POLITECNICO
MILANO 1863

A.Y. 2021-2022

Computer science and engineering

Systems and Methods for Big and Unstructured Data
Delivery 3

Report

Beretta Davide
Bollini Matteo
Caleffi Nicolò
De Nicolò Andrea
Madhkour Amor

January 10th 2022

Table of content

1.Introduction 2

2. Specification 3

2.A Assumptions: 4

3. Database Structure 5

3.A Documents structure 5

3.B Database creation 9

3.C Kibana Dashboard 9

3.C.1 Information not divided by region 9

3.C.2 Information divided by region 13

4. Queries and commands 16

4.A Queries 16

4.A.1 Vaccinated people each month 16

4.A.2 People who can take the booster dose 17

4.A.3 Male and female vaccinations 18

4.A.4 People who joined the vaccination campaign 19

4.A.5 Administered doses for supplier 20

4.A.6 Administered doses for region and age group 21

4.A.7 Average vaccinations over day in the last week 22

4.A.8 Region with the highest number of administered booster doses 23

4.A.9 Average number of doses by a supplier 24

4.B Commands 27

4.B.1 Uploading a new record with the same data of a line of the dataset 27

4.B.2 Uploading a new record from another source 28

regions index 29

Add and execute the enrich policy 30

Create the ingest pipeline 30

Command 32

4.B.3 Uploading multiple records 32

1.Introduction

This document describes the main features of a database supporting a data analysis scenario about Covid-19 vaccinations statistics. What follows is the result of combined efforts of the students in group number **26**:

- Beretta Davide (10656835)
- Bollini Matteo (10615991)
- Caleffi Nicolò(10588181)
- De Nicolò Andrea(10837851)
- Madhkour Amor(10794343)

More specifically, in **chapter two** a brief explanation of the problem and the specification of the project work is provided, along with the description of the dataset.

In **chapter three** the structure of the database is described , with particular attention to the mapping and the pipelines used to import the data.

In **chapter four** there is a brief description of the Kibana dashboard and how it works.

In **chapter five** there is a description of the queries and commands we decided to present to analyze and update the data.

2. Specification

On GitHub there is a repository which contains an open dataset filled by various information about shipments, deliveries and administrations of anti-Covid-19 vaccines in all the italian regions. The dataset is continuously updated as the vaccines are delivered and administered and in particular is made by 10 tables containing:

- Deliveries, divided by date and area;
- Administrations, divided by date, area and gender, age and category of the subject;
- Data about vaccinable population of the various areas;
- Data about the vaccination sites

What is asked is to import in an Elasticsearch environment one table of this dataset (**somministrazioni-vaccini-latest**) about the vaccine administrations and write some queries in order to obtain statistics about the vaccination in the various regions and age ranges, developing a basic Kibana dashboard for the visualization of the results.

The table is described as: "data about daily administration of the vaccines, divided by area and age of the vaccinated subjects.". In particular the table is structured as shown below:

somministrazioni-vaccini-latest

FIELD	**DOMAIN**	**DESCRIPTION**
Area	String	Acronyms of the region of delivery
Supplier	String	Complete provider's name
Administration_date	Datetime	Administration date of the vaccines
Age_group	String	Age range of the vaccinated subjects
Male_count	Integer	Number of vaccinations administered to males
Female_count	Integer	Number of vaccinations administered to females.
First_doses	Integer	Number of people administered with the first dose
Second_doses	Integer	Number of people administered with the second dose

Post_infection_doses	Integer	Number of people administered with a dose after they have been infected
Booster_doses	Integer	Number of people administered with the additional booster dose
NUTS1_code	String	European NUTS 1 code: https://en.wikipedia.org/wiki/NUTS_statistical_regions_of_Italy
NUTS2_code	String	European NUTS 2 code: https://en.wikipedia.org/wiki/NUTS_statistical_regions_of_Italy
ISTAT_region_code	Integer	ISTAT code of the region
Region_name	String	Name of the region (bilingual, when necessary)

2.A Assumptions:

- A person completes the vaccination cycle either by taking the second dose or the post infection dose
- A person can take the booster dose since 4 months after completing the vaccination cycle
- No one has already taken two booster doses
- The field "first_doses" includes only the first doses administered to people who were not infected (i.e., it doesn't include the post infection doses). This is coherent with the fact that in the dataset *first_doses* + *second_doses* + *post_infection_doses* + *boost_doses* = *male_count* + *female_count*

3. Database Structure

3.A Documents structure

In our database we store one document for each row of the dataset. These documents have all the fields that are present in the csv file, plus two extra fields:

- administerd_doses, that is computed as male_count+female_count, is the number of doses that have been administered in that record
- fully_vaccinated_count, that is computed as second_doses+post_infection_doses, is the number of people that have completed their vaccination cycle in that record

These fields are helpful to compute queries efficiently. Without them we would need to use scripts or runtime fields, which, being not indexes, are slower (even if, being the dataset not too big, the execution time is still short). Another important fact is that Kibana visualizations don't allow us to use such queries, or at least to create and manage them via the Lens visual tool. In particular, the Map visualization doesn't even allow the execution of some computations on the queries result before showing them, which makes it impossible to show something like the number of administered doses without the administered_doses field. Since the database is still small even with these extra fields, we decided that the best way to explore the data was to add and use them.
The mapping of the index is reported below:

```
{
  "properties": {
    "@timestamp": {
      "type": "date"
    },
    "NUTS1_code": {
      "type": "keyword"
    },
    "NUTS2_code": {
      "type": "keyword"
    },
    "administration_date": {
      "type": "date",
      "format": "iso8601"
    },
    "age_group": {
      "type": "keyword"
    },
    "area": {
      "type": "keyword"
    },
    "booster_doses": {
      "type": "long"
    },
    "female_count": {
      "type": "long"
    },
    "first_doses": {
      "type": "long"
    },
    "male_count": {
      "type": "long"
    },
    "post_infection_doses": {
      "type": "long"
    },
    "region_ISTAT_code": {
      "type": "keyword"
    },
    "region_name": {
      "type": "keyword"
    },
    "second_doses": {
      "type": "long"
    },
    "supplier": {
      "type": "keyword"
    },
      "administered_doses": {
      "type": "long"
    },
      "fully_vaccinated_count": {
      "type": "long"
    }
  }
}
```

