent
17672087	17672086	1	0.0	100.0





I risultati sul campo state.created\_date:

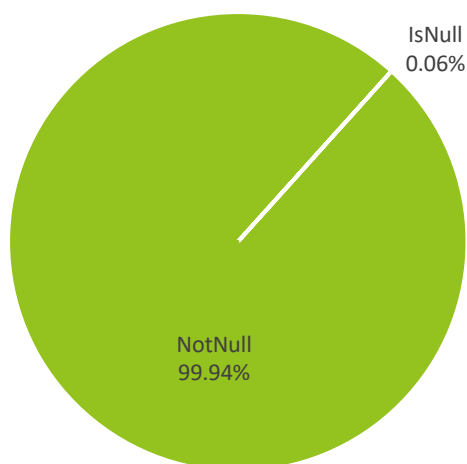
```
SELECT total, NotNull, IsNull, ROUND(IsNull * 100.0 / total,2) AS Null_percent,
ROUND(NotNull * 100.0 / total,2) AS NotNull_percent
FROM (SELECT
      (SELECT COUNT(*) AS deliveries
        FROM `database-dev-332416.database_prod.product` as p,
        UNNEST(state) AS s)
      AS total,

      (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 Comparazione e percentuale dati presenti e dati mancanti su state.created\_date

total	NotNull	IsNull	Null_percent	NotNull_percent
17672087	17662120	9967	0.06	99.94



I risultati su state.delivery\_send\_state:

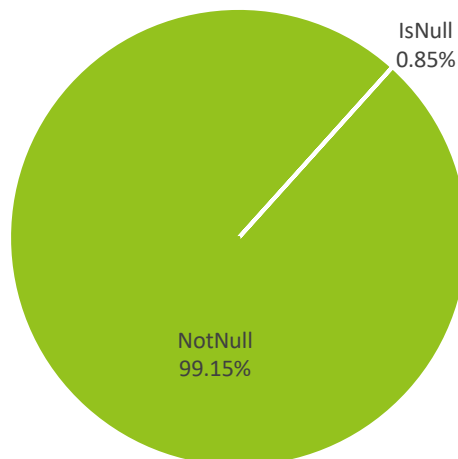
```
SELECT total, NotNull, IsNull, ROUND(IsNull * 100.0 / total,2) AS Null_percent,
ROUND(NotNull * 100.0 / total,2) AS NotNull_percent
FROM (SELECT
      (SELECT COUNT(*) AS deliveries
        FROM `database-dev-332416.database_prod.product` as p,
        UNNEST(state) AS s)
      AS total,

      (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 Comparazione e percentuale dati presenti e dati mancanti su state.delivery\_send\_state

total	NotNull	IsNull	Null_percent	NotNull_percent
17672087	17522727	149360	0.85	99.15



E infine i risultati su state.delivery\_created\_date:

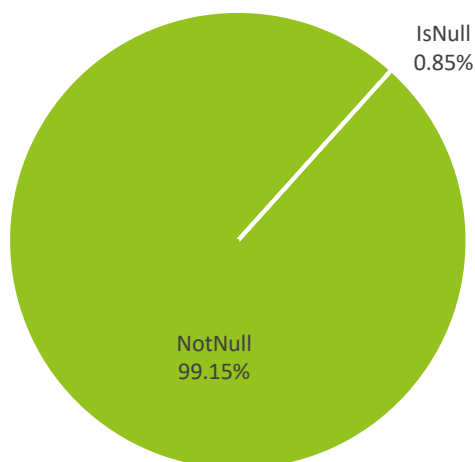
```
SELECT total, NotNull, IsNull, ROUND(IsNull * 100.0 / total,2) AS Null_percent,
ROUND(NotNull * 100.0 / total,2) AS NotNull_percent
FROM (SELECT
      (SELECT COUNT(*) AS deliveries
        FROM `database-dev-332416.database_prod.product` as p,
        UNNEST(state) AS s)
      AS total,

      (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 Comparazione e percentuale dati presenti e dati mancanti su state.delivery\_created\_date

total	NotNull	IsNull	Null_percent	NotNull_percent
17672087	17522727	149360	0.85	99.15



## 2.6 Analisi numero scansioni

Nonostante, quindi, un forte numero di dati apparentemente sporchi, il database aveva, quantomeno, una bassissima percentuale di dati mancanti, specialmente nei campi per me più rilevanti.

A questo punto ho deciso di fare una verifica sulle scansioni giornaliere effettuate da ogni postino. I risultati ottenuti erano, prevedibilmente, assurdamente alti: i picchi massimi raggiungevano quasi le 20.000 scansioni giornaliere, un numero ridicolmente alto, ma anche escludendo le decine di risultati così alti, si contavano comunque centinaia di scansioni giornaliere sopra le 600 (numero massimo indicativo dichiarato dall'azienda stessa).

