SAINT MARY'S UNIVERSITY SINCE 1802

One University. One World. Yours.

MSc in Computing and Data Analytics

MCDA 5580 – Data and Text Mining

Assignment – 2

Submitted to:

Trishla Shah

Prepared by:

Allen Mathew	- A00432526
Gyaneshwar Rao	- A00433014
Shubham Chumber	- A00433094
Devvrat Deshwal	- A00431503

Table of Contents

Executive Summary: 3

Objective: 3

Data Preparation Tools 4

OpenRefine 4

A. Dataset Used: 4

B. Tasks Performed: 5

C. Brief on Work Done on OpenRefine: 29

Google Data Prep: 30

A. Tasks Performed: 30

B. Brief on Work Done on Google Data Prep: 70

SSIS / Data Preparation (ETL Pipeline) 71

A. Dataset Used: 71

B. Schema: 71

C. ETL Pipeline Process: 72

1. Empty All Tables: 75

2. Load new Data into Dimension Tables: 77

3. Load new Data into fact table: 84

4. Save data to CSV file: 89

D. Testing of the ETL pipeline: 96

E. CSV File generated: 100

F. Brief on Work Done on SSIS: 101

Comparison of the Data Preparation Software: 102

Conclusion: 103

References: 103

Appendix: 104

Executive Summary:

Data in the Real-World often tends to be inconsistent, may contain incomplete/missing records furthermore it is probably going to contain numerous errors, Data Preparation could be done to resolve such issues. It is the process of transforming uncleaned/raw data into a meaningful format. Data Preparation is used for Data Cleaning, Integration, Transformation and Reduction.
For this assignment we used various tools such as OpenRefine, Google Data Prep, Microsoft SSIS to gain a better understanding of the concepts involved in Data Preparation. Each of the Data Preparation Tools was used to complete a set of tasks on different Datasets. On completing the tasks, the results were recorded and documented. Finally, based on the performance and experience in using the three Data Preparation tools a comparison table was made.

Objective:

To use different software tools like Google Data Prep, OpenRefine and Microsoft SSIS to gain a better understanding of the concepts for Data Preparation and ETL process. The OpenRefine is first used to audit the data using with different filters and judged accordingly to remove/delete/edit row data which in turn is called as normalizing the data or cleaning the data. Google Data Prep is another tool which will allow us to merge different files into one file and perform operations on it in bulk, where the processing is handled in cloud and data flow pipeline is evaluated accordingly with different filters for generating steps to be processed in every processing milestone in the event flow to generate the required combined documents using jobs on the cloud. The Microsoft SSIS is an ETL tool which is used to data clean and automate the process of data management into relational database using “Windows work flow” framework. The tool provides impeccable debugging features and has ‘try catch’ simulators which provides error details at each row level, which helps further in debugging the flow and correcting the automation process. This enterprise tool stands out with all kinds of GUI tools for efficient ETL implementation which helped us to put data into data tables.

Data Preparation Tools

To have a better understanding of the Data Preparation concepts we will be using various tools to perform tasks such as Data Cleaning, Integration, Transformation and Reduction. The following are the Data Preparation tools that will be used:

1. **OpenRefine**
2. **Google Data Prep**
3. **SSIS**

OpenRefine

OpenRefine is a desktop application(open source) that can be used as a robust tool for data preparation.

A. Dataset Used:

We have used the Dataset (i.e. question1.csv) that was provided to us for this Assignment. The data consists of client details. The dataset provides us information such as:

1. ID
2. Full Name
3. Email
4. Sex
5. Birth Date
6. Credit Card
7. Latitude
8. Longitude

B. Tasks Performed:

The following tasks are performed to gain a better understanding of the Data Preparation concepts and the OpenRefine tool:

1) Install OpenRefine

The current stable version of **OpenRefine(v3.1)** was downloaded and the *EXE* file was made to run on the desktop system.

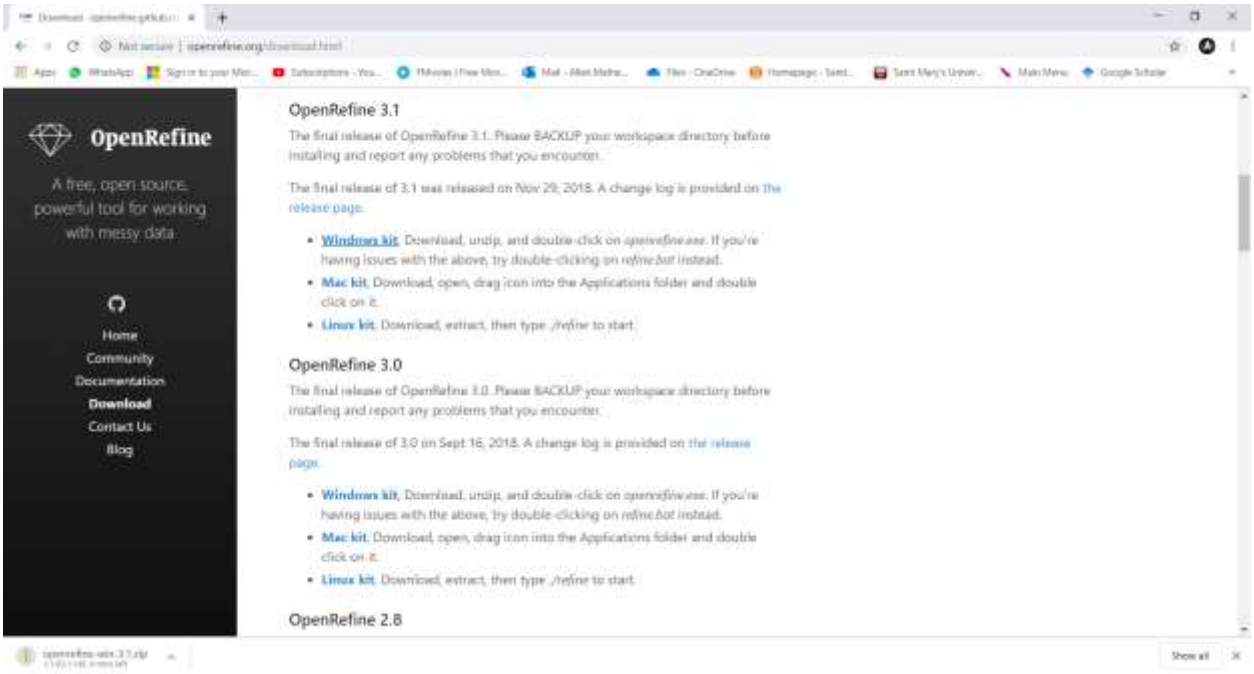

Fig 1: Open Refine Download Page

2) Create Project "**question1**" in OpenRefine
&
3) Load question1.csv

A new project can be created in OpenRefine by navigating to the ***"Create Project"*** section. Then the data file (i.e. ***"question1.csv"***) is selected on clicking the ***"Choose File"*** button, once the data file is loaded it can be pre-viewed by clicking the ***"Next"*** button.

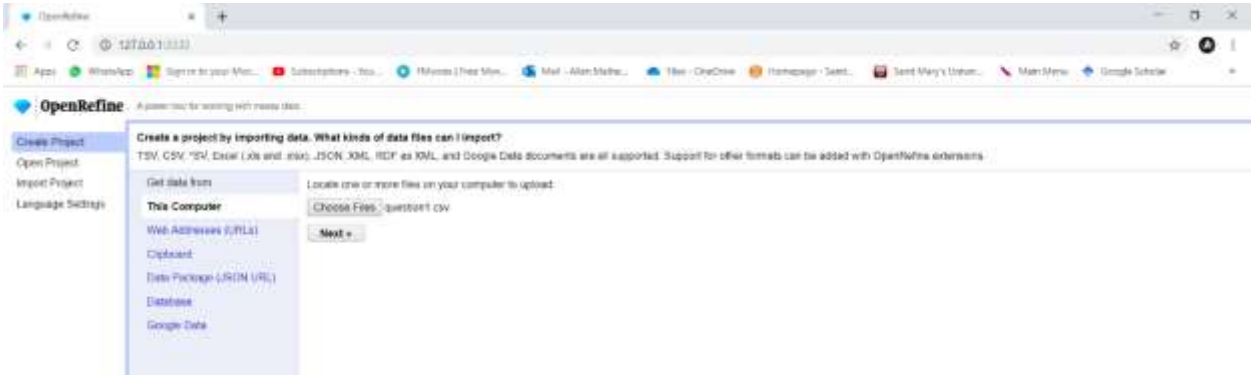

Fig 2: Choose the data file(i.e. question1.csv).

As per the question's requirement, the name of the project is titled as **"*question1*"**. Then we select ***"commas(CSV)"*** since the data file is a *CSV* file and the columns/values are separated by commas. Additionally, we select few more parameters to store the blank rows as nulls. After reviewing the data, click on ***'Create project'*** to proceed.

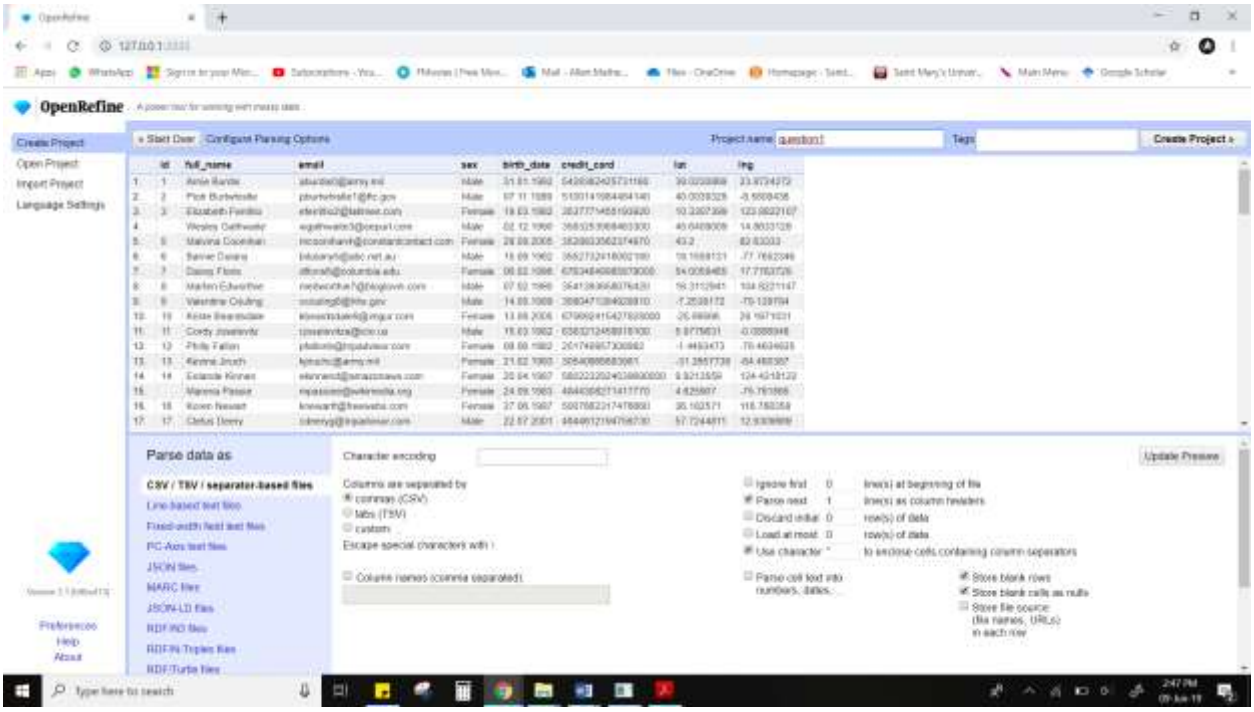

Fig 3: Pre-view contents in the question1.csv data file.

The OpenRefine project was created successfully. The ***"Show as"*** operation (located in the toolbar which is below the total number of records) is used to represent the data in either rows or records. The project consists of 973 records and 1000 rows.

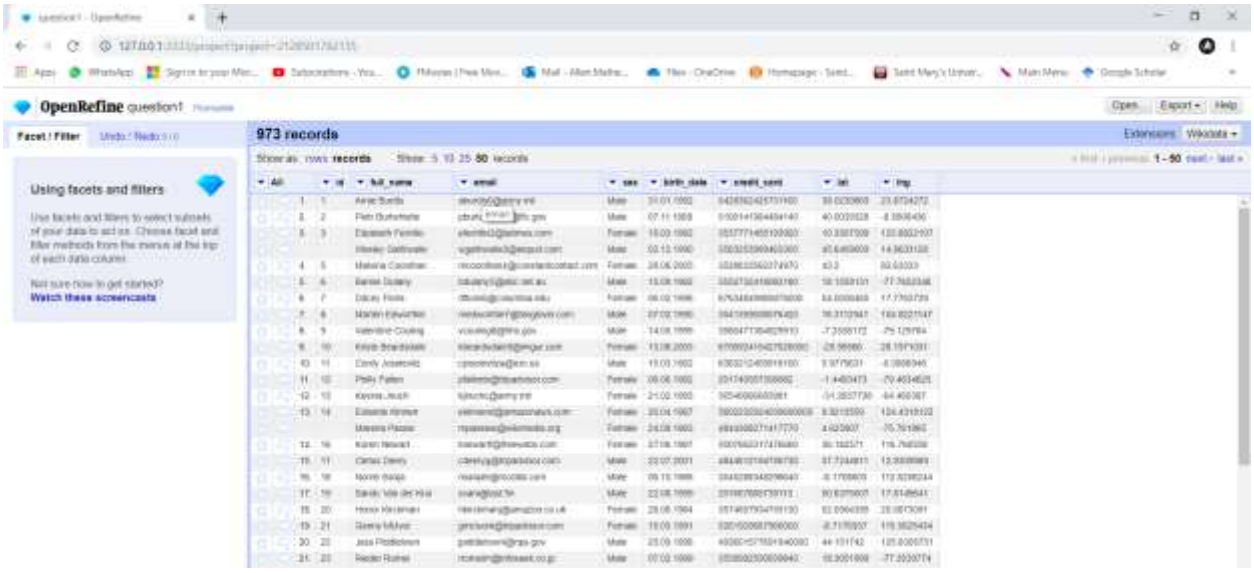

Fig 4: Start page of OpenRefine, Displaying 973 records.

To perform the further tasks/operation on each individual row of the project, the data needs to be represented as rows rather than records. (Williamson, 2019)

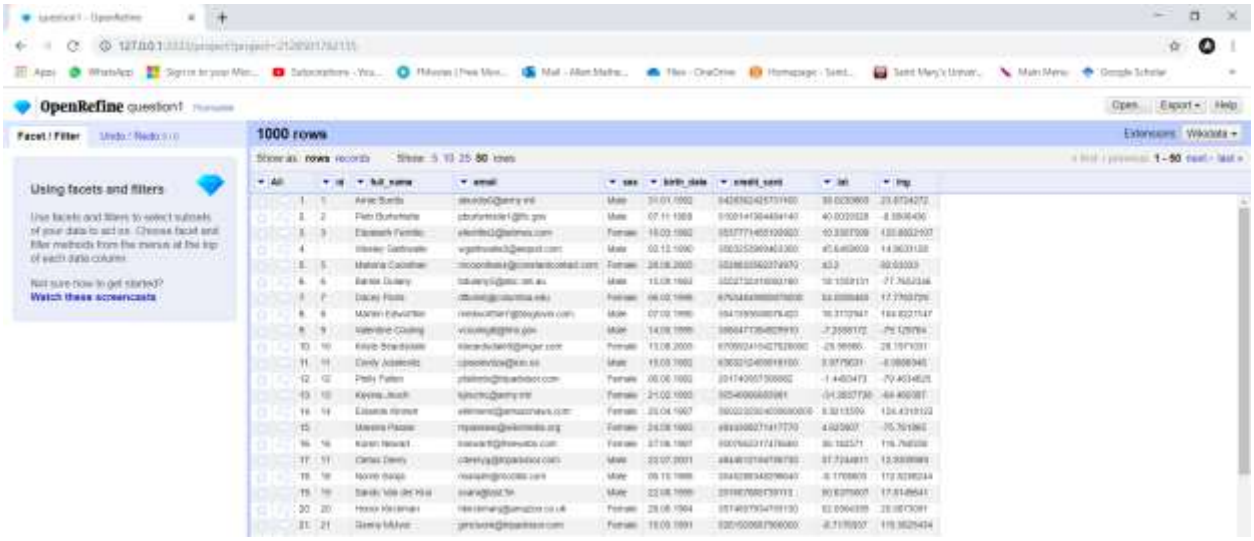

Fig 5: Start page of OpenRefine, Displaying 1000 rows.

4) Remove all rows in which **id** is empty

A facet was created on the ***"id"*** column (**Facet > Customized facets > Facet by blank [null or empty string]**).

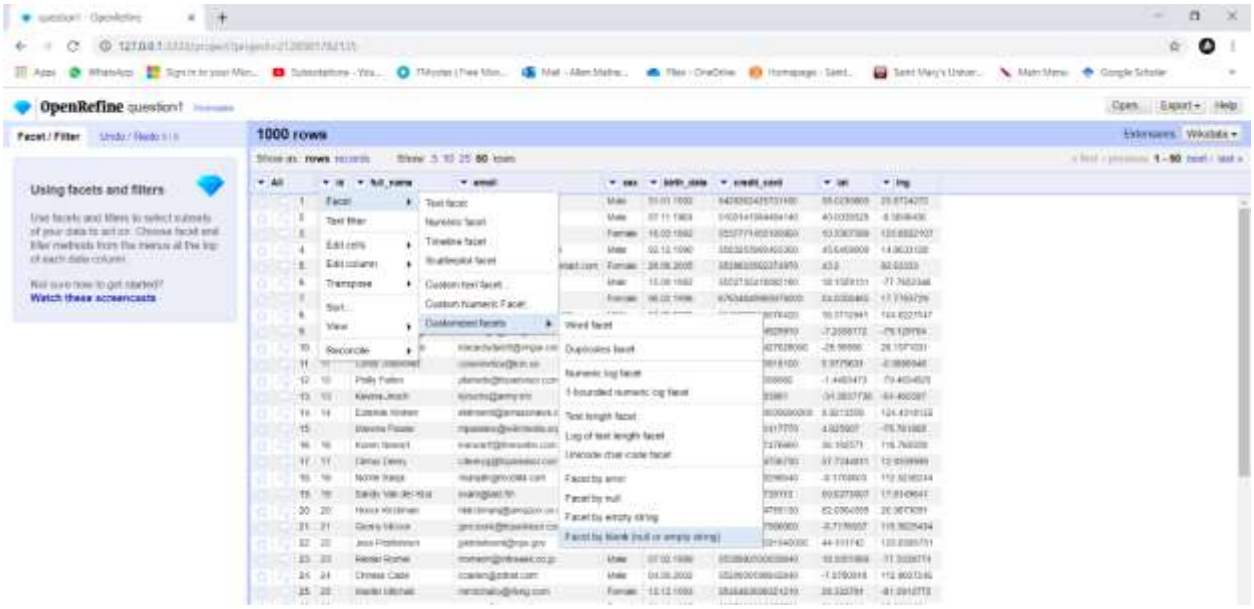

Fig 6: Creating a Facet to check for 'null' or 'empty string'.

The facet clearly distinguishes the number of rows where the ***"id "*** column has an empty record (true) and does not have an empty record (false). Then click on ***"true"*** to select the 27 rows with blank records in the 'id' column and remove them using the '**All**' dropdown ('**Edit rows**' > '**Remove all matching rows**').

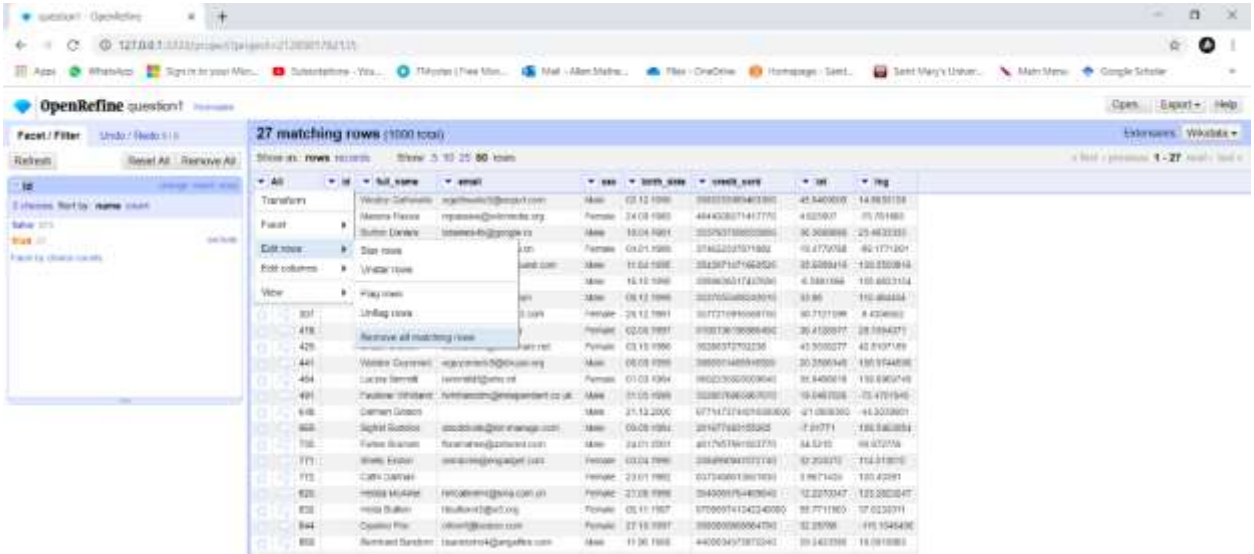

Fig 7: Selecting "Remove all matching rows" within Edit rows from All dropdown, when there are blank records in the 'id' column

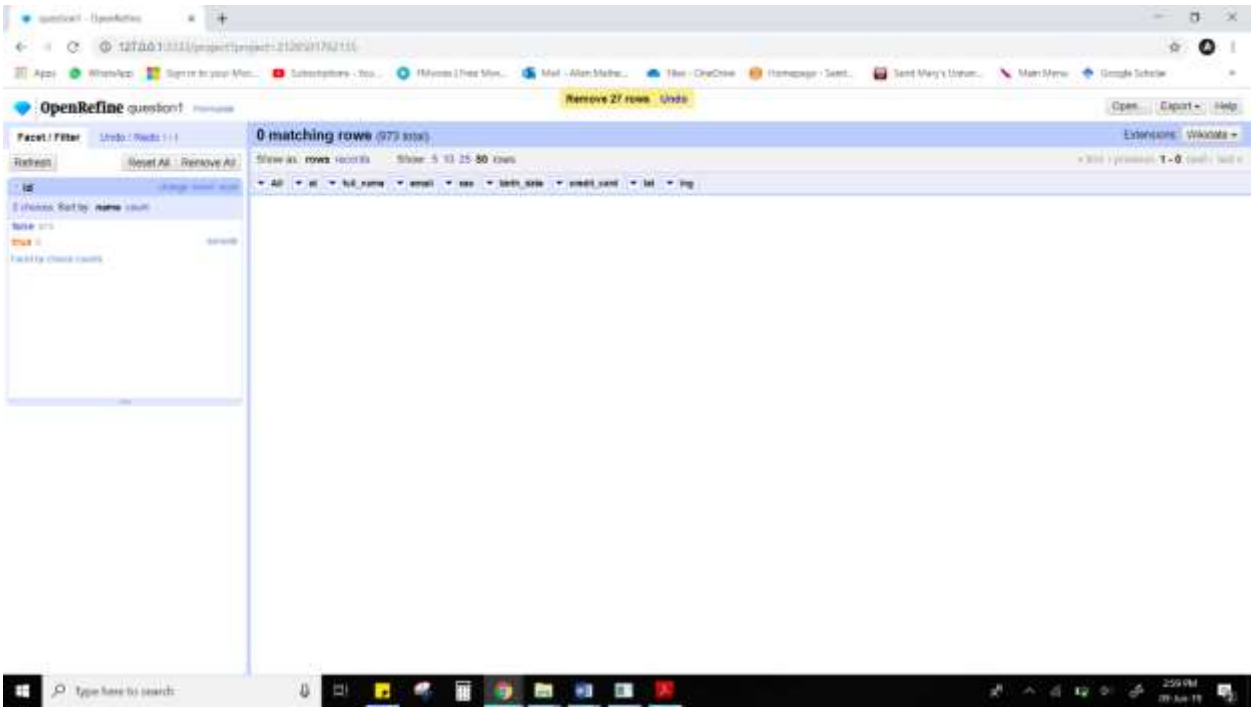

Fig 8: Confirmation pertaining to the Removal of the 27 rows with blank records in the 'id' column

There are 973 unique rows can be seen upon closing the facet. After performing a quick overview, it is evident that there is no row containing an empty record in the "***id***" column.

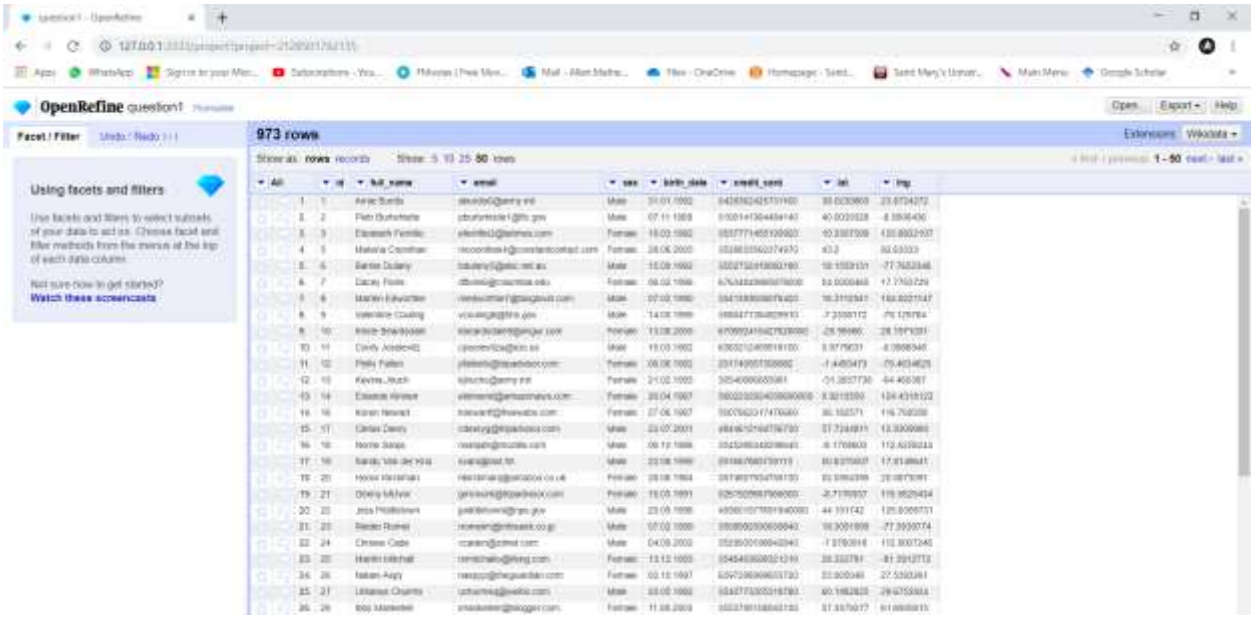

Fig 9: There exists 973 Unique rows after removing all rows with blank records in the 'id' column

5) Remove all rows in which **email** is empty

Like the previous task, A facet is created on the "***email***" column (**Facet** > **Customized facets** > **Facet by blank [null or empty string]**).

