SMBUD

Vaccinations Analysis

Specification, data model and Elasticsearch analysis

POLITECNICO
MILANO 1863

Filippo Lazzati (10629918) - Martina Magliani (10682333) - Christian Grasso (10652464) - Sofia Martellozzo (10623060) - Giacomo Lombardo (10674987)

Year: 2021/2022

Contents

1 Problem specification

The purpose of this project is to design, store and query data on a NoSQL database in order to analyze data about COVID-19 vaccination statistics.
Starting from an open dataset containing raw data about the distribution and administration of COVID-19 vaccines in Italian regions, the main goal of the project is to obtain relevant information through queries and graphs.
To perform this task, the dataset has been loaded into **ElasticSearch**, a search engine with an internal database focused on high performances that interacts with external components through RESTful APIs. Once loaded into the search engine, information can be retrieved from the dataset not only through queries but also through Kibana, a powerful user interface that allows to visualize ElasticSearch data. More specifically, through **Kibana** it is possible to create Dashboards that generate and display graphs and other significant information in an intuitive way.

1.1 Dataset

The used dataset is provided by the Italian Government and contains the official data about vaccines distribution and administration. More specifically it provides a regional overview of the amount of doses administered every day in the time period between December 27th, 2020 and today. The dataset contains 15 fields, which are briefly summarized in the following:

- **index**, a unique identifier of the record;
- **area**, an acronym of the region at issue;
- **fornitore**, the brand provider of the vaccine;
- **data_somministrazione**, the date when the vaccine doses have been administered;
- **fascia_anagrafica**, the age range the people who received the vaccination belong to;
- **sesso_maschile**, the number of males vaccinated;
- **sesso_femminile**, the number of females vaccinated;
- **prima_dose**, the number of first doses;
- **seconda_dose**, the number of second doses;
- **pregressa_infezione**, the number of doses administered to people who came down with the disease between 3 and 6 months ago;
- **dose_addizionale_booster**, the number of third doses admininstered;

- **codice_NUTS1**, European classification of NUTS level 1;
- **codice_NUTS2**, European classification of NUTS level 2;
- **codice_regione_ISTAT**, ISTAT code of the region;
- **nome_regione**, the name of the region.

The way this dataset has been imported into `ElasticSearch` is shown in section 2.
According to the project's specifications, the possibility of integrating the project with an additional dataset is contemplated. Therefore, we have decided to add a secondary dataset, consegne-vaccini-latest, that contains additional information about the amount of daily delivered doses in the same time period, to better contextualise the primary dataset. Such secondary dataset contains 8 field, briefly explained in the following:

- **index**, a unique identifier of the record;
- **area**, an acronym of the region at issue;
- **fornitore**, the brand provider of the vaccine;
- **data_consegna**, the date when the vaccine doses have been delivered;
- **numero_dosi**, the number of doses delivered;
- **codice_NUTS1**, European classification of NUTS level 1;
- **codice_NUTS2**, European classification of NUTS level 2;
- **codice_regione_ISTAT**, ISTAT code of the region;
- **nome_regione**, the name of the region.

2 Conceptual model

In `ElasticSearch`, there are two ways to create a schema (model) of the data we are importing in the database. We have to remember that `ElasticSearch` is an Information Retrieval system that contains an inner document-oriented database, which means that we can store data at the granularity level of documents, the same of `MongoDB`. In the `ElasticSearch` world, the schema of the data is called *mapping*, and can be of two types:

- dynamic mapping, which is automatically carried out by `ElasticSearch`, which autonomously tries to infer the datatypes of the fields;
- explicit mapping, in which we have to explicitly define the type of the fields when creating an index[1].