```
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 Scansioni giornaliere per postino in ordine decrescente

	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 righe

Dato che i risultati così alti erano legati specialmente ad un periodo più lontano (prevalentemente marzo e aprile 2020), ho voluto verificare se la situazione migliorava in tempi più recenti, ripetendo la query ma limitandola prima a tutto ciò che è stato consegnato da settembre ad oggi, e poi tutto ciò che è stato consegnato da Novembre ad oggi, per verificare se la patch rilasciata avesse migliorato la situazione.

```
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 Scansioni giornaliere per postino in ordine decrescente dopo Settembre

	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 righe

Da settembre in poi la situazione migliorava, ma non quanto avrei sperato.

Mi ha sorpreso ancora di più il risultato ottenuto nella query successiva, riferita al periodo da novembre in poi, in cui, come avevo anticipato nel paragrafo precedente, i numeri si riducono ridicolmente. Basti pensare che sono state restituite solo 41 righe.

```
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 Scansioni giornaliere per postino in ordine decrescente dopo Novembre

	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 righe

Per comprendere meglio il problema, ho deciso di entrare maggiormente nel dettaglio, selezionando alcuni dei postini con il più alto numero di scansioni.

Inizialmente ho voluto vedere se tutte le loro scansioni giornaliere fossero così elevate, e ho voluto visionarle dalle più recenti alle più vecchie.

```
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 Scansioni giornaliere del postino 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 righe

```
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 Scansioni giornaliere del postino 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 righe

Ho anche deciso di verificare le scansioni settimanali e mensili per ogni postino, nel dubbio che le scansioni possano essere state caricate tutte in un giorno ma effettuate in più giorni. Il risultato, però, mostra chiaramente che non è questo il caso e che i numeri esageratamente alti di scansioni si mantengono per tutti i giorni della settimana. Anche nell'analisi su un mese calano di poco, per quanto in quest'ultima, tolti i primi eccessivi risultati, gli altri sembrano essere verosimili.

```
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 Scansioni settimanali per postino

	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 righe



```

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 Scansioni mensili per postino

	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 righe

A seguire, ho voluto ispezionare nello specifico le scansioni effettuate basate su un'unica data e un unico 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;

```

Di seguito riporto solo una parte del risultato, dato che la query mi ha restituito tutti i campi del database e, di seguito, non riuscirei a mostrarli tutti.

2.6.8 Scansioni giornaliere del postino lambda

	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 righe

La query così formulata ha estratto tutti i barcode connessi al postino Ni, ma anche tutti i movimenti di questi barcode, anche quindi quando il postino era diverso. Per questa ragione ho voluto verificare anche con quale frequenza venisse mostrato il nome del postino in questione.

```
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 Frequenza postino nel risultato

Frequenza_postino
874

Ho ripetuto questa verifica per svariate combinazioni di postini e date, e ho notato la presenza di dati duplicati tra i risultati. Questo mi ha fatto insospettire sulla vera causa di questi numeri così elevati, e ho ricercato la causa di queste duplicazioni, il che mi ha condotto al problema dell'UNNEST.

## 2.7 Il problema dell'UNNEST

Come si è visto finora, per accedere al contenuto della cartella annidata 'state' ho usato la formula UNNEST(state): il comando UNNEST è stato, infatti, l'unico metodo con cui sono riuscita ad

accedere al contenuto della tabella annidata. Il problema dell'UNNEST, però, è che crea un cross join con la tabella principale, duplicando in questo modo i dati. Questa duplicazione potrebbe falsare i risultati e, quindi, essere la causa dei numeri di scansioni eccessivi che ho riscontrato finora.

Come ovviare, però, a questo problema? Ho impiegato molto tempo nel ricercare una possibile soluzione.

Il mio primo tentativo è stato quello di utilizzare un CONCAT per connettere il giorno e l'ID, creando così una chiave identificativa combinata che consenta ai singoli valori di ripetersi, ma non alla combinazione. Come si può vedere dai risultati, questo metodo non è stato efficace, e in più mi sono resa conto che, se nella migliore delle ipotesi avesse funzionato, avrebbe comunque escluso dalla ricerca tutti i possibili secondi tentativi di consegna avvenuti in una giornata, restituendo comunque un risultato falsato.

```
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 Scansioni giornaliere per postino con Concat

	Scansioni	Giorno	Postino
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

Il tentativo di maggior successo è consistito in una soluzione più semplice e più efficace: ho creato una subquery in cui ho voluto visionare state.created\_date e state.user\_username, raggruppando poi per created\_date e user\_username. A questo punto, nella query principale, ho voluto visionare state.created\_date, e poi fare un count sullo stesso dato: questo, infatti, dal momento che è obbligatorio e segnala ogni movimento della lettera, può facilmente essere utilizzato per tracciare il numero di scansioni.