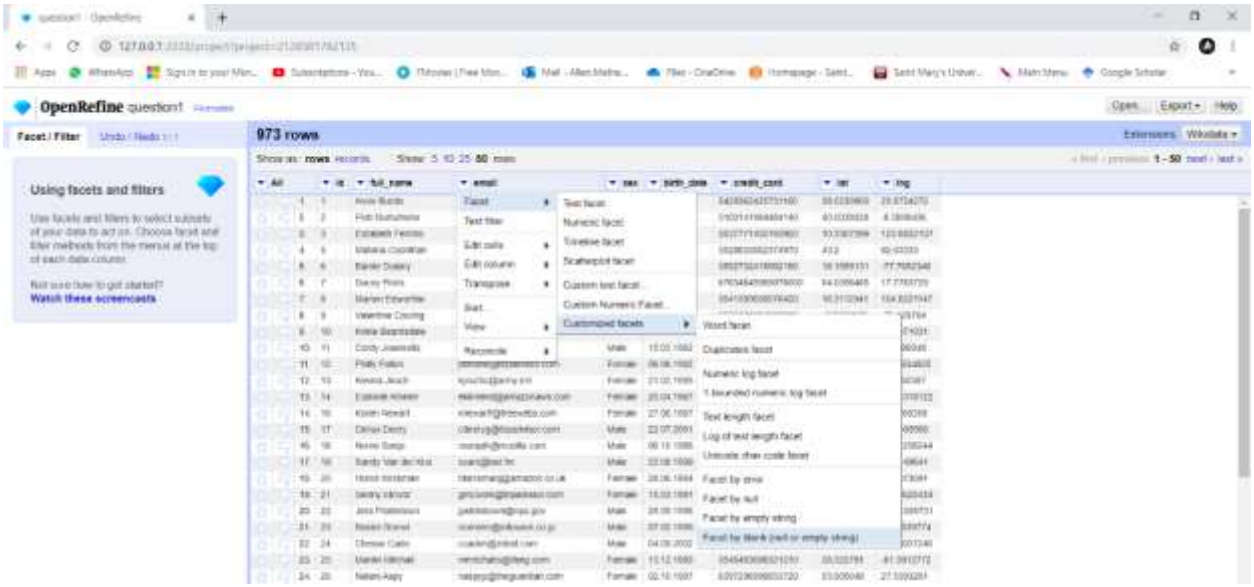

Fig 10: Creating a Facet to check for 'null' or 'empty string'.

The facet clearly distinguishes the number of rows where the ***"email "*** column has an empty record (true) and does not have an empty record (false). Then click on ***"true"*** to select the 33 blank records in the 'email' column and remove them using the '**All**' dropdown ('**Edit rows**' > '**Remove all matching rows**').

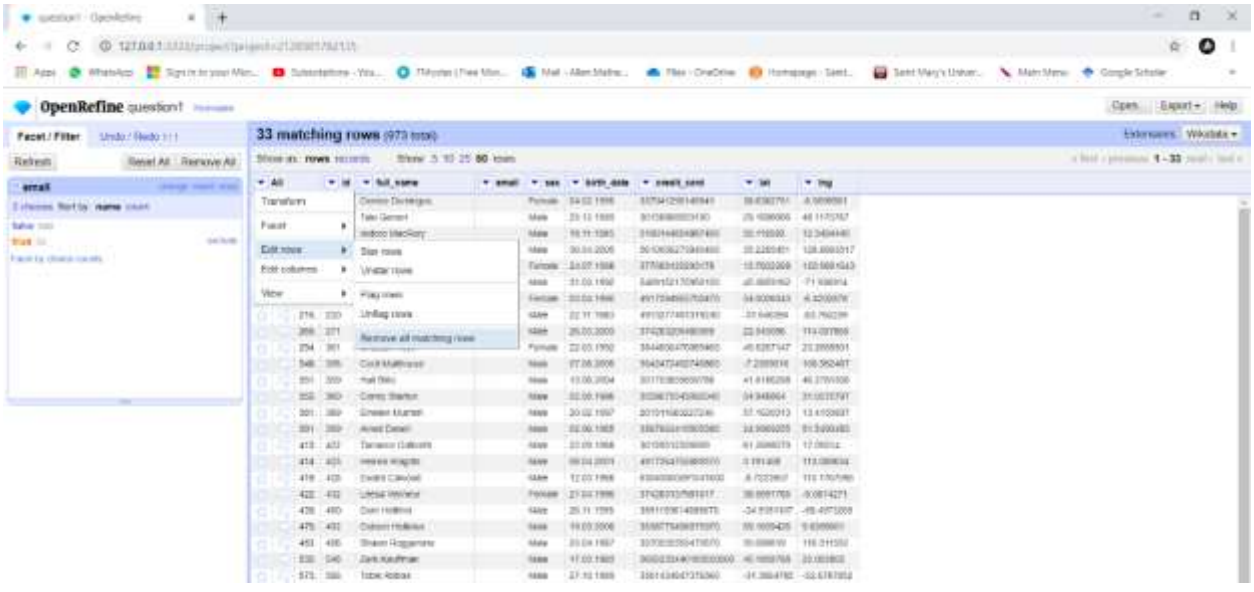

Fig 11: Selecting "Remove all matching rows" within Edit rows from All dropdown, when there are blank records in the 'email'' column

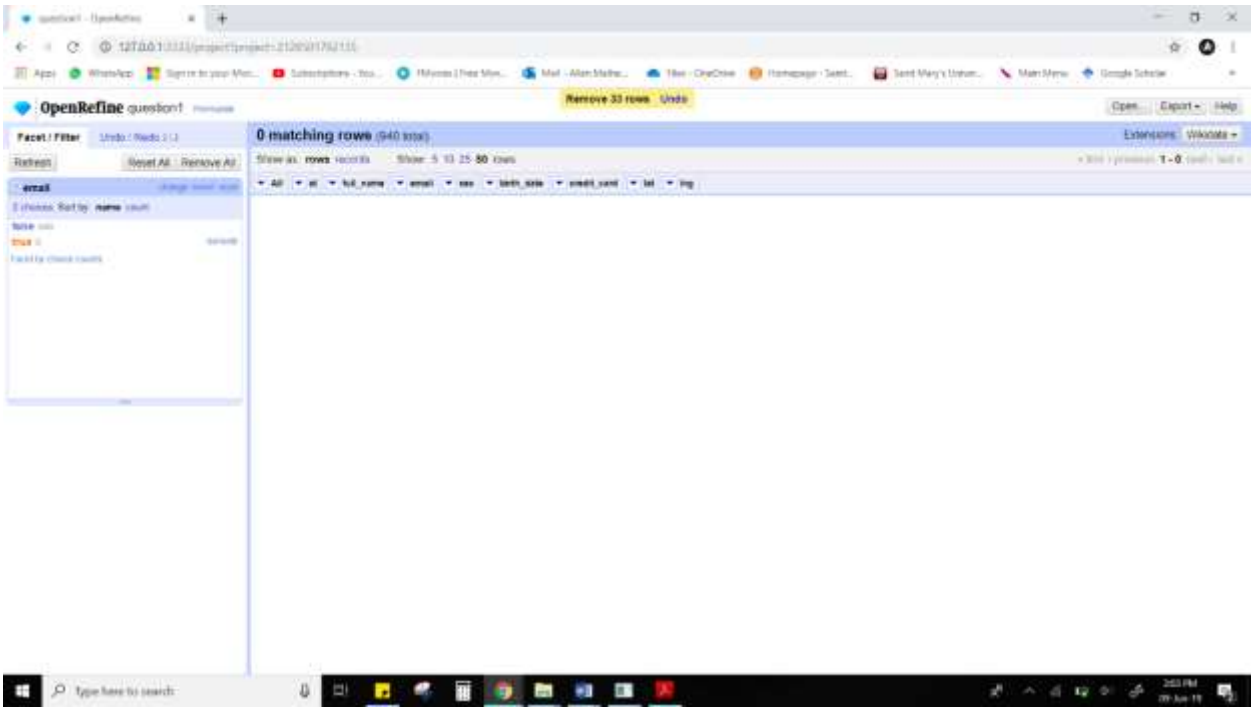

Fig 12: Confirmation pertaining to the Removal of the 33 rows with blank records in the 'email' column

There are 940 unique rows can be seen upon closing the facet. After performing a quick overview, it is evident that there is no row containing an empty record in the ***email*** column.

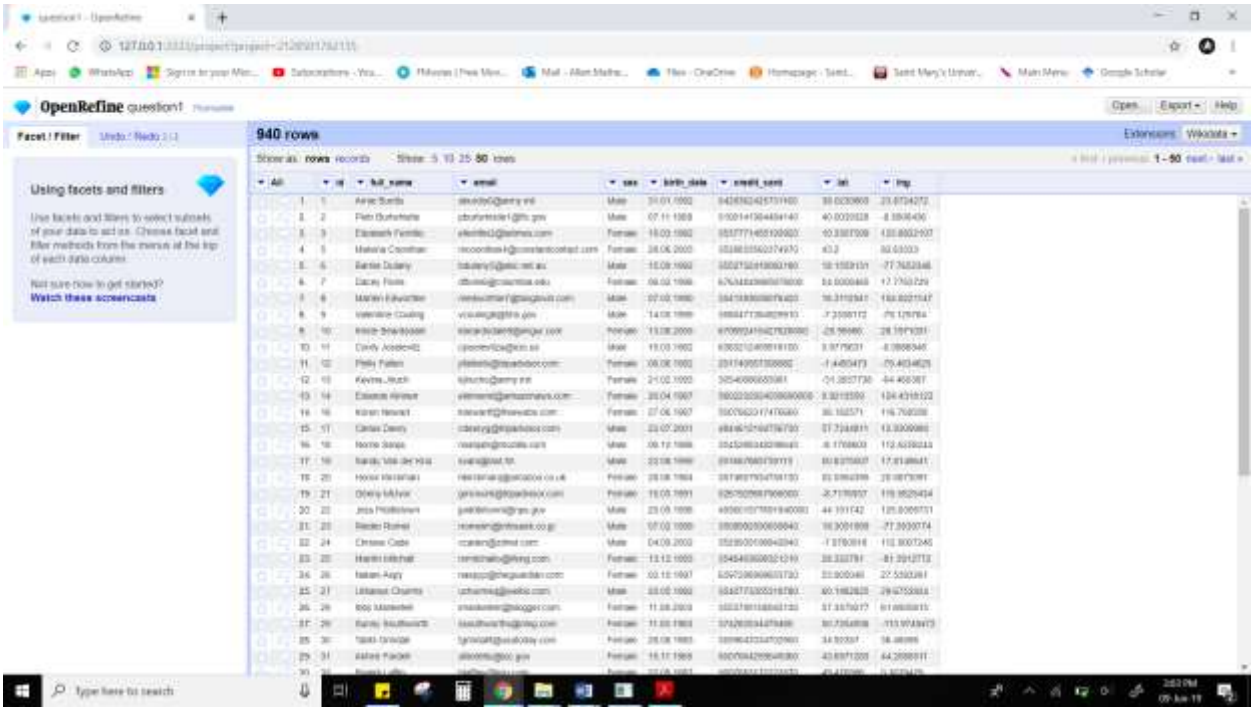

Fig 13: There exists 940 Unique rows after removing all rows with blank records in the 'email' column

6) Split Full Name in to First name and Last Name. Store first name into **f_name** and last name into **l_name** (you will create these two columns).

To complete this task, data from the 'full_name' column will be used to create the additional column 'f_name' and 'l_name' (**Edit column** > **Add column based on this column**).

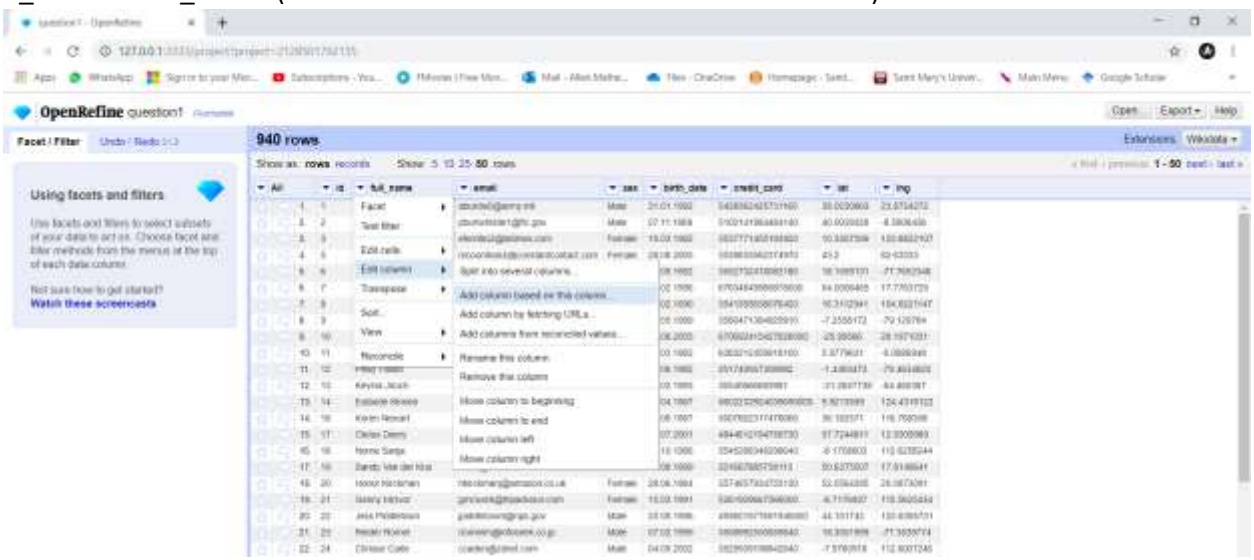

Fig 14: Creating new column f_name and l_name based on data from full_name column.

The following expression (GitHub, 2019) is used to extract the first name from the full name:

value.split("")[0]

For the above expression the "value" is the data of each individual row in the ***full_name*** column. On utilizing the "value" with the ***split*** function it returns an array of strings obtained by splitting the value for every occurrence of a blank space (i.e. ""). The first element (i.e. position [0]) from this array of strings will be the first name and it is the new value for the respective row in the ***f_name*** column.

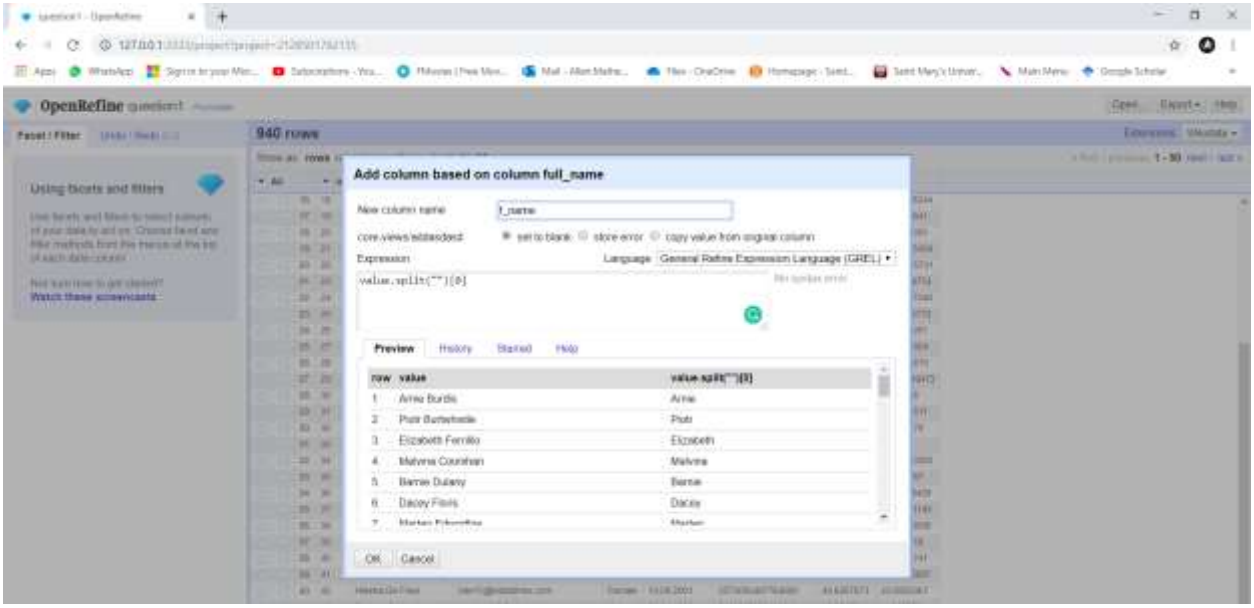

Fig 15: Expression to derive first name from the full_name column

The following expression is used to extract the last name from the full name:

value.split("").slice(1).join(" ")

For the above expression the “value” is the data of each individual row in the ***full_name*** column.

On utilizing the “value” with the ***split*** function it returns an array of strings obtained by splitting the value for every occurrence of a blank space (i.e. “”). Then the first element of the array is sliced since it will be the first name. The rest of the elements in the array would constitute to parts of the last name, hence the ***join*** function is used to concatenate each element with a blank space (i.e. “ “) as a separator to represent the last name as a whole. It is then the new value for the respective row in the ***l_name*** column.

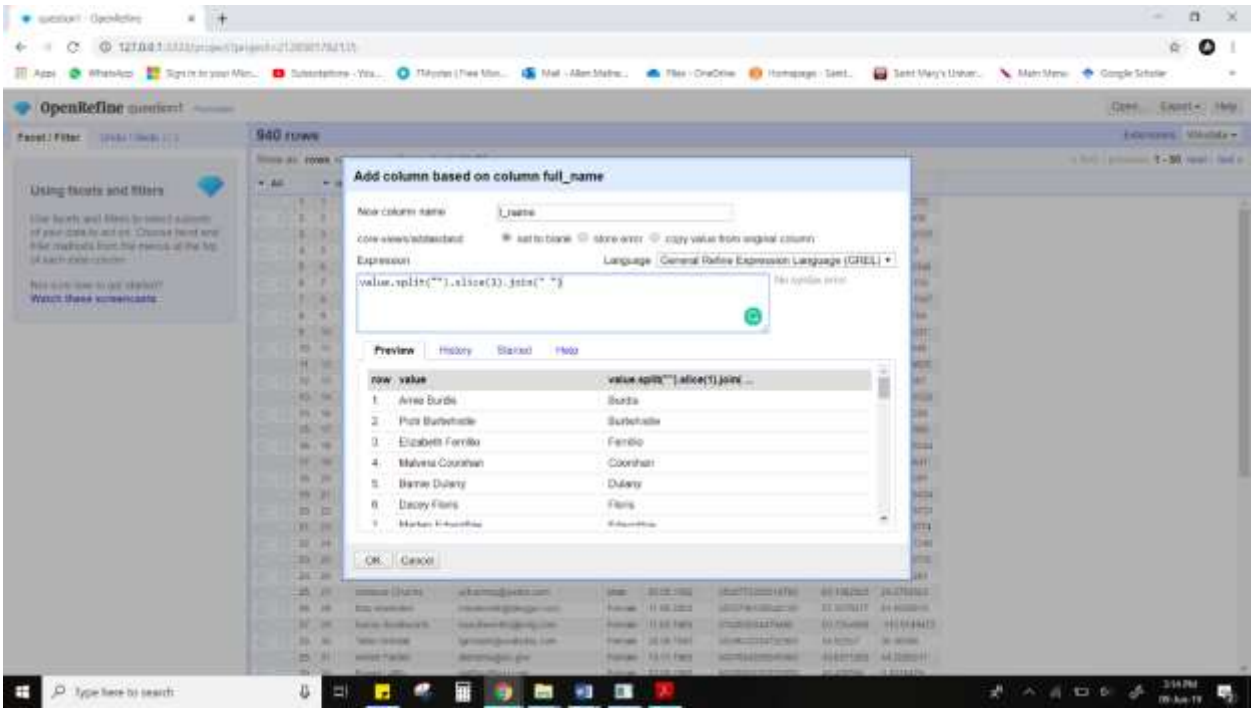

Fig 16: Expression to derive last name from the full_name column

The ***f_name*** and ***l_name*** column were successfully created.

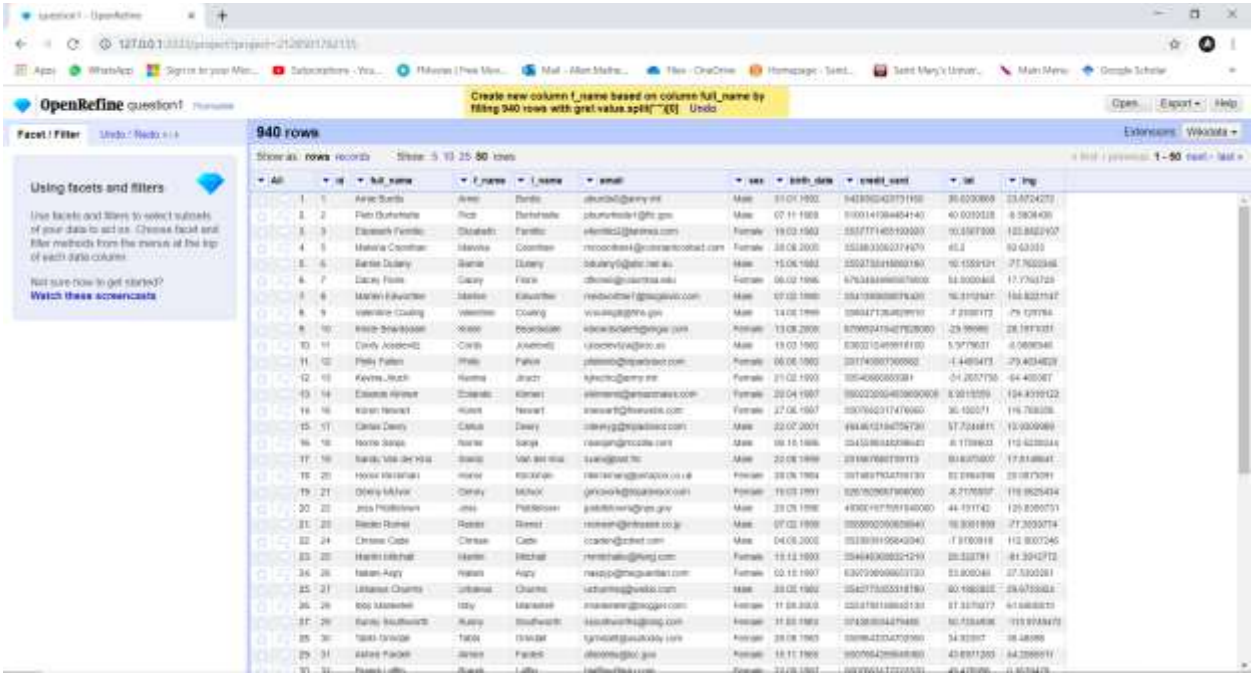

Fig 17: f_name Column was Created Successfully.

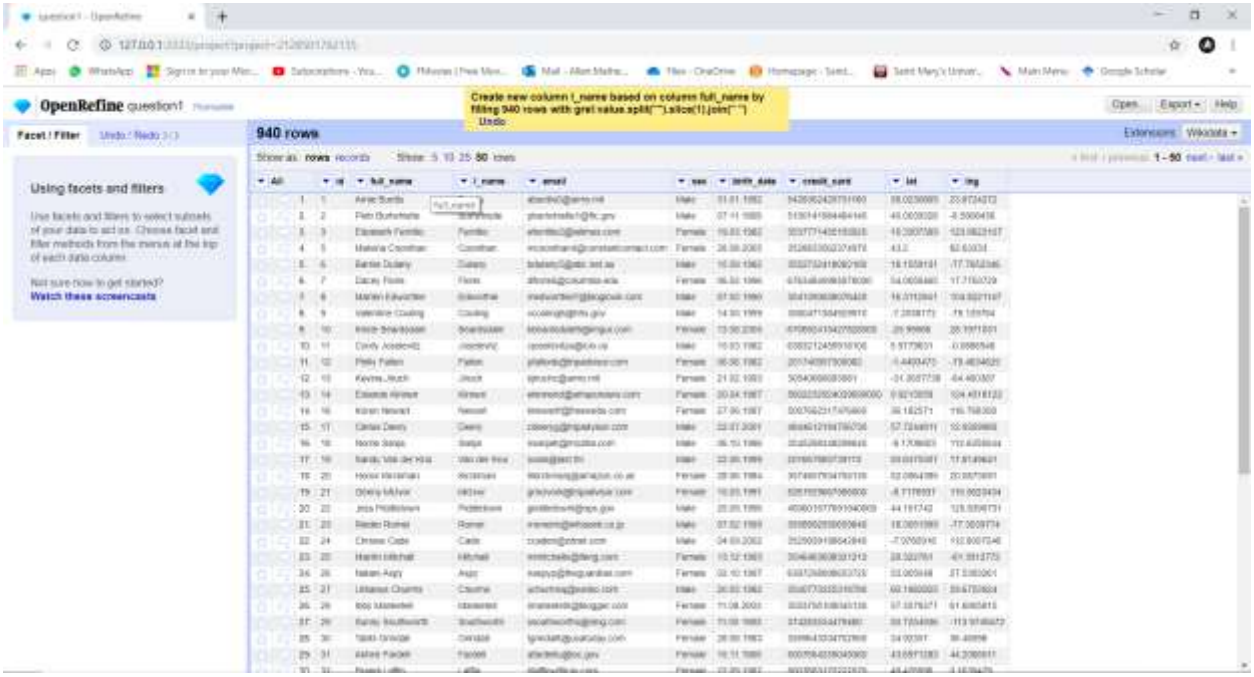

Fig 18: l_name Column was Created Successfully.

After performing a quick overview, it is evident that the ***full_name*** column was properly split into the ***f_name*** and ***l_name*** column.

7) Code Gender (**sex** column) into **M** and **F** instead of Male and Female (Do not create a new column).

To complete this task, individual cells from the ***'sex'*** column will be Transformed (**Edit cells** > **Transform…**).

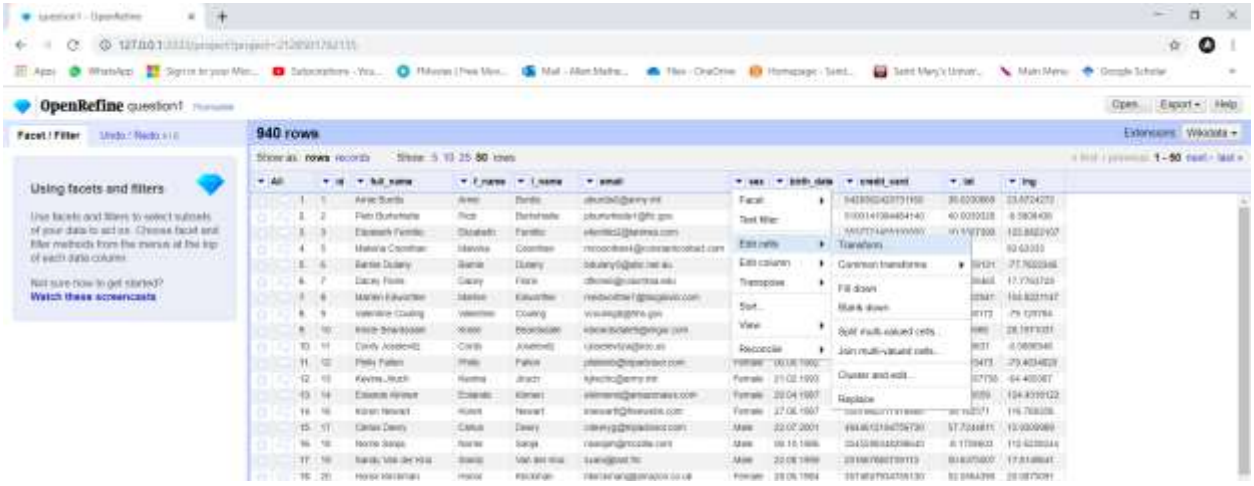

Fig 19: Selecting Transformation within Edit Cells from sex column.