We decided also to use the keyword type for string fields instead of the text one. These fields don't allow arbitrary text values, but just specific values. We expect all the documents that will be imported to have the correct values (including, for instance, the transactions of the region names and the correct capital letters) and we usually perform aggregations on them, making the keyword type the more suited one.
The "region_ISTAT_code" field is a keyword too (it's made by digits, but we don't compute math operations or range queries on it, so the keyword type is more appropriate). To use this field in the Kibana dashboard map, we used the ingest pipeline to slightly modify its format with respect to the one used in the dataset, storing each code with 2 digits (e.g. 03 instead of 3).
In order to store the istat code with two digits and to compute the two extra fields, we used the following ingest pipeline:

```
{
  "description": "Ingest pipeline created by text structure finder",
  "processors": [
    {
      "csv": {
        "field": "message",
      "target_fields": [
        "administration_date",
        "supplier",
        "area",
        "age_group",
        "male_count",
        "female_count",
        "first_doses",
        "second_doses",
        "post_infection_doses",
        "booster_doses",
        "NUTS1_code",
        "NUTS2_code",
        "region_ISTAT_code",
        "region_name"
      ],
      "ignore_missing": false
    }
  },
  {
    "date": {
      "field": "administration_date",
      "timezone": "Europe/Rome",
      "formats": [
        "ISO8601"
      ]
    }
  },
  {
    "convert": {
      "field": "booster_doses",
      "type": "long",
      "ignore_missing": true
    }
  },
  {
    "convert": {
      "field": "female_count",
      "type": "long",
      "ignore_missing": true
    }
  },
  {
    "convert": {
      "field": "first_doses",
      "type": "long",
      "ignore_missing": true
    }
  },
  {
    "convert": {
      "field": "male_count",
      "type": "long",
        "ignore_missing": true
      }
    },
    {
      "convert": {
        "field": "post_infection_doses",
        "type": "long",
        "ignore_missing": true
      }
    },
    {
      "convert": {
        "field": "second_doses",
        "type": "long",
        "ignore_missing": true
      }
    },
    {
      "script": {
         "description": "pad the istat code to two digit, for a problem with
java overloading I need that def array",
        "lang": "painless",
        "source": "ctx['region_ISTAT_code'] = String.format('%02d', new def[]
{Integer.parseInt(ctx['region_ISTAT_code']) });"
      }
    },
        {
      "script": {
        "description": "administered_doses field",
        "lang": "painless",
                             "source":    "ctx['administered_doses']    =
ctx['male_count']+ctx['female_count'];"
      }
    },
        {
      "script": {
        "description": "add fully_vaccinated_count field",
        "lang": "painless",
                          "source":    "ctx['fully_vaccinated_count']    =
ctx['post_infection_doses']+ctx['second_doses'];"
      }
    },
    {
      "remove": {
        "field": "message"
      }
    }
  ]
}
```

3.B Database creation

To populate the database from Kibana, create an index **vaccinations** importing the data from **somministrazioni.csv** (under the Data folder) using the mapping and ingest pipeline we provided before.
The file contains the dataset at 10/01/2022.
Since all the preprocess steps are performed by the ingest pipeline, the same procedure can be used to import a more recent version of the dataset as well.

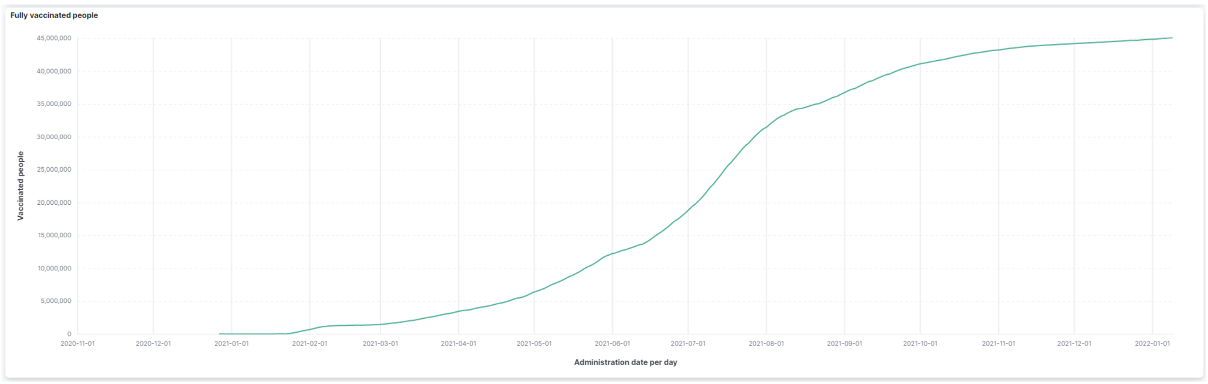
3.C Kibana Dashboard

The dashboard can be imported from the file **dashboard.ndjson**. The controls inside the dashboard may cause issues with older versions of Kibana. If the dashboard can't be imported, you can import the visualizations to the library from the file **visualizations.ndjson** and create a new dashboard from them. Both files are inside the Data folder.

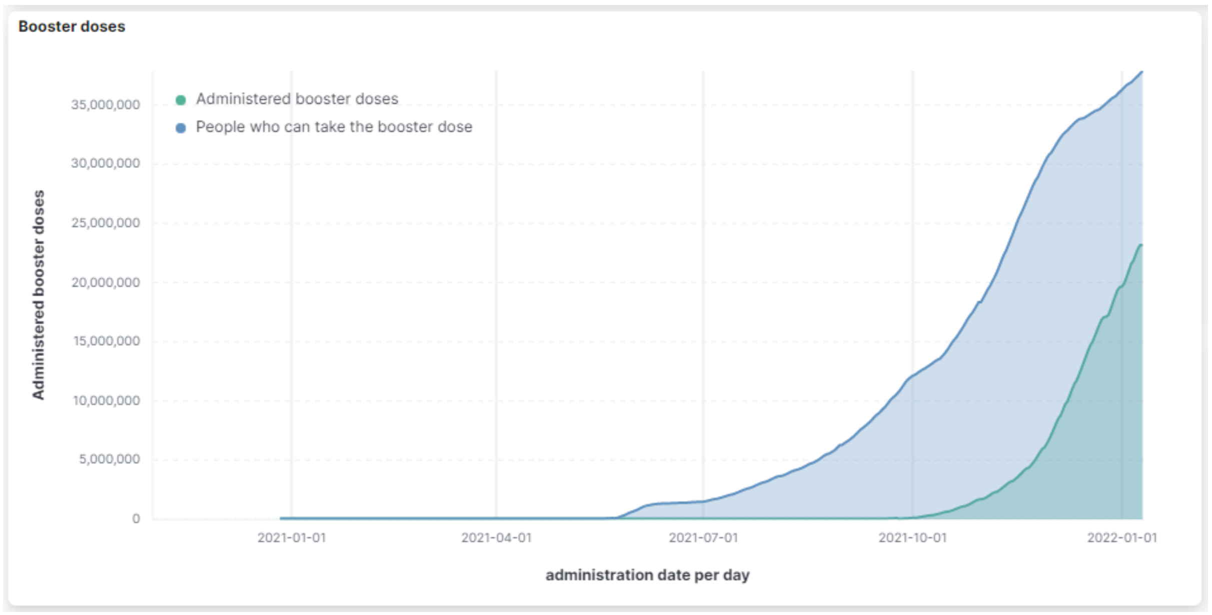
The dashboard allows you to select which regions and age groups you want to consider in your analysis. If none is selected, everyone is considered. The time interval can be set from the usual Kibana control, and you may further refine the documents to be considered using the Kinana query panel. This control is experimental: in case of any issue, you can ignore it and use the Kibana query box to define the regions to consider.
We believe that the dashboard is more useful if the time filter includes the whole available documents: on the 10 of January, we consider 14 months (rounded to the month) till now.
The dashboard is divided in two parts: the first shows cumulative data for all the regions you have selected, while the second one divides the data between the regions, allowing you to make comparisons.
We will indicate which visualizations are based on the queries that we provide in section 4. Note that Kibana Lens doesn't generate exactly the same query. First of all, it filters the time interval, region and age group you have selected. Then, it may also use a different version of the query. For instance, Kibana doesn't ask Elasticsearch for cumulative sums, instead it computes it on its own.