```
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 Scansioni giornaliere per postino usando una subquery

	Giorno	Scansioni	Postino
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

I risultati, purtroppo, continuano ad essere insoddisfacenti, ma mostrano un miglioramento a partire dal periodo di settembre.

```
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 Scansioni giornaliere per postino usando una subquery dopo Settembre

	Giorno	Scansioni	Postino
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 righe

Ancora una volta, invece, i risultati riferiti al periodo dopo la patch dell'app restituiscono risultati troppo bassi.

```
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 Scansioni giornaliere per postino usando una subquery dopo Novembre

	Giorno	Scansioni	Postino
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 righe

## 2.8 Pulizia dei dati

Mi stavo convincendo sempre più, a questo punto, che il problema fossero i dati stessi e che, qualunque risultato io potessi ottenere con questi, sarebbe comunque stato un risultato inaffidabile. Spronata dal mio relatore, però, ho deciso comunque di non arrendermi e fare un ultimo tentativo nella mia analisi: ho deciso di calcolare la quantità e la percentuale di giornate in cui risulti che un singolo fattorino abbia effettuato un numero di scansioni compreso tra 10 e 600, e quelle che si trovano al di fuori di questo intervallo.

```

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);

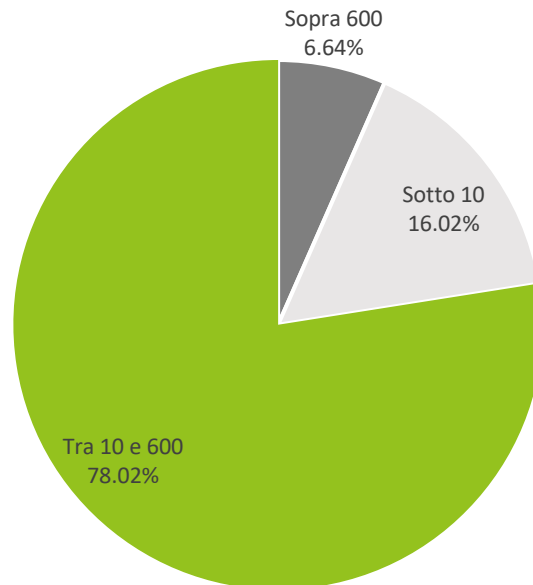
```



2.4.5 Comparazione e percentuale scansioni giornaliere per postino comprese tra 10 e 600, e quelle che si trovano al di fuori di questo intervallo.

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



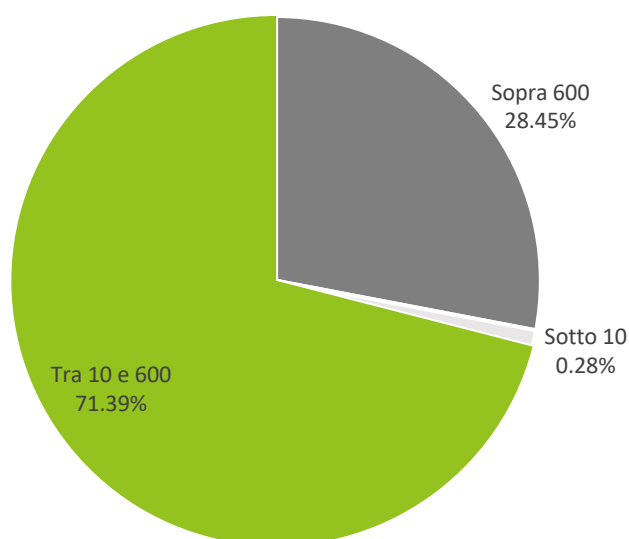
I risultati così ottenuti, sfruttando la subquery precedentemente creata, danno un quadro più positivo di quanto, a questo punto avrei immaginato.

Ho ripetuto la query facendo un SUM invece che un COUNT, per vedere non solo in quante occasioni i dati fuori scala fossero presenti, ma anche a quanto ammontassero, e nuovamente il risultato ottenuto mi ha piacevolmente sorpreso: nonostante il numero di dati inutilizzabili non è indifferente, costituisce comunque la minor parte del database, lasciando un buon 70% di dati utili.

### 2.8.2 Comparazione e percentuale scansioni fuoriscala.

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



Per completezza, ho deciso di fare un'analisi anche sui singoli lavoratori, per tracciare l'andamento generale delle loro scansioni. Ne ho selezionati 5, scegliendoli tra quelli con il numero più alto di scansioni complessive e quelli che risultano attivi dopo settembre.

