Università di Pisa

Laboratory of Data Science

Report for the project assignment

Laboratory of Data Science
AY 2022/2023

Group 14
Lorenzo Pieri 578814
Rocco Tiesi 636678

1 Part 1: Data Preparation

Introduction

The first part of the "Laboratory of Data Science" project, consists in creating and populating a database, and to perform various activities on two files. We were given two csv files:

- `answers_full`: it is a table with data regarding answers given by students to different multiple – choice questions. There are also data about the questions, the students, and the subject of the questions.
- `subject_metadata`: it is a table that contains information about the subject of each question. The subject is given in a way that can be used to index the other file to retrieve the topic of the question.

1.1 Assignment 0

As concerns assignment 0, we follow the representation of the Data Warehouse schema to reproduce it using SQL Server Management Studio and upload it in the server of the university's department. Firstly, we create each dimension table and the fact table adding the corresponding primary and foreign keys. Secondly and lastly, we assign the correct relations between tables.

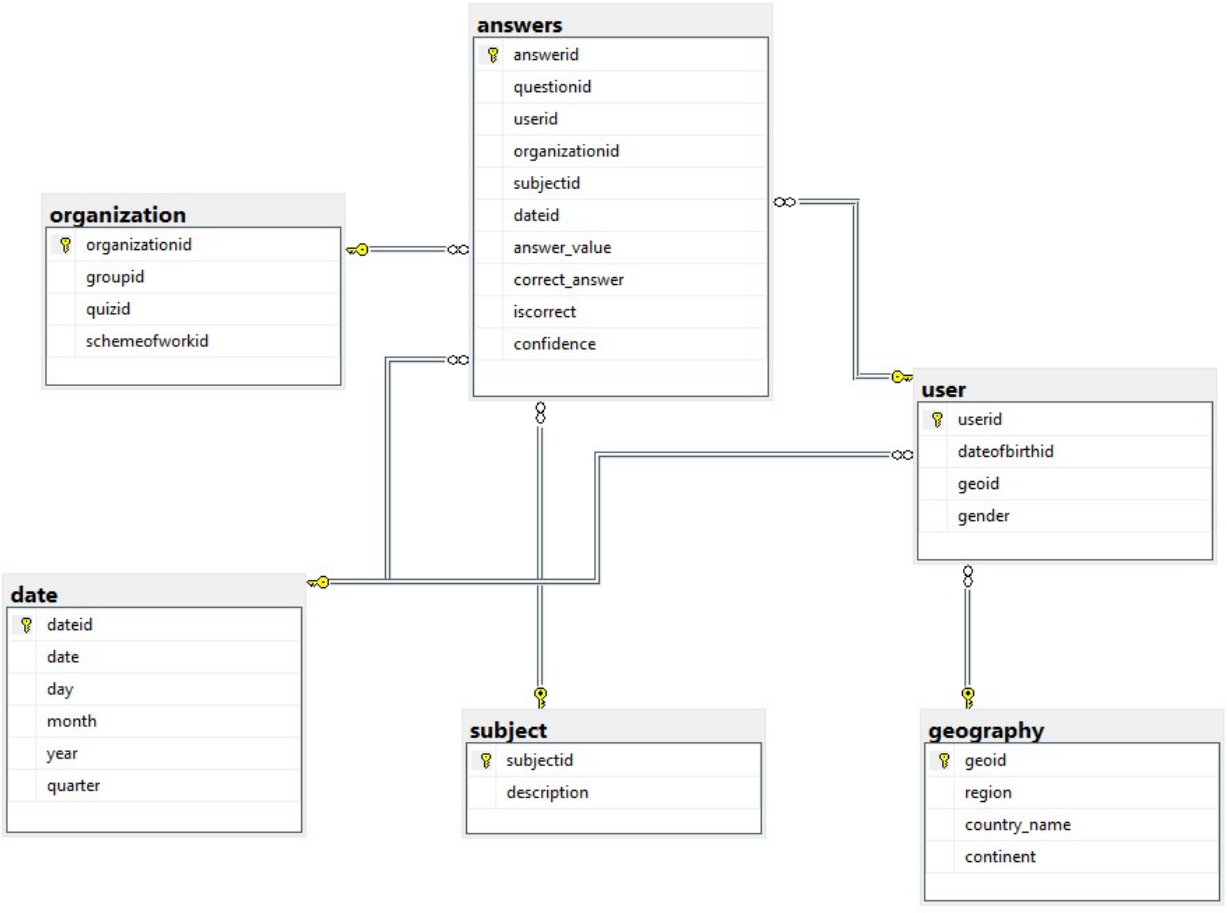

Figure 1: Database Schema

answers	
answerid	*int*
questionid	*int*
userid*	*int*
organizationid*	*int*
subjectid*	*int*
dateid*	*int*
answer_value	*int*
correct_answer	*int*
iscorrect	*bit*
confidence	*float*

user	
userid	*int*
dateofbirthid*	*int*
geoid*	*int*
gender	*int*

subject	
subjectid	*int*
description	*nvarchar(MAX)*

date	
dateid	*int*
date	*date*
day	*int*
month	*int*
year	*int*
quarter	*nvarchar(7)*

organization	
organizationid	*int*
groupid	*int*
quizid	*int*
schemeofworkid	*float*

geography	
geoid	*int*
region	*nvarchar(50)*
country_name	*nvarchar(50)*
continent	*nvarchar(50)*

Table 1: Tables and attribute types

1.2 Assignment 1

Assignment 1 requires writing a Python program that creates the six tables of the Data Warehouse from the `answers_full` file. The first function *"openCSVfile"* is used to extract all the content of the file and put it in a list of lists. Each list contains all the elements of a single row of the csv file.
Since we need to find a way to integrate the name of the continents related to each country, we use an additional csv file "country-codes" to extrapolate the names of the continents. The function that associates each country to the relative continent is *"generateCountryDict"*. Given the list extracted from the csv file of country codes, it generates a dictionary with country code as a key and the corresponding continent as value. It returns also another dictionary with the country codes as keys and the country names as values.