The following expression (GitHub, 2019) is used to Transform the sex column:

value.replace("Male","M").replace("Female","F")

For the above expression the "value" is the data of each individual row in the **"sex"** column.
On utilizing the "value" with the ***replace*** function it returns a string obtained by replacing every occurrence of "Male" & "Female" with "M" & "F" respectively.

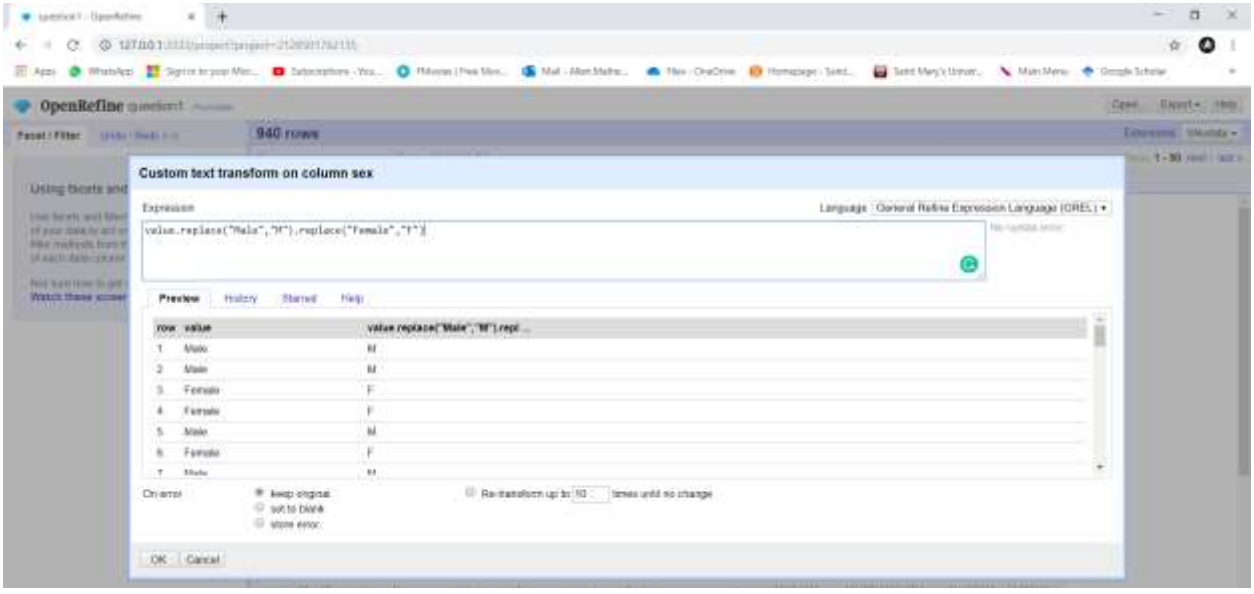

Fig 20: Expression to transform Male and Female to "M" and "F" respectively

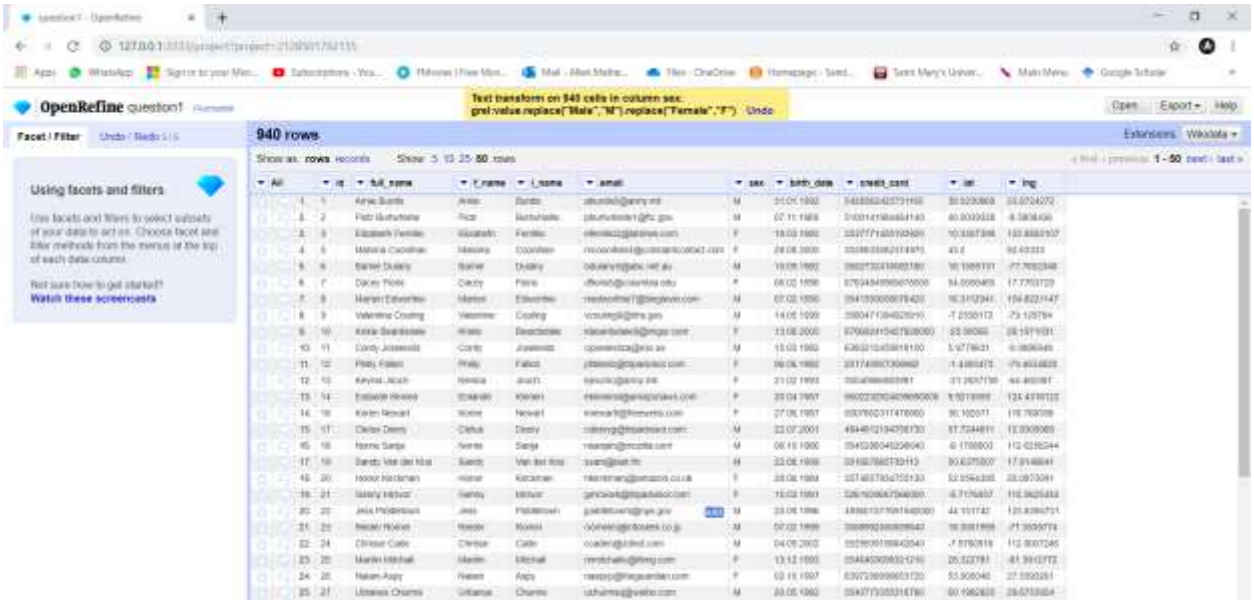

Fig 21: The sex column was successfully transformed form "Male"/"Female" to "M"/"F".

After performing a quick overview, it is evident that all the rows in the **"sex"** column was properly transformed.

8) Calculate age of the person on 24th May 2019 from the **birth_date** column. Store calculated age in **age** column (You will create this column).

To complete this task, data from the '***birth_date***' column will be used to create the additional column '***age***' (**Edit column** > **Add column based on this column**).

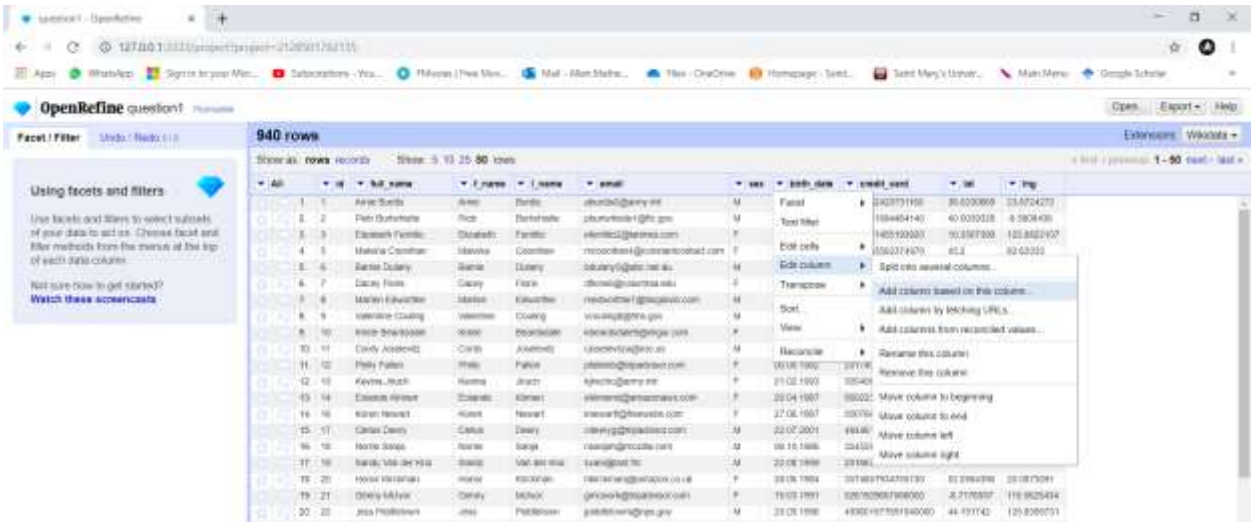

Fig 22: Creating new column age based on data from birth_date column.

The following expression (GitHub, 2019) is used to calculate the age from the ***birth_date***:

diff(toDate("24/05/2019","dd/MM/yyyy"),toDate(value.split(".").join("/"),"dd/MM/yyyy"), "days")/365

For the above expression the “value” is the data of each individual row in the ***birth_date*** column.

On utilizing the “value” with the ***split*** function it returns an array of strings obtained by splitting the value for every occurrence of a full stop (i.e. “.”). Then the ***join*** function is used to *concatenate* each element of the array with a blank slash (i.e. “/“), this is done to create a consistent format for date. After this the ***toDate*** function is used to create a date object of a particular format (i.e. dd/MM/yyyy) from the date string of “value”.

The string “***24/05/2019***” is used in the above expression as a bench mark (according to the question) to calculate the age from the date of birth to this particular date.

The ***diff*** function is used to get the difference between the benchmark date the birth date. The result is then divided by 365 to get the actual age.

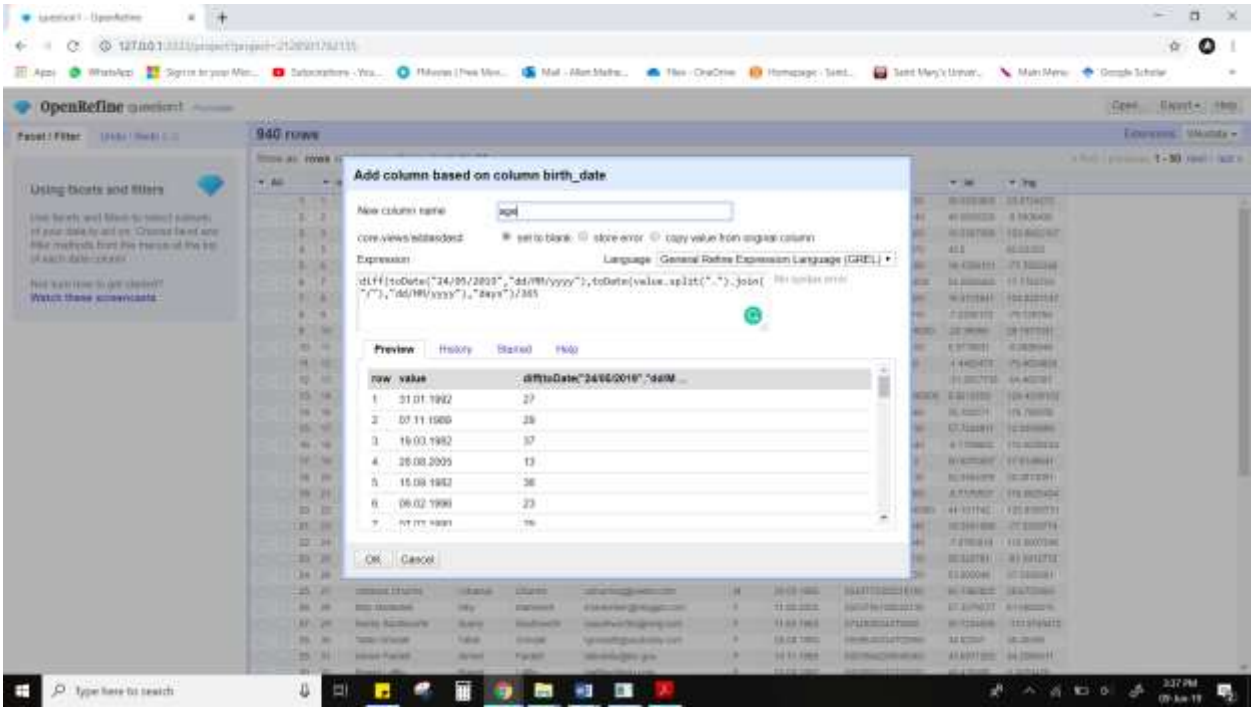

Fig 23: Expression to derive age from birthdate

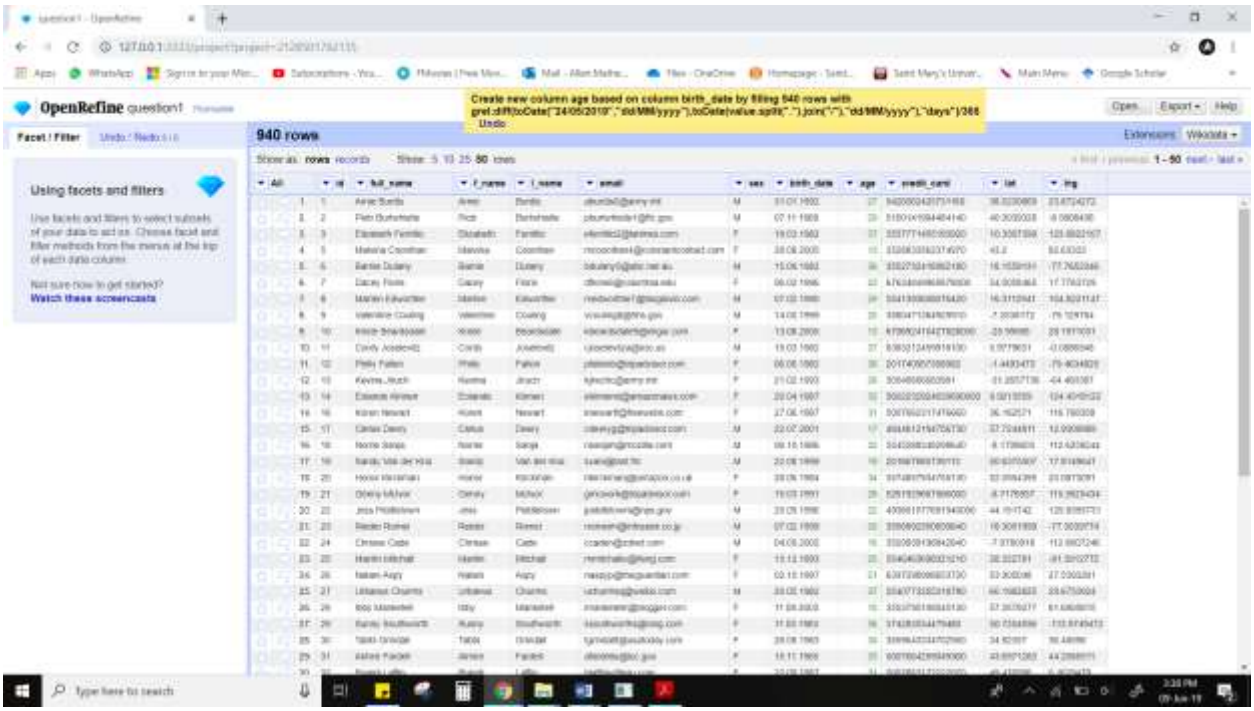

Fig 24: The age column was successfully created.

After performing a quick overview, it is evident that all the rows in the ***age*** column was properly calculated based on the data from the ***birth_date*** colume.

9) Only keep last 4 digits of credit card in the **credit_card** column (Do not create a new column).

To complete this task, individual cells from the '***credit_card***' column will be Transformed (**Edit cells** > **Transform…**).

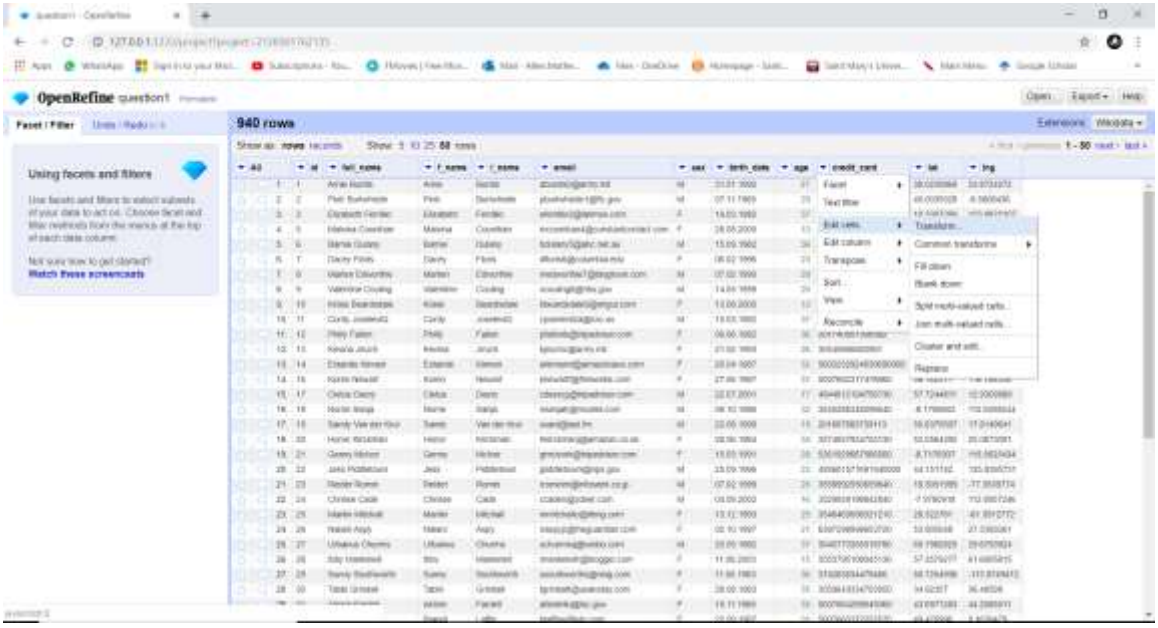

Fig 25: Selecting Transformation within Edit Cells from credit_card column.

The following expression (GitHub, 2019) is used to Transform the credit_card column:

value.substring(value.trim().length()-4)

For the above expression the "value" is the data of each individual row in the ***credit_card*** column. On utilizing the "value" with the ***trim*** function it returns a copy of the string, with leading and trailing whitespace removed. Then the ***substring*** function is used to create a substring from the 4th last index (calculated by subtracting 4 from the total string length) of the "value" string data all the way to the end of the string. This string is the last 4 digits of the credit card.

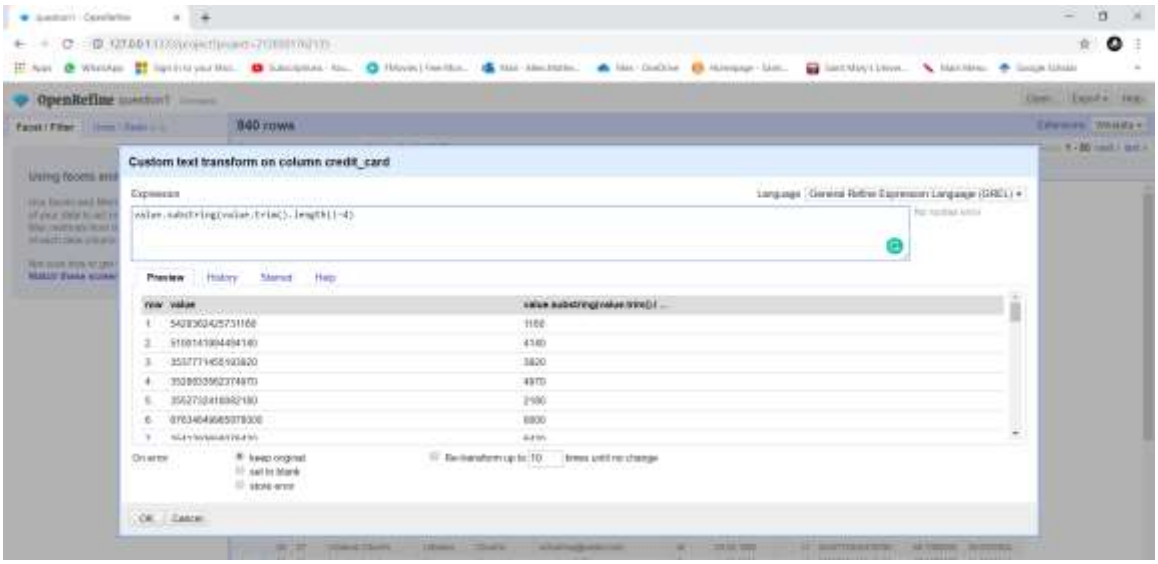

Fig 26: Expression to transform credit card string to the last 4 digits of the credit card

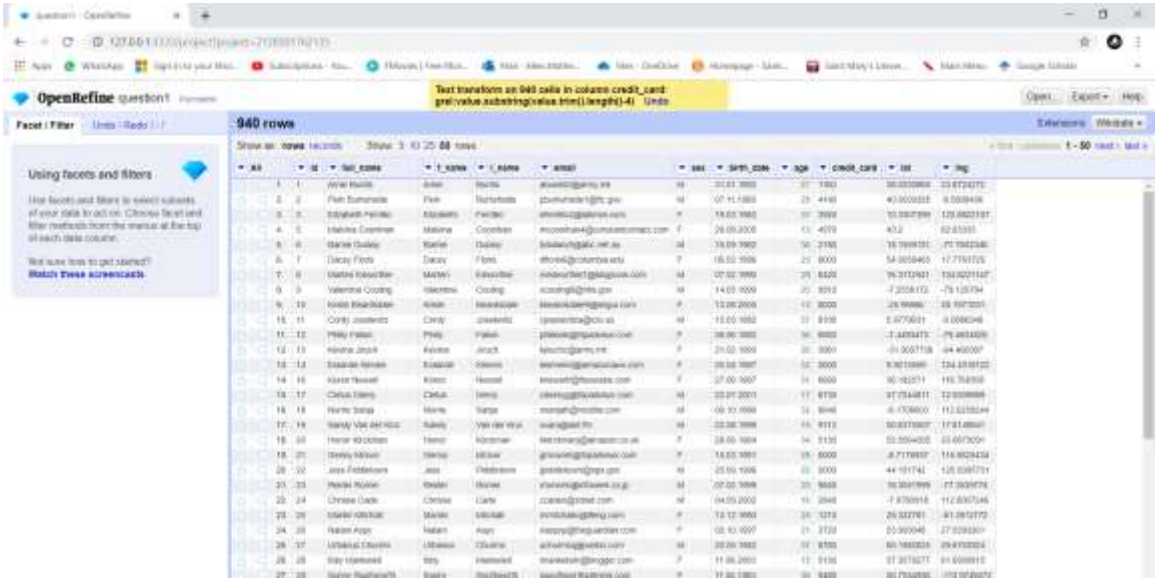

Fig 27: The credit_card column was successfully transformed

After performing a quick overview, it is evident that all the rows in the ***credit_card*** column was properly transformed.

10) Delete **full_name** column.

To complete this task, the ***full_name*** column needs to be deleted (**Edit column** > **Remove the column)**.

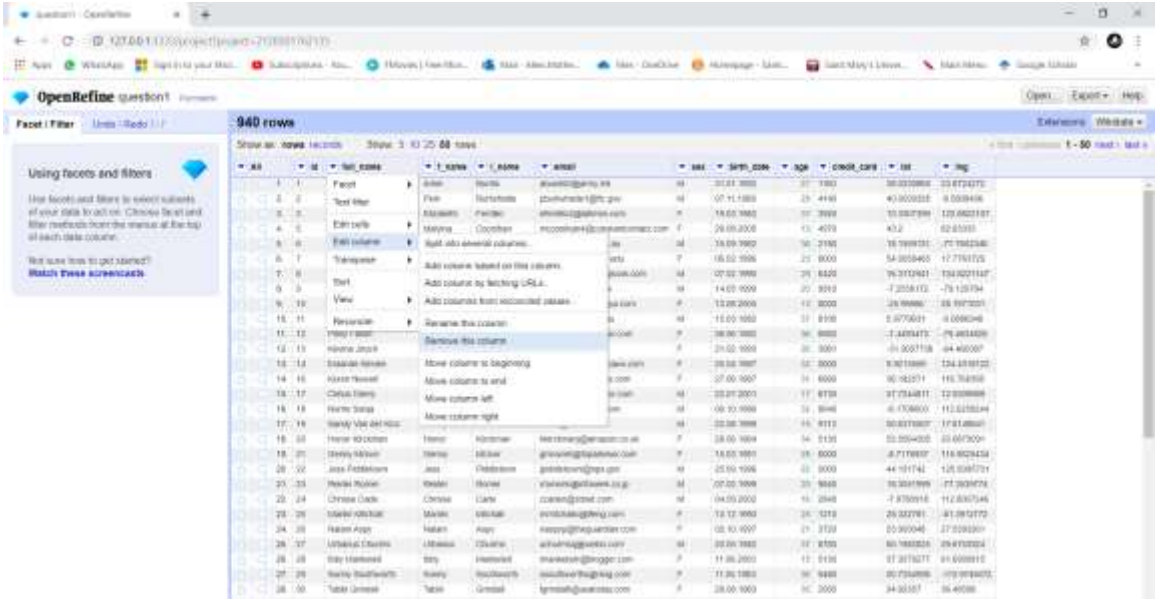

Fig 28: Selecting Remove the column within Edit Column from full_name column.

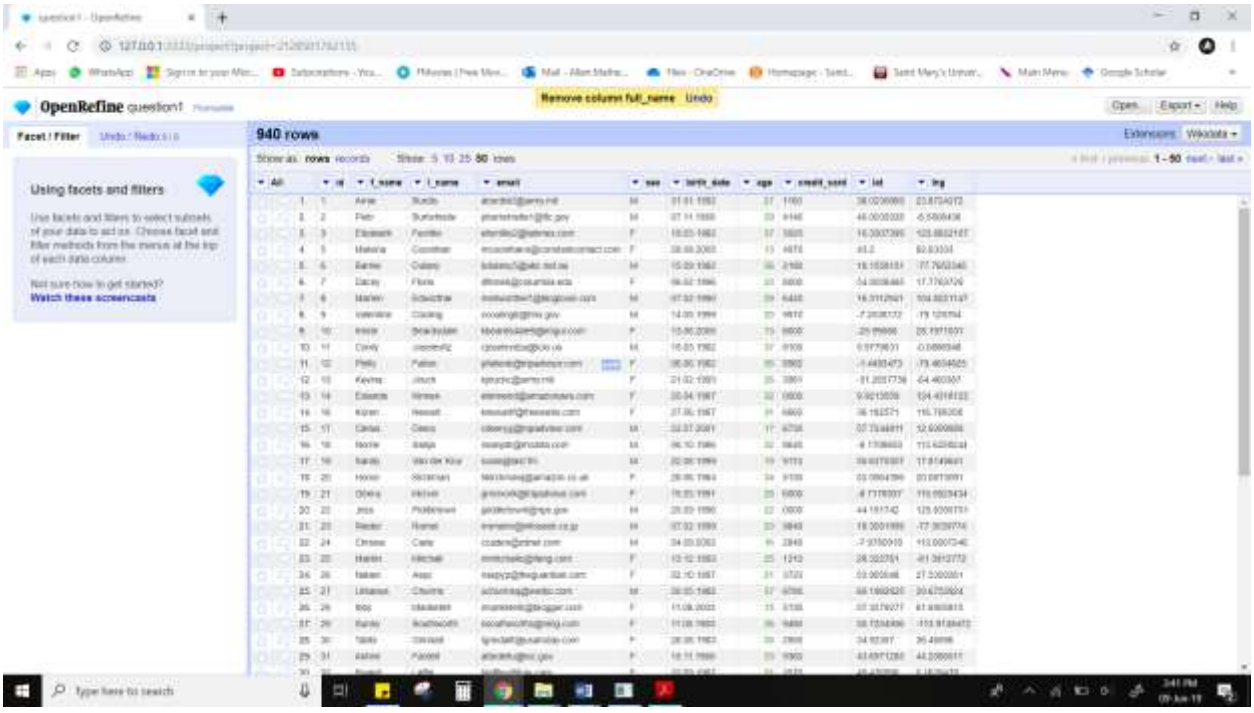

Fig 29: The full_name column was deleted successfully

11) Delete **birth_date** column.

To complete this task, the ***birth_date*** column needs to be deleted (**Edit column** > **Remove the column)**.

