UNIVERSITÀ DI PISA

DEPARTMENT OF COMPUTER SCIENCE
Data Science and Business Informatics

Laboratory Of Data Science Project

Data Integration, Analysis and Visualization

Assessment Data Analysis of Student Performances

Ludovico Lemma 637757
Davide Innocenti 640090

l.lemma@studenti.unipi.it
d.innocenti6@studenti.unipi.it

Academic year 2022/2023

Contents

1 Introduction

In this report we are going to describe the various pre-processing and ETL steps we performed on a given dataset regarding answers of individuals to various multiple-choice questions delivered by different organizations. We wil then answer some business questions with a set of multiple tools.
The aim of the first step is to create detailed tables to synthesize a Data Warehouse into SQL Server Management Studio so that it could be queried for business purposes.

The data was provided through two different CSV files: "answer_full.csv" and "subject_metadata.csv". The former contains the main body of the of data, 538.835 and 17 dimensions, regarding both information about individuals and organizations, it was presented as a single table with the following features:

```
ColumnIndex(['QuestionId', 'UserId', 'AnswerId', 'CorrectAnswer', 'AnswerValue',
'Gender', 'DateOfBirth', 'PremiumPupil', 'DateAnswered', 'Confidence', 'GroupId',
'QuizId', 'SchemeOfWorkId', 'SubjectId', 'RegionId', 'Region', 'CountryCode'])
```

The latter is instead used to provide details about the “SucjectId” column since it contains an explicit reference to a hierarchy of 4 different levels for a total of 388 subjects. The columns were presented as it follows:

```
ColumnIndex(['SubjectId', 'Name', 'ParentId', 'Level'])
```

A preliminary Data Exploration task has been performed in order to check for incongruities. Two dimensions have been taken into account:

- Missing Values
- Data Integrity

Luckily enough, no missing values have been found in the dataset, while for the data integrity check, every column has been analyzed and no record had consistency issues.

1.1 Creating the Data Warehouse Schema

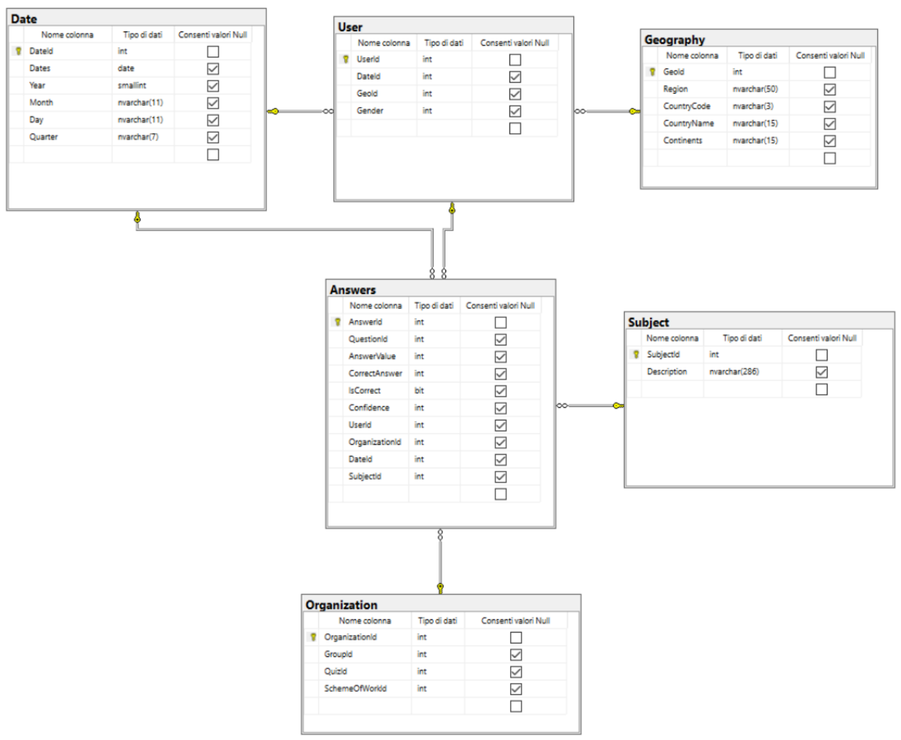

Figure 1: Data Warehouse Schema

For the true first step of the project we decided to design the Snow Flake schema of the Data Warehouse on Microsoft SQL Server Management Studio like depicted in the above picure.
We set integers as the data types of the primary keys of the tables we had to create, we discarded from the original Dataset the features "PremiumPupil" and "RegionId", the first one because it wasn't requested according to the business objectives, the second one because it wasn't meaningful as it was a direct mapping of the country code which wasn't the feature on which we needed to build the key in the Geography table, and it was semantically wrong as the feature Region had an higher grain of detail.
For the Description column in the subject table we assigned a place holder at first but later we determined nvarchar(286) by measuring the largest description we had as a result of the explicitation of the ids.