The function *"generateContinentColumn"* creates the column "Continents" in the original list with all the data extracted from the `answers_full` file. Firstly, we add at the first list of the original list (therefore, the list regarding the header) the name of the column (i.e., "Continents"). Subsequently, for each row of the list, the function checks the country code at position 17, and then adds to the list the correct continent name extracted from the dictionary. Also, we decide to use this loop to replace every country code with its correspondent country name, using the dictionary created before. Eventually, it returns the update list with the new column.

To create the values of the "isCorrect" attribute, we need to compare the variables `"answers_value"` and `"correct_answer"`. For doing this task, we write the function *"generateIsCorrectColumn"*. With this function a new column regarding the attribute is added to the list. Going into details, each element of "isCorrect" contains value 1 if the correct answer and the answer given by the user is the same. Whereas, they have value 0 otherwise.

The function *"getQuarter"* returns the correct quarter given the month and the year (e.g., January 2020 = `Q1_2020`). The table "Date" is created using firstly the function "generateDateDict". It extracts all the dates from the original file and generates the list of dictionaries regarding all the elements that will form the final table. The second function that we implement to achieve our goal is *"getDateKey"*, it returns the "dateid" given the list of dates and a single date.

The table "Organization" is created using the function *"generateOrganizationColumn"*. It is used to create the two new columns "organization" and "organizationid". The first one contains the tuples of the three values needed to generate the table organization. To generate their ids, we first order all the records by the values of the tuples. In this way, we can assign quickly the id comparing the current tuple with the previous one. If they are the same, we assign the current id, otherwise we increment the id "counter" and we assign it to the record.

The function *"generateTables"* creates all the tables (answers, organization, subject, user, geography; date was already generated). It generates the tables in the form of a list of dictionaries according to the attributes of the tables on the SQL server. We specify a list of attributes for each table, so that the function

can generate a list of dictionaries for each of them.