Come si può vedere nel primo esempio riportato (immagine 2.7.6), i risultati riferiti al postino lambda sono nettamente sproporzionati, con una media che si mantiene costantemente ben sopra le duemila scansioni giornaliere. Per i quattro postini a seguire, però, la situazione migliora: ci sono ancora dei picchi di scansioni eccessivi, ma sono più limitati e la media sembra aggirarsi di più su una scala di risultati fattibili. C'è da notare che, mentre le scansioni del primo postino si riferiscono solo ai mesi di Marzo e Aprile 2021, gli altri postini sono stati attivi in periodi più recenti.

```

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

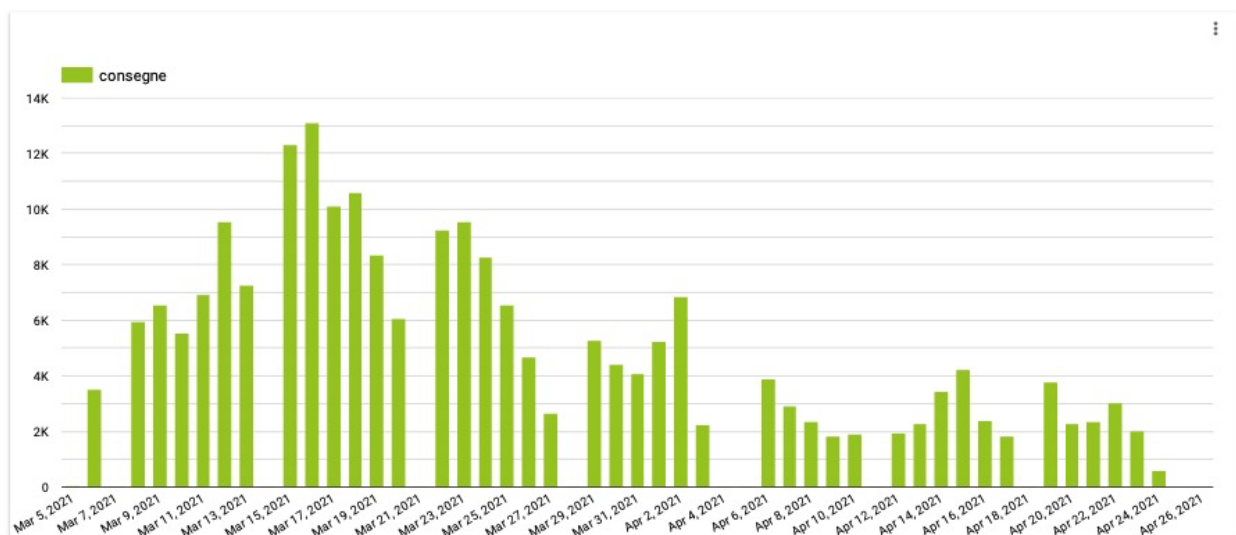
GROUP BY giorno, postino
ORDER BY scansioni DESC;

```

2.8.3 Scansioni giornaliere del postino lambda + Istogramma

	Giorno	Scansioni
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 righe



```

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

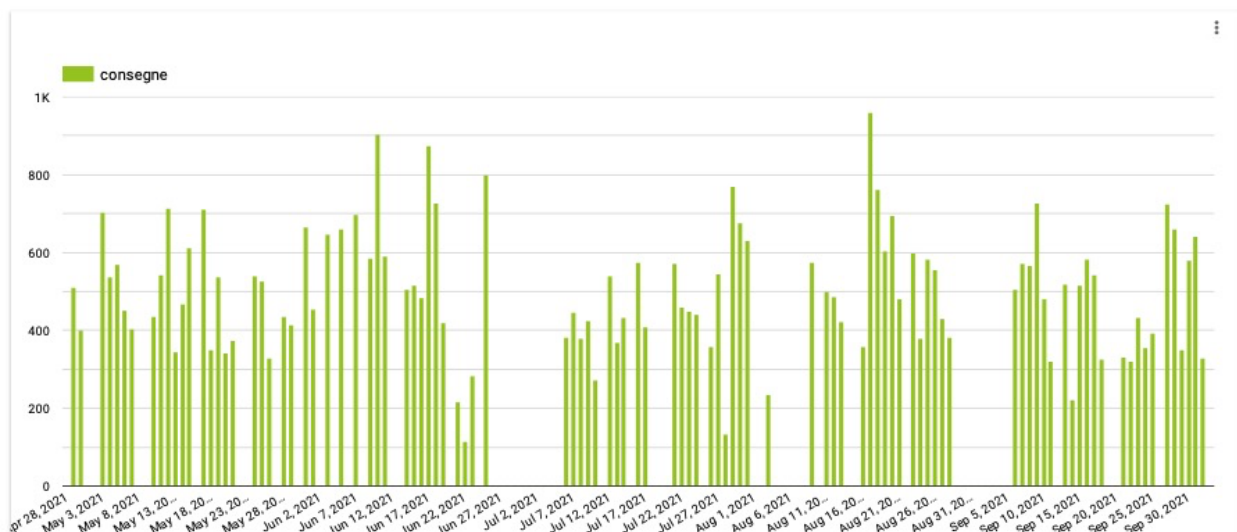
GROUP BY giorno, postino
ORDER BY scansioni DESC;