3.C.1 Information not divided by region

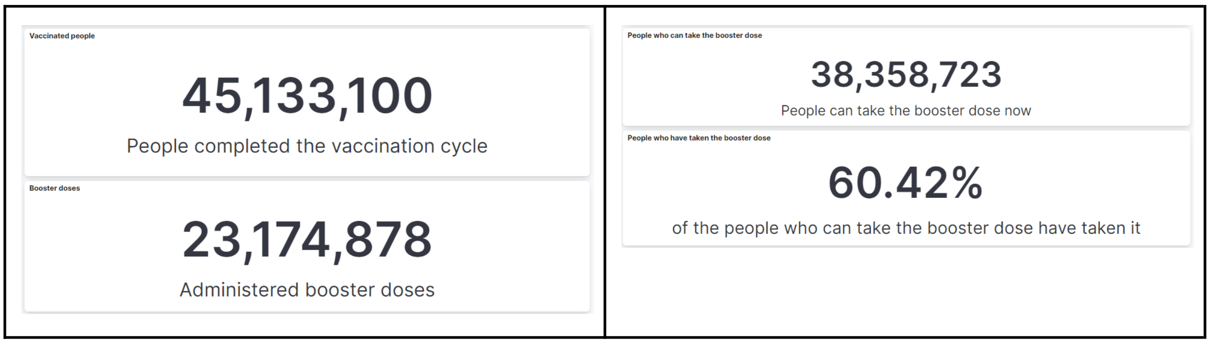
The first part includes visualizations about the cumulative data of all the regions and age groups selected.



The first visualization is a graph showing the number of people that completed the vaccination cycle over time. This visualization refers to the query 4.A.1, but Kibana uses a different version of it. In particular, the granularity is changed to the single day, and Kibana computes the cumulative sum by itself.



The dashboard continues with another graph on the left, showing the number of administered booster doses (green area) and the number of people that can take them (blue area) over time. This way, you can easily compare how many booster doses were administered and how many people could take them.


Vaccinated people

45,133,100

People completed the vaccination cycle

Booster doses

23,174,878

Administered booster doses

People who can take the booster dose

38,358,723

People can take the booster dose now

People who have taken the booster dose

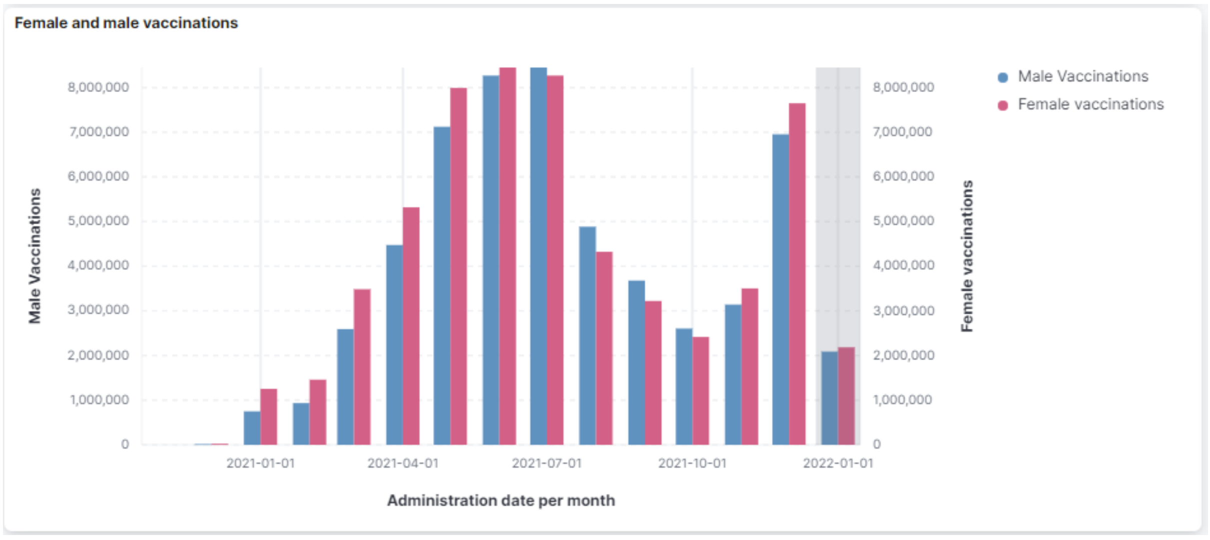
60.42%

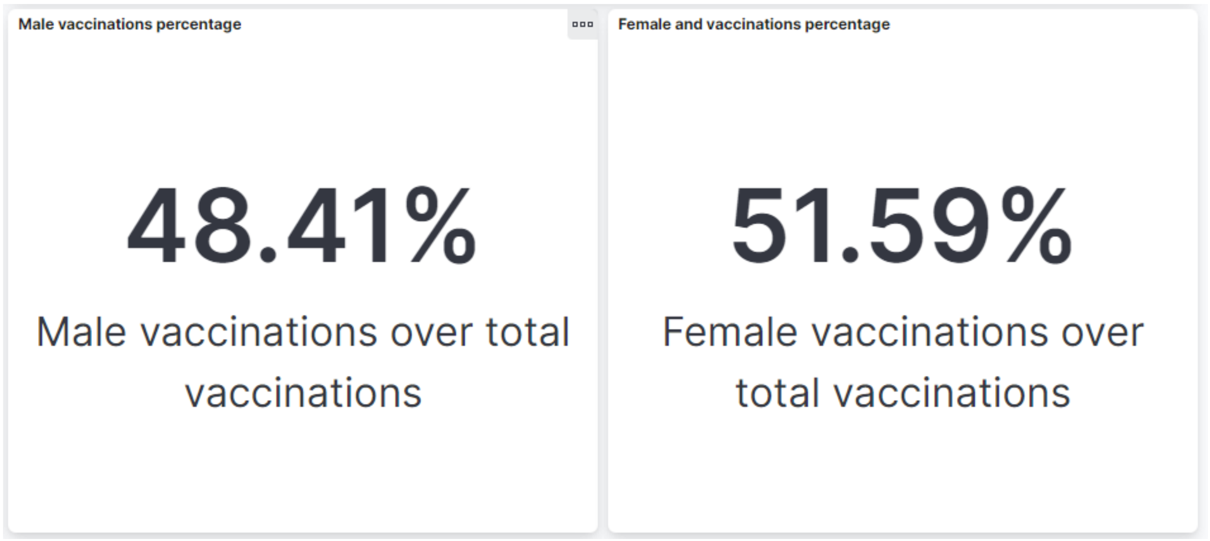
of the people who can take the booster dose have taken it

On the right of the booster doses graph there are four metrics:

- the number of people that completed the vaccination cycle
- the number booster doses that were administered
- the number of people that can take the booster dose
- the percentage of people who have taken the booster dose (over the ones who can take it)

The first metric is a simple sum, while the other 3 are based on query 4.3.2 (Kibana divides it in two queries for the first two metrics, and computes the percentage of the last metric by itself from the two results provided by the query).