In order to solve the problem of reducing the size of the list, we create the function *"dictUniqueValues"*. Given the list of dictionaries and a key, it generates a new list of those dictionaries that contain unique values for that specific key. For instance, we use this function to generate the table user with unique "userid" from the main csv file.

"geoid" is created using the function *"getRegionKey"*. It returns the geoid of a region, given the table geography and a specific region name. The functions *"assignDateKey"* and *"assignRegionKey"* assign the correct id to each date and region in the table user. The first function is also used to assign to the answers table the correct dateid given the Date of Answer value.

The Subject table is created using different functions. The first function is *"generateSubjectDict"*, it generates a list of dictionaries from the file of subjects, and it returns the list of dictionaries with all the subjects. These 4 functions, *"getSubjectName"*, *"getSubjectId"*, *"getLevel"*, *"getParent"*, return the name, the id, the level, or the parent id of a subject given the name or the id.
The function *"generateSubjectList"* creates a description that will be present in the final subject table. Each value is a string with the names of the subject present in the sequence and ordered by level extracted from the dictionary, respecting the hierarchy. *"getDescriptionId"* returns the id assigned to a specific sequence of ids of subjects. *"assignDescriptionKey"* assigns to each record of the list of dictionaries the correct id of the sequence of subject ids. *"addSubjectDescription"* adds the list of sequence of subjects to the dictionaries, one for each sequence.

answers	538835
user	13630
organization	24640
date	596
geography	76
subject	412

Table 2: Number of records for each table

Eventually, we create two functions to export everything in csv format. In particular, the function *"dropKey"*, given the list of dictionaries and the name of a key, removes that key and its associated value from each dictionary. In this way we remove those columns that are present in the original file (that we use for assigning foreign keys and for generating the new data) but are not present in the final schema. It returns a new list of dictionaries. The last function, *"saveToCsv"*, simply saves the table list in a new csv file.

1.3 Assignment 2

For the last assignment we need to populate our database with the data already prepared in the previous assignment writing a Python program. First of all, we create the function *"csvToList"* to extract the content of a file and put it into a list of dictionaries. The lists created with this function will be used for the following operations. Subsequently, we create *"getAttributesList"* that gets the list of the attributes of the table that will be used in the queries. This is useful to execute the next functions.
Since we need to rename all the keys of the dictionaries in lowercase, due to the fact that the columns of the tables in the schema (created in assignment 0) are in this format, whereas some attributes in the original file are in uppercase. To solve this problem, we generate the function *"renameKeysLower"*. This is always called by the main function (the function to prepare the csv to upload) and it puts the values of a key in a new key with its name in lowercase, and then it removes the original key. The above function is different than the function *"renameKey"*, that it use to rename a single attribute (for instance, to transform correctanswer in `"correct_answer"`).
The last four functions are used to upload all the data into the corresponding tables on the server.

2 Part 2: SSIS Solutions

Introduction

In Part 2 of the "Laboratory of Data Science" project, we are required to solve some problems on the database that we created previously. In order to solve the following assignments, we need to use SQL Server Integration Services (SSIS).
We provide a SSIS solution project file with a SSIS package for each of the following Assignments.