It is clear that dynamic mapping can fail, therefore we have to be cautious when using it.
In our project, we have decided to exploit dynamic mapping because the mapping inferred perfectly coincides with the one we would have defined. In fact, you can notice this in the following json code, which is provided by `ElasticSearch` to justify the dynamic mapping:

```
{
  "properties": {
    "@timestamp": {
      "type": "date"
    },
    "area": {
      "type": "keyword"
    },
    "codice_NUTS1": {
      "type": "keyword"
    },
    "codice_NUTS2": {
      "type": "keyword"
    },
    "codice_regione_ISTAT": {
      "type": "long"
    },
    "data_somministrazione": {
      "type": "date",
      "format": "iso8601"
    },
    "dose_addizionale_booster": {
      "type": "long"
    },
```

[1] In the `ElasticSearch| world, an index can be seen as a table of RDBS.`

```
    "fascia_anagrafica": {
      "type": "keyword"
    },
    "fornitore": {
      "type": "keyword"
    },
    "nome_area": {
      "type": "keyword"
    },
    "pregressa_infezione": {
      "type": "long"
    },
    "prima_dose": {
      "type": "long"
    },
    "seconda_dose": {
      "type": "long"
    },
    "sesso_femminile": {
      "type": "long"
    },
    "sesso_maschile": {
      "type": "long"
    }
  }
}
```

The mapping requires some observations:

- the type **keyword** is used for structured content such as IDs, email addresses, hostnames, status codes, zip codes, or tags, therefore it perfectly fits for codes like "codice_NUTS1", or "area", or "fornitore", ... etc.; notice that it is ok for us also to have "fascia_anagrafica" as **keyword**, since we do not need to perform text search on it;

- the type **long** is a numeric datatype that represents a signed 64-bit integer with a minimum value of -2^{63} and a maximum value of $263-1$; basically, it is type **integer** but enlarged with more bytes, therefore it is suited for all the numeric values we have in the dataset;

- the only date field, "data_somministrazione", has been properly inferred as a date, thanks to the fact that it was encoded in the standard format iso8601;

- it is better to spend some words about field "area". As a matter of fact, it has been inferred of type "keyword", which is ok for our purposes; however, it can be matched 1-to-1 to a region of Italy, therefore in order to provide a map view in the `Kibana` dashboard, later on we have provided

some rows of explanation about how to match field "area" to "geo_shape" datatype, which allows to draw it as a polygon.

As far as the second dataset is concerned, there are few differences with what previously stated:


```
{
  "properties": {
    "@timestamp": {
      "type": "date"
    },
    "area": {
      "type": "keyword"
    },
    "codice_NUTS1": {
      "type": "keyword"
    },
    "codice_NUTS2": {
      "type": "keyword"
    },
    "codice_regione_ISTAT": {
      "type": "long"
    },
    "data_consegna": {
      "type": "date",
      "format": "iso8601"
    },
    "fornitore": {
      "type": "keyword"
    },
    "nome_area": {
      "type": "keyword"
    },
    "numero_dosi": {
      "type": "long"
    }
  }
}
```


As it can be easily seen by the json provided, the date "data_consegna" has been properly inferred, as well as all the other fields, therefore we can easily adopt dynamic mapping also for this dataset.

3 Queries and Commands

In this section we present some queries and some commands that we consider useful to manage the considered data. It should be remarked that we can interact with `ElasticSearch` through REST APIs, since it is RESTful, and that there are various ways to interfacing with it; the most used are:

- a REST client application like Postman;
- a client like curl;
- `Kibana`, the graphical user interface of `ElasticSearch`. This is the way we adopt.

You have to notice also that you have to upload the datasets with names "`somministrazioni_vaccini`" and "`consegne_vaccini`" in order to use the following queries and commands.

3.1 Queries

We have identified the following queries:

1. **find the number of vaccines administered by brand in the last 30 days and also the brand with the highest number**.
   In order to perform this query, we have to use the `GET` method (like all the other queries). Moreover, we have to filter the results through "query" to retrieve data that is not older than 30 days and then we have to exploit a script to sum the values of fields "sesso_maschile" and "sesso_femminile". Then, we have to aggregate by brand ("fornitore") to compute meaningful sums. Finally, to retrieve the brand with the highest number of vaccines administered we have to exploit also the "max_bucket" construct:

```
GET /somministrazioni_vaccini/_search
{
  "size": 0,
  "query": {
    "range": {
      "@timestamp": {
        "gte": "now-30d/d",
        "lt": "now/d"
      }
    }
  },
  "runtime_mappings": {
    "vaccini": {
      "type": "long",
      "script": {
        "source": "emit(doc['sesso_maschile'].value + doc['
    sesso_femminile'].value);"
      }
    }
  },
  "aggs": {
```

```
    "vaccinations_per_brand": {
      "terms": {
        "field": "fornitore"
      },
      "aggs": {
        "total_vaccinations": {
          "sum": {
            "field": "vaccini"
          }
        }
      }
    }
  ,
    "most-used_brand": {
      "max_bucket": {
        "buckets_path": "vaccinations_per_brand >
    total_vaccinations"
      }
  }
}
}
```