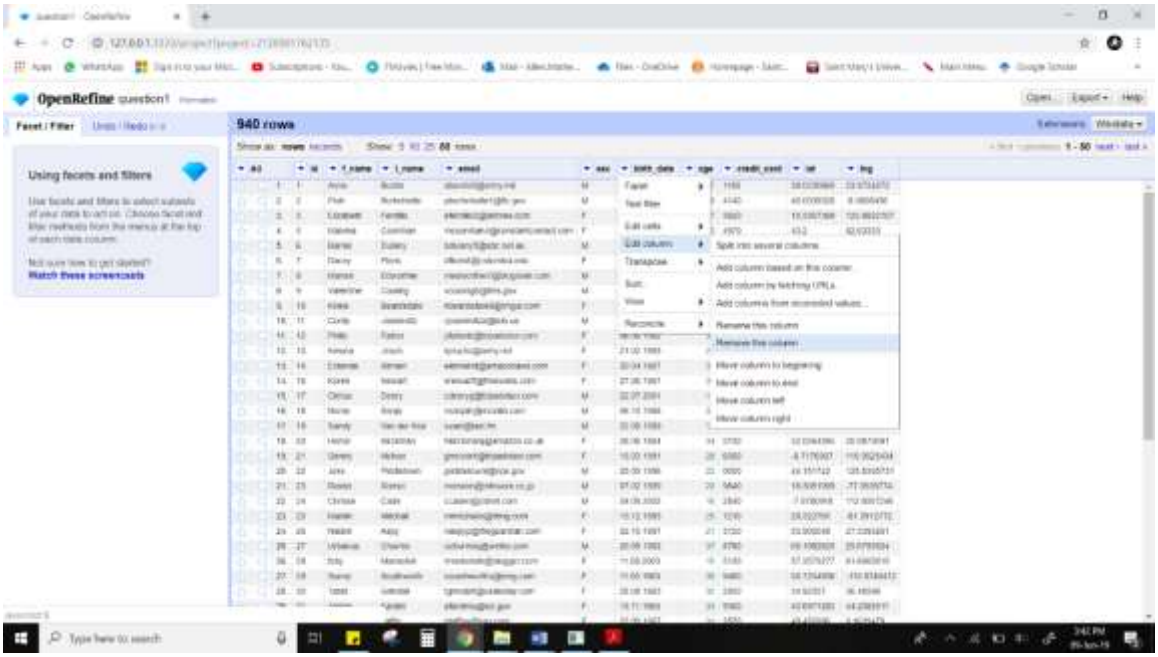

Fig 30: Selecting Remove the column within Edit Column from birth_date column.

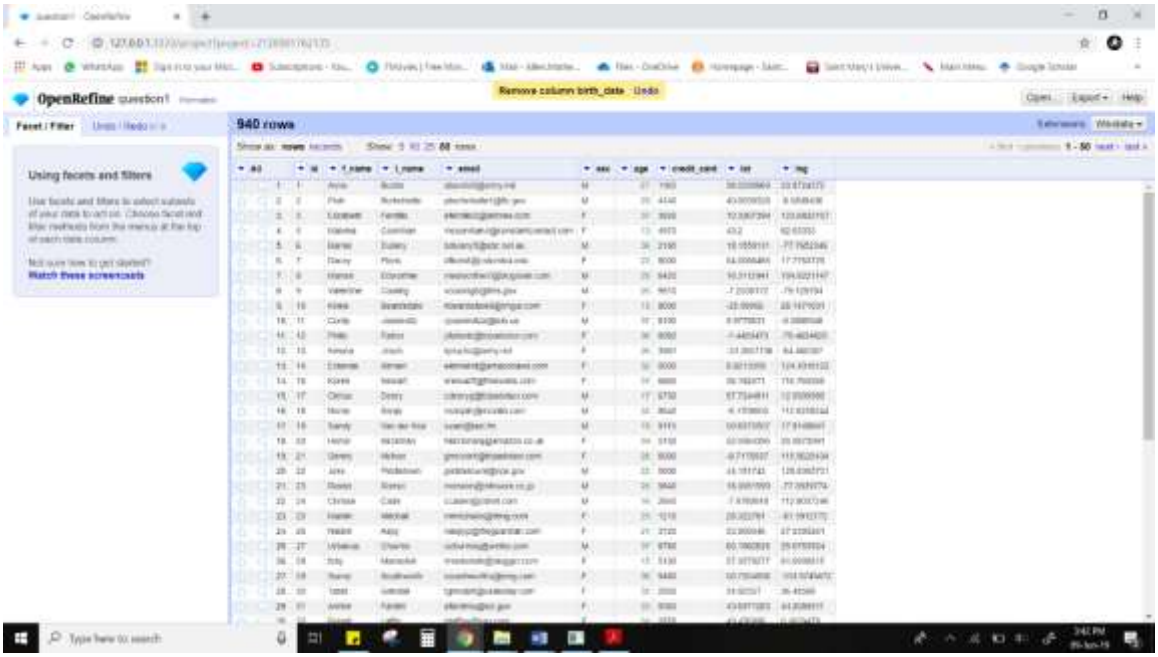

Fig 31: The birth_date column was deleted successfully

12) Signup for OpenCage Reverse Geocoding API (https://opencagedata.com/api)and get API Key.

The following Figure shows that an account was successfully created with OpenCage, from which we were able to get an API Key (i.e. 512728198e3f4ebf869f382852c8c949) that would be utilized for this project.

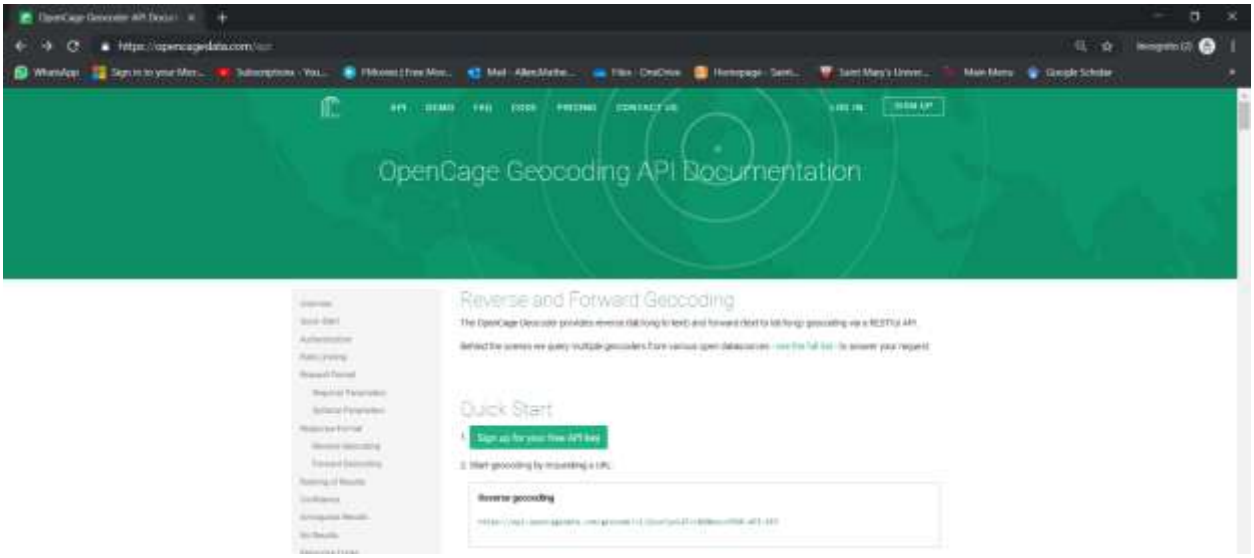

Fig 32: Open Cage Geocoding API Account/ API Key.

13) URL format to retrieve location (JSON Data): https://api.opencagedata.com/geocode/v1/json?q=LAT+LNG&key=YOUR-API-KEY &
14) You can replace the text highlighted in red using expression and retrieve location detail (JSON data). You can store this JSON data into **raw_data** column (You will create this column).

To complete this task, data from the '***lat***' and '***lng***' column will be used to fetch JSON from a web service and create the additional column '***raw_data***' (**Edit column** > **Add column by fetching URLs**).

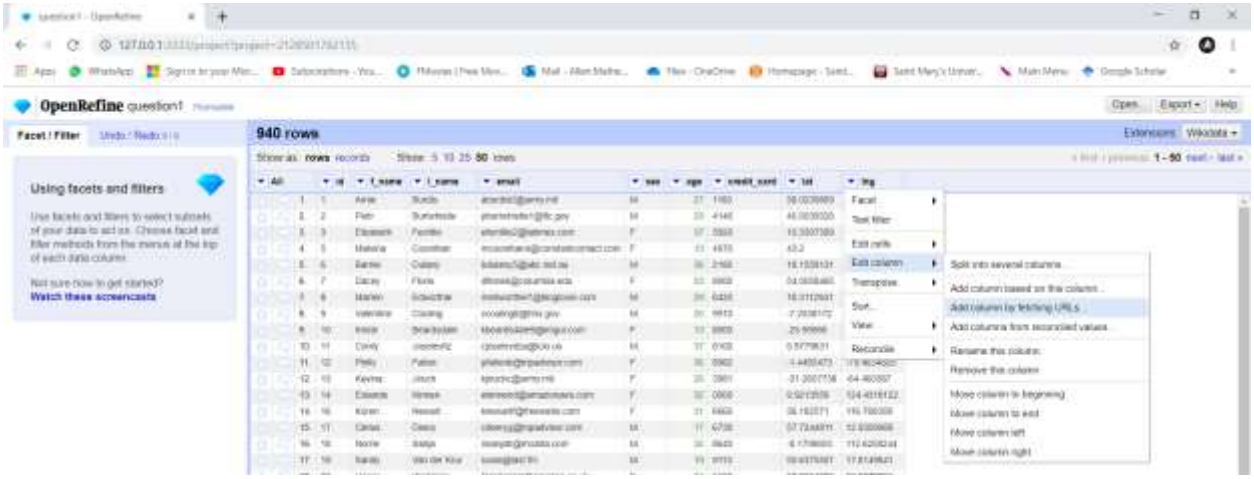

Fig 33: Add column by fetching URL within Edit Column from lng column.

The following expression (GitHub, 2019) is used to retrieve the geographical location detail (JSON data) using URLs:

'https://api.opencagedata.com/geocode/v1/json?q='+cells['lat'].value+'+'+value+'&key=512728198e3f4ebf869f382852c8c949'

For the above expression the "value" is the data of each individual row in the ***lng*** column and "**cells['lat'].value"** is the data of each individual row in the ***lat*** column. This latitude and longitude data are concatenated with the URL string containing the API key.

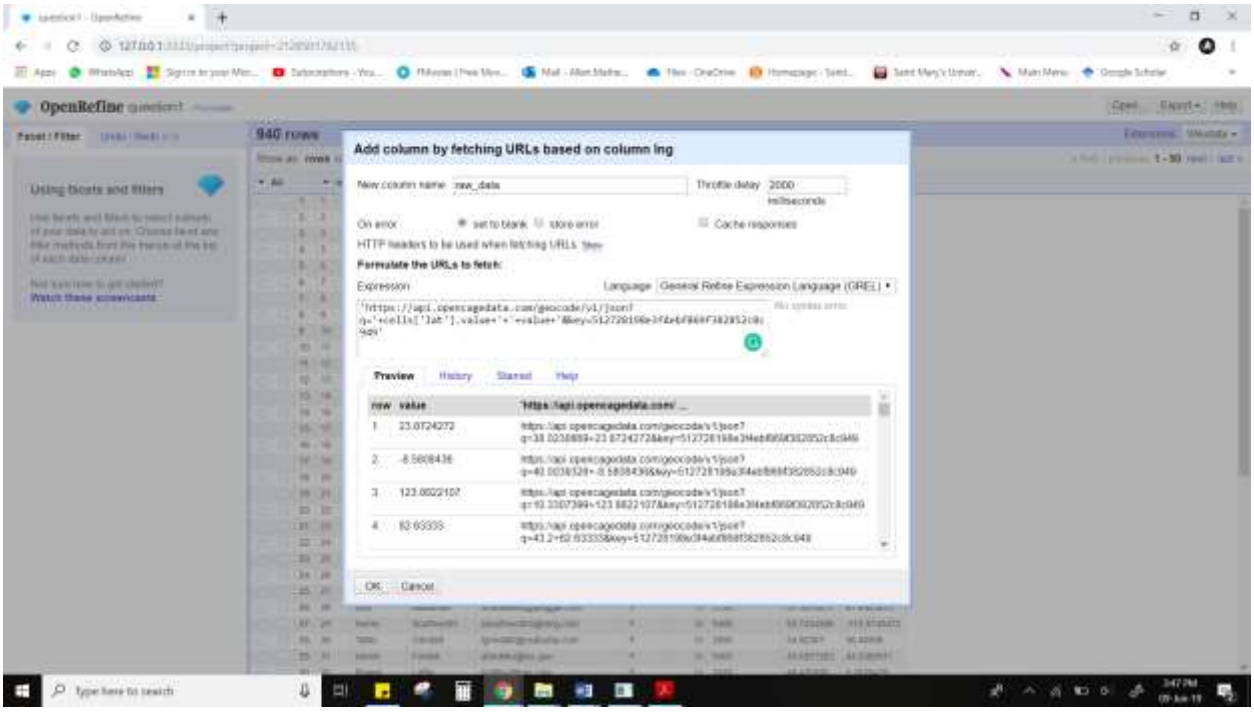

Fig 34: Expression to derive the URL address from the lat and lng column.

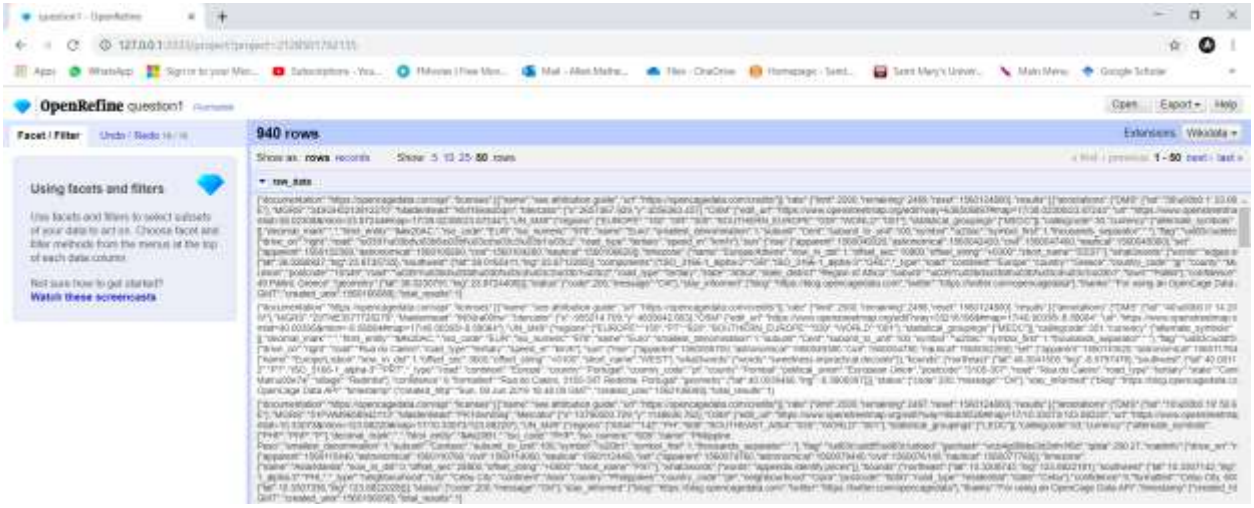

Fig 35: The raw_data column was created successfully.

After performing a quick overview, it is evident that all the rows in the ***raw_data*** column have properly fetched the JSON Data from the webservice.

15) Filter continent and country from the JSON data and store into **continent and country** columns (You will create these two columns).

To complete this task, data from the '***raw_data***' column will be used to create the additional column 'continent' and 'country' (**Edit column** > **Add column based on this column**).

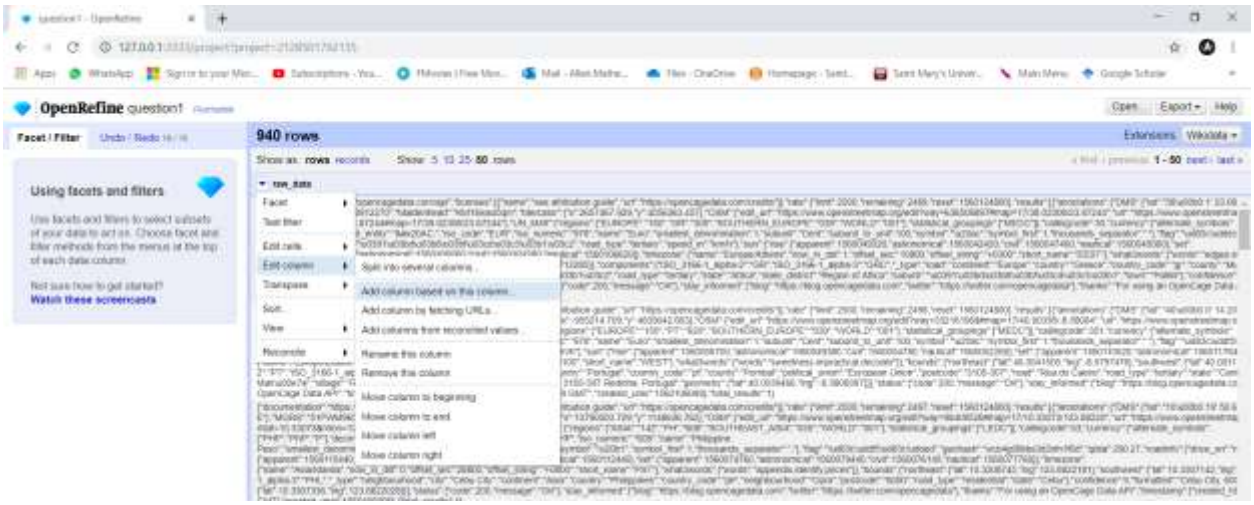

Fig 36: Creating new column continent andcountry based on data from raw_data column.

The following expressions are used to extract the continent and country from the JSON Data:

value.parseJson().results[0].components.continent

For the above expression (GitHub, 2019) the "value" is the data of each individual row in the ***raw_data*** column. On utilizing the "value" with the ***parseJson*** function it returns a JSON Object of the string. The first element (i.e. position [0]) from the result array is selected, it has the "components" array. Finally, the data pertaining to the continent and country can be retrieved by traversing the "components" array.

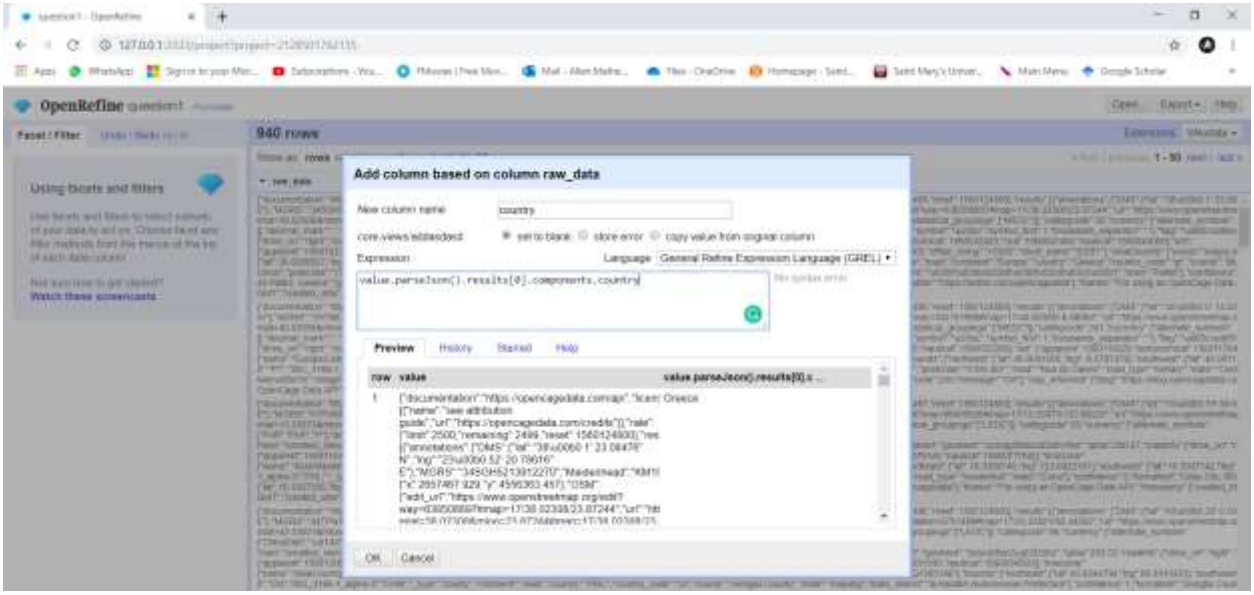

Fig 37: Expression to extract country data from JSON data

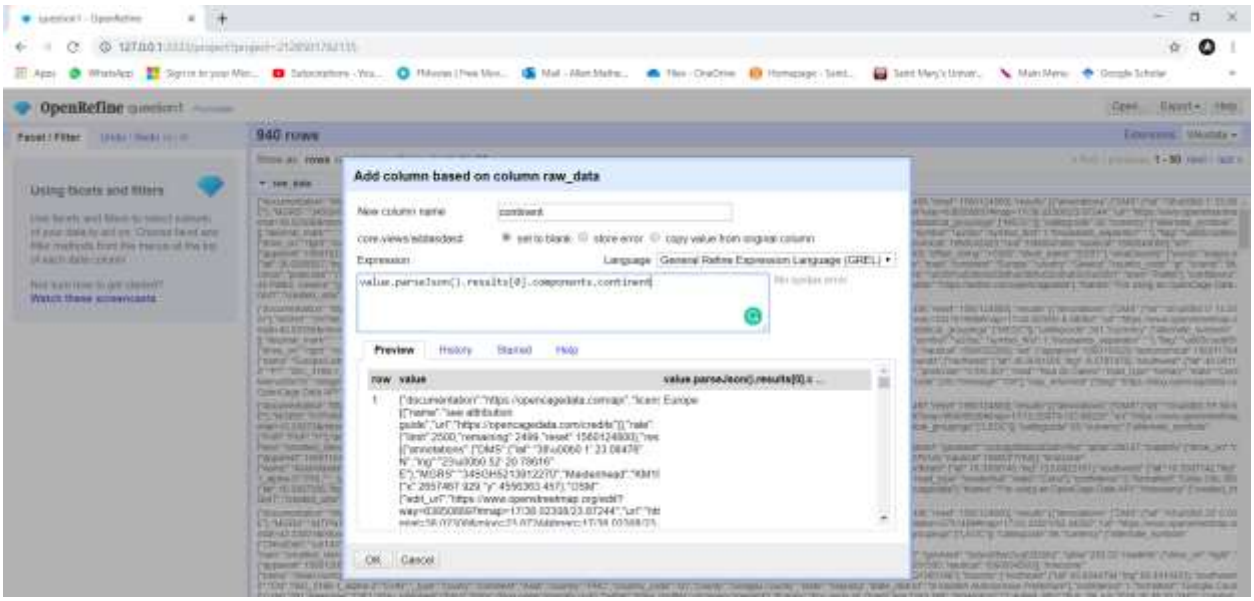

Fig 38:Expression to extract continent data from JSON data

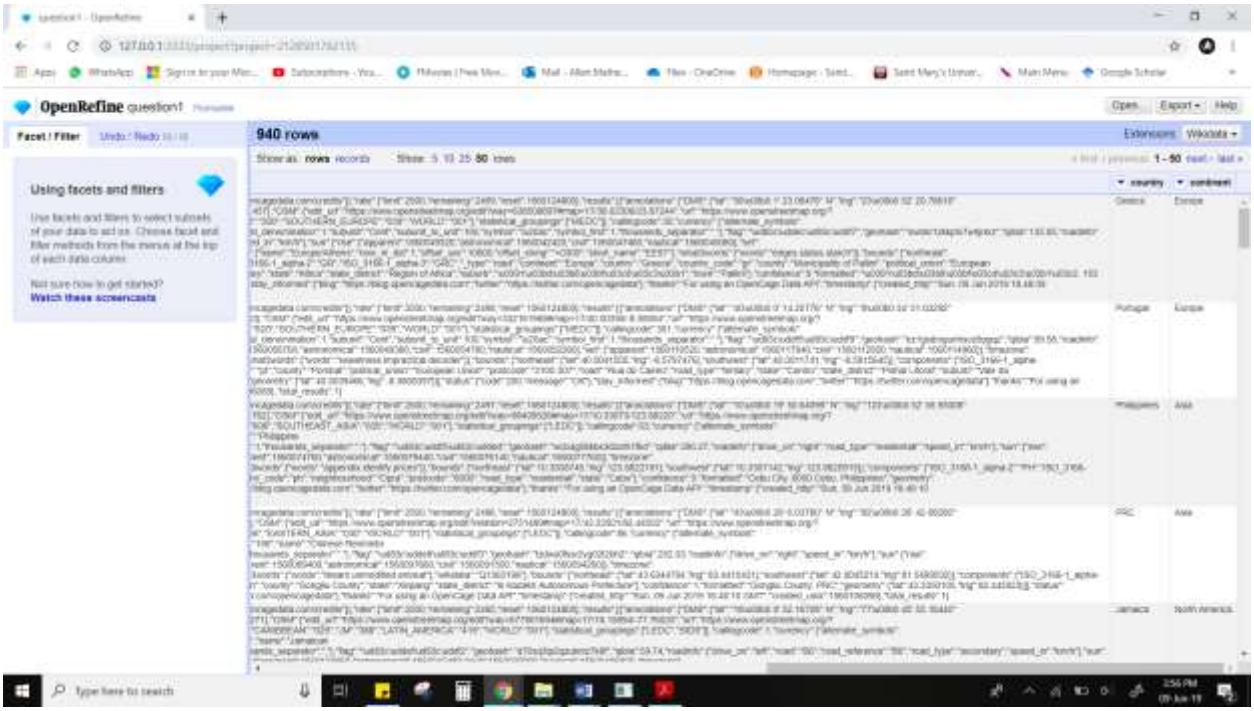

Fig 39: The continent and country column were successfully created

After performing a quick overview, it is evident that the country and continent column data were properly extracted from the ***raw_data*** column.

16) Delete **raw_data** column.

To complete this task, the ***raw_data*** column needs to be deleted (**Edit column** > **Remove the column)**.

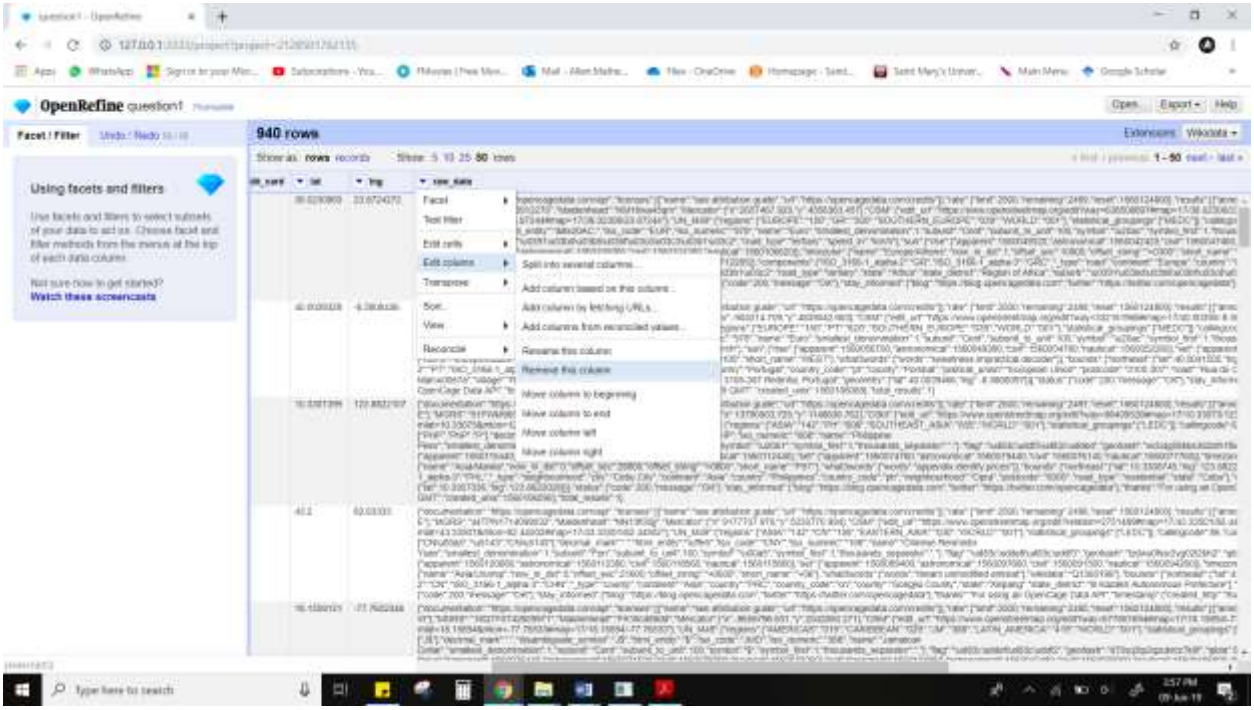

Fig 40: Selecting Remove the column within Edit Column from raw_data column.