2.1 Assignment 0

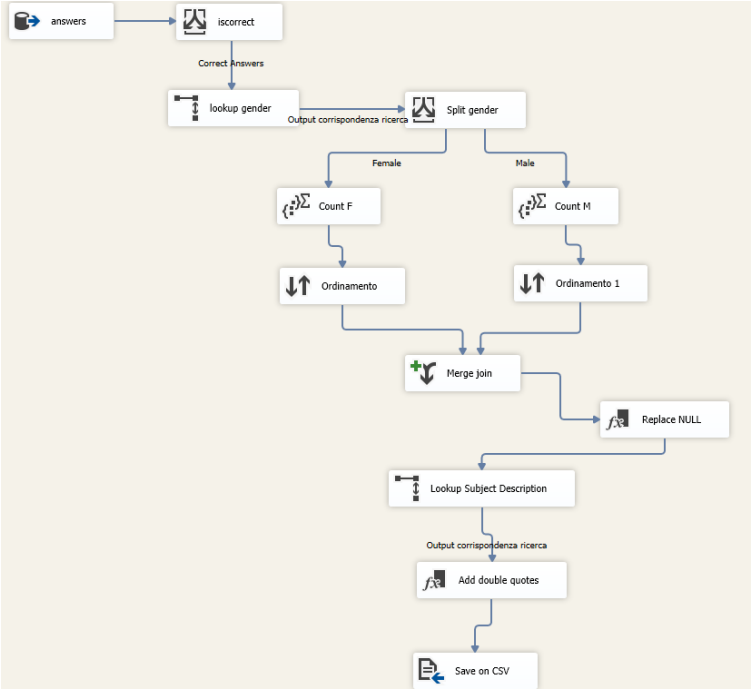

Figure 2: Assignment 0 - Data Flow

The initial component of the Data Flow is an *OLE DB Source* that connects to the SQL Server database. We make the link by selecting the columns *answerid*, *userid*, *subjectid*, and *iscorrect* from the table "answers". Then, using a Conditional Split, we choose only the correct answers using the *iscorrect* attribute.
The next step is to import the *gender* column from the table "user" into our data flow. This is done to divide the data by gender using another Conditional Split. Gender of 1 implies *Male* and a gender of 2 indicates *Female*.
We group by *subjectid* for each split and count the number of correct answers for each gender. These two aggregations are renamed *CountF* and *CountM*.
We update the NULL values in the *subjectid* column after the merging join since some subjects do not have correct answers from female users, and the full outer join cannot handle this condition. Using a derived column, we check if the *subjectid* for the female split's aggregation function is NULL; if so, it replaces the NULL value with the male split's correspondent *subjectid*. The final step before exporting the result to a CSV file is to lookup the *description* of each *subjectid* and then use a derived column to put a quotation mark before and after the string. This is required to avoid problems with the CSV file caused by the presence of commas in the *description*'s string.