2. **find the region with the highest number of first vaccinations in the last 3 months**.
   The query is similar to the previous one. We have to filter the results by date through "query", and then exploit nested "aggs" to compute the sum of field "prima_dose" aggregated by region; finally we draw the maximum in the same way as before:

```
GET /somministrazioni_vaccini/_search
{
  "size": 0,
  "query": {
    "range": {
      "@timestamp": {
        "gte": "now-90d/d",
        "lt": "now/d"
      }
    }
  },
  "aggs": {
    "first_vaccinations_per_region": {
      "terms": {
        "field": "area"
      },
"aggs": {
        "first_vaccinations": {
          "sum": {
            "field": "prima_dose"
          }
        }
}
  },
    "most-vaccinated_region": {
      "max_bucket": {
```

```
        "buckets_path": "first_vaccinations_per_region >
    first_vaccinations"
      }
  }
}
}
```

3. **find the total number of boosters in Italy in last 30 days**.
   This query is quite easy; indeed, we have to filter the results by date and then return the sum of field "dose_addizionale_booster":

```
GET /somministrazioni_vaccini/_search
{
  "size": 0,
  "query": {
    "range": {
      "@timestamp": {
        "gte": "now-30d/d",
        "lt": "now/d"
      }
    }
  },
  "aggs": {
    "total_number_of_boosters": {
      "sum": {
        "field": "dose_addizionale_booster"
      }
  }
  }
}
```

4. **find the age range with the highest number of vaccinations in the last 30 days.** The pattern is similar to the first query; namely, we use a script after having filtered the results by date to compute the sum of male and female vaccine doses and finally we return the maximum value:

```
GET /somministrazioni_vaccini/_search
{
  "size": 0,
  "query": {
    "range": {
      "@timestamp": {
        "gte": "now-30d/d",
        "lt": "now/d"
      }
    }
  },
  "runtime_mappings": {
    "vaccini": {
      "type": "long",
      "script": {
        "source": "emit(doc['sesso_maschile'].value + doc['
    sesso_femminile'].value);"
      }
    }
  },
```

```
  "aggs": {
    "vaccinations_per_age_range": {
      "terms": {
        "field": "fascia_anagrafica"
      },
      "aggs": {
        "total_vaccinations": {
          "sum": {
            "field": "vaccini"
          }
        }
      }
    }
  ,
    "most-used_brand": {
      "max_bucket": {
        "buckets_path": "vaccinations_per_age_range >
    total_vaccinations"
      }
  }
}
}
```

5. **find the total number of Moderna doses received in the last 3 days in Italy**.
   We simply have to filter the data both by "fornitore" and date before computing the sum of field "numero_dosi":

```
GET /consegne_vaccini/_search
{
  "size": 0,
  "query": {
    "bool": {
      "filter": [
        {"term": {
          "fornitore": "Moderna"
        }
        },
        {"range": {
      "@timestamp": {
        "gte": "now-3d/d",
        "lt": "now-1d/d"
      }
     }
    }
  ]
    }},
  "aggs": {
    "vaccine_doses_received": {
      "sum": {
        "field": "numero_dosi"
      }
    }
  }
}
```

6. **find the region with the highest disparity of vaccinations between males and females**.
   This query is more tricky than the previous one because it involves a script in which we have to compute the absolute value of a difference (we do not know whether there are more males that undergo vaccination or females, and we do not want to return negative values). The rest of the query is similar to the others:

```
GET /somministrazioni_vaccini/_search
{
  "size": 0,
    "runtime_mappings": {
      "vaccini": {
        "type": "long",
        "script": {
          "source": "long total = doc['sesso_maschile'].value
    - doc['sesso_femminile'].value; if(total < 0){total =
    total * - 1; } emit(total)"
        }
      }
    },
    "aggs": {
      "vaccinations_per_region":{
      "terms": {
          "field": "nome_area"
        },
        "aggs": {
        "max_dif_of_vaccinations_between_male_female": {
          "max": {
            "field": "vaccini"
          }
     }

        }},
      "highest_disparity_of_vaccinations": {
        "max_bucket": {
          "buckets_path": "vaccinations_per_region >
    max_dif_of_vaccinations_between_male_female"
     }
    }
  }
}
```