Male vaccinations percentage

48.41%

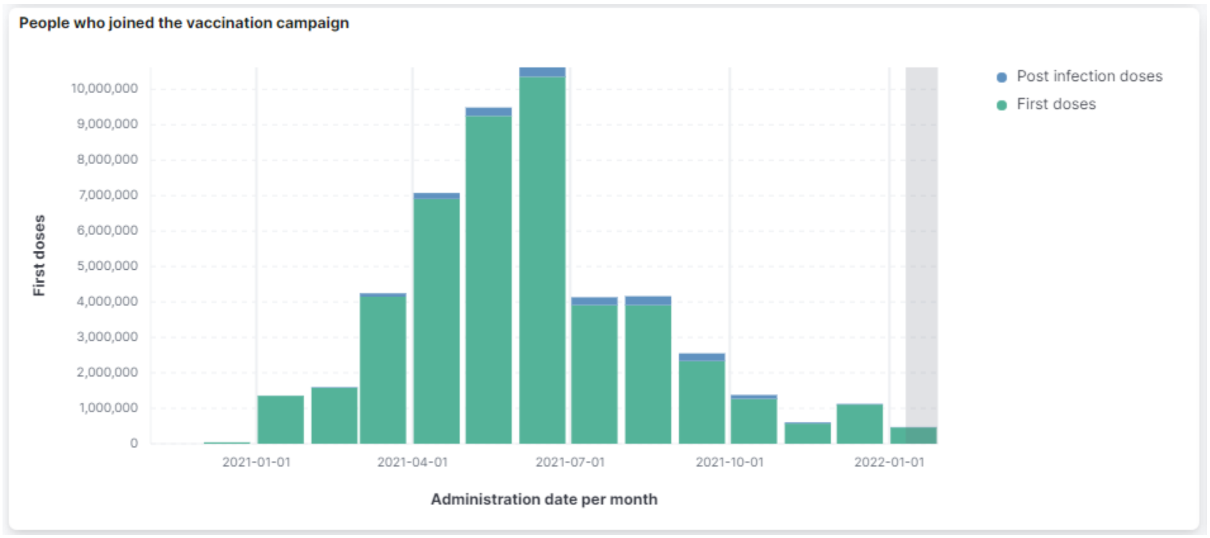
Male vaccinations over total vaccinations

Female and vaccinations percentage

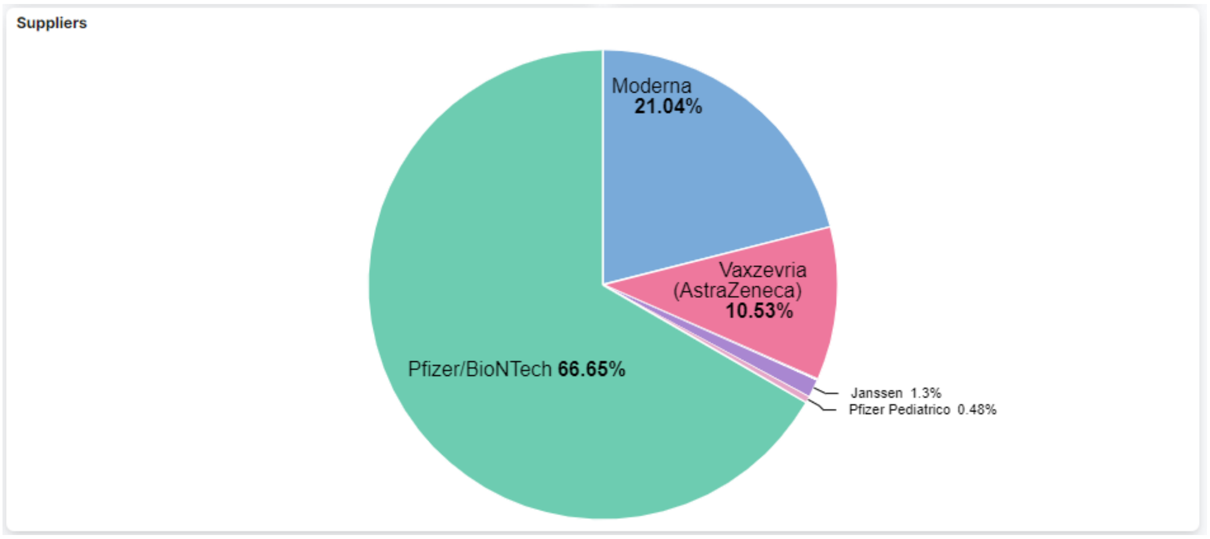
51.59%

Female vaccinations over total vaccinations

The dashboard continues with visualizations regarding male and female vaccinations. An histogram shows, for each month (identified by its first day), the number of doses administered to males (blue) and to females (pink) during the month. The graph will alert you if the time filter you set truncates a month. On the right you see two metrics that show the percentage of doses administered to males and to females respectively. The histogram is based on query 4.A.3.



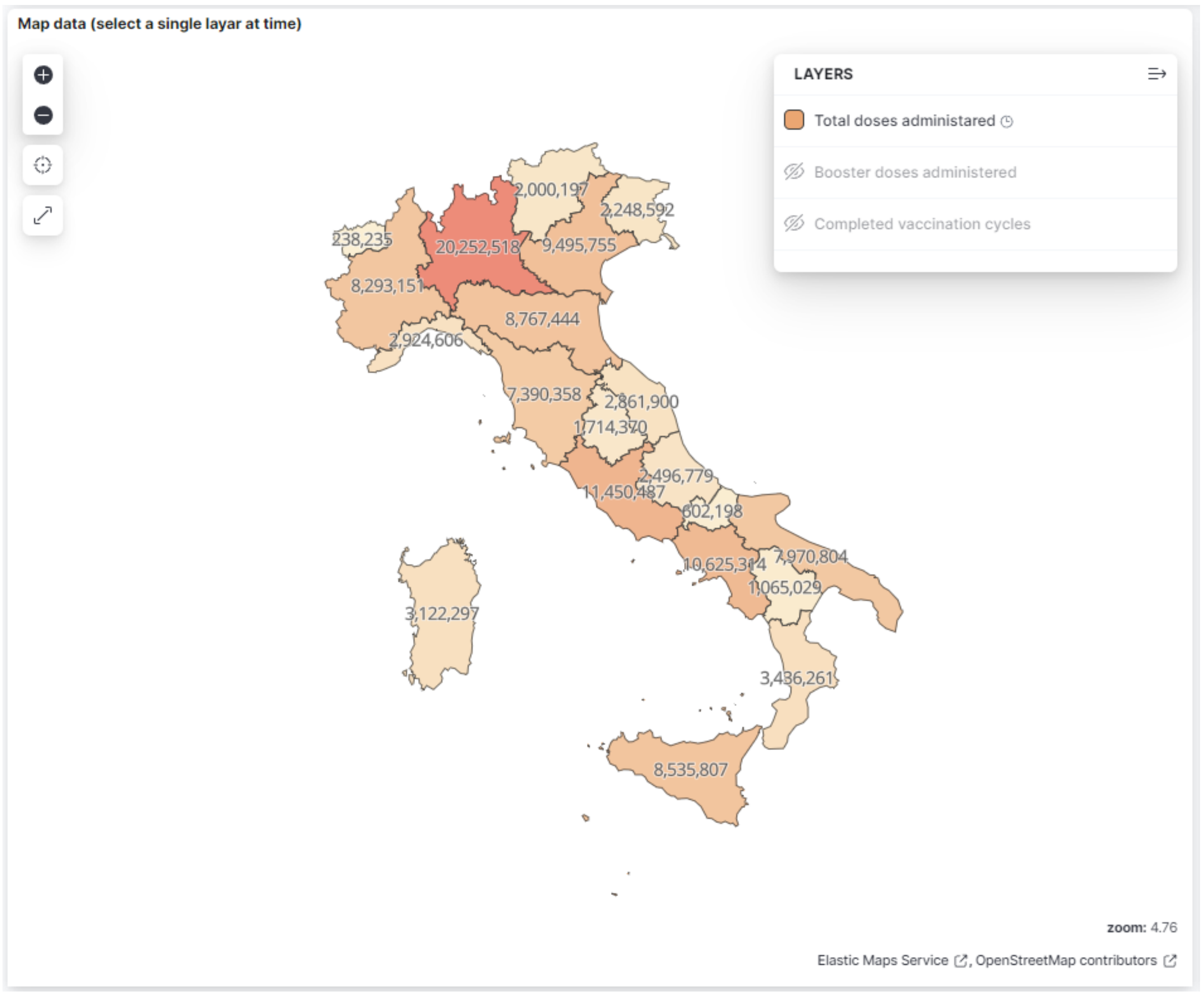
The last histogram shows the first and post infection doses that were administered each month. As before, Kibana alerts you if the time filter truncates a month. This chart is useful to see how many people joined the vaccination campaign each month. First doses (green) and post infection doses (blue) are shown separately, one on top of the other. This is based on query 4.A.4.