subjectid	subjDescription	CorrectF	CorrectM

2.2 Assignment 1

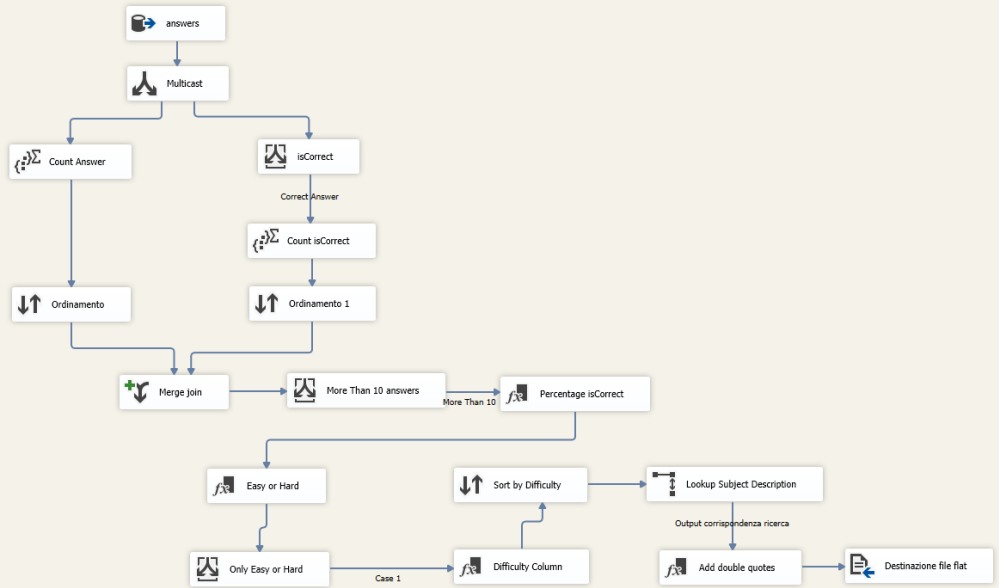

Figure 3: Assignment 1 - Data Flow

The initial Data Flow Component is an *OLE DB Source* to import the fact table once more, choosing to import the columns *answerid*, *subjectid*, *iscorrect*.
Following that, we utilise a Multicast component to group by *subjectid* and tally the number of answers for each *subjectid* on one side.
On the other hand, we first choose only the correct responses (through a Conditional Split), then group by *subjectid* and count the correct answers for each *subjectid*.
Finally, we finish the Multicast with a Merge join and use another Conditional Split to choose only the subject with more than 10 total answers.
The next step is to compute the ratio of correct answers to total number of answers for each subject as a derived column. This is required to construct two additional derived columns with boolean values.
The first, *isEasy*, is TRUE if the percentage of right answers is greater than 90%. The second one, *isHard*, is TRUE if this percentage is less than 20%.
Given these two boolean values for each topic, a Conditional Split selects only those subjects that have a TRUE value in one of the two derived columns.
Finally, a new column "Difficulty" is added with the string "Easy" or "Hard" based on the boolean column values.
The *description* of each subject is then added, and the final output is exported to a CSV file, as before.

subjectid	**subjDescription**	**subjDifficulty**

2.3 Assignment 2

After using the same *OLE DB Source* for the fact table "answers" (selecting the columns *answerid*, *userid*, *iscorrect*), we need to select only the correct answers using a Conditional Split.
The *geoid* and *country name* are then imported from the *user* and *geography* tables using two Lookup Components. An Aggregate Transformation Component is used to group all the answers by *country* and *userid*, as well as to count the number of correct answers for each grouping.
We decide to start with a Multicast to conduct some sort of ranking on this outcome. The maximum value of correct answers is calculated for each country, and these values are then added to the preceding grouping. This join produces a table containing all of the *userid*s for each country and their number of correct answers,

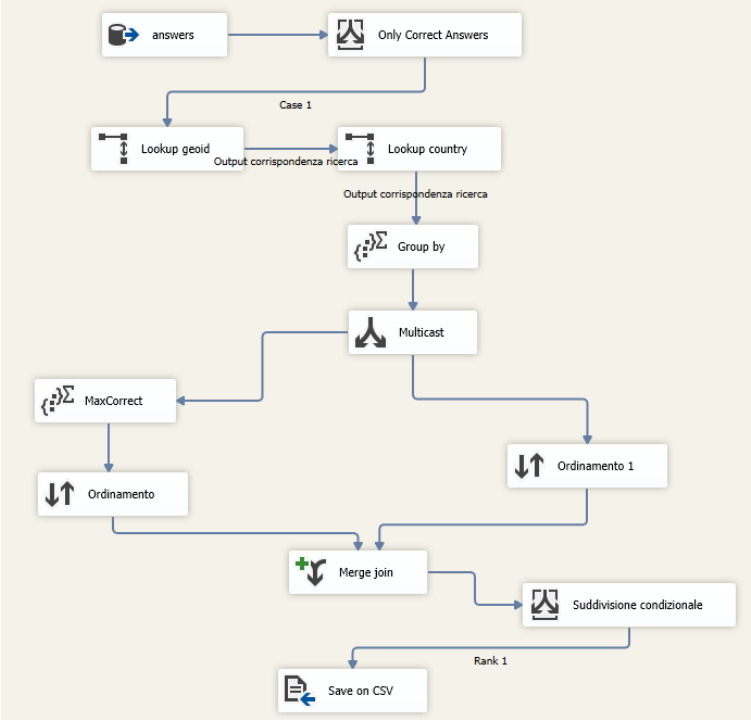

Figure 4: Assignment 2 - Data Flow

followed by a column containing the maximum number of correct answers for that country.
At this point, just a Conditional Split is required to choose only the rows where the number of accurate answers (for that user) equals the maximum number of correct answers for that country.
The result is the list of all the *userid*s with the highest number of correct answers for all the countries.

country_name	**userid**	**iscorrect**

3 Part 3: Multidimensional Data Analysis

Introduction

The final stage of the project entails answering some business questions on a datacube that we must construct based on the previously created database. To answer those problems, we must employ *MultiDimensional eXpression (MDX)* in *SQL management studio*. Finally, we must use *Power BI* to develop a dashboard for showing some distributions of our data.