1.2 Preprocessing the Dataset

In this step we dealt with the general tasks of reading and processing the CSV files, generating the tables and the primary keys, managing the peculiarities of the different tables and finally mapping the primary keys to the values they referred to as foreign keys in the original fact table or the tables higher in their hierarchy.
The functions were written with the scope of being as general as possible to mantain and adapt them to other contexts in the future. The only libraries we used in the development of the functions were the python standard "re", used to manage the CSV default reading standard (we implemented only the two cases of "commas" and "double quotes"), and the pycountry_convert library, used to get both the country and continent name from the ISO country codes.
Several tasks have been performed to adapt the tables and satisfy the requirements of the business questions for the Data Warehouse:

- **IsCorrect** measure has been generated to check, for each individual, the correctness in answering the questions it was computed by comparing values of the "CorrectAnswer" and "AnswerValue" columns.
- The **Subject** table has been made by only using the "SubjectId" and "Description" dimensions. The latter is an extended representation of sorted subjects according to the level order, from the most general one (e.g. Math) of the subject list to most specific ones. In order to do that every distinct list of the old "SubjectId" column in the answer_full.csv has been overwritten with the explicit description sorted by level.
- The **Geography** table primary key ("GeoId") has been generated by mapping a concatenation of the "Region" and "CountryName" columns (with Region being the most important having the highest grain). The additional "Continent" and "CountryName" dimensions were also requested. The library "pycountry_convert" has been exploited to map the ISO country codes to the desired features, with only some minor adjustments needed for the United Kingdom.
- In the **Date** table, we accommodated both dates of birth of users and for dates of answers to save space. For this reason, we to performed a set union of both columns, for a total amount of 596 dates. From this column we then built the derived dimensions requested by also preserving the hierachy (Year, Month, Day and Quarter). The corresponding "DateId" was later mapped on the User table "DateOfBirth" dimension and to the Fact Table dimension "DateOfAnswer".

In the end, every table has been assigned its own integer progressive primary key by considering the distinct values of sorted records, then the primary key was mapped back to the original (not distinct) dimensions used to build it, to the source data, so that a foreign key could be obtained for a related table.

1.3 Uploading the Tables to the Server

In this final step of this part of the project we developed a single class to manage the entire process of loading the table to the server. The class accepts as input a CSV loaded as a dictionary as in the preprocessing step (with each column as a value of the column name), and the destination table in the server which will store a Data Warehouse. In order for the class to work a schema as to be already defined. After establishing a connection (for which a credentials.txt file with user id and password is needed in the same folder) the schema's types will be fetched from the server in order to be casted so that the input data will match the requirements (if possible), there is also a check on the header to be uploaded and the one expected from the server so that an error would be raised if the two wouldn't match.

Then a parametric query is gonna be generated to insert the rows into the remote table, by matching the header's columns and the number parameters "?" to be inserted as in the following example:

```
INSERT INTO [Table_Name] (Column0, Column1, ...)  VALUES (?,?, ...)
```

In order to keep things simple we decided to intend the Class object to work as a "first-only upload", indeed we established a routine of deleting all previous data from the table expected to be uploaded to avoid inconsistencies. We managed also to catch possible errors about integrity of possible related tables, so that also data present in the table's hierarchy will be deleted. Finally the class uploads the records, making a maximum of 10 attempts to execute the `INSERT INTO` query, and committing every 100 queries. After a few tests with some functionalities we concluded that the most efficient way to insert data was executing multiple queries at once, with multiple `INSERT INTO` query. We reduced the upload time to 2 minutes from 15 minutes, also reducing the log strains on the server.

2 SISS Solutions

In this section we are going to use Sequel Server Integration Services (**SSIS**) to solve some business question on the warehouse we just created. All the computation are executed on client side without the aid of any SQL query.

2.1 Business Question 1

For every subject, the number of correct answers of male and female students.