7. **number of children between 5-11 vaccinated this week in Basilicata**.
   We have to filter the results by "fascia_anagrafica", by "nome_area" and by date. Then we have to exploit a script to sum the values of fields "prima_dose", "seconda_dose" and "dose_addizionale_booster". Finally we can find the number of children through the sum of "total_doses" with the "aggs" function.

```
  GET /somministrazioni_vaccini/_search
  {
    "size": 0,
    "query": {
```

```
      "bool": {
        "filter": [
          {"term": {
            "fascia_anagrafica": "5-11"
          }
          },
          {"range": {
        "@timestamp": {
          "gte": "now-7d/d",
          "lte": "now-1d/d"
        }
       }
      },{"term":{
        "nome_area": "Basilicata"
      }}
    ]}},
    "runtime_mappings": {
      "total_doses": {
        "type": "long",
        "script": {
            "source": "emit(doc['prima_dose'].value + doc['
    seconda_dose'].value + doc['dose_addizionale_booster'].
    value);"
        }
      }
    },
    "aggs": {
      "number_of_children": {
        "sum": {
          "field": "total_doses"
        }
      }
    }
  }
```

8. **Percentage of vaccinated people in Lombardia compared to total vaccines each month**.
   For this query we have aggragate by a date interval specified with calendar-aware time intervals ("calendar_interval": "month"). Then, for each time interval, we retrive the total amount of vaccinated people ("vaccinati") using the "aggs" function and we find the vaccineted people in Lombardia exploiting the nested "aggs". Finally, to retrive the percentage of vaccinated people in Lombardia up to the total people in Italy, we have to exploit the "bucket_script" aggregation to compute the division of the parameters "vaccinatiLombardia" and "vaccinati", multiplying the result by 100.

```
GET /somministrazioni_vaccini/_search
{
  "size": 0,
  "aggs": {
    "percentage_per_year": {
      "date_histogram": {
        "field": "data_somministrazione",
        "calendar_interval": "month"
```

```
      },
      "aggs": {
        "vaccinati": {
          "sum": {
            "field": "prima_dose"
          }
        },
        "vaccinati_Lombardia": {
          "filter": {
            "term": {
              "nome_area": "Lombardia"
            }
          },
          "aggs": {
            "dosi": {
              "sum": {
                "field": "prima_dose"
              }
            }
          }
        },
        "vaccini_percentage": {
          "bucket_script": {
            "buckets_path": {
              "vaccinatiLombardia": "vaccinati_Lombardia>dosi
    ",
              "vaccinati": "vaccinati"
            },
            "script": "params.vaccinatiLombardia / params.
    vaccinati * 100"
          }
        }
      }
    }
  }
}
```

9. **brand of the less-used vaccine with young people (¡30 years old)**. We have to filter the results through "bool" that retrieves data from all the specified fields ("fascia_anagrafica") thanks to the "should" clause. Then we have to exploit a script to sum the values of fields "sesso_maschile" and "sesso_femminile". Then, we have to aggregate by brand ("fornitore") to compute meaningful sums. Finally, to retrieve the brand with the lowest number of vaccines administered we have to exploit the "min_bucket" construct:

```
GET /somministrazioni_vaccini/_search
{
  "size": 0,
  "query": {
    "bool": {
      "should": [
       {"match" : {
            "fascia_anagrafica": "5-11"
       }},
       {"match" : {
```

```
            "fascia_anagrafica": "12-19"
        }},
        {"match" : {
            "fascia_anagrafica": "20-29"
        }}
      ]
    }
  },
  "runtime_mappings": {
    "vaccini": {
      "type": "long",
      "script": {
        "source": "emit(doc['sesso_maschile'].value + doc['
    sesso_femminile'].value);"
      }
    }
  },
  "aggs": {
    "vaccinations_per_brand": {
      "terms": {
        "field": "fornitore"
      },
      "aggs": {
        "total_vaccinations": {
          "sum": {
            "field": "vaccini"
          }
        }
      }
    }
  ,
    "less-used_brand": {
      "min_bucket": {
        "buckets_path": "vaccinations_per_brand >
    total_vaccinations"
      }
  }
}
}
```