This section ends with a pie chart showing how many doses were administered for each supplier. The pie chart shows you the percentage of doses, but you can see the actual number of doses placing the cursor over it. It's based on query 4.A.5.

3.C.2 Information divided by region

This section of the dashboard shows data divided by region. Remember that it considers only the regions you have selected.



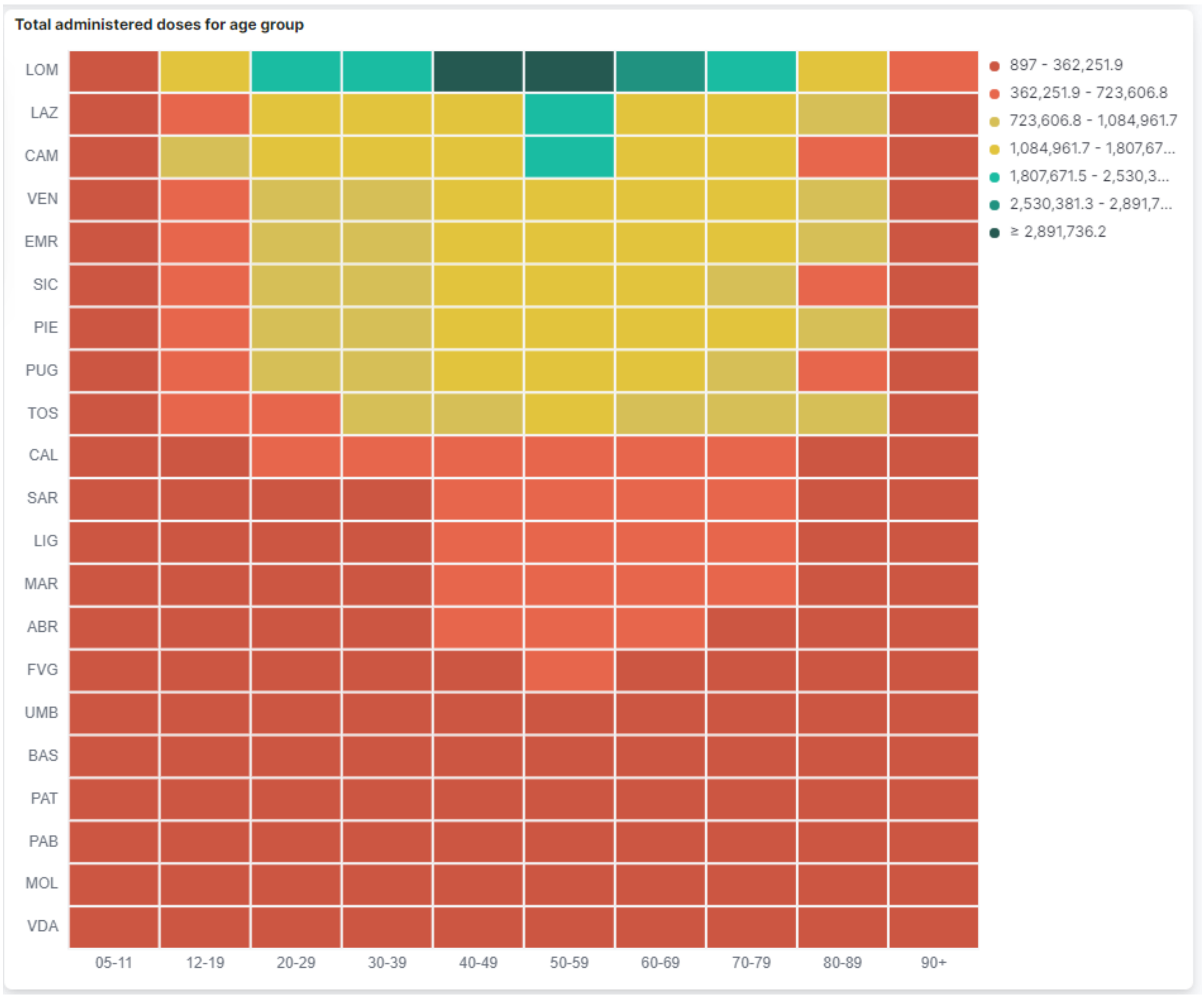
The section starts with a map showing the regions you have selected. The map has three levels. Each level shows a particular metric for each region and colors the map based on it (darker color for region with higher metric). The three metrics are:

- the total number of administered doses in the region (red layer)
- the total number of booster doses administered in the region (blue layer)
- the total number of people that completed the vaccination cycle in the region (green layer)

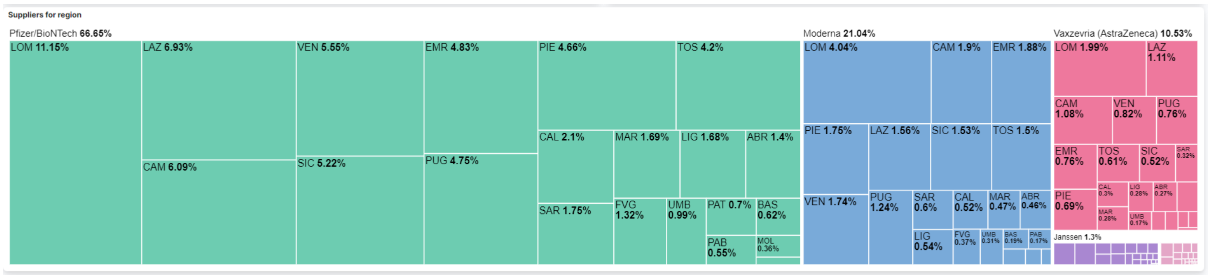
The visualization allows you to select which levels to show. We suggest selecting only one level at a time.