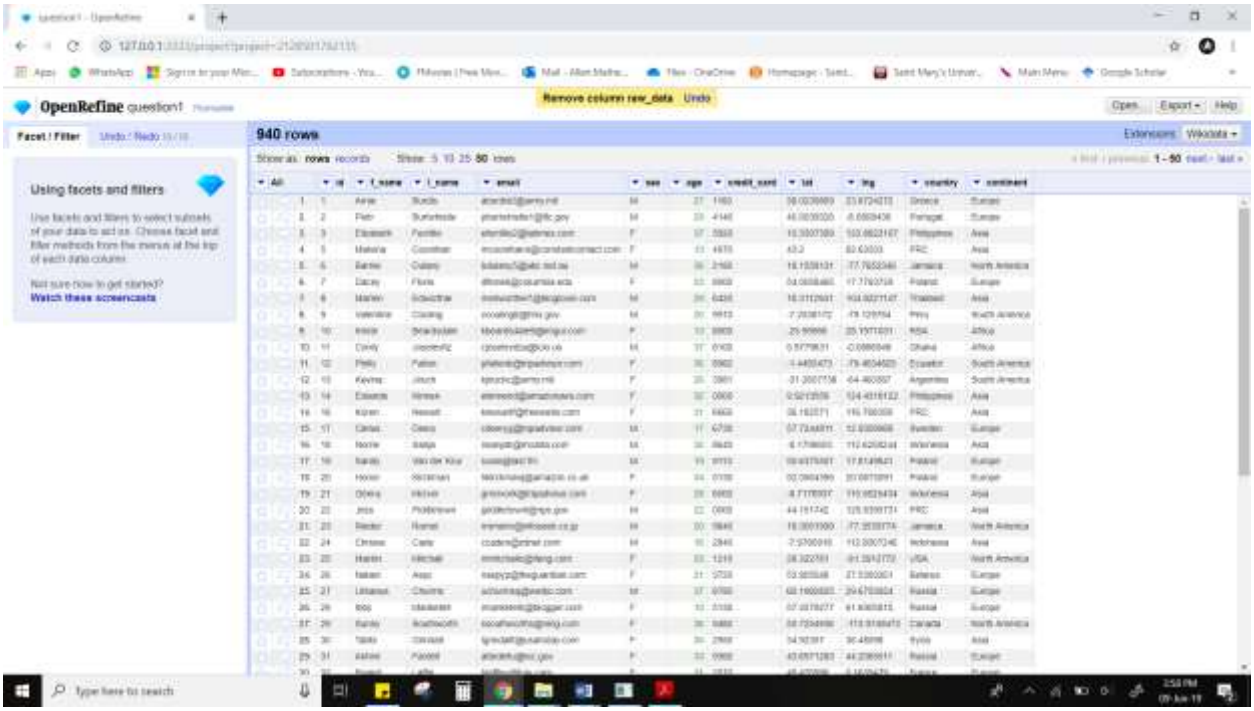

Fig 41: The raw_data column was successfully deleted/removed

17) Export project as well as CSV file. Rename project file as **solution1.tar.gz** and rename CSV file as **solution1.csv**

The whole OpenRefine project can be exported as an *.tar.gz* file, it will contain the history of the changes made as well as the data used in the project (**Export > Export project > Export to local**).

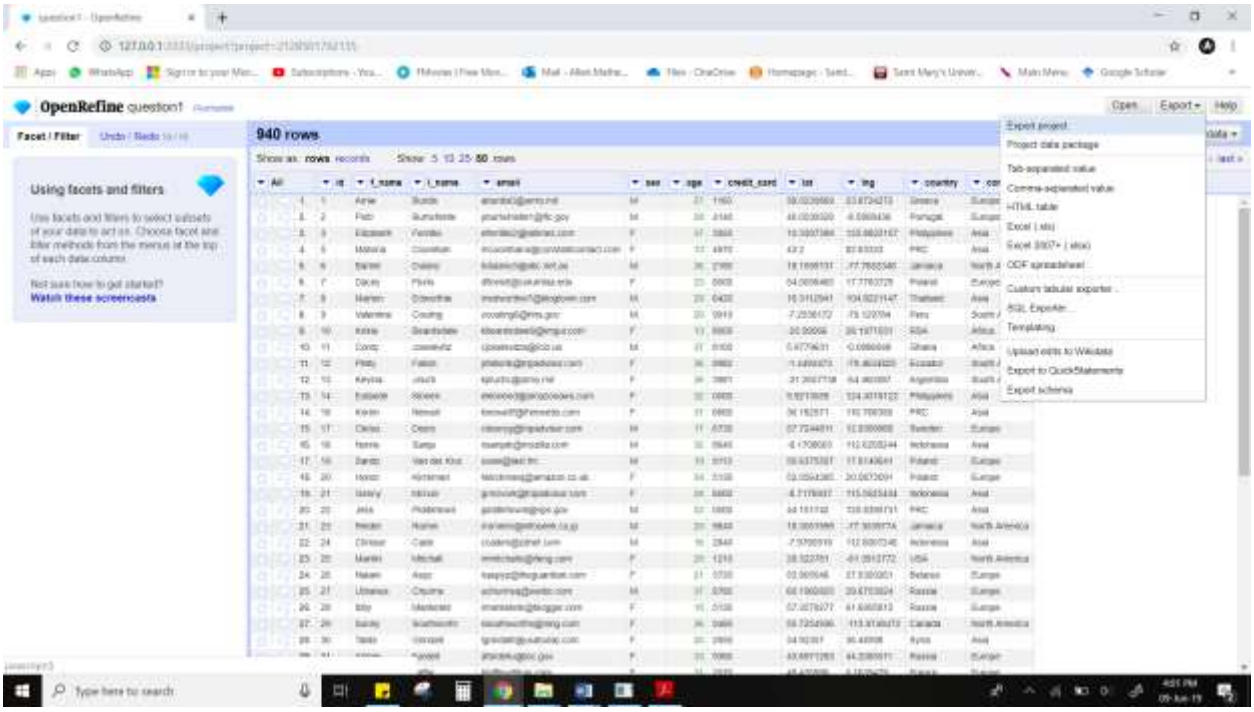

Fig 42:Exporting the OpenRefine Project as tar.gz file

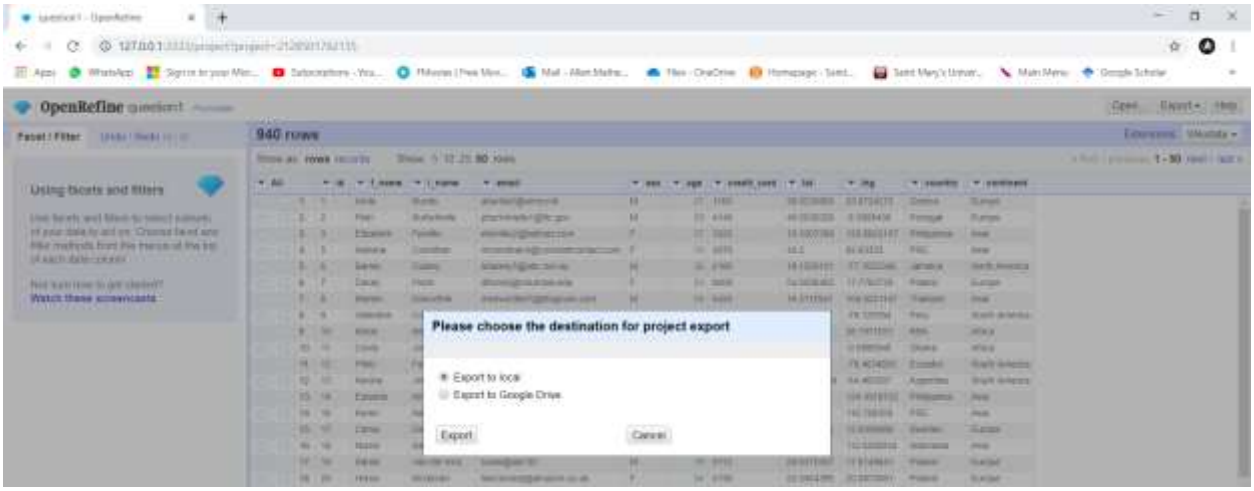

Fig 43: Exporting project to local device

The data itself can be exported as a *csv* file (**Export > Comma separated value**).

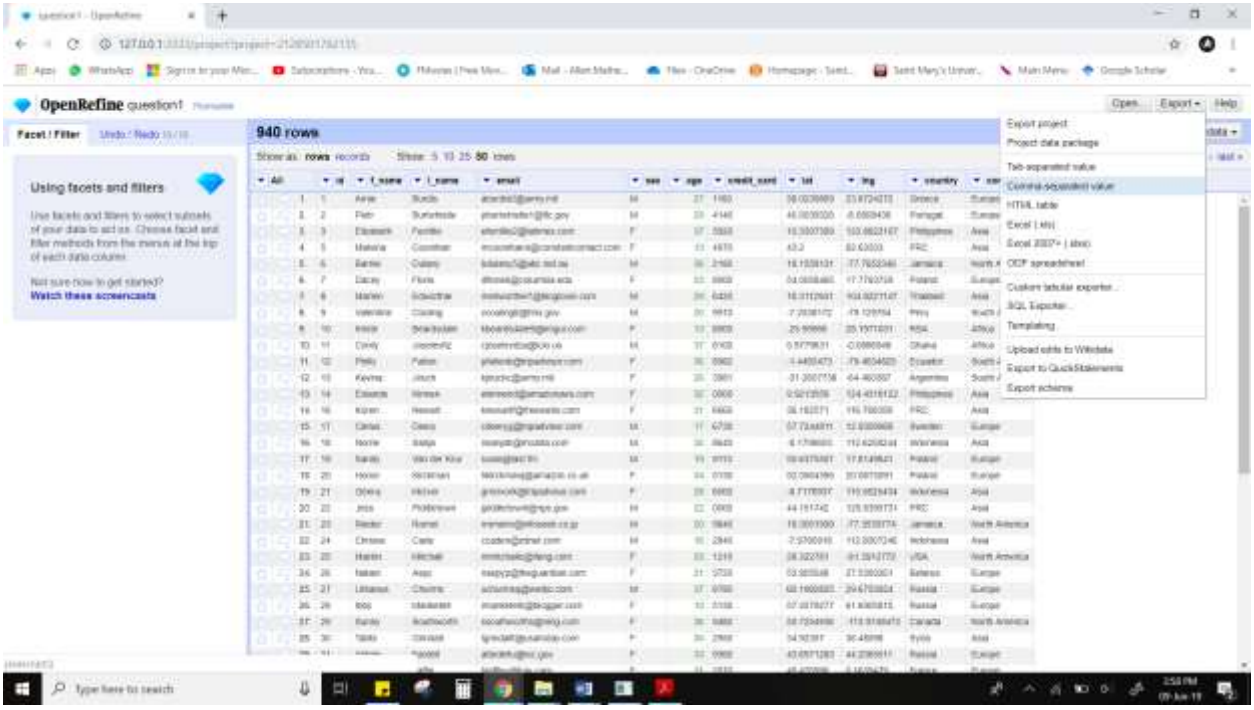

Fig 44:Export to CSV file as .csv

The **solution1.tar.gz** and **solution1.csv** were successfully exported.

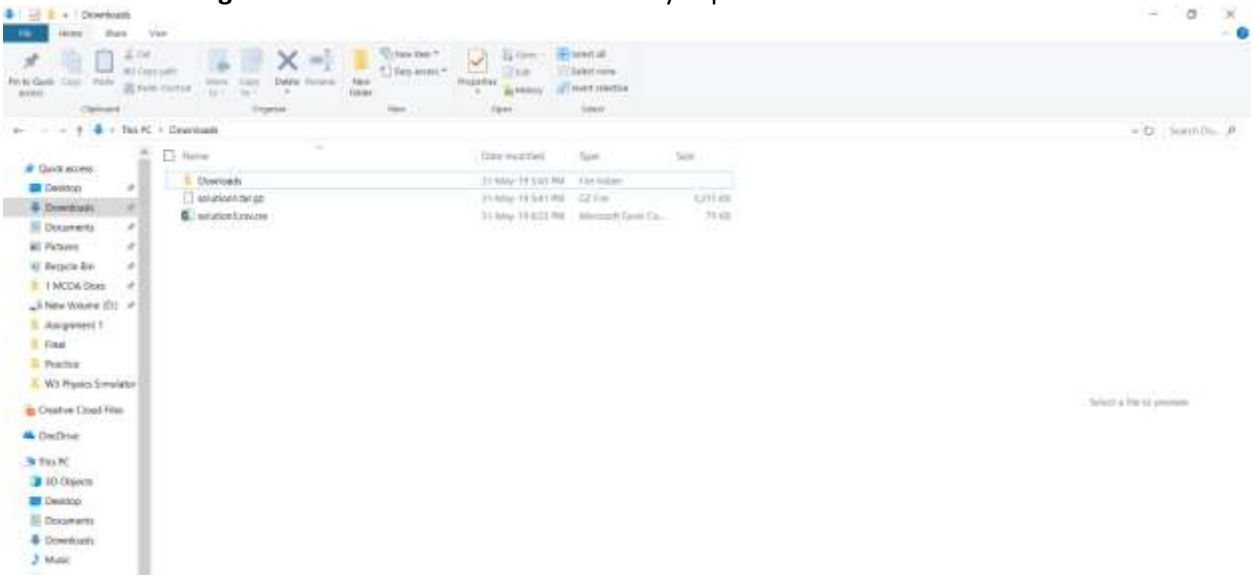

Fig 45: Saved Data.

After performing a quick overview, it is evident that the **solution1.tar.gz** and **solution1.csv** files were properly exported from the question1 OpenRefine project column.

C. Brief on Work Done on OpenRefine:

OpenRefine is a robust tool that is used for data preparation. It was installed on the desktop systems, then a new project was created where the dataset that was provided was loaded into the project.

A set of tasks were carried out on this dataset such as, various data cleaning operation were performed, a few columns were transformed from one format into another, few new columns were created where its records were based on the value derived from existing columns. Web services such as OpenCage Reverse Geocoding API was used to get the location details (i.e. JSON Data), to retrieve the data pertaining to the location an API Key is required which can be attained by creating an account with OpenCage as well as the latitude and longitude details are required which were provided from the respective columns in the dataset. Once the JSON data was retrieved, new columns are created where its records were extract from certain components from the entire JSON data. The project was exported as .tar.gz file and the cleaned dataset was export as a CSV File.

Google Data Prep:

Google Data Prep is a cloud-based application that can be used for data preparation. It can accurately predict and provide suggestions about the tasks to use for Data Preparation based on the user's interaction with the UI of the dataset that was loaded.

A. Tasks Performed:

The following tasks are performed to gain a better understanding of the Data Preparation concepts and the Google Data Prep tool:

1) Signup for Google Cloud Dataprep

The following Figure shows that an account was successfully created with Google Cloud Dataprep.

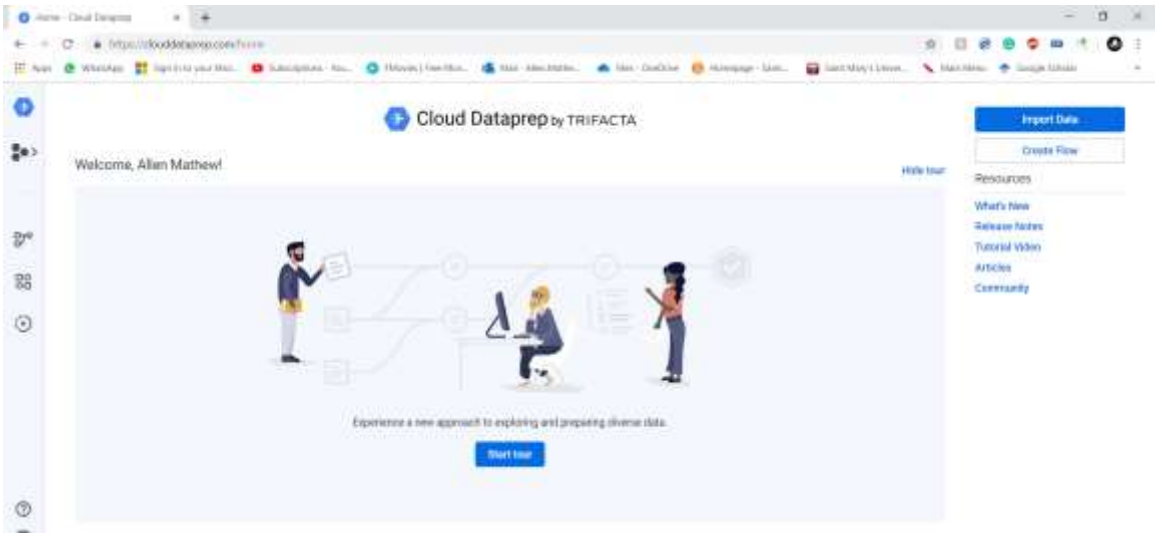

Fig 46: Start Page/ Console age of Cloud data Prep.

A flow called "Question 2" was created, to carry out the tasks for this project

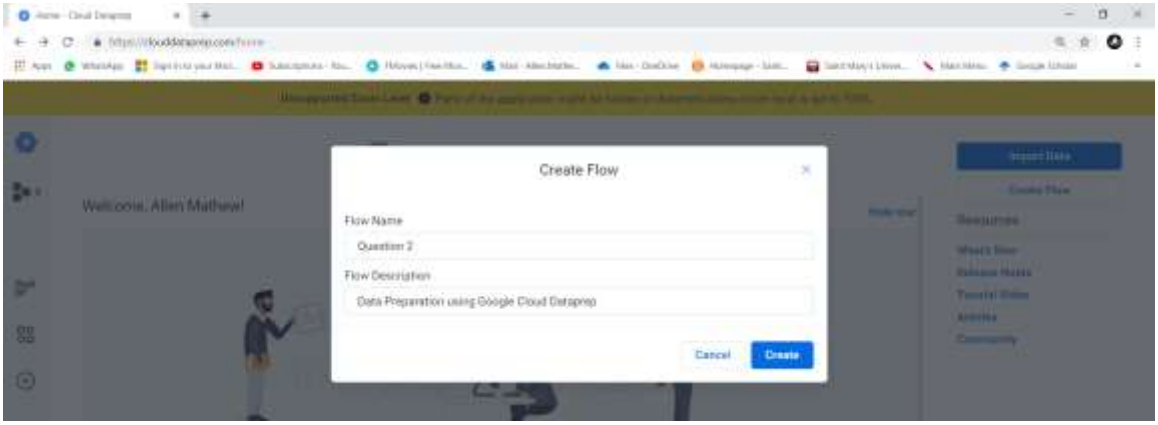

Fig 47: A flow is created with title as Question 2.

2) Load question2_1.csv, question2_2.csv, question2_3.csv

The following Figure shows that the respective Excel files were uploaded to the Google Cloud Dataprep.

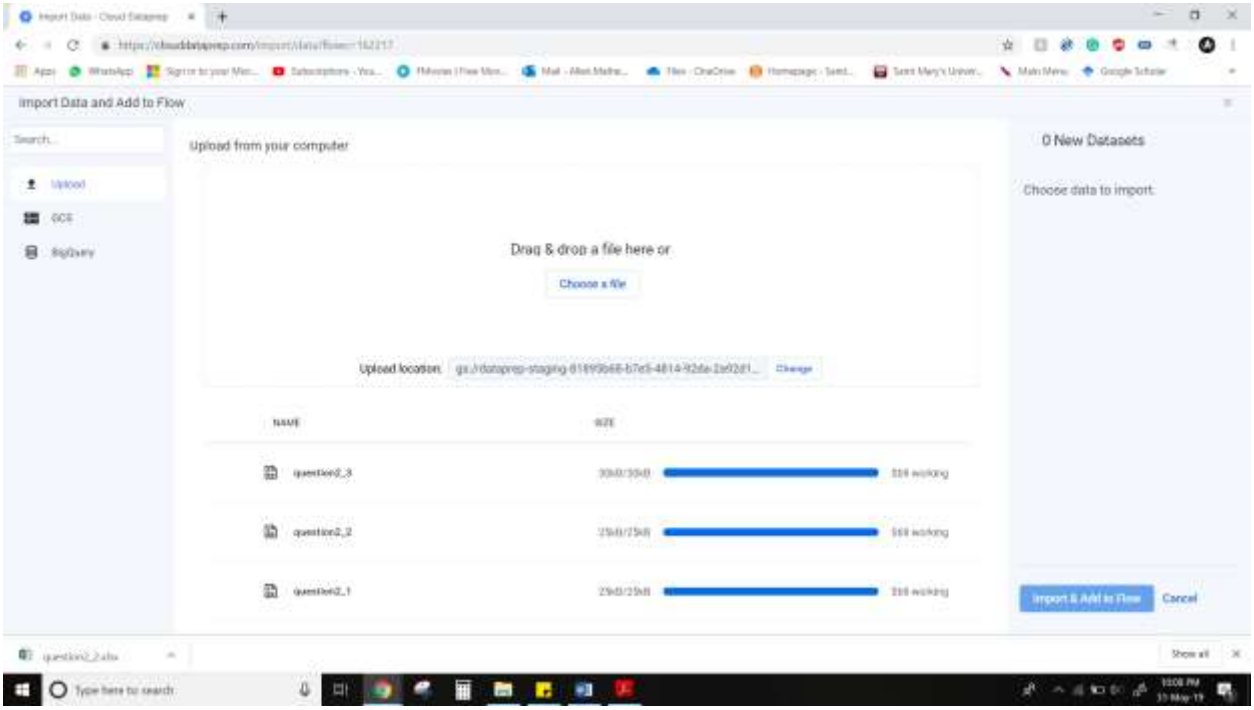

Fig 48: Imported Files.

The following Figure shows that the flow was successfully created, and the respective Excel files were imported correctly.

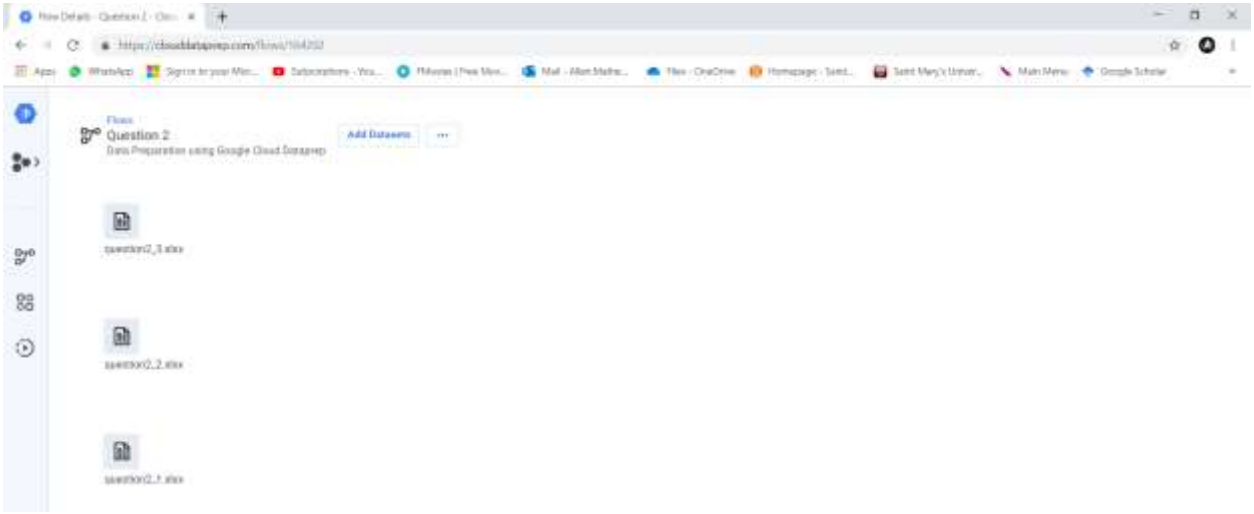

Fig 49:Dataset is uploaded to the flow

3) Data Preparation on Question2_3 Dataset

The question2_3 dataset consists of 2 columns:

- **stock_symbol**:
  - A text string used to uniquely identify the stock.
- **stock_market_cap**:
  - A text string that displays the amount of capital present for the stock. Data is displayed in either millions or billions.

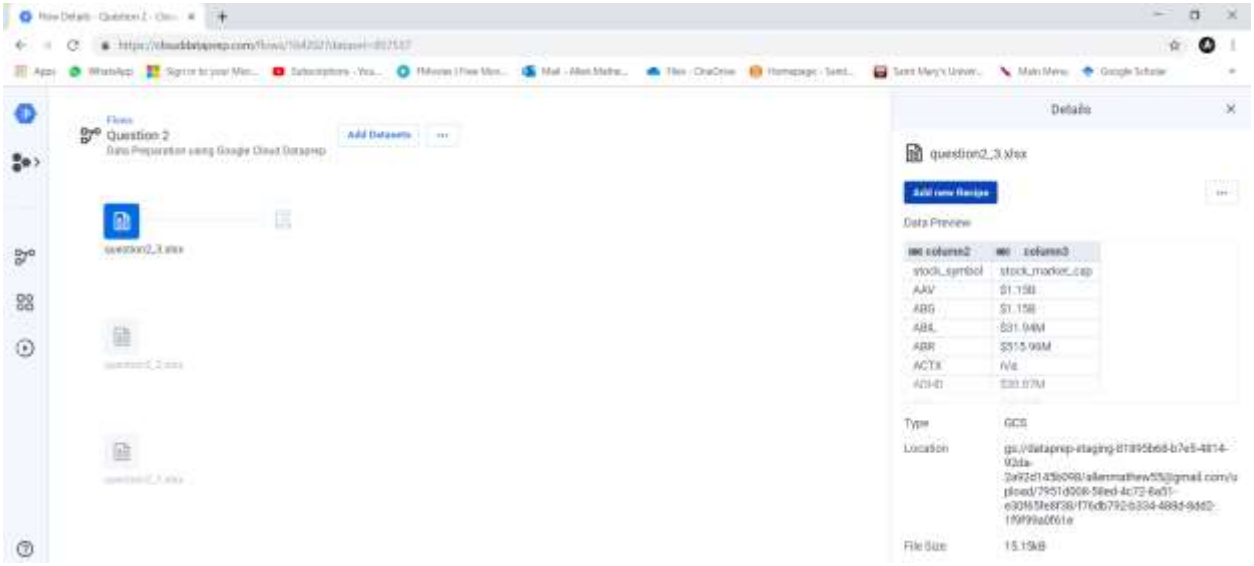

Fig 50: Reviewing Column Details for dataset question2_3.