```

2.8.4 Scansioni giornaliere del postino mi + Istogramma

	Giorno	Scansioni
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 righe



```

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

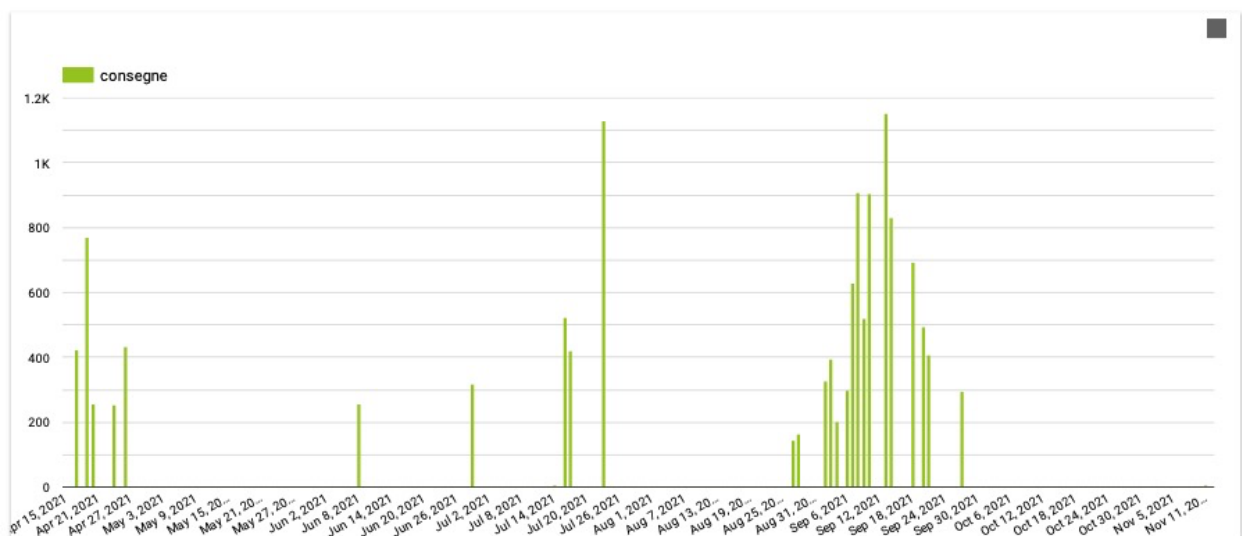
GROUP BY giorno, postino
ORDER BY scansioni DESC;

```

2.8.5 Scansioni giornaliere del postino Chi + Istogramma

	Giorno	Scansioni
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 righe



```

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

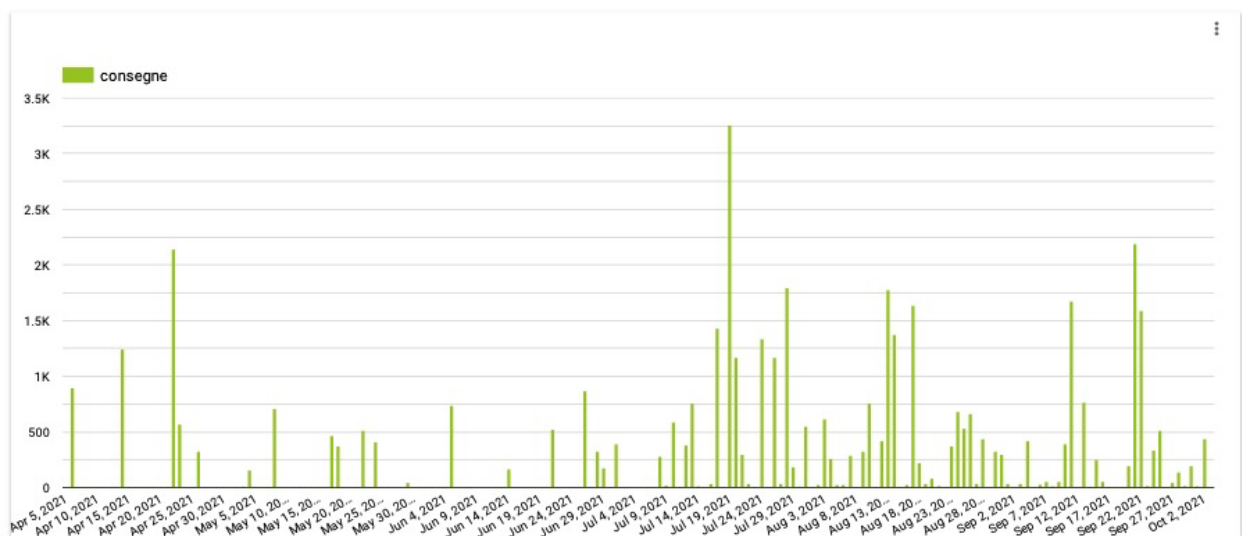
GROUP BY giorno, postino
ORDER BY scansioni DESC;