The file `MDXQueries_Group14.mdx` contains all three MDX queries for the assignments 1, 2 and 3; the file `Dashboard_Group14.pbix` contains two pages named "Assignment 4" and "Assignment 5" with the dashboards for the correspondent assignments.

3.1 Assignment 0

Assignment 0 concerns the creation of the datacube.
In order to solve this first question, we utilize *Visual Studio* with the Analysis services tool to connect to the database and create our cube.
We implement two computed fields in the View to make the creation of the two major measures simpler, since the *iscorrect* attribute has a boolean data type. In particular, a straightforward query is used to construct *iscorrectValue*:

```
case iscorrect
when 'True' then 1
else 0 end
```

We develop a similar query also for the case where the attribute is False, assigning the value 1 in the new field *incorrectValue* if the answer given by the user is incorrect.

Before building the OLAP cube, we establish the dimensions Date and User.
The first one represents the dates for the answers in the fact table. We include a hierarchy with the following relations: *Year* → *Quarter* → *Month* → *Day* → *Dateid*. The same hierarchy is applied in the dimension User where we associate the primary key *dateid* with the foreign key *dateofbirthid*. As the name of the attribute suggests, in this case the date represents the date of birth of the user. Moreover, another hierarchy, *"RegionCountryContinent"*, is included in User to establish the relations between the geographical attributes: *Continent* → *Country Name* → *Region* → *Geoid*.
Finally, we generate the other two dimensions, Organization and Subject.

Concerning the measures, four of them are necessary for the implementation of the MDX queries: *Answers Count, IsCorrect, IsNotCorrect, NumberOfUsers.*

Anwsers Count represents the number of anwsers. It is a simple aggregation function of Count.

IsCorrect represents the number of correct anwsers. We use an aggregation function of Sum because the value of the computed field is 1 when the answer is correct and 0 when it is wrong.

IsNotCorrect represents the number of wrong anwsers. As the previous measure, it is the Sum of the field incorrectValue to obtain the total number of wrong answers.

NumberOfUsers represents the number of users/students. It is a Count Distinct function that counts the distinct value of the field *userid*.

All the measures have as *FormatString* property the value "Standard".
The cube is uploaded with the name `Group_14_Cube_2022`.

3.2 Assignment 1

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

```
WITH MEMBER Mistakes AS
[Measures].[Answers Count] - [Measures].[IsCorrect]
SELECT {Mistakes} ON COLUMNS,
NONEMPTY(GENERATE([User].[RegionCountryContinent].[Country Name],
TOPCOUNT(([User].[RegionCountryContinent].CURRENTMEMBER,
[User].[Userid].[Userid]),1,Mistakes))) ON ROWS
FROM [Answers]
```

Firstly, on the rows we show all the countries and the User with the higher value of *Mistakes* for each Country. The `NONEMPTY` function allows to return only rows with values, while with the `GENERATE` function the set of countries is applied to each member of the set of users, and the resulting sets are joined by union. Finally, `TOPCOUNT` selects the top 1 User for each country. *Mistakes* is a member that we define as the difference between the count of answers and the count of correct answers of the specific user, using the two correspondent measures of the cube. Alternatively, we can use the measure *IsNotCorrect* without introducing the member.

		Mistakes
Belgium	86016	256.00
France	53200	242.00
Germany	113435	254.00
Ireland	79098	273.00
Italy	67433	392.00
Spain	105010	330.00
UK	54492	502.00
Canada	21412	254.00
US	72434	278.00
Australia	23352	331.00
New Zealand	96080	216.00

Figure 5: Query 1 - Results

3.3 Assignment 2

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

```
WITH MEMBER MaxCorrect AS
MAX(([Subject].[Subjectid].CURRENTMEMBER, [User].[Userid].[Userid]), [Measures].[IsCorrect])
SELECT {MaxCorrect} ON COLUMNS,
NONEMPTY(FILTER(([Subject].[Subjectid].[Subjectid], [User].[Userid].[Userid]),
[Measures].[IsCorrect] = MaxCorrect)) ON ROWS
FROM [Answers]
```