10. **Percentage of vaccinated people with booster compared to total vaccines each year**.
    The pattern of this query is similar to the eighth one. We filter by "year" as date interval, then we retrive the total vaccinated people and the boostered ones to compute the percentage with the "bucket_script" aggregation.

```
GET /somministrazioni_vaccini/_search
{
  "size": 0,
  "aggs": {
    "percentage_per_year": {
      "date_histogram": {
        "field": "data_somministrazione",
        "calendar_interval": "year"
      },
      "aggs": {
```

```
        "vaccini": {
          "sum": {
            "field": "prima_dose"
          }
        },
          "booster": {
            "sum": {
              "field": "dose_addizionale_booster"
              }
          },
        "vaccini_percentage": {
          "bucket_script": {
            "buckets_path": {
              "vacciniBooster": "booster",
              "vaccini": "vaccini"
            },
            "script": "params.vacciniBooster / params.vaccini
    * 100"
          }
        }
      }
    }
  }
}
```

3.2 Commands

We have identified the following commands to show how the system works:

1. **Update one specific document (data of Abruzzo 2020/03/30 for age range 40-49 for Moderna) with specific data (100 males vaccinated and 2012 second doses)**.
   The construct we use is the "_update_by_query" one, which can be used with the `POST` method; we have to perform a query to find the document to update, and then we exploit a script (with parameters) to update the document:

```
POST /somministrazioni_vaccini/_update_by_query
{
  "query": {
    "bool": {
      "filter": [
        {"term": {"area": "ABR"}},
        {"term": {"fascia_anagrafica": "40-49"}},
        {"term": {"fornitore": "Moderna"}},
        {
          "range": {
            "@timestamp": {
              "time_zone": "+02:00",
              "gte": "2020-03-30T00:00:00",
              "lt": "2020-03-31T00:00:00"
            }
          }
        }
```

```
      ]
    }
  },
  "script": {
    "source": "ctx.sesso_maschile = params.sesso_maschile; ctx
    .seconda_dose = params.seconda_dose",
    "lang": "painless",
    "params": {
      "sesso_maschile": 100,
      "seconda_dose": 2012
    }
  }
}
```

2. **Create one new document**.
   This command is the easiest; we simply have to make a `POST` with "_doc", and then specify a value for all the fields:

```
POST /somministrazioni_vaccini/_doc
{
  "data_somministrazione": "2021-12-28",
  "fornitore": "Moderna",
  "area": "BAS",
  "fascia_anagrafica": "30-39",
  "sesso_maschile": 910,
  "sesso_femminile": 1211,
  "prima_dose": 93,
  "seconda_dose": 563,
  "pregressa_infezione": 33,
  "dose_addizionale_booster": 1465,
  "codice_NUTS1": "ITF",
  "codice_NUTS2": "ITF5",
  "codice_regione_ISTAT": 17
}
```

3. **Add a runtime field**.
   Runtime fields are fields evaluated at query time. They allow us to add fields to existing documents without reindexing data and to define additional fields without modifying the schema. Here, a useful runtime field might be the sum of the total doses, since in each record the doses are grouped by type of administration. By creating the runtime field `dosi_totali`, we add an additional field to each record containing the desired information and to facilitate the creation of other queries. To create a runtime field, we have to use a script along with the `PUT` method, specifying the name of the field, its type and its source:

```
PUT somministrazioni_vaccini/_mapping
{
  "runtime": {
    "dosi_totali": {
      "type": "long",
      "script": {
        "source": "emit(doc['prima_dose'].value + doc['
    seconda_dose'].value + doc['pregressa_infezione'].value +
    doc['dose_addizionale_booster'].value);"
```

```
      }
    }
  }
}
```

4 Kibana Dashboard

By using the Kibana tool we realised a simple dashboard containing some relevant graphs to visualise the data provided by the dataset. The dashboard displays the following information and graphs:

- total number of administered vaccines;
- administered doses of vaccines per region;
- administered doses per supplier;
- administered doses per week, grouped by:
  - supplier;
  - age range;
  - gender.
- total number of delivered doses;
- delivered doses per region;
- delivered doses per week.