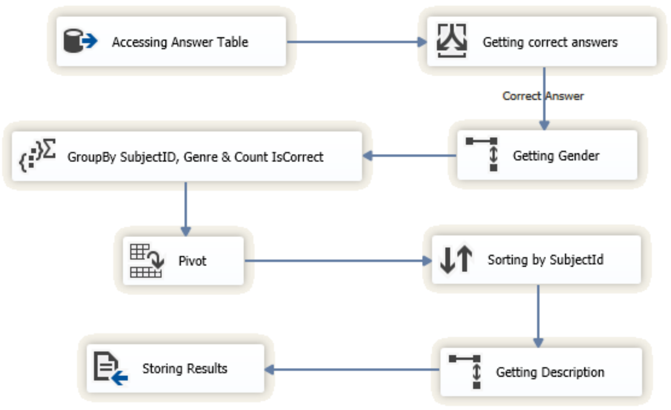

Figure 2: Business Question 1

After having accessed to the Answer Table by an **Origin OLE DB** node we first discard incorrect answers to make the computation lighter by cutting off almost half of the records, then we retrieve Gender attribute by **Looking Up** for that column in the User Table, assuming 1 as Female and 2 as Male. Thereafter we proceed with grouping up tuples by SubjectID and Genre, counting the number of correct answers for each of them.
At this point of the computation the task is already completed but we added few more nodes for visualization purposes:
A **Pivot** node has been used in order to have each SubjectID on a single line followed by two more columns (*Female_nCorrectAnswers* and *Male_nCorrectAnswers* respectively) and to drop the 0 genre, followed by a **Sorting** node and a final Look Up to the Subject Table to get the full Description of the SubjectID itself.
Finally, the resulting table is canalized into a *Destination File Flat* node to output the answer in a .csv file.

2.2 Business Question 2

A subject is said to be easy if it has more than 90% correct answers, while it is said to be hard if it has less than 20% correct answers. List every easy and hard subject, considering only subjects with more than 10 total answers.

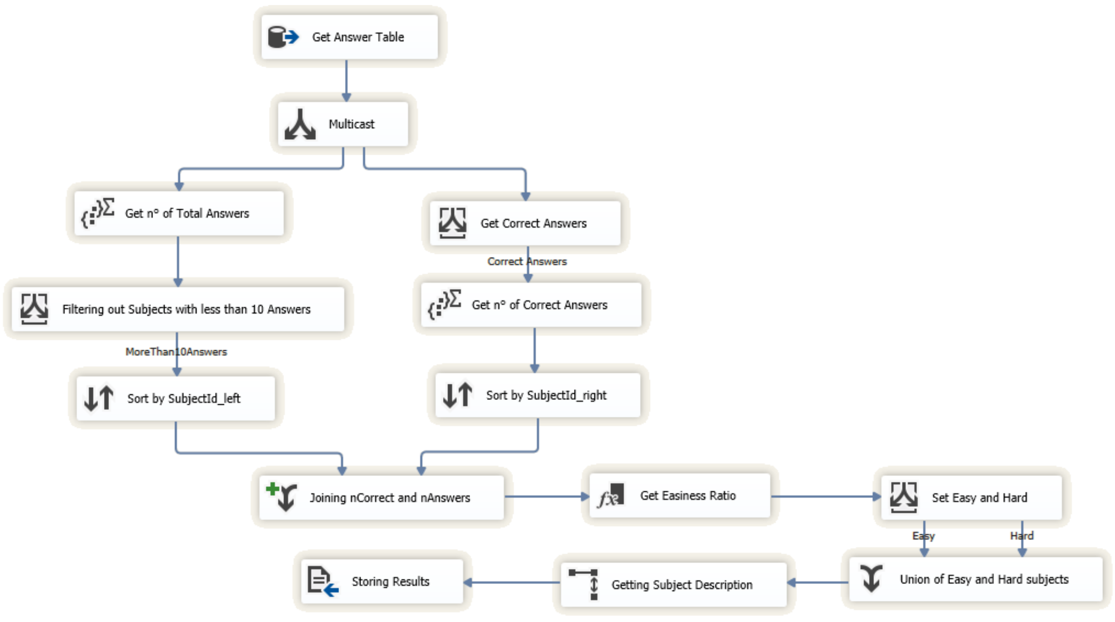

Figure 3: Business Question 2

The first connection is once again to the Answer table, followed by a **Multicast** node.
In this case we had to retrieve few information in order to answer the business question, we therefore split the flow by computing on the left the amount of Total Answers by SubjectId filtering on those with more than 10 answer given, and, on the right, we kept only the correct answers grouped by Subject id, in order to finally join them both in a single table, being ready to compute the ratio of correctness (a.k.a EasinessRateo) for every SubjectId.
The latter is estimated in the following **Derived Column** node, as well as two more columns (EasyCol and HardCol) which are subsequently used to split on Easy answers (*easiness ratio* >0.9) and Hard ones (*easiness ratio* <0.2).

Then, the two columns have been collapsed in a single one by using a **Union** node, resulting in a table with only few SubjectId left with their own easiness label attached to. For a better understanding, the Description column in the Subject table has been again retrieved, and everything has been stored in the corresponding .csv file.

2.3 Business Question 3

For each country, the student or students that answered the most questions correctly for that country.