For this assignment, using the `TOPCOUNT` function provides a unsatisfactory outcome because it returns only one record for every subject while, in many cases, there are more users with the same highest value of correct answers for a single subject. To avoid this problem, we use a member *MaxCorrect* that compute the maximum value of the measure *IsCorrect* for each subjectid. Then, a `FILTER` function is applied on the set of users for each subject with the condition `IsCorrect = MaxCorrect`.
The downside of this solution is the execution time of about 9 minutes.

		MaxCorrect
Maths, Advanced Pure, Functions, Composite Functions	97793	3.00
Maths, Advanced Pure, Functions, Function Notation	77990	8.00
Maths, Advanced Pure, Functions, Inverse Functions	85945	3.00
Maths, Algebra, Algebraic Fractions, Adding and Subtracting Algebraic Fractions	18052	9.00
Maths, Algebra, Algebraic Fractions, Adding and Subtracting Algebraic Fractions	109348	9.00
Maths, Algebra, Algebraic Fractions, Multiplying and Dividing Algebraic Fractions	18052	4.00
Maths, Algebra, Algebraic Fractions, Simplifying Algebraic Fractions	24738	16.00
Maths, Algebra, Algebraic Fractions, Simplifying Algebraic Fractions	61809	16.00
Maths, Algebra, Algebraic Fractions, Solving Equations with Algebraic Fractions	107293	5.00
Maths, Algebra, Co-ordinates, Co-ordinates-Others	83476	8.00
Maths, Algebra, Co-ordinates, Distance Between Two Co-ordinates	54492	2.00
Maths, Algebra, Co-ordinates, Distance Between Two Co-ordinates	101574	2.00

Figure 6: Query 2 - Results (only first rows)