4.1 Dashboard initialization

To create the Kibana dashboard:

- Install the two datasets as it is explained in Section 3 (remember to name them `somministrazioni_vaccini` and `consegne_vaccini`);
- From the Dev Tools, run the command 3 (Add a runtime field), since some visualizations require the field `dosi_totali`;
- In the Maps tool, select "Add layer" and then "Upload GeoJSON". Then upload the `.geo.json` file provided in the exports section and name the index "`italyregions`";
- Go to Stack Management (you can leave the Maps page without saving) and then to Saved Objects. Then upload the `.ndjson` file provided in the exports section. It is not necessary to edit any import setting;
- Go to the Dashboard section and open the Italy Vaccines dashboard;
- Set a relevant time period (e.g. January 1st, 2021 - December 27th, 2021).

4.2 Dashboard visualizations

In this section are provided screenshots of the various visualizations created on the Kibana Dashboard.

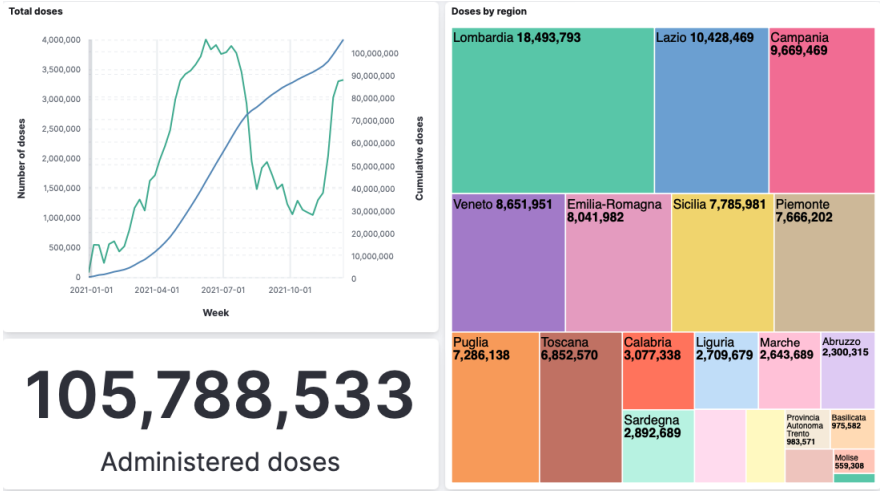

Figure 1: Overview of the vaccine doses administrations

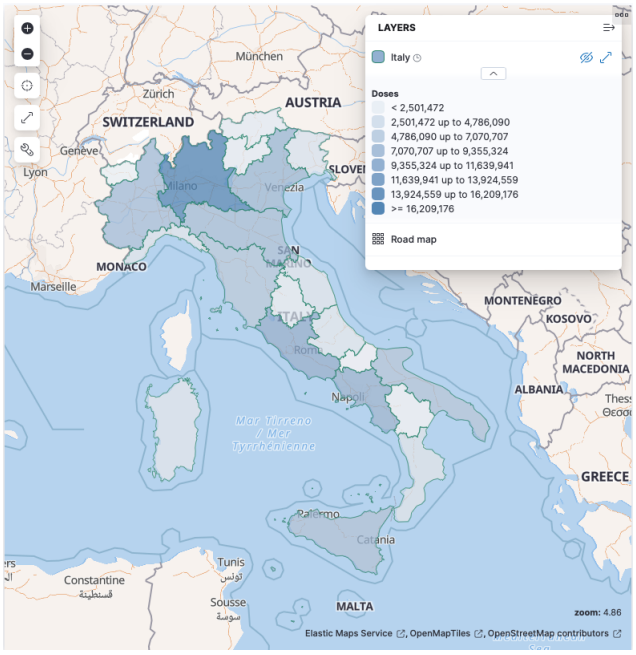

Figure 2: Doses administration by region, visualized on the Italian territory

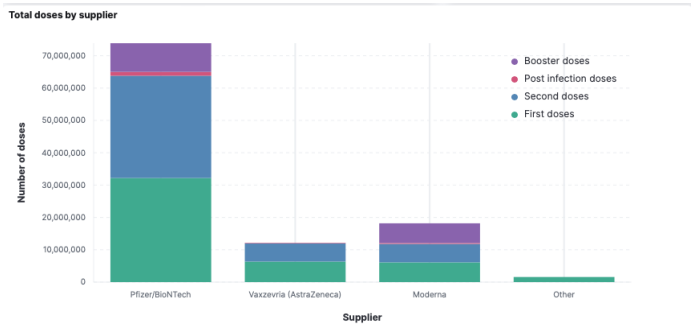

Figure 3: Administered doses per supplier

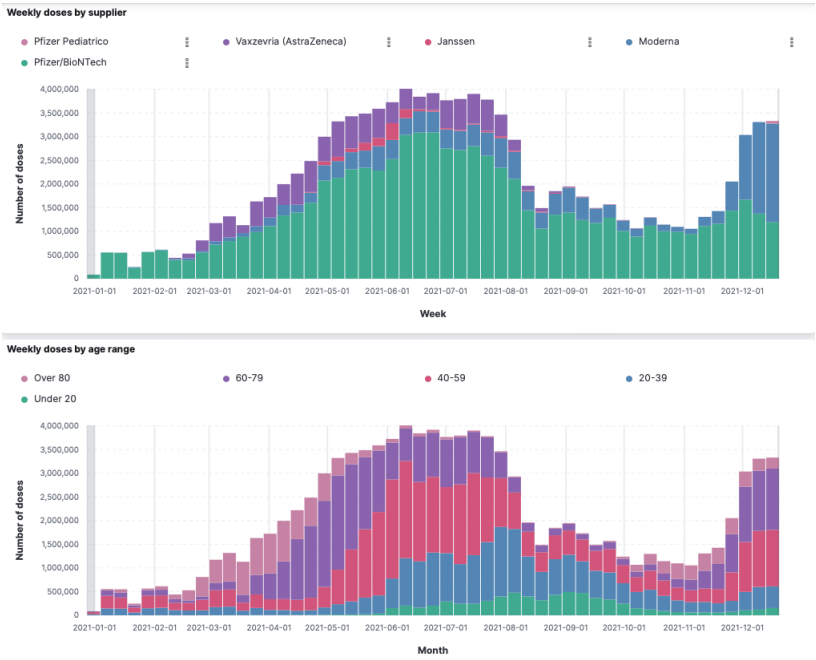

Figure 4: Weekly administered doses, grouped by supplier and age range

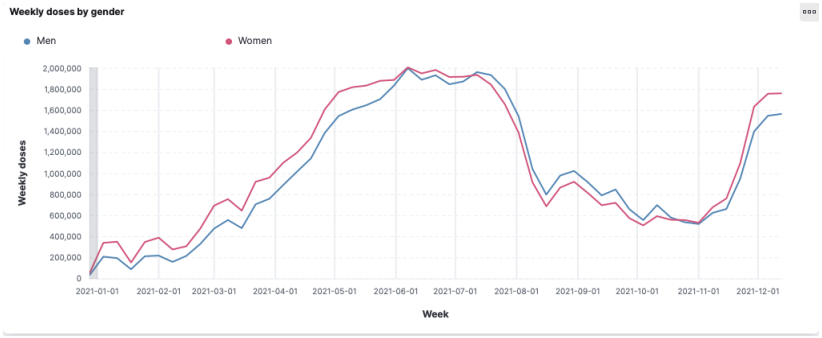

Figure 5: Weekly administered doses, grouped by gender

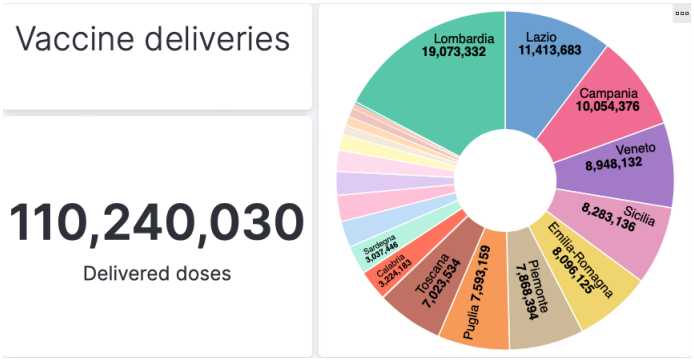