This visualization uses the istat code to map our documents to the map. Since the map uses a two digits code, we need to format it correctly when importing the dataset.

The map doesn't allow complex queries, so we needed to create the two redundant fields to show them.



On the right there is a heatmap showing, for each region and for each age group, the number doses that the region administered to that age group. You can see the actual value of a cell placing the cursor over it. The cell color ranges from red (low number of doses) to green (high number of doses). It's based on query 4.A.6.



The dashboard ends with a tree map. This map shows the percentage of doses administered from each supplier (similarly to the last visualization of the first section). Then it shows, for each supplier, how its doses were distributed across the regions (considering the doses administered so far). Note that it shows only the actually used doses, not the stored, lost or expired ones.

4. Queries and commands

4.A Queries

This section describes some queries that retrieve relevant statistical information from the database. Some of them are not well-suited to be represented in a Kibana visualization and have been omitted in the dashboard. We are interested in aggregations only, so they specify “size”:0 for the number of documents to retrieve. As a convention, we use a limit of 100 buckets in aggregations instead of the default one (10). This makes sure that we return all the buckets when we aggregate by the region, supplier or age group. It also makes it possible to upload a dataset of another nation or with another age group subdivision and still be able to use these queries, as long as the limit of 100 regions, age groups and suppliers is respected.

For parameterized queries, we express the parameters as {{parameter_name}}, that is the same syntax used by search templates. This way they can be easily converted to search templates if required. We prefer to report the query (GET method) directly so that it can be tested using a single http method only, instead of requiring a PUT to upload the template and a GET to use it.

If not otherwise specified, we identify the regions by their area (the acronymous).

We provide a short description of the result of each query, focusing on the relevant fields inside the **aggregations** inner document. This is a simplified description that should be enough to interpret the result, but the actual response contains many other fields with additional data. Describing the complete answer is outside the scope of this document. We also provide a partial version of the result, with the most relevant values in bold.
Many queries compute the trend of some metrics over time. When this is done for each month, each month is identified by its first day.

4.A.1 **Vaccinated people each month**

This query returns, for each month, the number of people that completed the vaccination cycle that month (**fully_vaccinated_this_month**) and the cumulative number of people that completed the vaccination cycle from the beginning to that month included (**cumulative_fully_vaccinated**).

```
GET /vaccinations/_search
{
  "size": 0,
  "aggs": {
    "months": {
      "date_histogram": {
        "field": "administration_date",
        "calendar_interval": "month"
      },
      "aggs": {
        "fully_vaccinated_this_month": {
          "sum": {"field": "fully_vaccinated_count"}
        },
        "cumulative_fully_vaccinated":{
          "cumulative_sum": {"buckets_path": "fully_vaccinated_this_month"}
        }
      }
    }
  }
}
```

Partial result:

```
"aggregations" : {
    "months" : {
      "buckets" : [
        {
          "key_as_string" : "2020-12-01T00:00:00.000Z",
          "key" : 1606780800000,
          "doc_count" : 425,
          "fully_vaccinated_this_month" : {
            "value" : 54.0
          },
          "cumulative_fully_vaccinated" : {
            "value" : 54.0
          }
        },
        …..
```

4.A.2 **People who can take the booster dose**

This query returns the number of people that can currently take the booster dose (**people_who_can_take_the_booster_dose**), and the number of booster doses administered so far (**administered_booster_doses**).

```
GET /vaccinations/_search
{
  "size": 0,
  "aggs": {
    "records_before_4_months": {
      "filter": {
        "range":{
          "administration_date":{
            "lte": "now-4M/d"
          }
        }
      },
      "aggs": {
        "people_who_can_take_the_booster_dose": {
          "sum": {
            "field": "fully_vaccinated_count"
          }
        }
      }
    },
    "administered_booster_doses":{
      "sum": {
        "field": "booster_doses"
      }
    }
  }
}
```
Partial result:
```
"aggregations" : {
    "administered_booster_doses" : {
      "value" : 2.3174878E7
    },
    "records_before_4_months" : {
      "doc_count" : 121829,
      "people_who_can_take_the_booster_dose" : {
        "value" : 3.8358723E7
      }
    }
  },.....
```

4.A.3 **Male and female vaccinations**

This query returns the number of doses administered to males and females respectively each month. In the result, buckets are inside **vaccinations_for_month** and each one includes the number of male vaccinations (**male**) and female ones (**female**) for its own month.

```
GET /vaccinations/_search
{
  "size": 0,
  "aggs": {
    "vaccinations_for_month": {
      "date_histogram": {
        "field": "administration_date",
        "calendar_interval": "month"
      },
      "aggs": {
        "male": {
          "sum": {"field": "male_count"}
        },
        "female": {
          "sum": {"field": "female_count"}
        }
      }
    }
  }
}
```

Partial result:

```
"aggregations" : {
    "vaccinations_for_month" : {
      "buckets" : [
        {
          "key_as_string" : "2020-12-01T00:00:00.000Z",
          "key" : 1606780800000,
          "doc_count" : 425,
          "female" : {
            "value" : 24075.0
          },
          "male" : {
            "value" : 16692.0
          }
        },.......
```

4.A.4 **People who joined the vaccination campaign**

This query returns the number of people that joined the vaccination campaign each month (**people_who_joined_the_campaign**), that are people who got either the first dose or the post infection one. Notice that for each month it returns the number of people that joined the campaign that specific month only.

```
GET /vaccinations/_search
{
  "size": 0,
  "aggs": {
    "new_vaccinated_for_month": {
      "date_histogram": {
        "field": "administration_date",
        "calendar_interval": "month"
      },
      "aggs": {
        "people_who_joined_the_campaign": {
          "sum": {"script":
"doc['first_doses'].value+doc['post_infection_doses'].value"}
        }
      }
    }
  }
}
```

Partial result:

```
"aggregations" : {
    "new_vaccinated_for_month" : {
      "buckets" : [
        {
          "key_as_string" : "2020-12-01T00:00:00.000Z",
          "key" : 1606780800000,
          "doc_count" : 425,
          "people_who_joined_the_campaign" : {
            "value" : 40767.0
          }
        },......
```

4.A.5 **Administered doses for supplier**

This query returns the number of doses that have been administered for each supplier. The result consists in a list of buckets (**administered_doses_per_supplier**), each one indicating the supplier (**key**) and the number of doses administered by that supplier (**administered_doses**). Suppliers are ordered in lexicographical order.

```
GET /vaccinations/_search
{
  "size": 0,
  "aggs": {
    "administered_doses_per_supplier": {
      "terms": {
        "field": "supplier",
        "size":100,
        "order": {"_key": "asc"}
      },
      "aggs": {
        "administered_doses": {
          "sum": {"field": "administered_doses"}
        }
      }
    }
  }
}
```

Partial result:

```
"aggregations" : {
    "administered_doses_per_supplier" : {
      "doc_count_error_upper_bound" : 0,
      "sum_other_doc_count" : 0,
      "buckets" : [
        {
          "key" : "Janssen",
          "doc_count" : 18602,
          "administered_doses" : {
            "value" : 1501444.0
          }
        },.....
```