3.4 Assignment 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 AvgCorrect AS
([User].[Userid].[All], [User].[RegionCountryContinent].CURRENTMEMBER.PARENT.PARENT.PARENT,
[Measures].[IsCorrect]) /
([User].[Userid].[All], [User].[RegionCountryContinent].CURRENTMEMBER.PARENT.PARENT.PARENT,
[Measures].[NumberOfUsers])
MEMBER Ratio AS
[Measures].[IsCorrect] / AvgCorrect
SELECT {[Measures].[IsCorrect], AvgCorrect, Ratio} ON COLUMNS,
NONEMPTY(GENERATE([User].[RegionCountryContinent].[Continent],
TOPCOUNT(([User].[RegionCountryContinent].CURRENTMEMBER,
[User].[Userid].[Userid]),1,Ratio))) ON ROWS
FROM [Answers]
```

To calculate the ratio, we first need the average of correct anwsers. Therefore, we create a first member *AvgCorrect* that compute the average as the ratio of the total number of answers to the number of correct answers for the current continent, using the two measures mentioned before. We use the `PARENT` function on the geographical hierarchy to reach the level of the Continent. Alternatively, we can use the attribute Continent outside the hierarchy, avoiding the use of the function `PARENT`.
Then, the member *Ratio* is the simple ratio between the count of correct answers and the average. Repeatedly, with the `TOPCOUNT` function we select the User with the highest ratio for each Continent.

		IsCorrect	AvgCorrect	Ratio
Europe	99333	688.00	24.8859490698934	27.65
Nord America	39367	587.00	24.7856689917565	23.68
Oceania	24738	518.00	28.1532175689479	18.40

Figure 7: Query 3 - Results

3.5 Assignment 4

In order to show the geographical distribution of the answers, we decide to use two types of plots in our dashboard.
As it is possible to see in Figure 8, a map is represented twice to separate the two cases, with the relative size of the bubble for each country based on the number of correct (blue) or wrong (red) answers. We can notice, from the size of the bubbles, how the number of answers follows the same ratio in both the scenario for all the countries. This supposes that there is no country that prevails over the others. All of them maintain the same proportion of correct and incorrect answers.

Consequently, a bar plot is used to represent the correct and the wrong answers for each region. In this way, it is possible to have a higher level of granularity. By clicking on a bubble in the map, the regions of that specific country are highlighted in the bar plot to better understand the distribution of the correct/wrong answers.

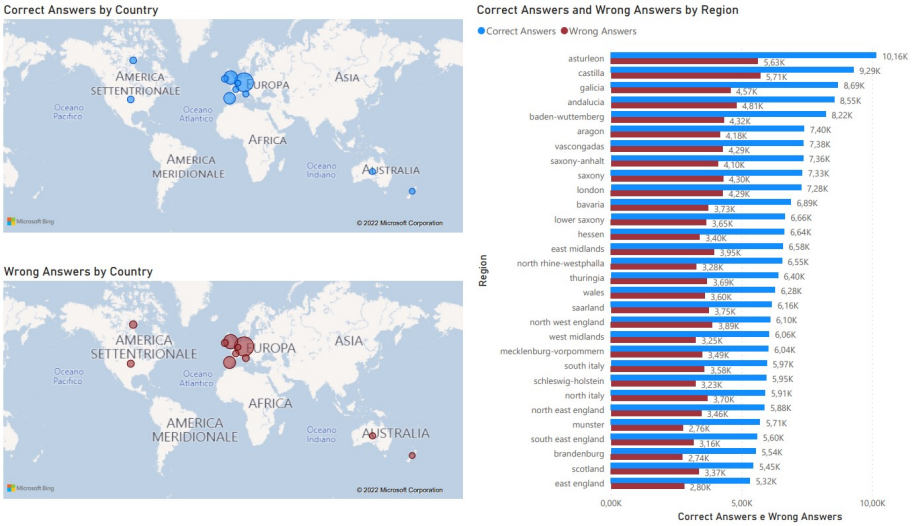

Figure 8: Dashboard 1 - Geographical representation

3.6 Assignment 5

For our free-choice dashboard, we implement three kinds of plots focusing on the gender of the students and the subjects. In Figure 9 we can observe that the distribution of correct answers by birth year and gender maintains a similar proportion between male (blue) and female (pink). However, it is possible to notice some differences for students that were born between 2005 and 2007. Whereas, for students born in 2009, female students have a better outcome.
The other two plots show us the distribution of the correct and wrong answers and also the number of total answers and students for each subjects.

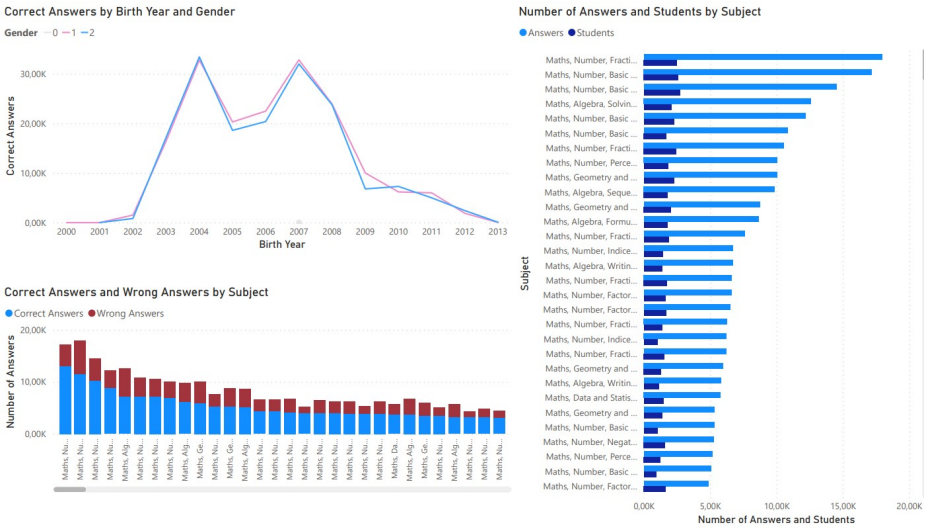

Figure 9: Dashboard 2