To clean the dataset a new recipe is added to the question2_3 dataset.

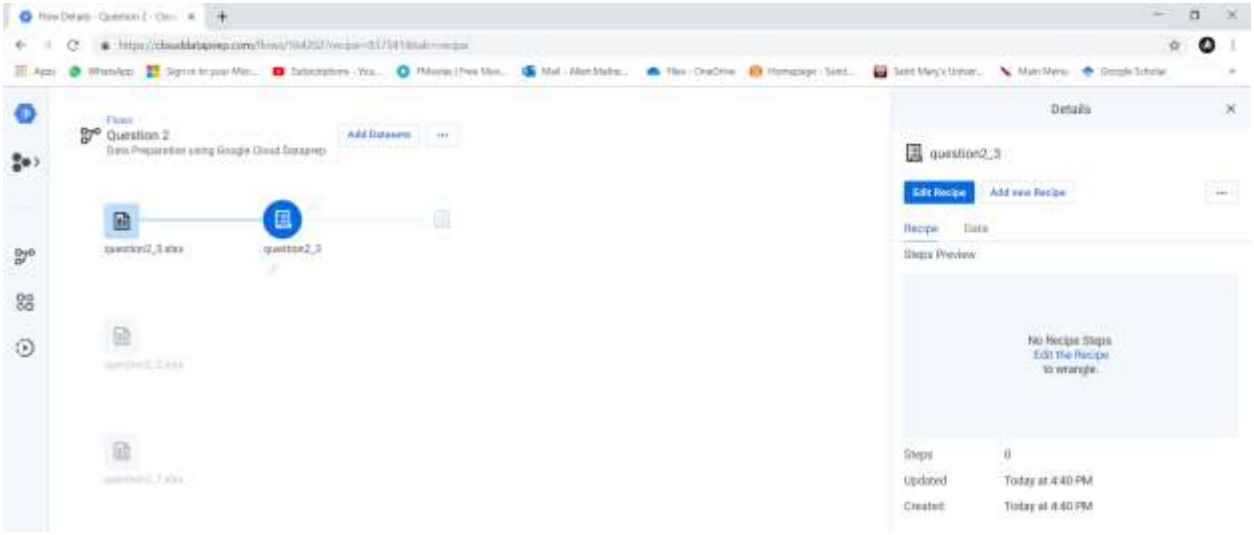

Fig 51: Adding new Recipe

On reviewing the initial dataset (seen in the following figures), few observations can be drawn:

1. The columns in the dataset have been given default headers while the first row in the dataset is not recognized as the appropriate headers of the respective columns.
2. The stock_market_cap column has multiple records with a value of not applicable(n/a).
3. The dataset has multiple duplicate records.
4. The dataset has no blank records.

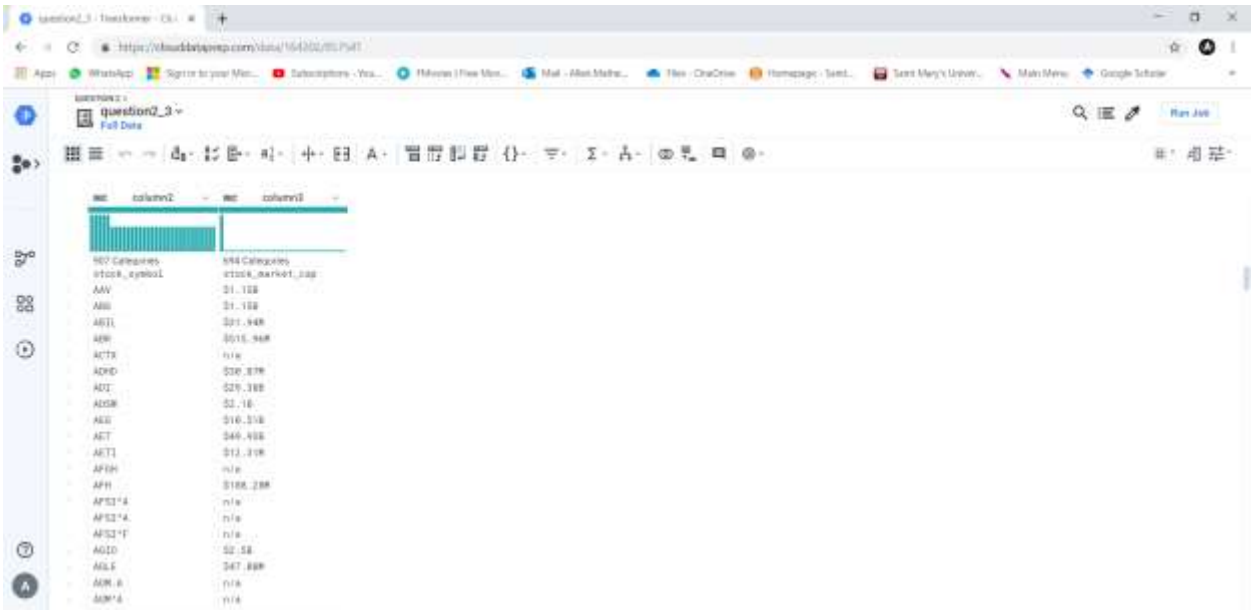

Fig 52: Reviewing stock symbol and stock market cap via Grid Format

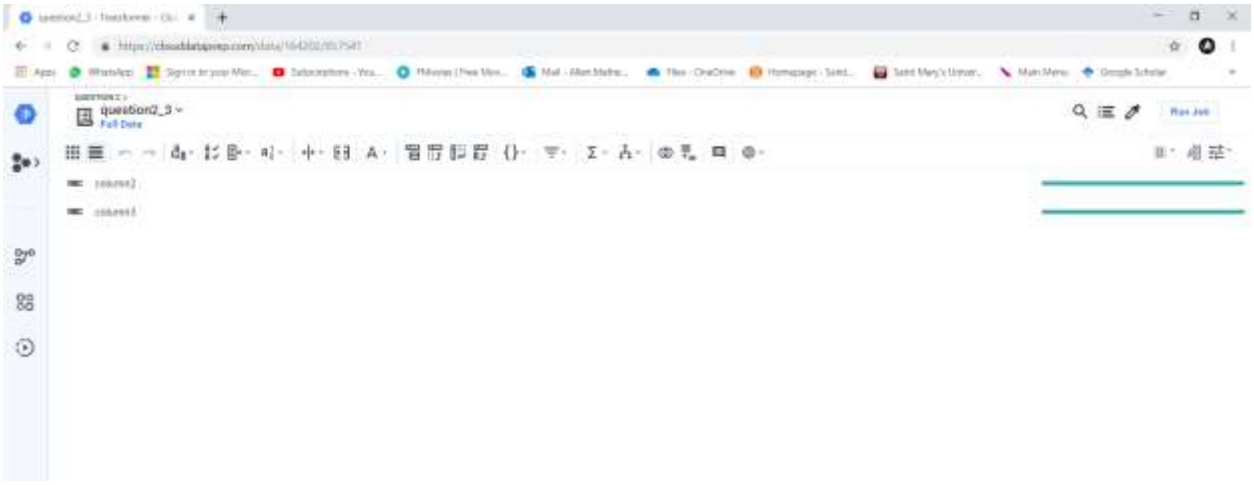

Fig 53: Reviewing stock symbol and stock market cap column via Column Format

3.1) Requirements for question2_3 Dataset :

Various steps need to be performed on the question2_3 dataset:

- The first row need to be made the header.
- The data needs to be cleaned (i.e. removing ‘n/a’ records as well as duplicate records)
- Remove all rows where stock_market_cap is greater than 30 million
- To maintain consistency all records in stock_market_cap are converted from billion to million

Step 1: Assign the 1st row as header

On selecting the first row in the dataset, Google Data Prep automatically provides several suggestions. The “Header” option that was suggested is added to the recipe, it converts the first row to the header.

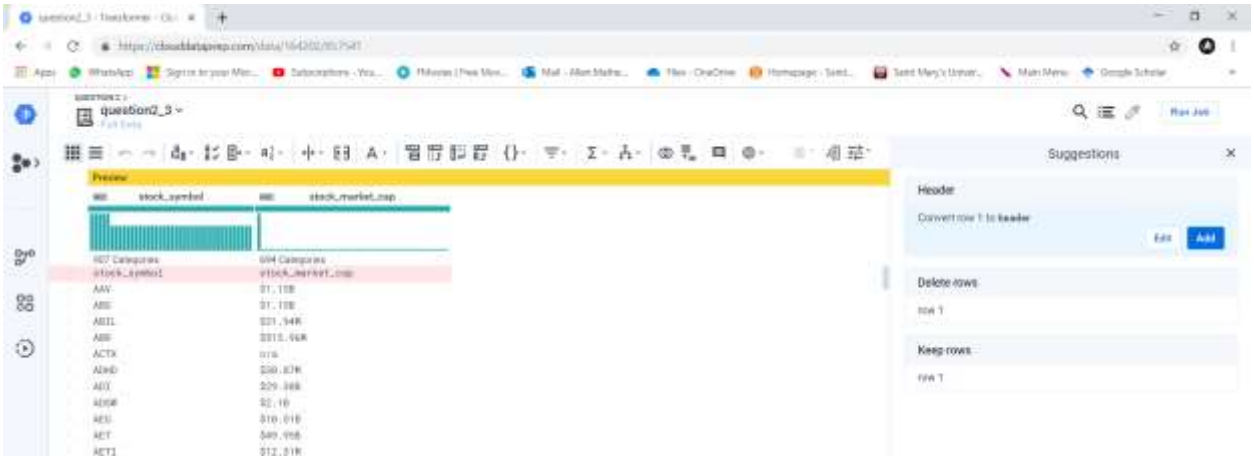

Fig 54: Assigning the 1st row of the dataset as header

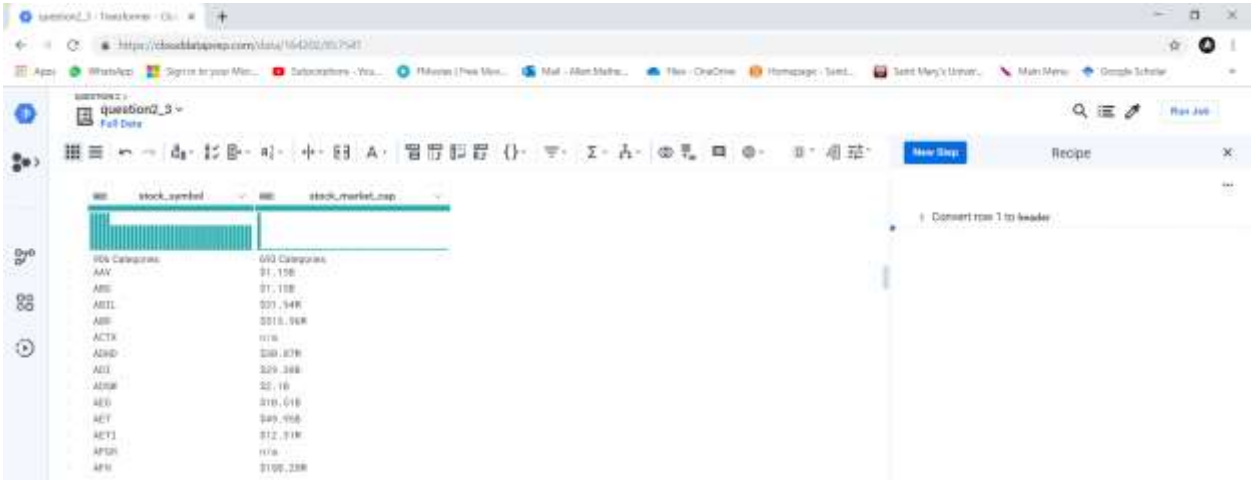

Fig 55: The first row was successfully assigned as the headers of the respective columns

Step 2: Remove rows with value as n/a

On selecting/highlighting a single “n/a” records in the stock_market_cap column, Google Data Prep automatically provides several suggestions.

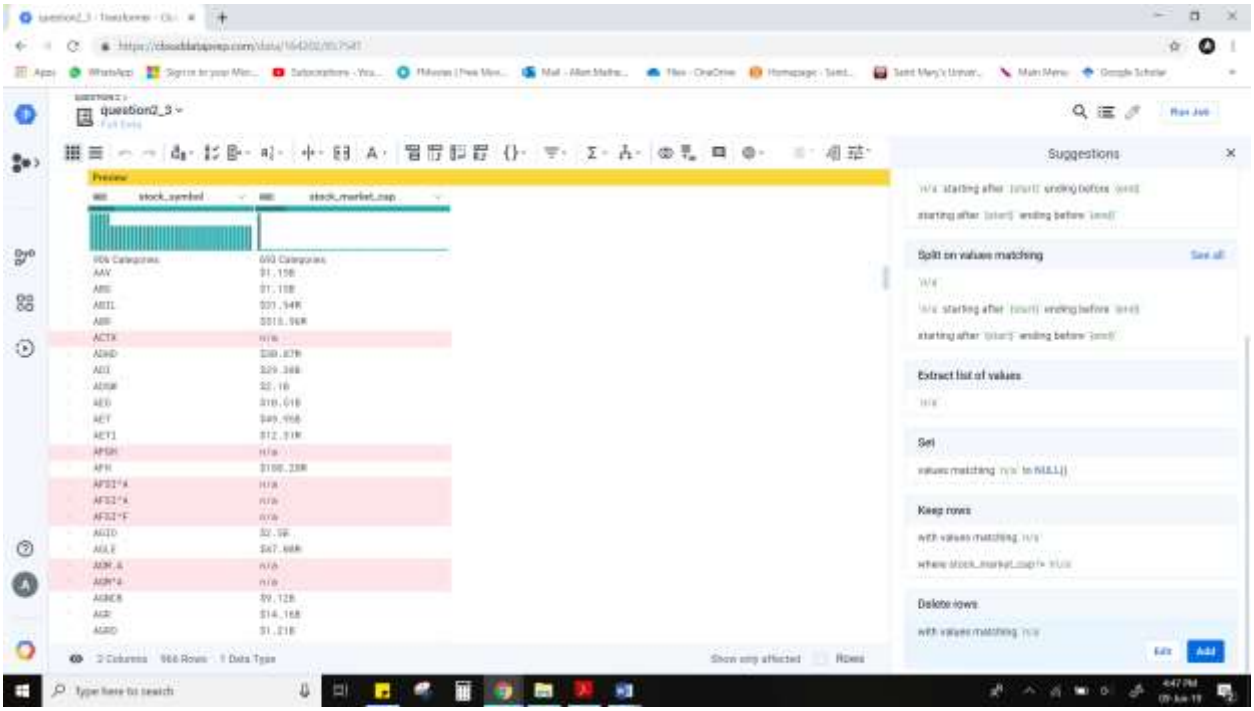

Fig 56: Removing Rows for every occurrence of NA’s in stock_market_cap column

The “Delete rows” option that was suggested, is added to the recipe, it deletes rows (i.e. 153 rows) from the dataset once an occurrence of “n/a” is observed in the stock_market_cap column.

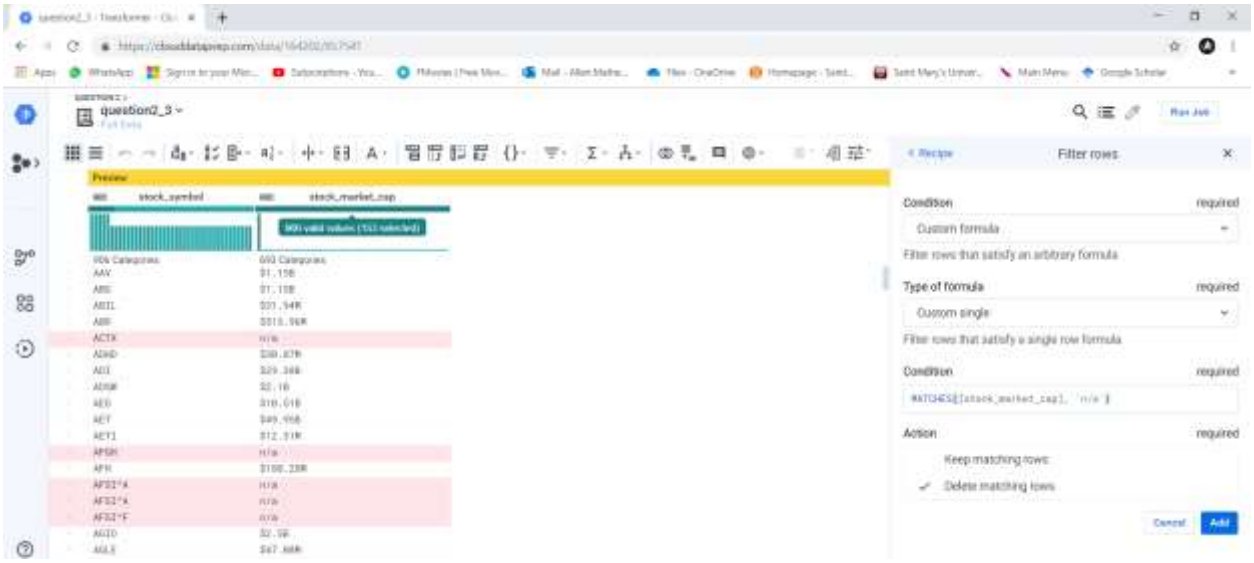

Fig 57: There are 153 rows where the occurrence of “n/a” is observed in the stock_market_cap column

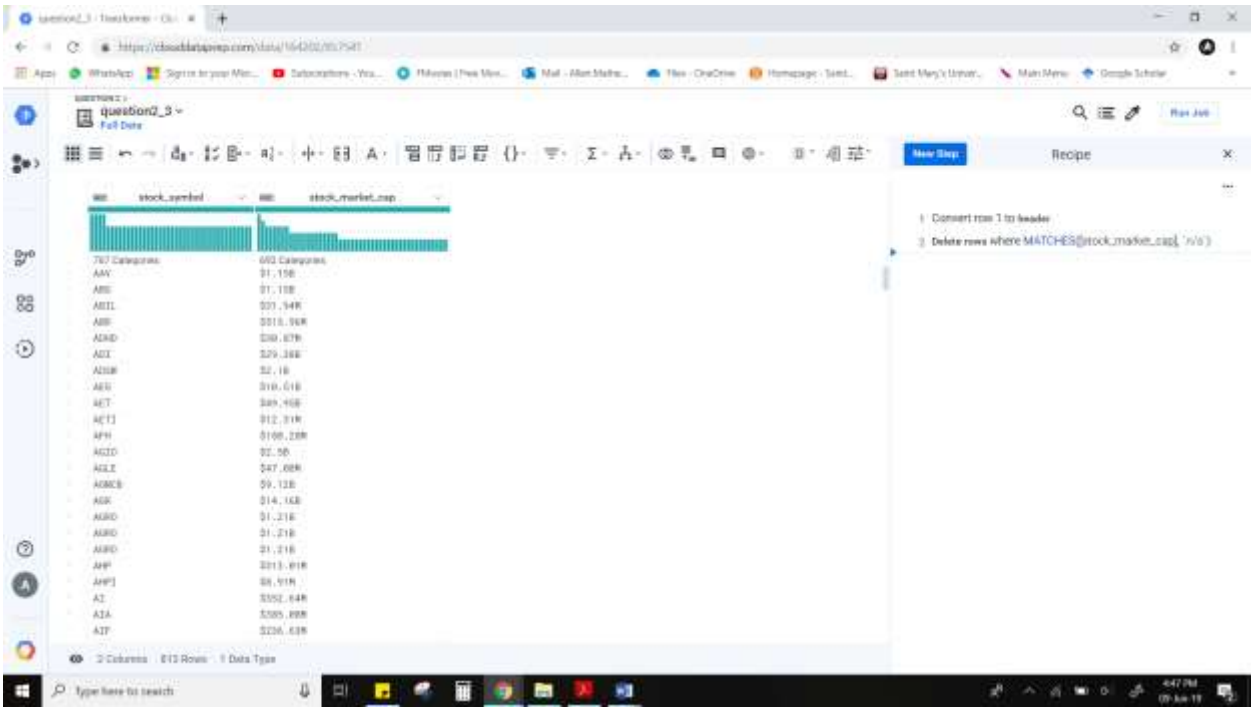

Fig 58: All rows with a record of "n/a" in the stock_market_cap column have been successfully deleted

Step 3: Remove rows with Duplicate values

The searching tool located at the top right corner of the browser can be used to select the "Remove duplicate rows" option.

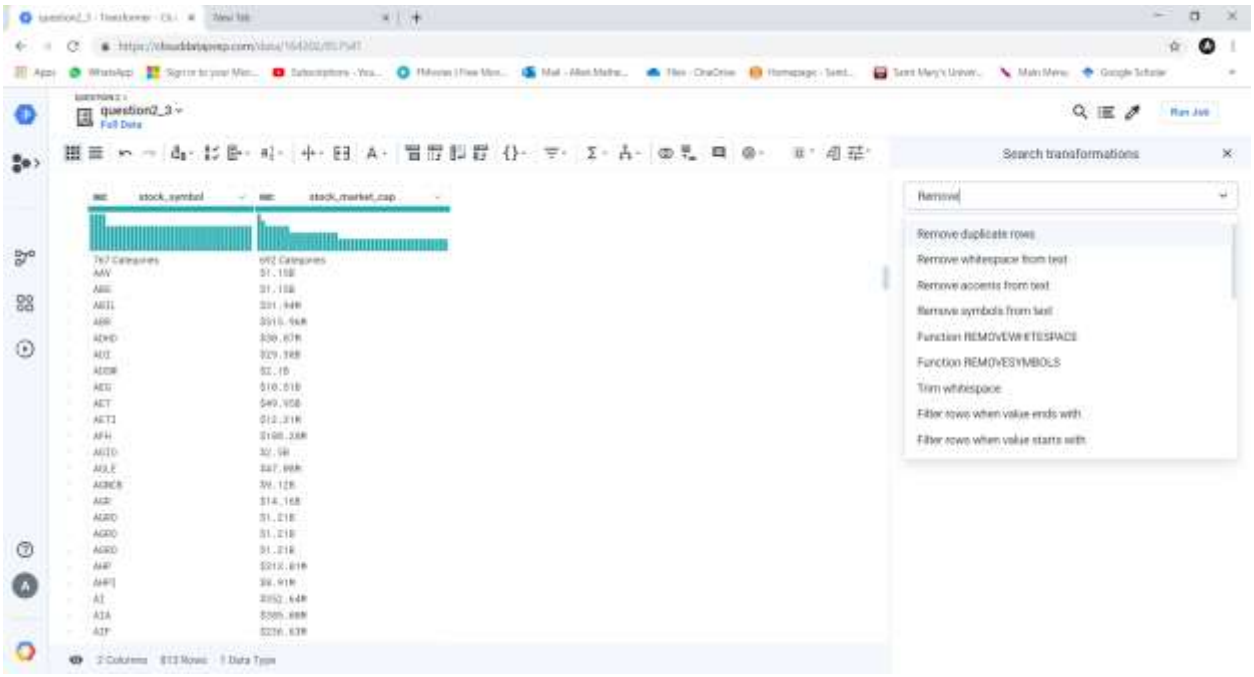

Fig 59: Searching for Remove duplicate option.

The “Remove duplicate rows” option is added to the recipe, it deletes all occurrences of duplicate rows (i.e. 46 rows) from the dataset.

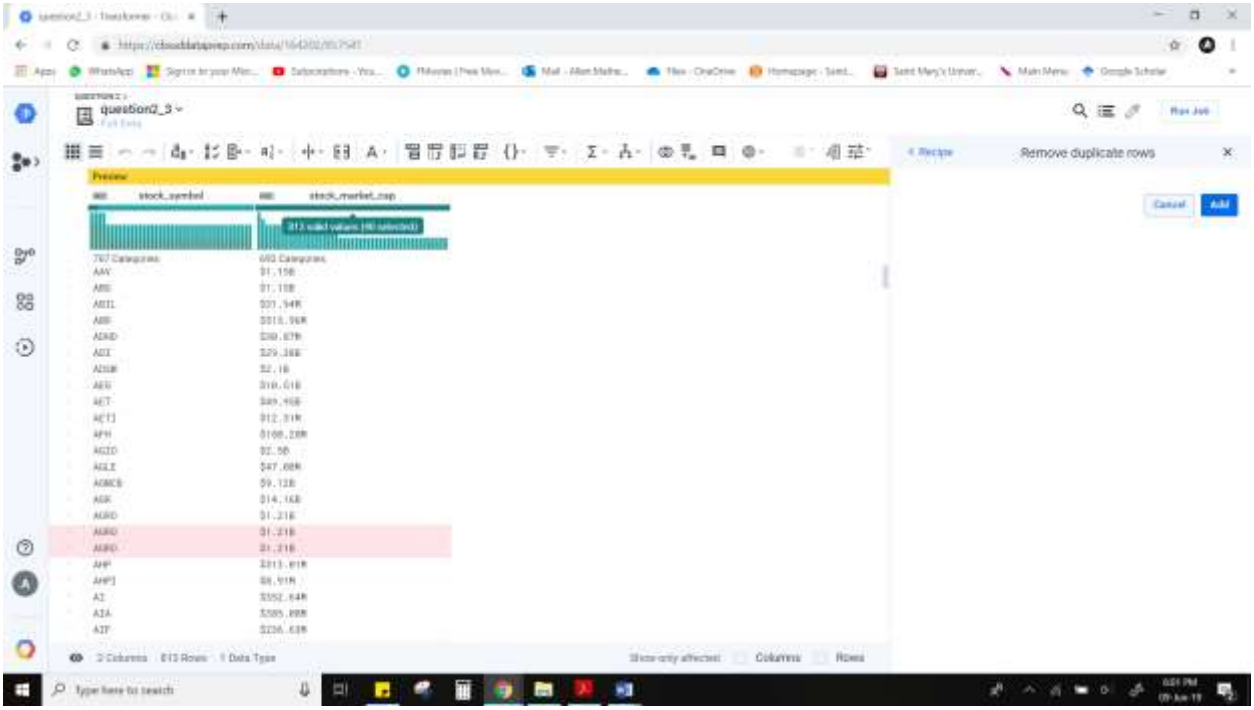

Fig 60: There are 46 rows in the dataset that consist of duplicate records.

All duplicate rows were successfully deleted.