4.A.6 **Administered doses for region and age group**

This query counts the number of doses that each region administered to each group. Regions and age groups follow the lexicographical order. The result consists of a list of buckets (**region**). Each of them contains the region (**key**) and another array of buckets (**age_group**). Inside this last list, each bucket contains the age group (key) and the number of administered doses (**administered_doses**). Regions are ordered alphabetically, while age groups are ordered from the younger one to the older one.

```
GET /vaccinations/_search
{
  "size": 0,
  "aggs": {
    "region": {
      "terms": {
        "field": "area",
        "size":100,
        "order": {"_key": "asc"}
      },
      "aggs": {
        "age_group":{
          "terms": {
            "field": "age_group",
            "size":100,
            "order": {"_key": "asc"}
          },
          "aggs": {
            "administered_doses": {
              "sum": {"field": "administered_doses"}
            }
          }
        }
      }
    }
  }
}
```

Partial result:

```
"aggregations" : {
    "region" : {
      "doc_count_error_upper_bound" : 0,
      "sum_other_doc_count" : 0,
      "buckets" : [
        {
          "key" : "ABR",
          "doc_count" : 8621,
          "age_group" : {
            "doc_count_error_upper_bound" : 0,
            "sum_other_doc_count" : 0,
            "buckets" : [
              {
```

```
            "key" : "05-11",
            "doc_count" : 19,
            "administered_doses" : {
              "value" : 9244.0
            }
          },
```

4.A.7 Average vaccinations over day in the last week

This query returns the average number of doses administered per day in the last week in a region (parameter {{region_area}}). If the data is not up to date and there aren't records in the last days, those days aren't considered. You may want to increase the number of considere days to test the query in that case.
In particular, the result consists of a list of the considered days (**day**) with the number of administered doses for each of them (**vaccinations**), while the average administered doses per day is inside **vaccinations_over_day**.

```
GET /vaccinations/_search
{
  "size": 0,
  "query":{
    "bool": {
      "filter": [
        {"term": {
          "area": "{{region_area}}"
        }},
        {"range":{
          "administration_date":{
            "gte": "now-7d/d"
          }
        }}
      ]
    }
  },
  "aggs": {
    "day": {
      "date_histogram": {"field": "administration_date",
"calendar_interval":"day"},
      "aggs": {
        "vaccinations": {
          "sum": {"field": "administered_doses"}
        }
      }
    },
    "vaccinations_over_day": {
      "avg_bucket": {
        "buckets_path": "day>vaccinations",
        "gap_policy" : "insert_zeros"
      }
      }
    }
}
```

Partial result (region = "VDA"):

```
"aggregations" : {
    "day" : {
      "buckets" : [
        {
          "key_as_string" : "2022-01-03T00:00:00.000Z",
          "key" : 1641168000000,
          "doc_count" : 18,
          "vaccinations" : {
            "value" : 1100.0
          }
        },
        …….
      ]
    },
    "vaccinations_over_day" : {
      "value" : 1323.142857142857
    }
```

4.A.8 **Region with the highest number of administered booster doses**

This query considers the regions inside a group (identified by the NUTS1 code, parameter {{NUTS1_code}}). It returns the number of booster doses administered by each region of the group in the last week, and the region inside the group which administered the greatest number of them.
In particular, the result consists of a list of the considered regions (**region**) with the number of administered booster doses for each of them (**boosters**), while the one that administered the most of them is the **key** inside **best_region_for_boosters.**

```
GET /vaccinations/_search
{
  "size": 0,
  "query":{
    "bool": {
      "filter": [
        {"term": {
          "NUTS1_code": "{{NUTS1_code}"
        }},
        {"range":{
          "administration_date":{
            "gte": "now-7d/d"
          }
        }}
      ]
    }
  },
  "aggs": {
    "region": {
      "terms": {"field": "area","size": 100},
      "aggs": {
        "boosters": {
          "sum": {"field": "booster_doses"}
        }
      }
    },
    "best_region_for_boosters": {
      "max_bucket": {
        "buckets_path": "region>boosters",
        "gap_policy" : "insert_zeros"
      }
    }
  }
}
```

partial result (NUTS1_code=”ITC”):

```
"aggregations" : {
    "region" : {
      "doc_count_error_upper_bound" : 0,
      "sum_other_doc_count" : 0,
      "buckets" : [
        {
          "key" : "PIE",
          "doc_count" : 138,
          "boosters" : {
            "value" : 221674.0
          }
        },
        …….
      ]
    },
    "best_region_for_boosters" : {
      "value" : 556488.0,
      "keys" : [
        "LOM"
      ]
    }
  }
```

4.A.9 Average number of doses by a supplier

This query considers a single region (parameter {{region}}) and a single supplier (parameter {{supplier}}). It returns the average number of doses per month by that supplier used in that region.

In particular, the result consists of the number of doses by that supplier used each month in the region (**months**) and the average of them (**avg_doses_per_month**).

```
GET /vaccinations/_search
```

```
{
  "size":0,
  "query":{
    "bool": {
      "filter": [
        {"term": {
          "area": "{{region}}"
        }},
        {"term":{
          "supplier":{
            "value":"{{supplier}}"
          }
        }}
      ]
    }
  },
  "aggs":{
    "months":{
      "date_histogram": {"field": "administration_date","calendar_interval":
"month"},
      "aggs":{
        "monthly_doses":{
          "sum":{
            "field":"administered_doses"
          }
        }
      }
    },
    "avg_doses_per_month":{
      "avg_bucket": {
        "buckets_path" : "months>monthly_doses",
        "gap_policy" : "insert_zeros",
        "format" : "0"
      }
    }
  }
}
Partial result (region=”LOM”, supplier=”Moderna”):
"aggregations" : {
    "months" : {
      "buckets" : [
        {
          "key_as_string" : "2020-12-01T00:00:00.000Z",
          "key" : 1606780800000,
          "doc_count" : 1,
          "monthly_doses" : {
            "value" : 1.0
          }
        },
        …….
      ]
    },
    "avg_doses_per_month" : {
```

```
    "value" : 333030.5714285714,
    "value_as_string" : "333031"
  }
}
```

4.B Commands

This section describes some commands that may be used to update the database, uploading new documents. These commands make use of ingest pipelines and enrich indexes to automatically insert redundant data. For each command we first describe the steps required to create the pipelines and enrich indices (that should be executed only once), then we provide the command itself. Pipelines will also add the "@timestamp" field that is used by Kibana to filter the documents.
As for the queries, parameters are provided with the format "{{parameter_name}}".