```

2.8.6 Scansioni giornaliere del postino Zeta + Istogramma

	Giorno	Scansioni
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 righe



```

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

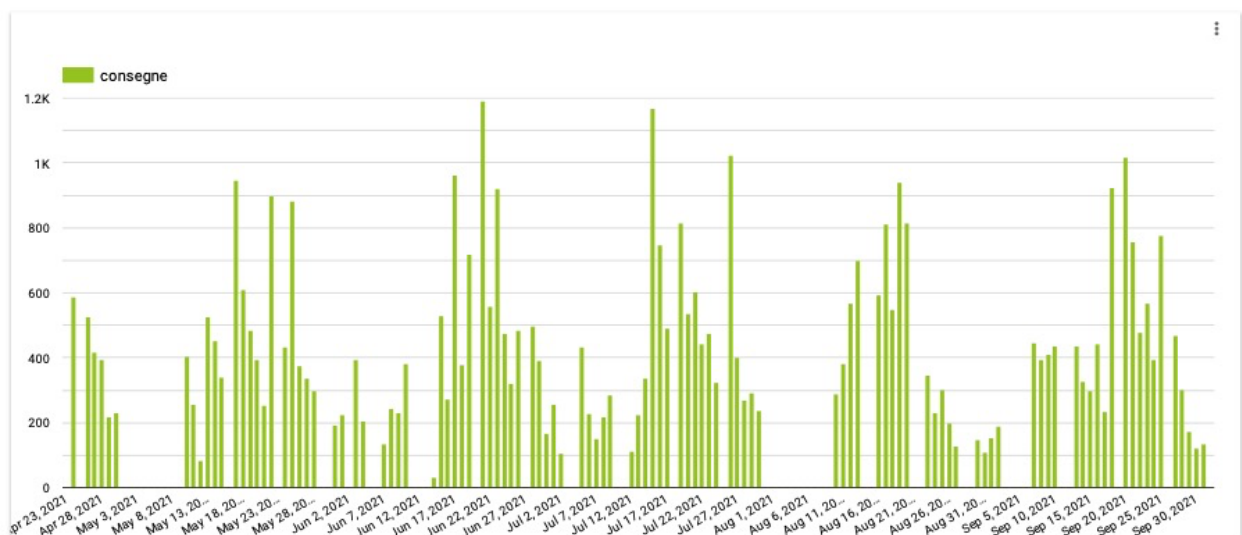
GROUP BY giorno, postino
ORDER BY scansioni DESC;