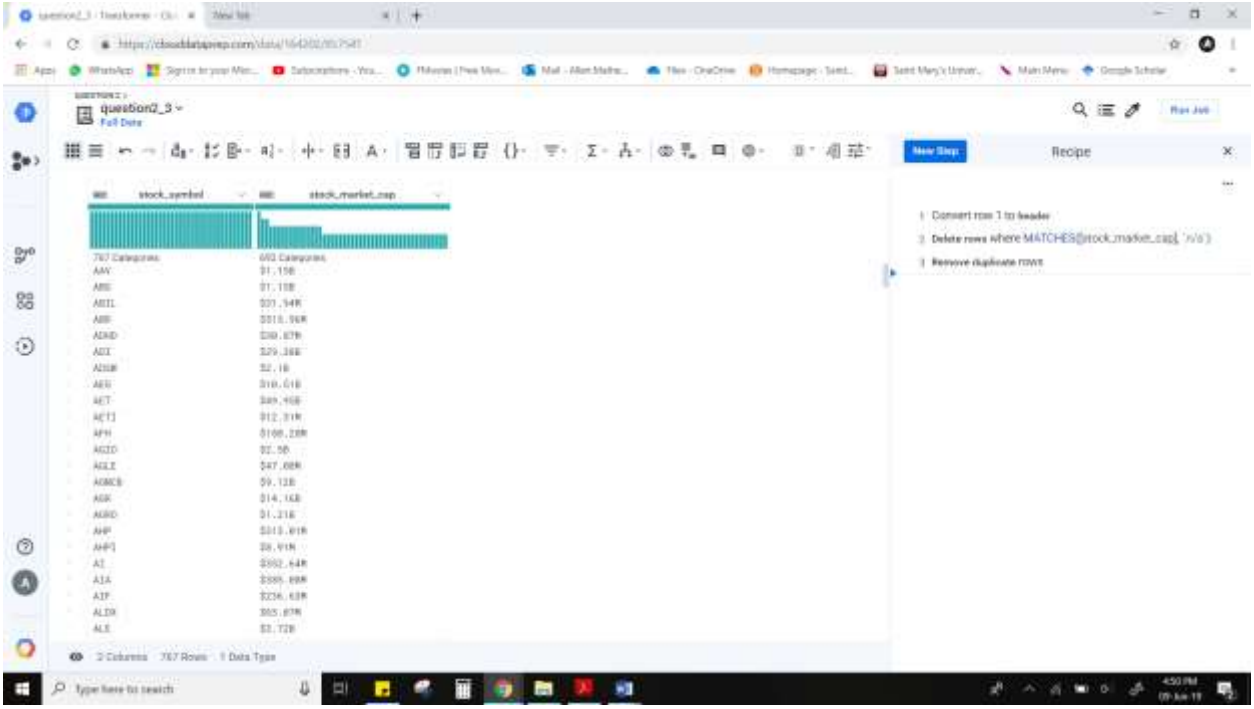

Fig 61: All rows with duplicate records were removed form the dataset

Step 4: Convert records in stock_market_cap from string to integer

The stock_market_cap column consists of string type records, the data in the records are either in million or in billions. To convert the string to integer it is first necessary to understand if the string is either in million or in billions which is indicated by the letter 'M' and 'B' respectively, to do so the "Conditions" option is selected with condition type "if… then. else". It creates a new column called "stock_market_numeric", data for this column is derived from the stock_market_cap column. The following expressions (Google Cloud, 2019) were used to complet this task:

```
If
MATCHES([stock_market_cap], 'B')
Then
ROUND(SUBSTRING(stock_market_cap, 1, LEN(stock_market_cap) - 1) *
1000000000, 0)
Else
ROUND(SUBSTRING(stock_market_cap, 1, LEN(stock_market_cap) - 1) * 1000000,
0)
New Column
stock_market_numeric
```

Hence, the MATCHES operation was used to check ***if*** the letter 'B' appears in the records of the stock_market_cap column, ***then*** the first and the last characters of the string in the record (i.e. '$' and 'B') are removed using the SUBSTRING operation moreover the number obtained from the string is multiplied by a billion then result is rounded off using the ROUND operation and added as a record to the stock_market_numeric column. ***Else*** the first and the last characters of the string in the record (i.e. '$' and 'M') are removed and the number is multiplied by a million then result is rounded off and added as a record to the stock_market_numeric column.

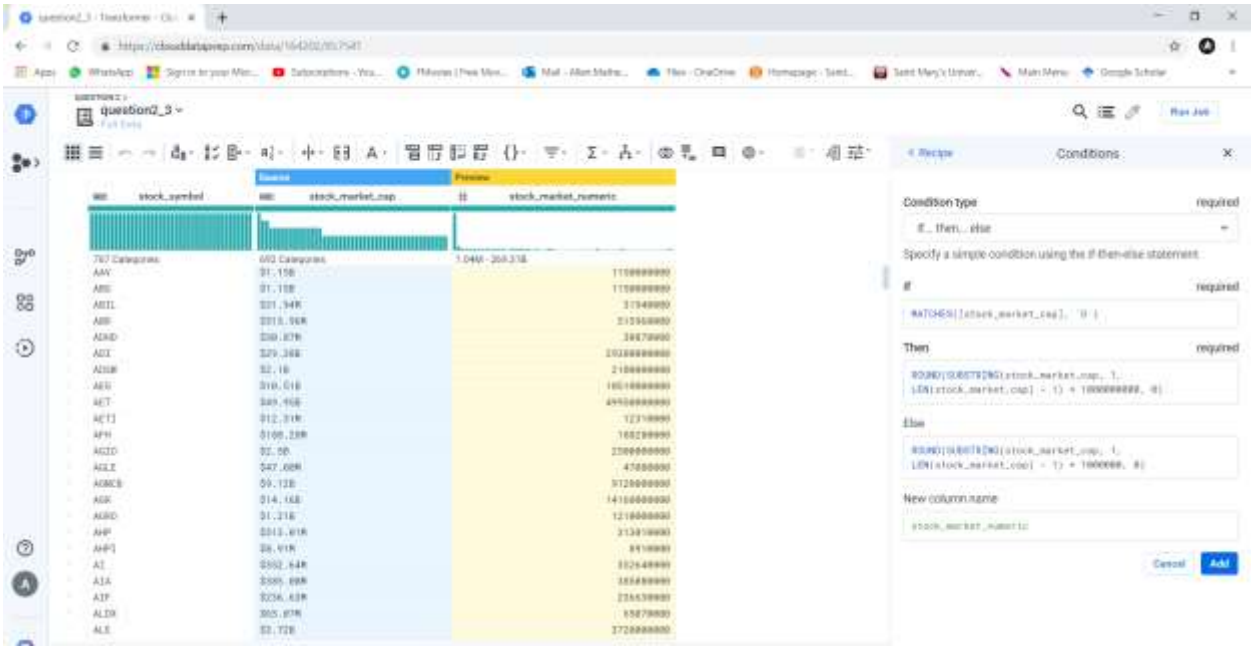

Fig 62: Converting the stock market cap string value into it numeric representation

The stock_market_numeric column was successfully created, and the string was converted to its numeric equivalent.

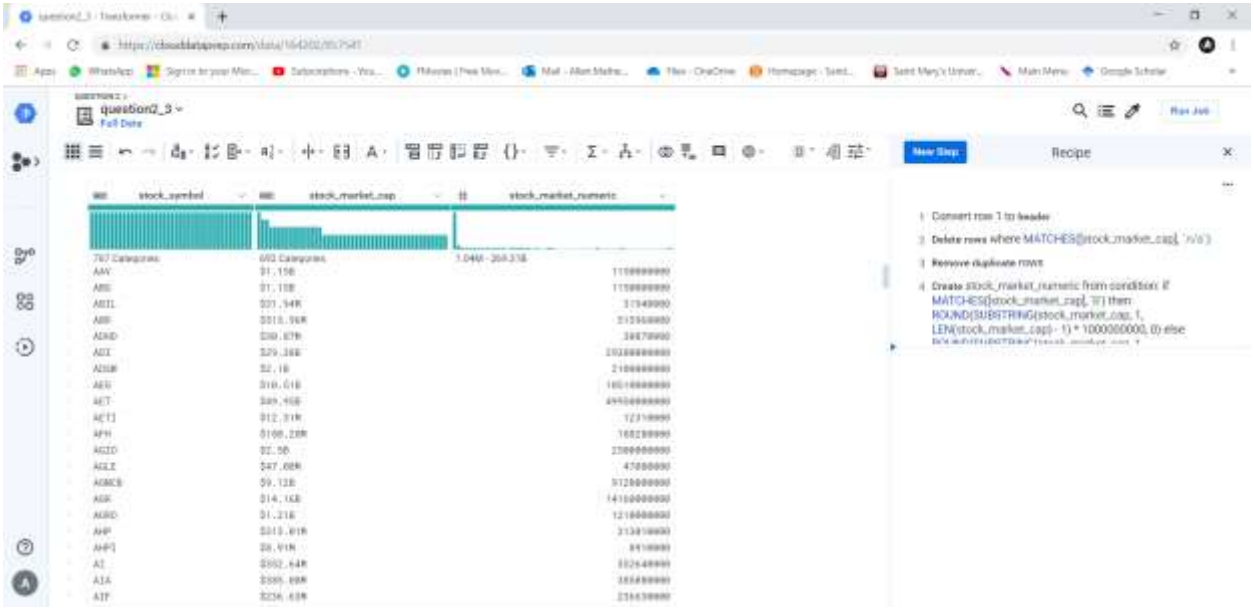

Fig 63: stock_market_numeric column was successfully created.

Step 5: Delete the Rows where the records in stock_market_numeric are lesser than 30 Million

The searching tool located at the top right corner of the browser can be used to select the “Filter rows using custom formula” option.

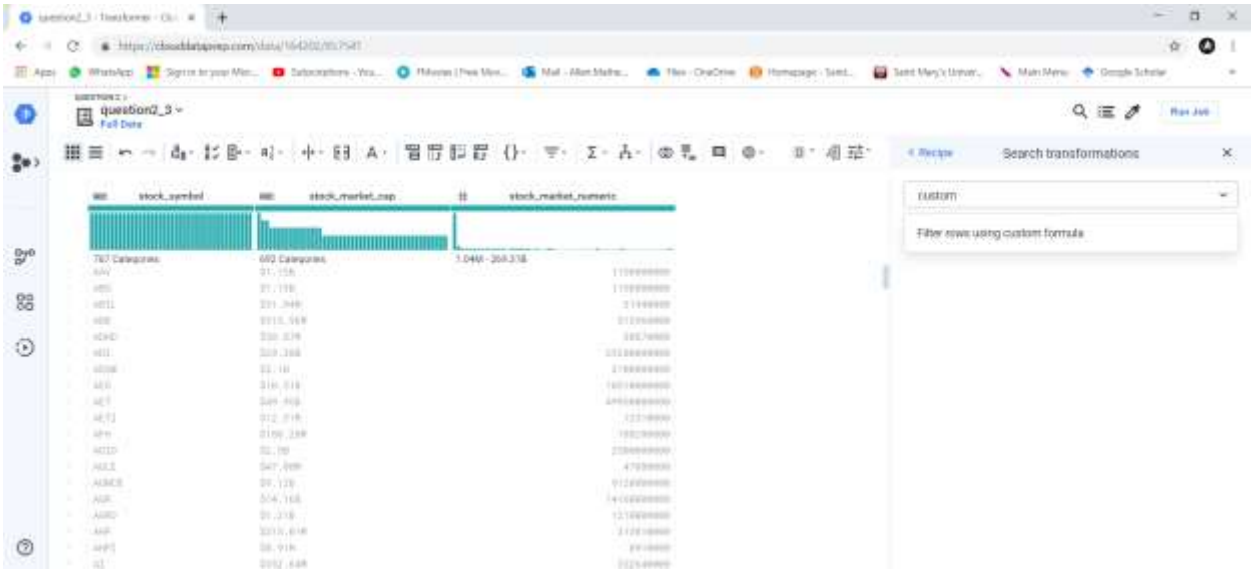

Fig 64: Searching for Filter rows using custom formula option

The “Filter rows using custom formula” option is added to the recipe, it deletes all rows (i.e. 53 rows) where the records in the stock_market_numeric is lesser than 30 million. The following is the expression (Google Cloud, 2019) used to filter the rows in the dataset:

Condition

stock_market_numeric < 30000000

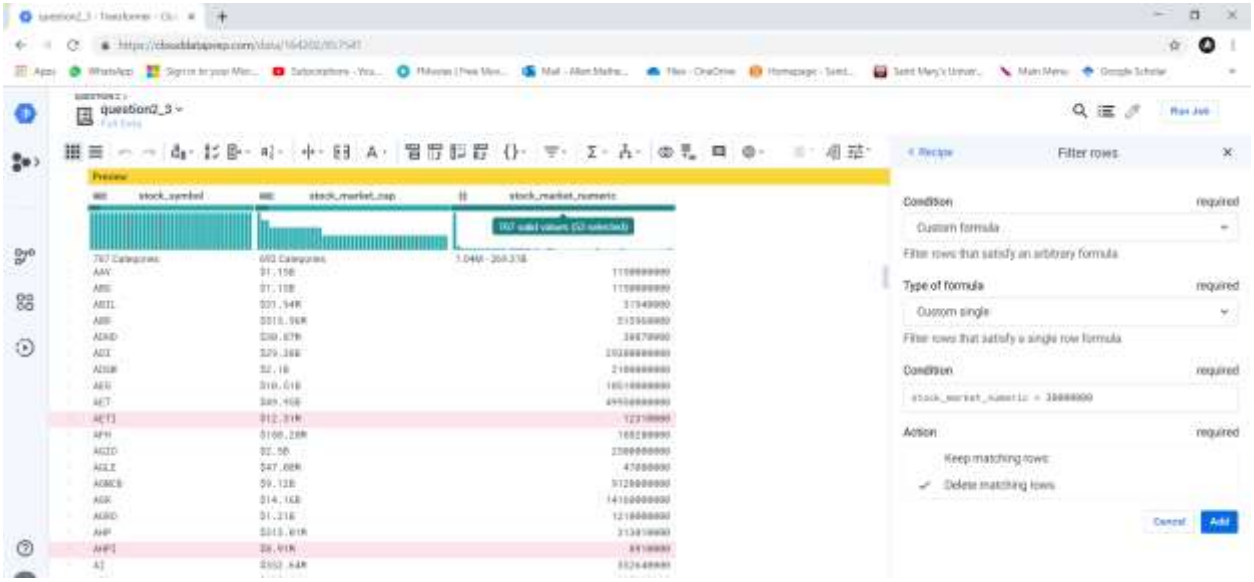

Fig 65: There are 53 rows that were deleted since the stock market numeric value was lesser than 30 million

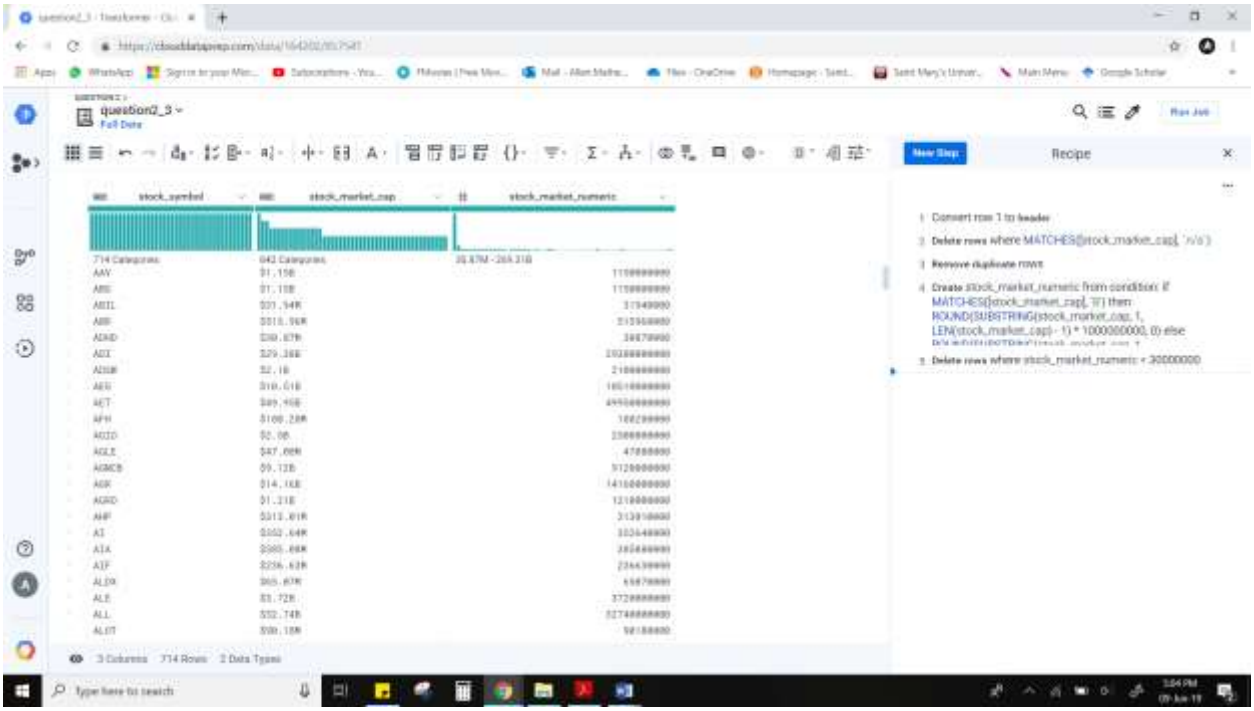

Fig 66: The 53 rows were successfully deleted where the records in the stock_market_numeric column were lesser than 30 Million

Step 6: The data in the stock_market_numeric column need to be represented in terms of millions to maintain consistency for all the records:

The searching tool located at the top right corner of the browser can be used to select the “New Formula” option.

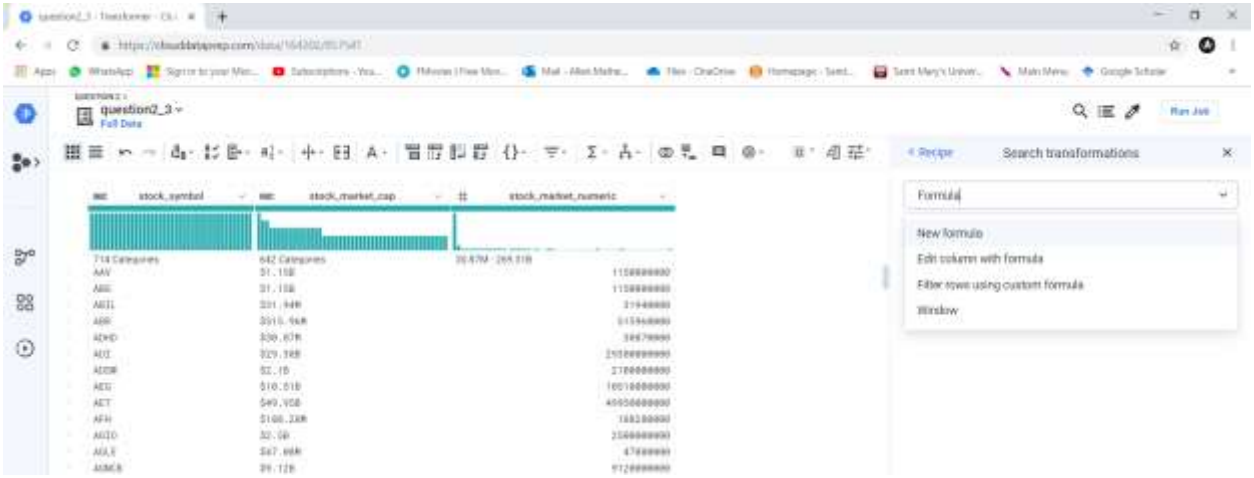

Fig 67: Searching for New formula option

The “New Formula” option is used to create a new column called “stock_market_cap_final”, data for this column is derived from the stock_market_numeric column. The stock_market_numeric column consists of integer type records, the data in the records are either in million or in billions. So, to maintain consistence all the records needs to be represented in millions. To do so the value in the stock_market_numeric column is divided by 1 million. Then the MERGE operation is used to concatenate the characters ‘$’ and ‘M’ at the start and end of the new value obtained after divided by 1 million. The following expressions (Google Cloud, 2019) were used to complete this task:

Formula

MERGE(['$',stock_market_numeric / 1000000,'M'])

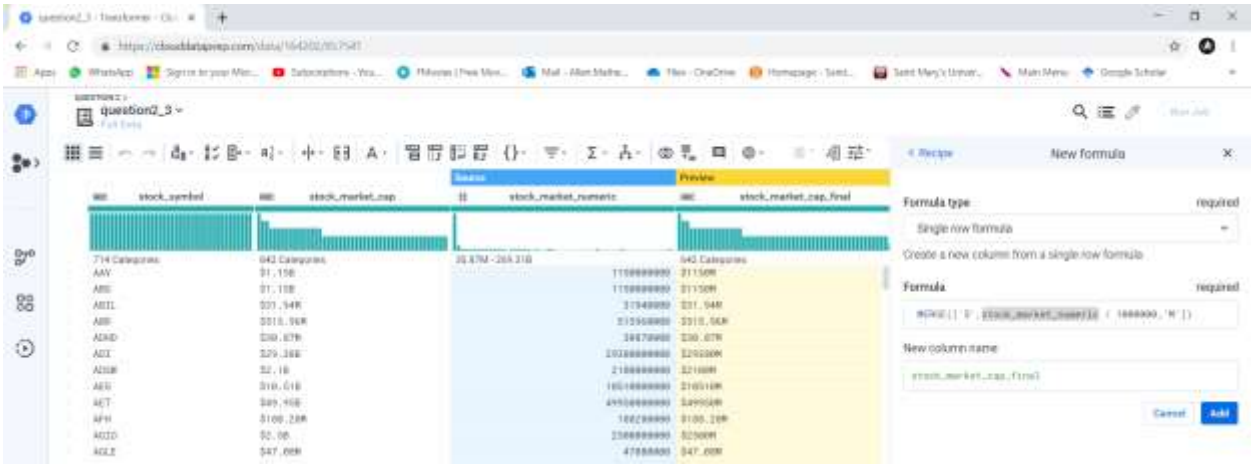

Fig 68:stock market cap final

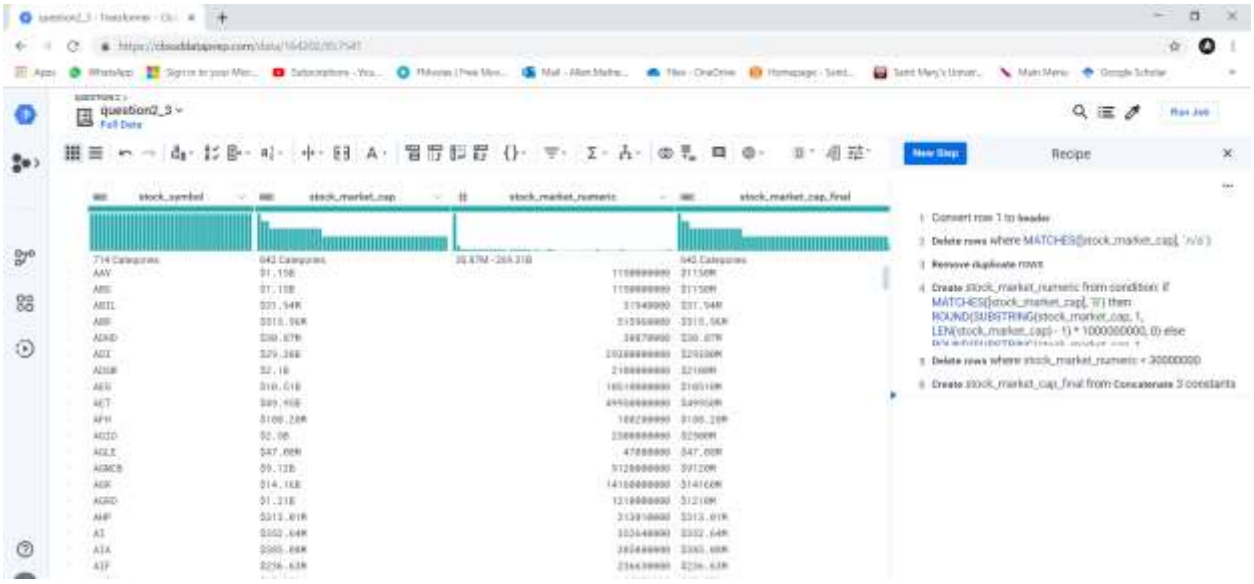

Fig 69: The new column stock_market_cap_final was created successfully

4) Data Preparation on Question2_2 Dataset

The question2_2 dataset consists of 2 columns:

- **stock_symbol**:
  - A text string used to uniquely identify the stock.
- **stock_sector**:
  - A text string that describes the sector in which the stock belongs in.

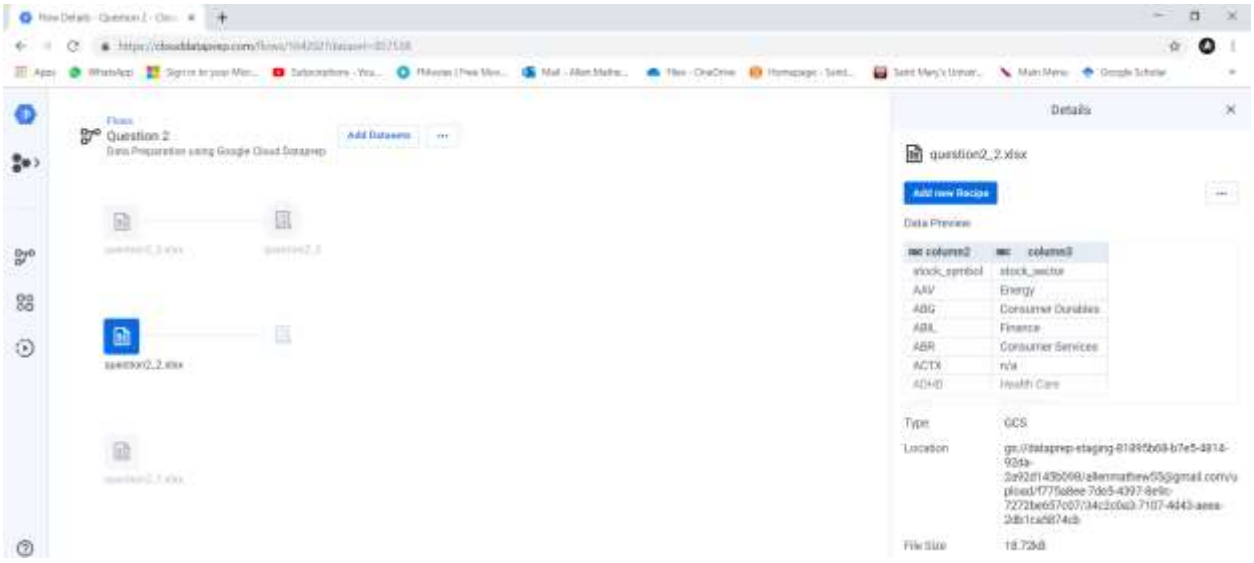

Fig 70: Reviewing Column Details for dataset question2_2

To clean the dataset a new recipe is added to the question2_2 dataset.

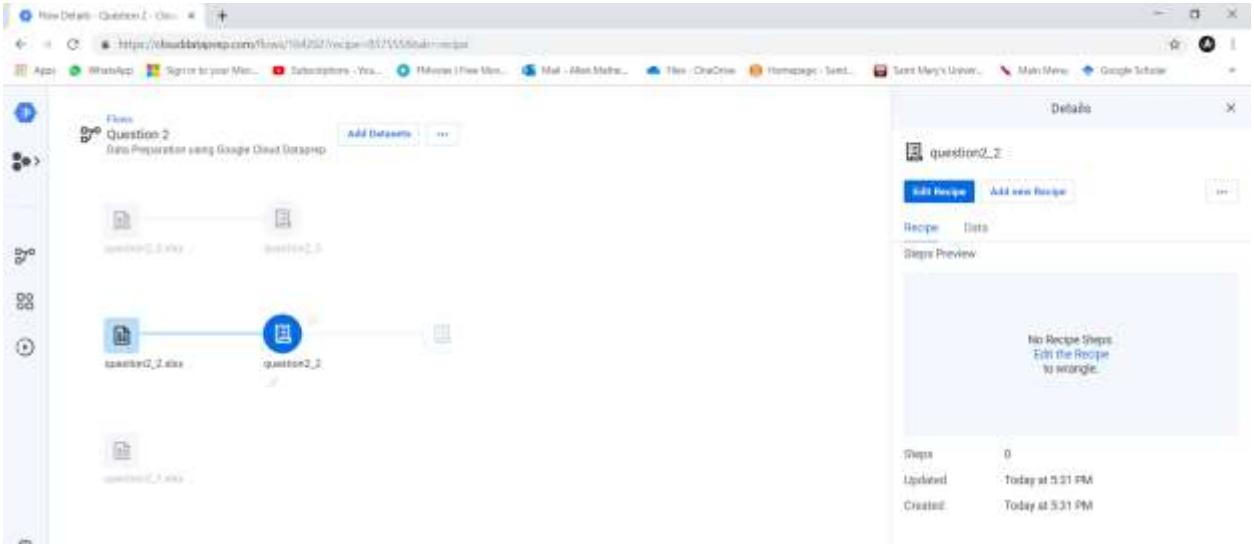

Fig 71: Adding new Recipe

On reviewing the initial dataset (seen in the following figures), few observations can be drawn:

1. The columns in the dataset have been given default headers while the first row in the dataset is not recognized as the appropriate headers of the respective columns.
2. The stock_sector column has multiple records with a value of not applicable(n/a).
3. The dataset has multiple duplicate records.
4. The dataset has no blank records.