4.B.1 Uploading a new record with the same data of a line of the dataset

This command can be used to upload a new document given the same fields that are used in the dataset. This can be useful to upload new documents from the same source in order to keep the data up to date.
We need a pipeline to automatically compute the redundant fields "fully_vaccinated_count" and "administered_doses", and to format the "region_ISTAT_code" field.
The pipeline can be uploaded with the following command:

```
PUT _ingest/pipeline/automatic_sum_fields_and_format_istat_code
{
  "description": "Ingest pipeline to automatically add the administered_doses
and fully_vaccinated_count fields and pad the istat code. It also add the
timestamp field",
  "processors": [
    {
      "script": {
        "description": "set administered_doses field",
        "lang": "painless",
        "source": "ctx['administered_doses'] =
ctx['male_count']+ctx['female_count'];"
      }
    },
        {
      "script": {
        "description": "set fully_vaccinated_count field",
        "lang": "painless",
        "source": "ctx['fully_vaccinated_count'] =
ctx['post_infection_doses']+ctx['second_doses'];"
      }
    },
        {
      "script": {
        "description": "pad the istat code to two digit",
        "lang": "painless",
        "source": "ctx['region_ISTAT_code'] = String.format('%02d', new def[]
{Integer.parseInt(ctx['region_ISTAT_code']) });"
      }
    },
    {
      "date": {
        "field": "administration_date",
        "timezone": "Europe/Rome",
        "formats": [
          "ISO8601"
        ]
      }
    }
  ]
}
```

The command itself is the following:

```
POST /vaccinations/_doc?pipeline=automatic_sum_fields_and_format_istat_code
{
  "administration_date": "{{administration_date}}",
  "supplier": "{{supplier}}",
  "area": "{{area}}",
  "age_group": "{{age group}}",
  "male_count": {{male count}},
  "female_count": {{female count}},
  "first_doses": {{first doses}},
  "second_doses": {{second doses}},
  "post_infection_doses": {{post infection doses}},
  "booster_doses": {{booster doses}},
  "NUTS1_code": "{{NUTS1 code}}",
  "NUTS2_code": "{{NUTS2 code}}",
  "region_ISTAT_code": "{{ISTAT code}}",
  "region_name": "{{region name}}"
}
```

Parameters shall respect the same format used in the dataset. In particular, the region name should use the same convention (it must contain the translation when needed and the right capital letters).

4.B.2 Uploading a new record from another source

This command is a more robust version of the previous one: it automatically computes redundant data. In particular it computes the NUTS codes, the ISTAT code and the region name from the area (i.e., the region acronyms). It can be useful if the data comes from a different source (e.g. directly from the region, which may not provide the region data since it's always the same for that

region: you have to simply add one code and other fields are added automatically) to make sure those fields are coerent.
In order to execute this command you have to prepare the appropriate enrich index and inget pipeline in Elasticsearch:

1. Create the regions index
2. Create and execute the enrich policy
3. Create the ingest pipeline

A detailed description of these steps is provided:

1. regions index

Create an index called **regions** uploading the data from the file **regions.csv** (that is under the Data folder), using the following mapping and ingest pipeline (they are not the automatic ones):
Mapping:

```
{
  "properties": {
    "NUTS1_code": {
      "type": "keyword"
    },
    "NUTS2_code": {
      "type": "keyword"
    },
    "area": {
      "type": "keyword"
    },
    "region_ISTAT_code": {
      "type": "keyword"
    },
    "region_name": {
      "type": "keyword"
    }
  }
}
```

Ingest pipeline:

```
{
  "description": "Ingest pipeline created by text structure finder",
  "processors": [
    {
      "csv": {
        "field": "message",
        "target_fields": [
          "area",
          "NUTS1_code",
          "NUTS2_code",
          "region_ISTAT_code",
          "region_name"
        ],
        "ignore_missing": false
      }
    },
    {
      "remove": {
        "field": "message"
      }
    }
  ]
}
```

2. Add and execute the enrich policy

Upload the enrich policy with the command:

```
PUT /_enrich/policy/region_area_policy
{
  "match": {
    "indices": "regions",
    "match_field": "area",
    "enrich_fields": ["NUTS1_code", "NUTS2_code","region_ISTAT_code",
"region_name"]
  }
}
```

And execute it to create the enrich index:

```
PUT /_enrich/policy/region_area_policy/_execute
```

3. Create the ingest pipeline

We need a pipeline to use the enrich index to enrich the data. The pipeline will also move the new fields from the inner object where they are created to the root of the json document, compute the other two redundant fields and add the timestamp field.

```
PUT _ingest/pipeline/automatic_region_enrich
{
  "description": "Ingest pipeline to enrich the data retrieving the region
data from the area. Need to use scripts to rename fields, moving them from
the inner document to the root one (using the rename processors cause
problems with the enrich index).",
  "processors": [
    {
      "enrich": {
        "policy_name": "region_area_policy",
        "field": "area",
        "target_field": "temp"
      }
    },
    {
      "script": {
        "lang": "painless",
        "source": "ctx['NUTS1_code'] = ctx.temp.NUTS1_code;"
      }
    },
    {
      "script": {
        "lang": "painless",
        "source": "ctx['NUTS2_code'] = ctx.temp.NUTS2_code;"
      }
    },
    {
      "script": {
        "lang": "painless",
        "source": "ctx['region_ISTAT_code'] = ctx.temp.region_ISTAT_code;"
      }
    },
    {
      "script": {
        "lang": "painless",
        "source": "ctx['region_name'] = ctx.temp.region_name;"
      }
    },
    {
      "script": {
        "description": "set administered_doses field",
        "lang": "painless",
        "source": "ctx['administered_doses'] =
ctx['male_count']+ctx['female_count'];"
      }
    },
        {
      "script": {
        "description": "set fully_vaccinated_count field",
        "lang": "painless",
        "source": "ctx['fully_vaccinated_count'] =
ctx['post_infection_doses']+ctx['second_doses'];"
      }
    },
    {
      "remove": {
        "field": "temp"
      }
    },
    {
      "date": {
        "field": "administration_date",
        "timezone": "Europe/Rome",
        "formats": [
          "ISO8601"
        ]
      }
    }
  ]
}
```

4. Command

The command can now be executed as many times as you want. The command is the following:

```
POST /vaccinations/_doc?pipeline=automatic_region_enrich
{
  "administration_date": "{{administration date (format yyyy-mm-dd)}}",
  "supplier": "{{supplier}}",
  "area": "{{area}}",
  "age_group": "{{age group}}",
  "male_count": {{male count}},
  "female_count": {{female count}},
  "first_doses": {{first doses}},
  "second_doses": {{second doses}},
  "post_infection_doses": {{post infection doses}},
  "booster_doses": {{booster doses}}
}
```

4.B.3 Uploading multiple records

This command is based on the previous one and can be used to import multiple records in a faster way. It can be useful, for instance, to upload, for the same record, one document for each age group. The command is the following:

```
POST /vaccinations/_bulk?pipeline=automatic_region_enrich
{ "create":{ "_index": "vaccinations"} }
{"administration_date": "{{administration date}}","supplier":
"{{supplier}}","area": "{{area}}","age_group": "{{age group}}","male_count":
{{male count}},"female_count": {{female count}},"first_doses": {{first
doses}},"second_doses": {{second doses}},"post_infection_doses": {{post
infection doses}},"booster_doses": {{booster doses}}}
{ "create":{ "_index": "vaccinations"} }
{"administration_date": "{{administration date}}","supplier":
"{{supplier}}","area": "{{area}}","age_group": "{{age group}}","male_count":
{{male count}},"female_count": {{female count}},"first_doses": {{first
doses}},"second_doses": {{second doses}},"post_infection_doses": {{post
infection doses}},"booster_doses": {{booster doses}}}
…
```

You can have multiple lines following the same pattern. Each even line (the second, the fourth…) has the parameters for a new document to be inserted.