```

2.8.7 Scansioni giornaliere del postino Ni + Istogramma

	Giorno	Scansioni
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 righe



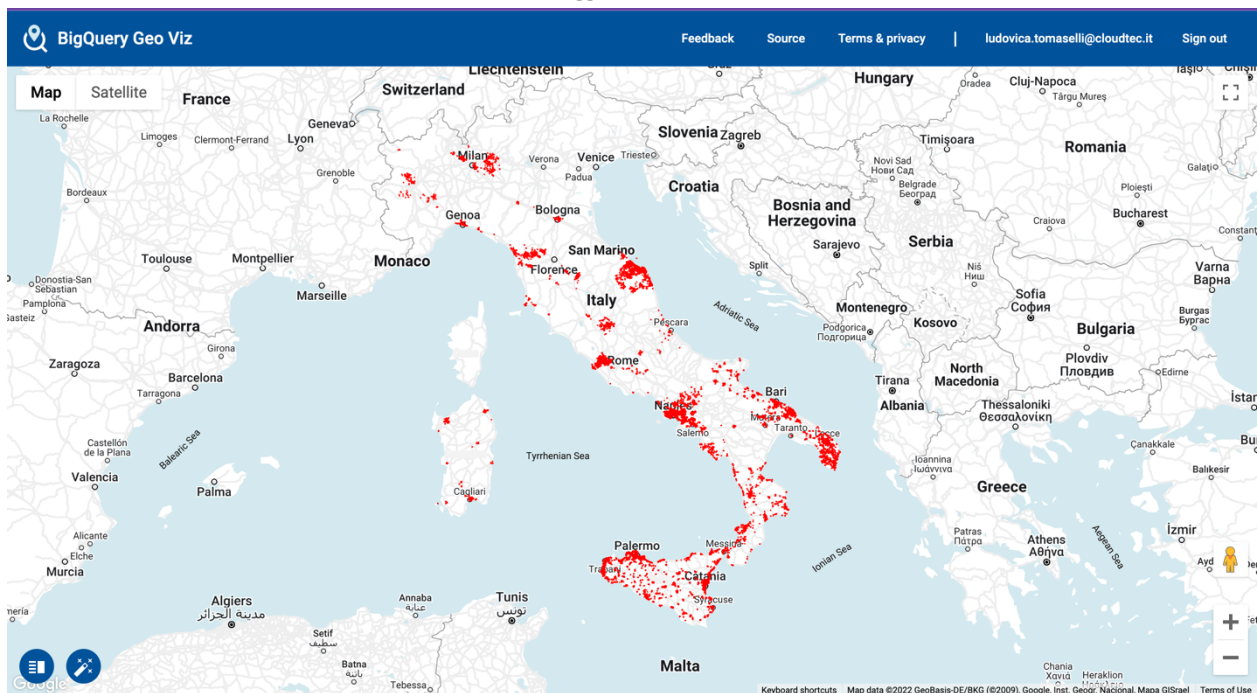
## 2.9 Mappare i dati

Nonostante le difficoltà, sembrerebbe che i dati del database siano, dopotutto, utilizzabili e che l'espedito della subquery sia efficace nell'eliminare i duplicati.

A questo punto, mi è stato possibile verificare anche le coordinate geografiche. Ho prima di tutto cercato un modo per trasformare i dati di latitudine e longitudine, che BigQuery identificava come FLOAT, in GEOGRAPHY, ovvero dati che potessero essere letti come coordinate e come tali analizzati su 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



Contro ogni aspettativa, quindi, sarà possibile lavorare con questi dati, escludendo dall'analisi quell'approssimativo 30% fuori scala.



## Chapter 3

### Conclusioni e Sviluppi Futuri

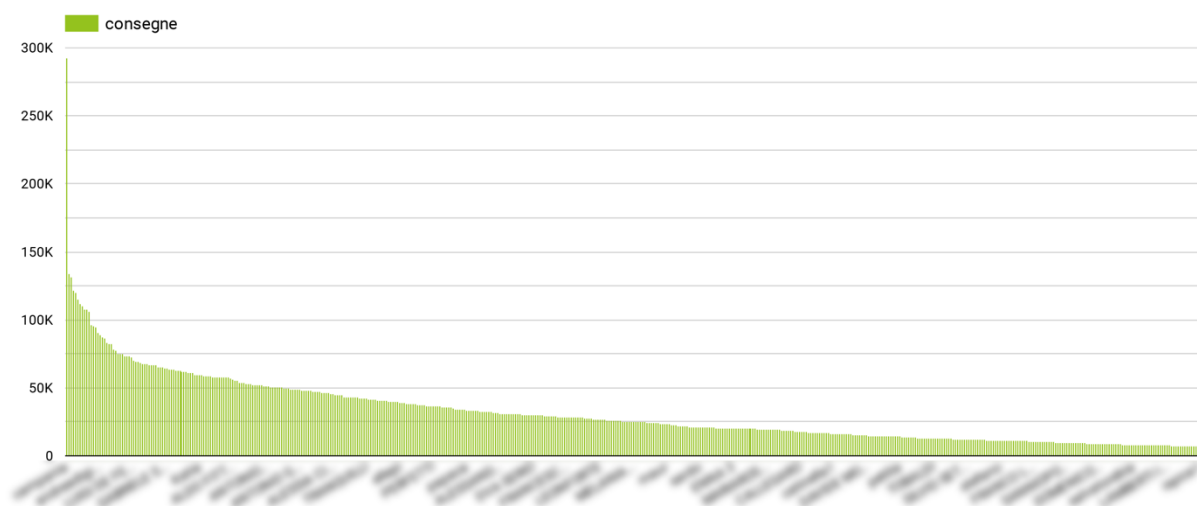
All'inizio del mio tirocinio mi è stato affidato un database di dubbia qualità, con l'intento di scoprire se l'azienda avrebbe potuto utilizzare i dati in esso contenuti, anche solo in parte. Si trattava del database di un'azienda postale, e idealmente i dati sarebbero dovuti servire per tracciare i movimenti delle lettere e dei postini

Il database era stato caricato su Google BigQuery e, per analizzarlo, ho usufruito anche di Google Data Studio e, in minor parte, di Google BigQuery Geo Viz.

Nel primo capitolo di questa tesi ho parlato brevemente di questi tre servizi, per dare almeno un'idea generale di cosa siano.