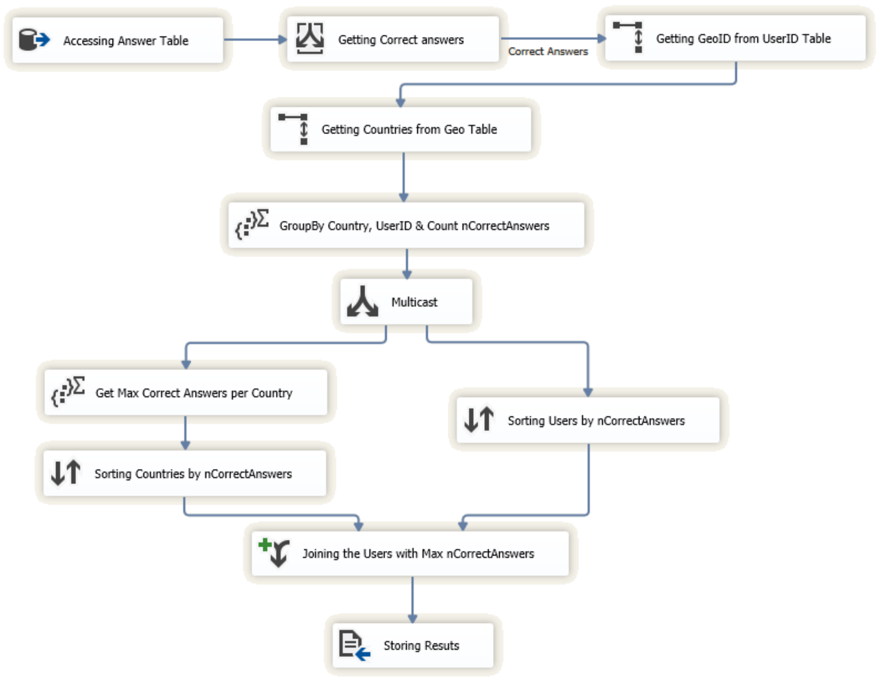
Accessing Answer Table
Getting Correct answers
Correct Answers
Getting GeoID from UserID Table
Getting Countries from Geo Table
GroupBy Country, UserID & Count nCorrectAnswers
Multicast
Get Max Correct Answers per Country
Sorting Users by nCorrectAnswers
Sorting Countries by nCorrectAnswers
Joining the Users with Max nCorrectAnswers
Storing Resuts

Figure 4: Business Question 3

The access to the Answer Table has been exploited once again to get the few columns needed for the computation: SubjectId and IsCorrect measure.
Since the question straightly ask for countries, we had to get them by retrieving few columns in order: User Table to get the GeoId FK and then Geography Table for the attribute CountryName.
After having grouped by Country and UserId and having counted the number of correct answers a **Multicast** node has been used to split the flow in two ways in order to compute the Maximum number of correct answer and the respective associated UserId eventually joining the two created flow by correspondence of the user id with the Maximum number of correct answers given.
Finally, the result is stored in the last .csv file.

3 Multidimensional Data Analysis

3.1 OLAP Cube

The first step taken to create the OLAP cube was to connect to our server **Group_10_DB** through Visual Studio and then we created a View on the data by selecting all the tables.
A few dimensions have been created, i.e. *User*, *Date*, *Organization*, *Subjects* and *Answers*; not all of them were really needed but we decided to keep them all for completeness purposes. Different consideration must be taken into account for the *Geography*: a first is due to the fact it is already embedded under User in a hierarchy and it is indeed needed to collect eventually user's region or country (unique for each user). In order to create cube's measures we then decided to insert some **Calculated fields** inside the **Answer** starting table:

- *CorrectIsCorrect* which by manipulating the *IsCorrect* measures now casts binary operator (i.e. True/False) as integers to easily count correct answers, by using the formula:

```
case when "IsCorrect" = 'True' then 1 else 0 end
```

- *WrongIsCorrect*, like in the previous example returns the opposite, inserted for computation purposes.

3.1.1 Creation of the Dimension User

This is our most important dimension due to the fact that Users have different links to temporal and geographical dimensions. Two Hierarchy have been created, one regarding Dates and the other strictly related to the geography dimension, as it follows:

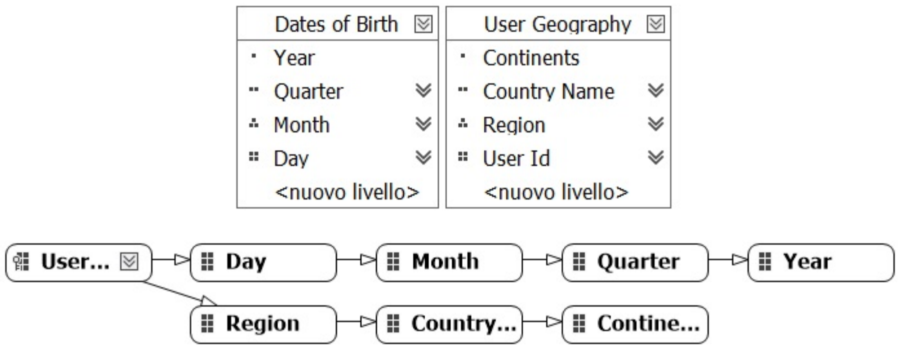

Figure 5: User Dimension

By developing this hierarchy every user is eventually linked to Geography dimension (a user is the highest spatial granularity), by following the schema: *Region* → *Country* → *Continent*, with the latter being the highest level of the hierarchy.
Also, a similar reasoning is applied to date connection in order to have a more complete view on users temporal dimension: every user will have its own **DateOfBirth** hierarchy by respecting the schema: *Day* → *Month* → *Quarter* → *Year*.
No further particular action have been taken w.r.t. ordering these ladders (i.e. ordering by *AttributeKey, Key, etc.*) since everything was loaded correctly sequenced.

3.1.2 Creation of the Dimension Date

This dimension, if available, is commonly imported and exploited for cubes operations since it comprehends different levels of granularity. For this reason and by considering the date w.r.t the answering occasion, a new hierarchy has been created.

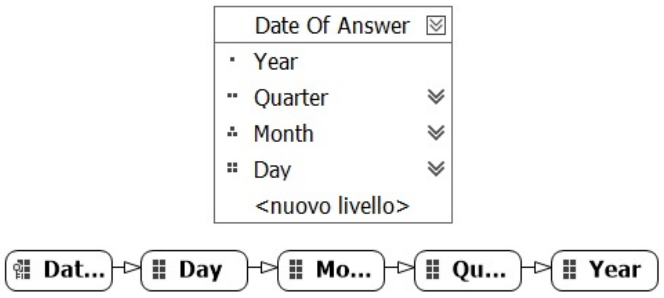

Figure 6: User Dimension

Once again, there was no need to adjust any intrinsic ordering since everything was already set up.

3.1.3 Cube Deployment

Finally, once the dimensions have been created, the OLAP cube has been deployed by exporting all the data on server side.
The cube has been set up as **MOLAP** (Multidimensional Online Analytical Processing) in order to make it optimized for "slicing and dicing" operations and data retrieval, while **Cache** Partitioning has not been taken into account since it is normally used for repeatedly queried cubes, while it was out of the scope of the work requested.
The **Answers** table is of course the one reported as fact table in which few new measures have been computed by scripting the previously mentioned *CorrectIsCorrect* and *WrongIsCorrect*.

Indeed, our available final measures are: **Correct**, **Wrong**, **AnswerCount**, respectively representing the sum of correct and wrong answers extracted from the Answer Table and the count of non-empty values w.r.t the known answer correctness (non null IsCorrect). Wrapping up, our cube is now composed by the above mentioned measures and 5 dimensions with their own internal hierarchy: **Answer**, **User**, **Date**, **Subjects** and **Organization** (the latter just for completeness).

4 MDX Queries

We were asked to answer some business question on the Datacube just deployed.
The queries were the following:

4.1 Query 1

Show the student that made the most mistakes for each country.

```
with member max_by_country as
                    max( ([User].[Country Name].CurrentMember,
                          [User].[User Id].[User Id]),
                          [Measures].[Wrong] )

select max_by_country on axis(0),
NONEMPTY( FILTER( ([User].[Country Name].[Country Name],
                   [User].[User Id].[User Id]),
                   [Measures].[Wrong] = max_by_country) ) on axis(1)
from [Group 10 DB];
```

This query has an execution time of few seconds. We exploited the **Wrong** measure we created before the cube deployment to use it during the creation of the member max_by_country, which gives us a metric to filter on during the select phase. Indeed, by cutting off nonempty tuples and filtering out those records not having a Wrong count equal to our member max_by_country we were able to answer the first query.

4.2 Query 2

For each subject, show the student with the highest total correct answers.

To answer this business question we proposed different solutions, considering various circumstances:

- Only the first 1st ranking student appears in list
- Every 1st ranking students in terms of correct answers appears in the list

The latter is obviously slower than the one depicted here, chosen for reporting reasons (the other strategy is simple: computing the max and instead of output 1 in the topcount output a number equal to the count of users that scores the maximum via a filter):

```
with set subject_user as
    NONEMPTY( GENERATE( [Subject].[Subject Id].[Subject Id].MEMBERS,
                    TOPCOUNT( [Subject].[Subject Id].CURRENTMEMBER *
                              [User].[User Id].[User Id].MEMBERS,
                              1,
                              [Measures].[Correct]) ))

select [Measures].[Correct] on axis(0),
order(subject_user, [Measures].[Correct], BDESC) on axis(1)
from [Group 10 DB];
```

By creating the **subject_user** set to subsequently use it in the select clause, we evaluate *TopCount* function w.r.t. Subject_Id members to extract for each subject only the single User_Id the the highest correctness count. We made the query easier to read by pre-computing the set in the set assignment.