Figure 6: Overview of the vaccine deliveries

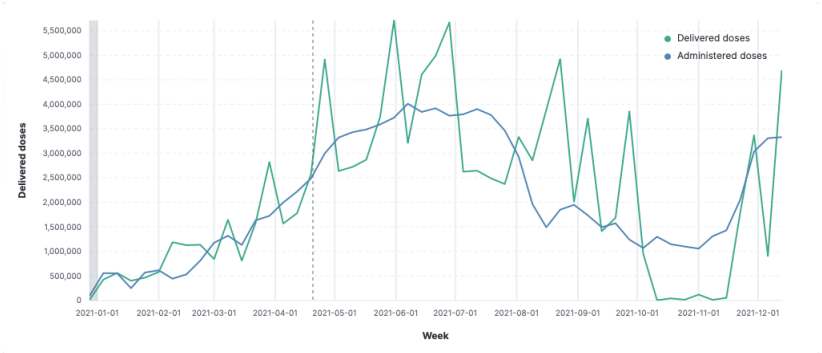

Figure 7: Weekly delivered/administered doses comparison

5 Cassandra

An alternative way to implement the vaccine administrations dataset can be to store it in a columnar database such as Cassandra. Columnar databases are particularly suitable for very large datasets and for analytical purposes such as aggregation operations on entire columns, such as retrieving the total number of administered doses.

5.1 Dataset implementation

To implement the dataset in Cassandra, first of all start Cassandra, then move to the dataset folder and start the `cqlsh` console.

- Create a keyspace:

```
        CREATE KEYSPACE somministrazioni WITH REPLICATION = {'class': 'SimpleStrategy', 'replication_factor':3};

```

- Select the keyspace:

```
        USE somministrazioni;

```

- Create a table:

```
        CREATE TABLE somministrazioni (
            data_somministrazione date,
            fornitore text,
            area text,
            fascia_anagrafica text,
            sesso_maschile varint,
            sesso_femminile varint,
            prima_dose varint,
            seconda_dose varint,
            pregressa_infezione varint,
            dose_addizionale_booster varint,
            codice_NUTS1 text,
            codice_NUTS2 text,
            codice_regione_ISTAT varint,
            nome_area text,
            PRIMARY KEY (codice_regione_istat, fornitore, fascia_anagrafica, data_somministrazione)
        );

```