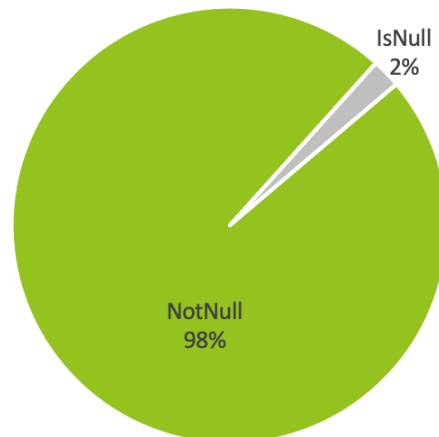
Nel secondo capitolo sono entrata nel merito della mia analisi: ho iniziato mostrando la struttura del database e poi sono passata ad illustrare le query che mi hanno permesso di conoscere i dati con cui avevo a che fare. Sin dalle prime fasi ho notato la presenza di dati anomali: il numero di consegne attribuite ai singoli impiegati in poco più di un anno raggiungeva picchi spaventosamente alti. Per fortuna, però, questi picchi occupavano la minor parte del database, come si può vedere nell'istogramma qui riportato creato tramite Data Studio.

2.4.5 Totale scansioni per ogni postino – Istogramma



Prima di approfondire la natura di questi picchi, ho voluto verificare la presenza di dati mancanti. In particolare, tenendo a mente uno degli obiettivi ultimi di questa analisi, ovvero scoprire se sarebbe stato possibile tracciare i movimenti dei postini, ho verificato in quale percentuale ci fossero dati mancanti nei campi di latitudine, longitudine, stato di consegna e data di consegna.

I risultati ottenuti da questa ricerca sono stati incredibilmente positivi: latitudine e longitudine mostravano appena il 2% di dati mancanti, e questo è stato il numero di dati mancanti più alto in questa ricerca, tutti gli altri campi sfioravano lo 0%.

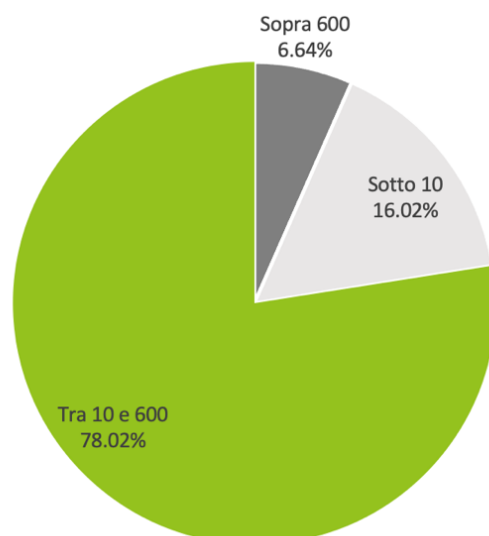


A questo punto era giunto il momento di comprendere la causa dei risultati troppo alti che avevo ottenuto: ho interrogato più volte il database per vedere il numero di scansioni per ciascun postino divise per giorno, per settimana, per mese. Ho analizzato singolarmente alcuni dei postini, verificando le consegne nelle singole giornate, e questo mi ha fatto notare la presenza di dati duplicati.

Ho ricercato l'origine di queste duplicazioni e ho scoperto un problema legato al comando 'UNNEST', comando che serve in BigQuery per leggere il contenuto di una cartella annidata ma che, sfortunatamente, crea un cross join che, quindi, duplica dati.

Sono riuscita a trovare una formula per bypassare questo problema, sfruttando una subquery con un 'GROUP BY': questo ha risolto il problema dei dati duplicati. Purtroppo, nonostante un certo miglioramento, i picchi di scansioni eccessive continuavano ad esserci ed era evidente, a questo punto, che il problema erano i dati stessi.

Tramite Data Studio, però, ho notato che i picchi eccessivi erano limitati ad una sola parte del database. Ho voluto, allora, indagare sulla quantità di dati utilizzabili, cioè quale percentuale delle scansioni giornaliere era compresa tra le 10 e le 600, numero indicativo che avevo ottenuto dall'azienda stessa. La percentuale di dati validi si è rivelata sorprendentemente alta, ammontando addirittura al 78%, un risultato notevole se si pensa che c'era poca speranza di riuscire ad ottenere un qualunque dato utilizzabile.



Grazie alla mia analisi sarà possibile utilizzare questi dati, appositamente filtrati, per svolgere varie analisi che beneficeranno l'azienda: contando che questi dati erano già stati dati per inutilizzabili, posso concludere che quest'analisi ha avuto un esito decisamente positivo.

Il prossimo obiettivo sarà quello di estrarre i percorsi dei postini con l'intento di migliorarli e si proverà ad applicare funzioni di machine learning per calcolare nuovi percorsi che possano ottimizzare le consegne. Sarà possibile verificare i tempi delle consegne, le aree più attive e valutare l'eventuale apertura di nuove filiali in queste zone.

Inoltre, si potrà analizzare il trend di crescita e compararlo ai dati economici dell'azienda, per verificare se alla crescita del fatturato corrisponde una crescita delle attività.

# Bibliografia

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

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