4.3 Query 3

For each continent, show the student with the highest ratio between his total correct answers and the average correct answers of that continent.

```
with member avg_correct as
(ANCESTOR([User].[Geography].CURRENTMEMBER, 3), [Measures].[Correct]) /
(ANCESTOR([User].[Geography].CURRENTMEMBER, 3), [Measures].[AnswerCount])

member correct_ratio as
([User].[Geography].CURRENTMEMBER, [Measures].[Correct]) /
 avg_correct

select {[Measures].[Correct], avg_correct, correct_ratio} on axis(0),
NONEMPTY( GENERATE( [User].[Continents].[Continents].MEMBERS,
                        TOPCOUNT( [User].[Continents].CURRENTMEMBER *
                                      [User].[Geography].[User Id],
                                      1,
                                      correct_ratio))) on axis(1)
from [Group 10 DB];
```

Here we had two possible interpretations: the case for which we were requested to count the continental average user correctness, or the continental average answer correctness. In the first case we would have needed a count of unique UserIds, while in the latter we opted for we needed AnswerCount with which we counted non null IsCorrect values (as we would need to know the correctness of an answer to consider it).

5 Data Visualization

As for the final task of the whole process we created a few dashboards to visualize some of the information of our Datacube by using a graphical tool such as Microsoft **PowerBI** which is commonly used to report purposes.

5.1 Geographical distribution of *correct answers* and *incorrect answers*

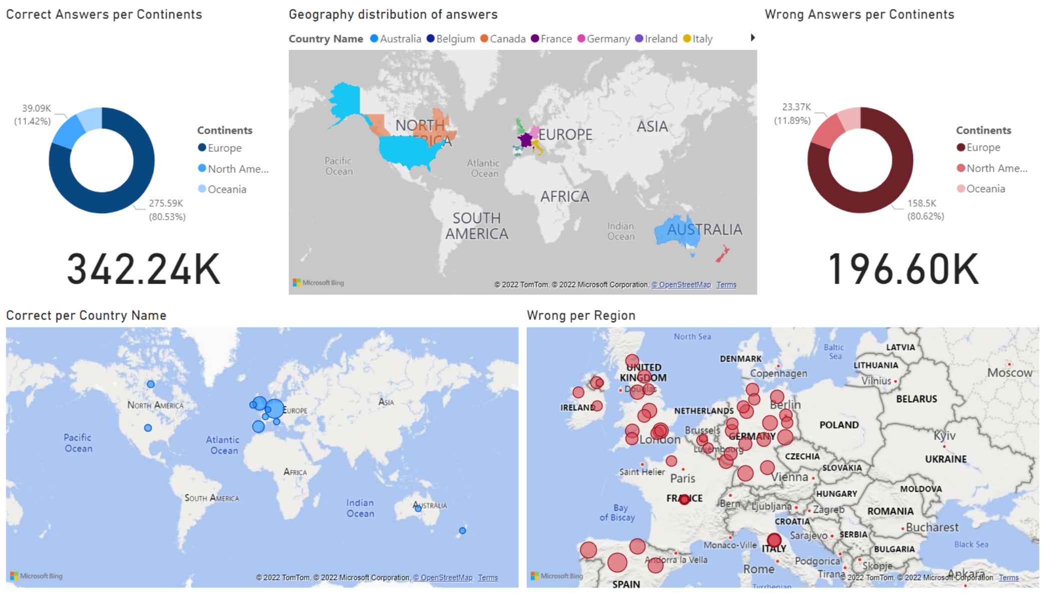

Figure 7: Wordwide Correctness Distribution

For this task we employed different charts: at first we can see on both sides donut charts, respectively counting *Correct Answers* on blue and *Wrong Answers* on red: these are stratified charts created to better explore hierarchical relationships, showing up that *Europe* is the one with the highest amount of answers given worldwide.
In the middle a Choropleth map is depicted, showing the Geography Distribution of the answers all over the word. The latter is useful to immediately show that, accordingly to the data we had at

the beginning, only data of a sub portion of continents was gathered; Continents such Asia, South America and Africa are not present in the visualization while *Europe*, *North America*, *Oceania* and the closest islands are instead depicted.
At the bottom of the report two World Heatmaps are proposed to further explore the data. Once again, correct answers (with National granularity) in blue on the left, and wrong ones in red on the right.
On the latter the granularity has been purposely set up to be per Region in order to have a deeper understanding of the regional distribution of answers, and how it is possible to observe (and confirm by considering the blue Heatmap) Europe is again the most interesting. It is not possible to import dynamic visualization inside this static document, we therefore depicted a "zoom-in" towards Europe as it follows:

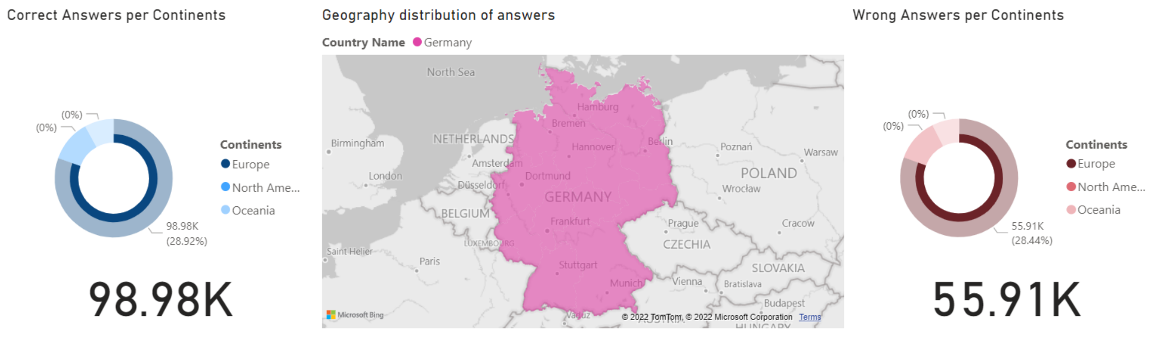

Figure 8: Germany Correctness Distribution

As it is shown, all the charts respect the proportion by going deeper inside the geographical hierarchy. More specifically, by only click on the designed are, is it possible to visualize counts of more specific area, such as Germany in this case.

5.2 Custom Dashboard

Finally, this last Dashboard has been created in order to have a wider view with reference to correct answers and Users gender/age.
First of all we displayed the distribution of correct answers with a stacked-bar plot. In this chart it is possible to observe that the 3 continents share the same distribution between correct and wrong answer, with a mean value of 60% of overall correctness.

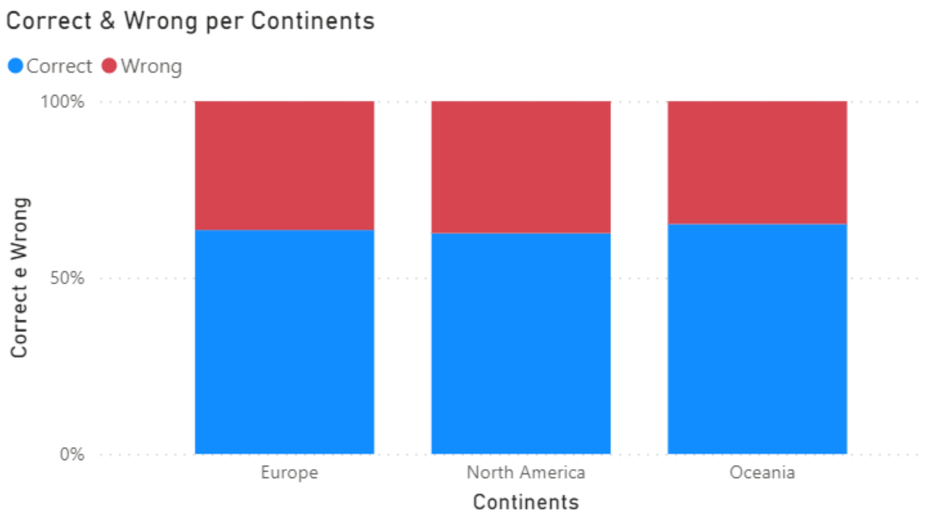

Figure 9: Correctness BarPlot

Then we moved to to visualize Users by year and gender.
At first we plotted, through the aid of a donut chart, how each gender is represented by each country; the legend is sorted based on the count of answers per country: Germany, UK and Spain obtain of course the podium in this representation. The charts shows how each of them actually preserve the same gender weight.
Our data seems to be well balanced when it comes to gender classes: forgetting about the 0 class (unidentified gender) it will be easy to check that both masculine and feminine gender share the same distribution, as well as in the correctness one.
Hence, this is clearly confirmed by the following ribbon chart:

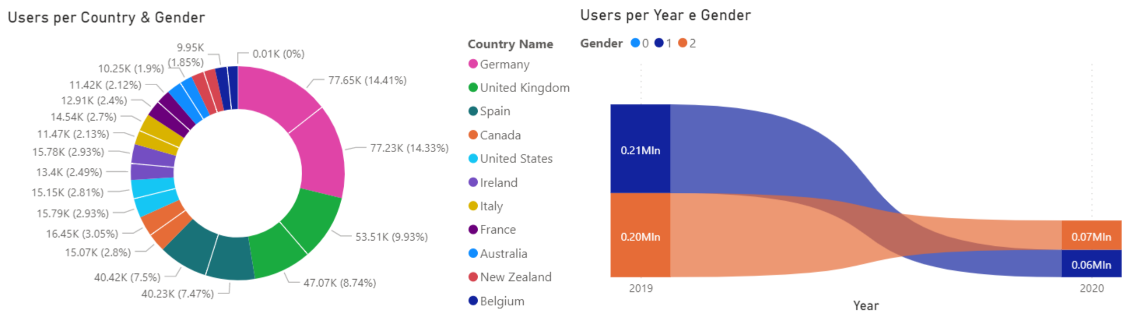

Figure 10: Gender distribution

As it can be observed, while the distribution is preserved, by adding up the temporal dimension it is instead shown that the amount of answering people tends to drop by more than the half, starting from more than 200k per gender in 2019 to no more than 70k in 2020.
Regarding temporal data, as we anticipated in previous sections the data warehouse kept a double date relation, both for the date of answer and the date of birth of the individuals.
The latter has been used for this plot to show that people interviewed were selected mostly from a small year range, i.e. around 2006-2008, considering the range 2000-2014.
Once again, the gender distribution is kept unmodified.

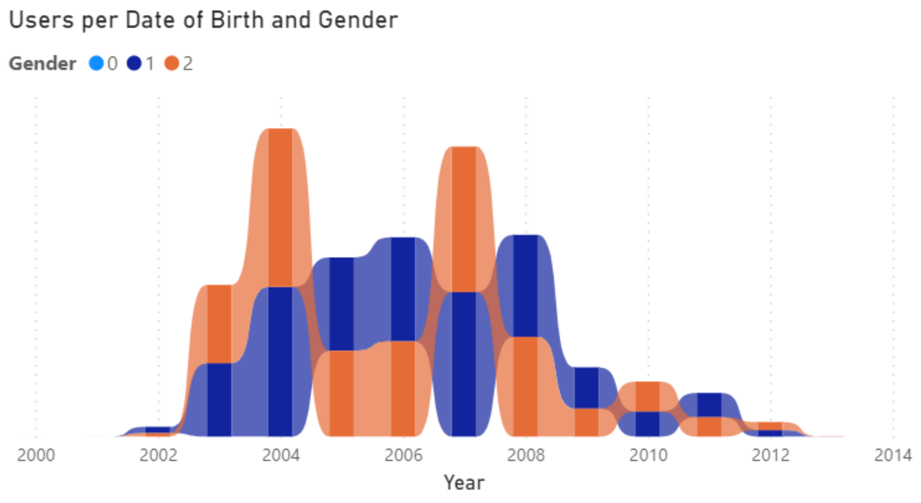

Figure 11: Gender distribution by DateOfBirth

In the end, as last plot we displayed the temporal evolution of the answers given across the years.

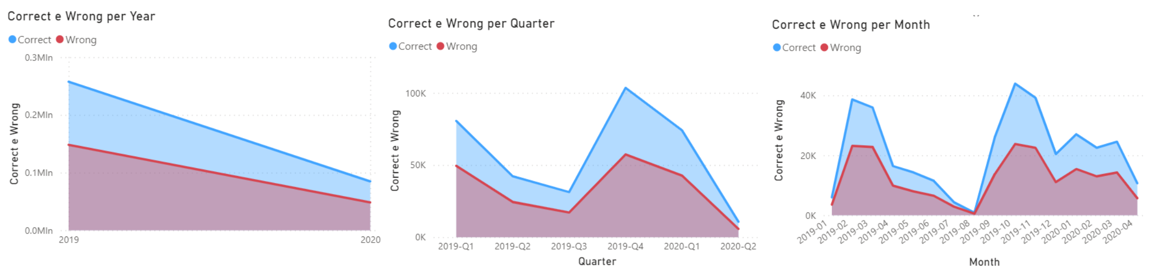

Figure 12: Dates of answering

Apart from Days, which is composed by a long list of elements, the entire Date's hierarchy, i.e. Years, Quarters and Months has been depicted in this chart showing the descending trend of registered answers from 2019-Q1 to 2020-Q4. As shown above, there is also a specific time frame in which there is a decisive drop in the answer count, more precisely in the Q3 of 2019, between August and September, probably meaning that during that specific period there was a drop in responses due to seasonality (probably summer holidays), at least for that specific year.