- Import the dataset:

```
        COPY somministrazioni (data_somministrazione, fornitore, area, fascia_anagrafica, sesso_maschile, sesso_femminile, prima_dose, seconda_dose, pregressa_infezione, dose_addizionale_booster, codice_NUTS1, codice_NUTS2, codice_regione_ISTAT, nome_area) FROM '<.csv dataset>' WITH HEADER = false;

```

- Create an index on `nome_area`:

```
        CREATE INDEX regione ON somministrazioni (nome_area);

```

Now the dataset is ready to be queried.

5.2 Queries examples

Here are some examples of possible queries in CQL. Note that in aggregation queries, it is not possible to order the results by the aggregated value since in Cassandra ordering is possible only on clustering columns.

- Total number of administered doses:

```
        SELECT sum(sesso_maschile)+sum(sesso_femminile) AS
    total_doses FROM somministrazioni;

```

- Administered doses per supplier in June, 2021:

```
        SELECT fornitore, sum(sesso_maschile)+sum(
    sesso_femminile) AS total_doses FROM somministrazioni
    WHERE data_somministrazione >= '2021-06-01' AND
    data_somministrazione <= '2021-06-30' GROUP BY
    codice_regione_ISTAT, fornitore ALLOW FILTERING;

```

- Total number of booster doses in Lombardia since December 1st, 2021:

```
        SELECT sum(dose_addizionale_booster) AS booster_doses
    FROM somministrazioni WHERE data_somministrazione >=
    '2021-12-01' AND nome_area = 'Lombardia' ALLOW FILTERING;

```

6 References and sources

- Elasticsearch
- Kibana
- Cassandra
- Dataset
- draw.io