Fig 72: Reviewing stock symbol and stock sector via Grid Format

Fig 73: Reviewing stock symbol and stock sector via Column Format

4.1) Requirements for question2_2 Dataset:

Various steps need to be performed on the question2_2 dataset:

- The first row need to be made the header.
- The data needs to be cleaned (i.e. removing 'n/a' records as well as duplicate records)

Step 1: Assign the 1st row as header:

On selecting the first row in the dataset, Google Data Prep automatically provides several suggestions. The "Header" option that was suggested is added to the recipe, it converts the first row to the header.

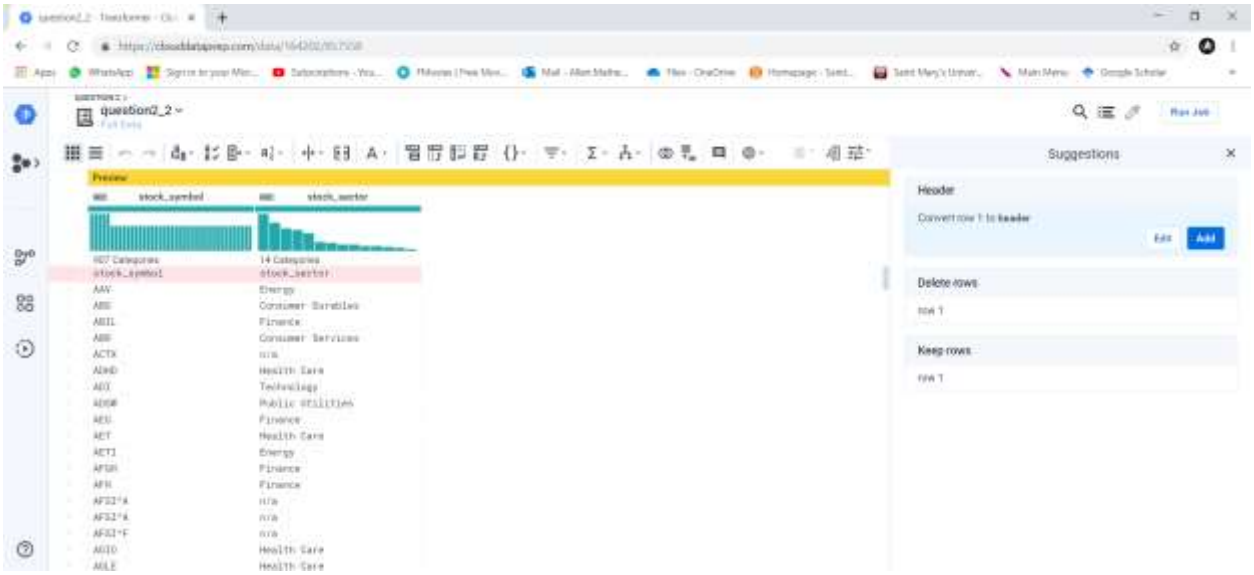

Fig 74: Assigning the 1st row of the dataset as header

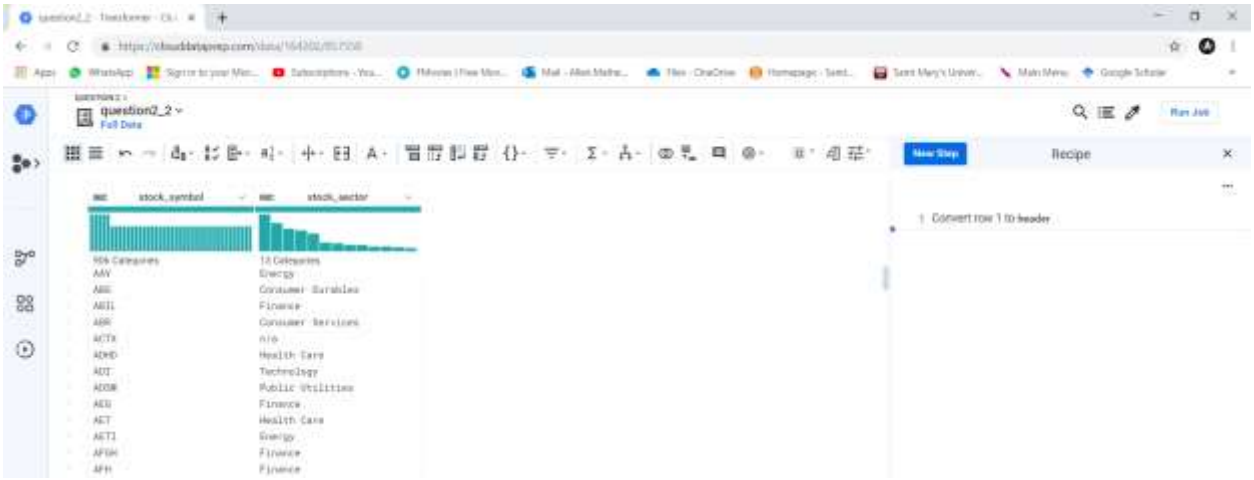

Fig 75: The first row was successfully converted as the headers of the respective columns

Step 2: Remove rows with value as n/a:

On selecting/highlighting a single “n/a” records in the stock_sector column, Google Data Prep automatically provides several suggestions.

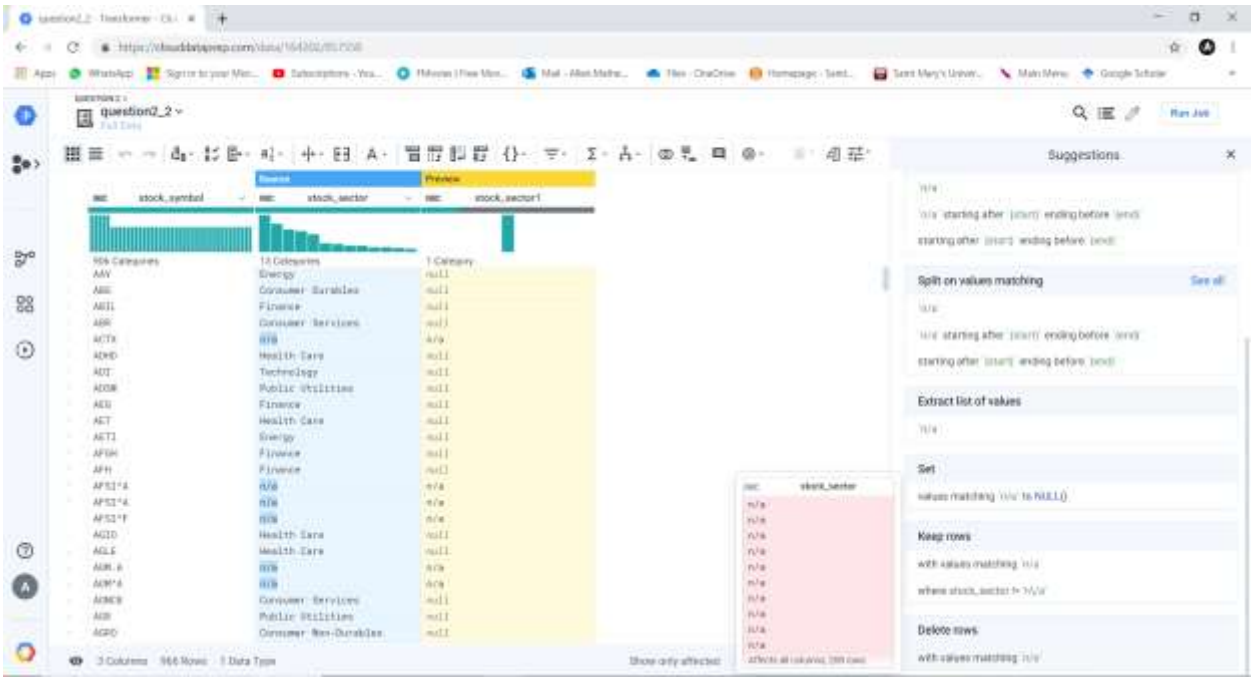

Fig 76: Removing Rows for every occurrence of NA’s in stock_sector column

The “Delete rows” option that was suggested is added to the recipe, it deletes rows (i.e. 208 rows) from the dataset once an occurrence of “n/a” is observed in the stock_sector column.

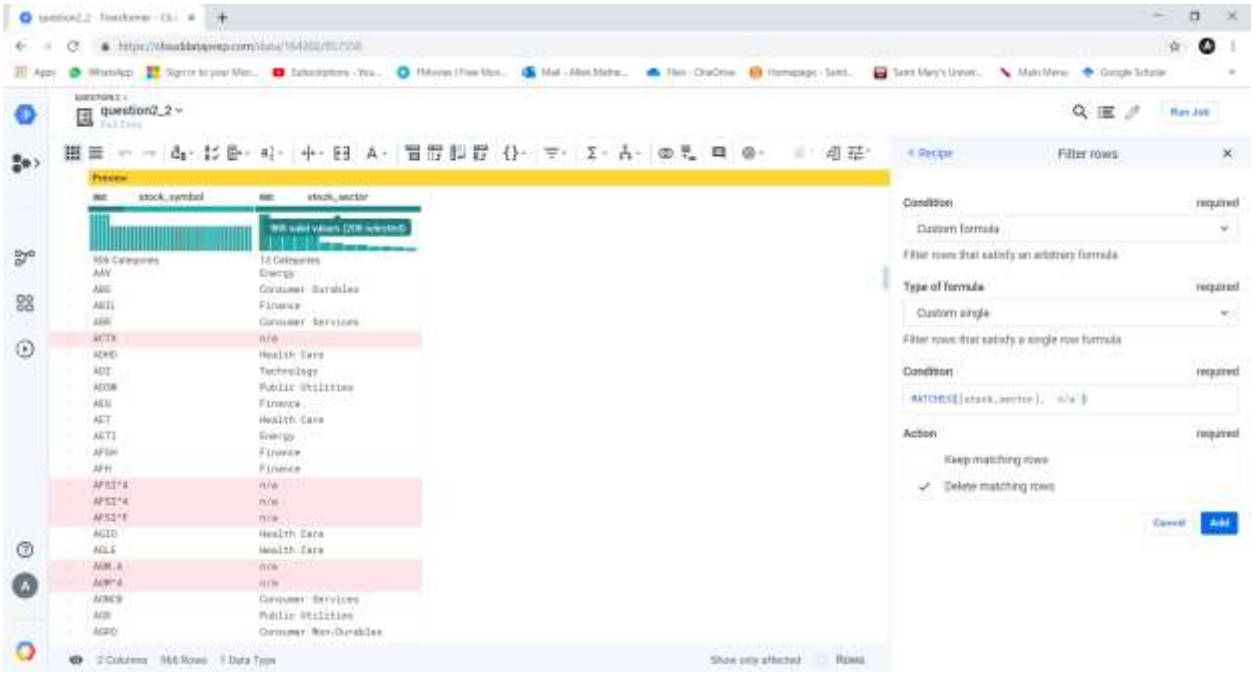

Fig 77: There are 208 rows where the occurrence of "n/a" is observed in the stock_sector column

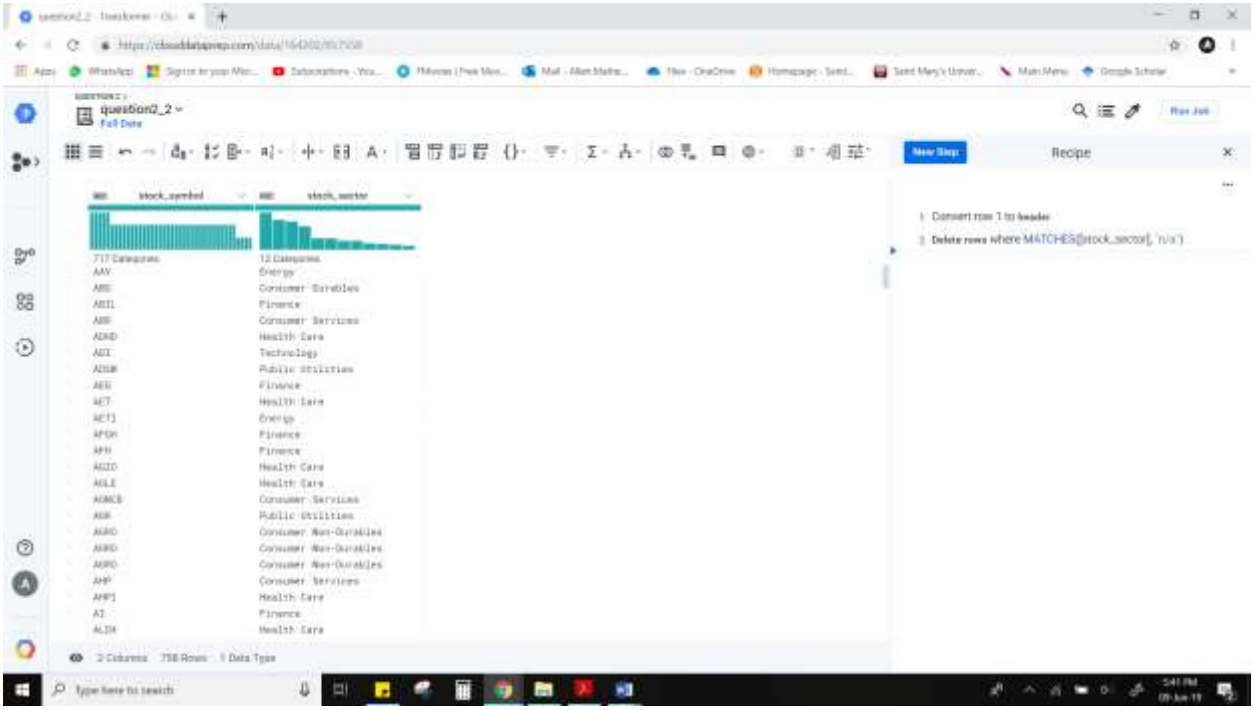

Fig 78: All rows with a record of "n/a" in the stock_sector column have been successfully deleted

Step 3: Remove rows with Duplicate values:

The searching tool located at the top right corner of the browser can be used to select the “Remove duplicate rows” option.

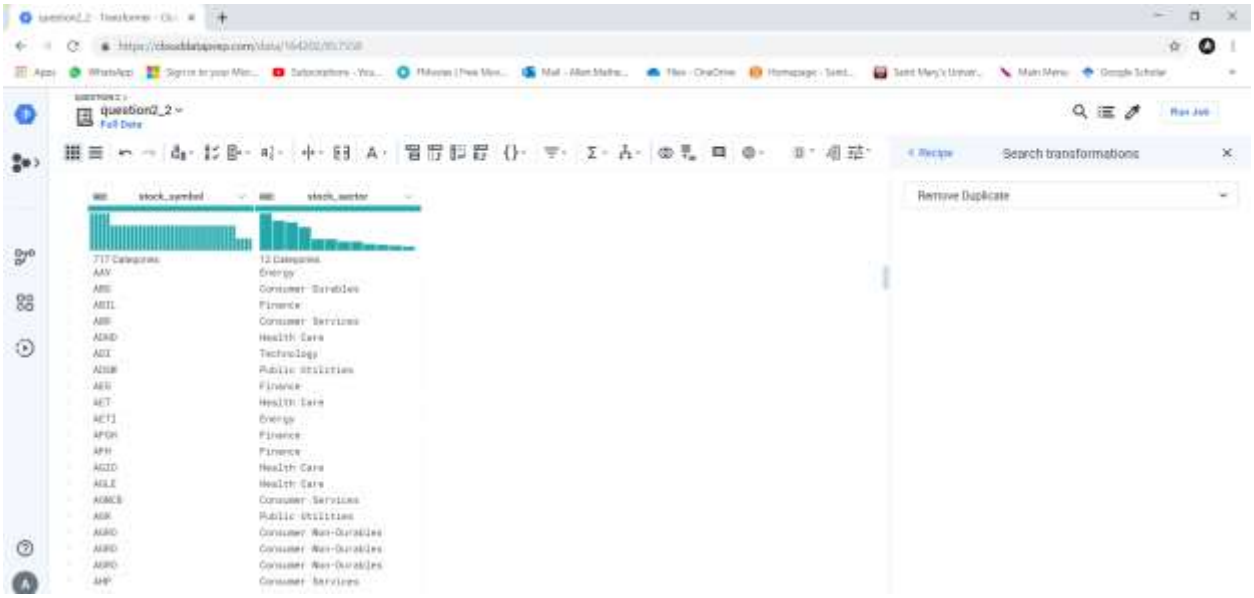

Fig 79: Searching for Remove Duplicates option

The “Remove duplicate rows” option is added to the recipe, it deletes all occurrences of duplicate rows (i.e. 41 rows) from the dataset.

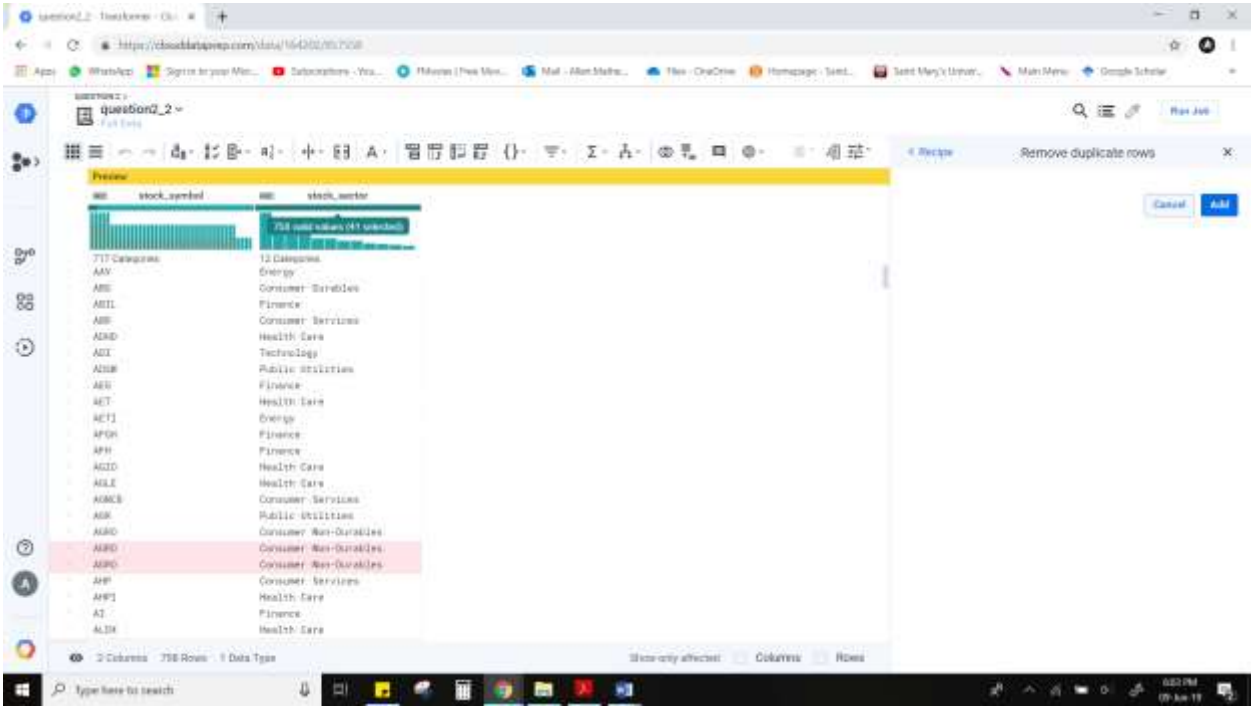

Fig 80: There are 41 rows in the dataset that consist of duplicate records.

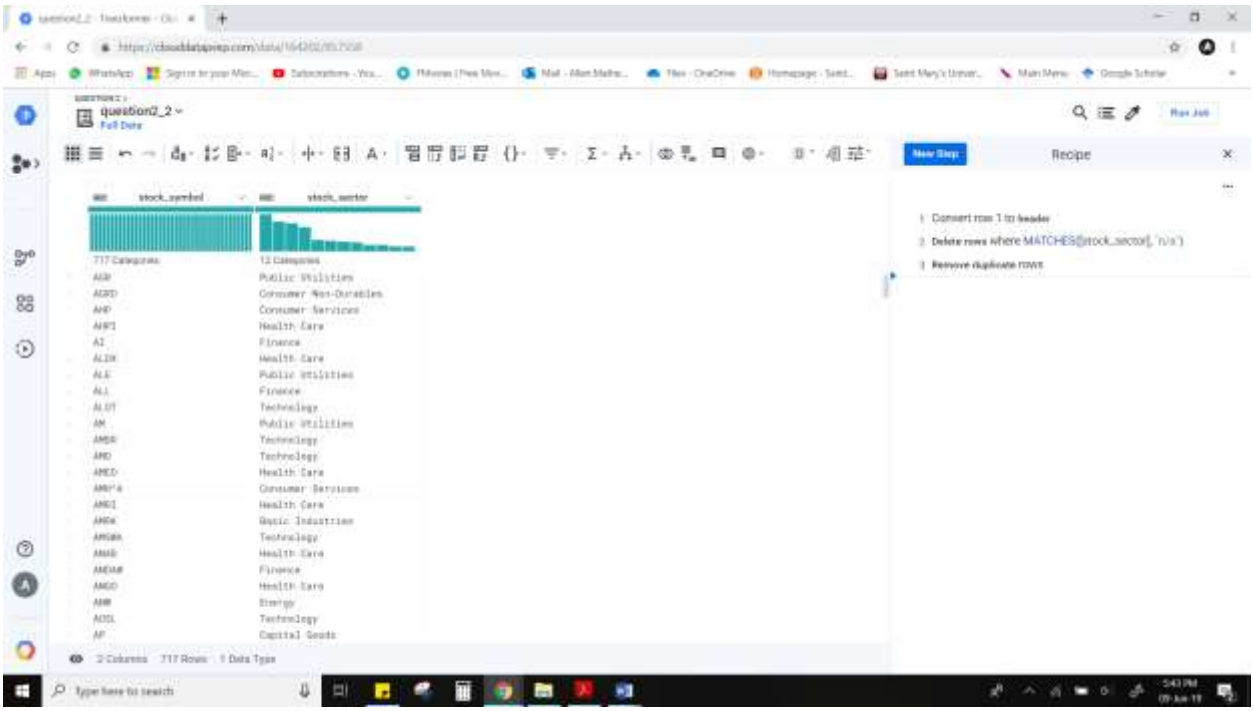

Fig 81: All duplicate rows were successfully deleted.

5) Data Preparation on Question2_1 Dataset

The question2_1 dataset consists of 2 columns:

- stock_symbol:
  - A text string used to uniquely identify the stock.
- stock_market:
  - A text string that describes which market the stock belongs in (i.e. NYSE or NASDAQ)

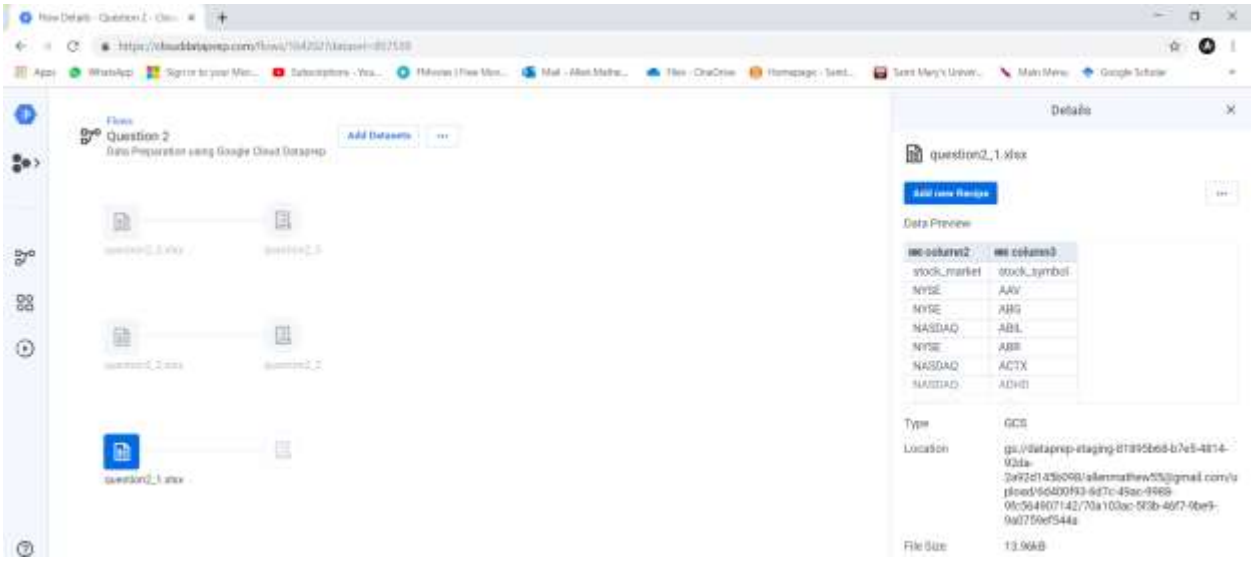

Fig 82: Reviewing Column Details for dataset question2_2

To clean the dataset a new recipe is added to the question2_3 dataset.

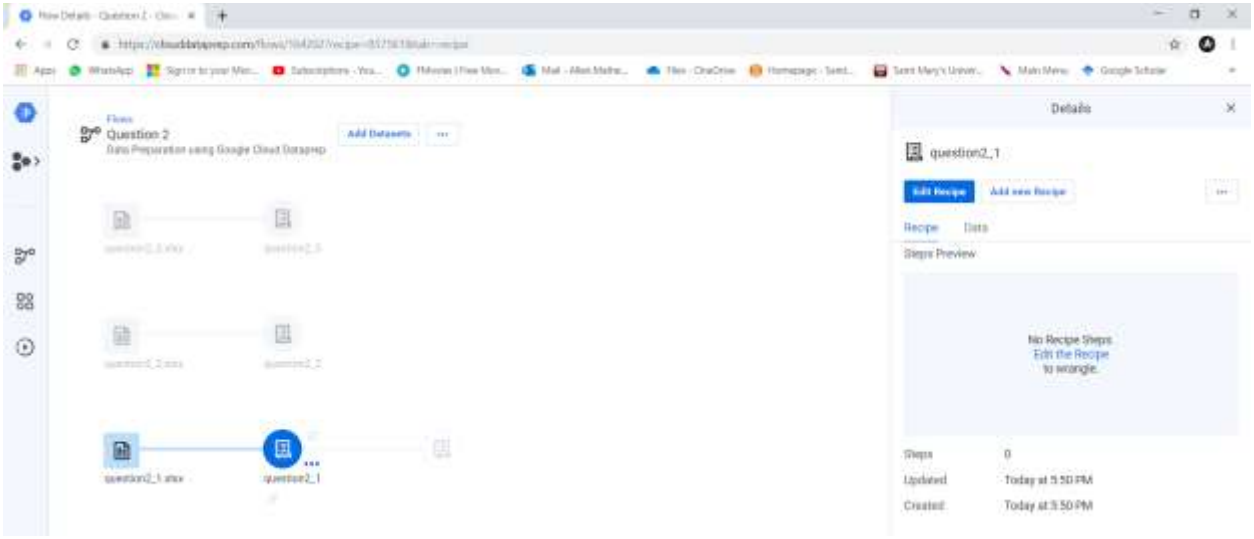

Fig 83: Adding new Recipe

On reviewing the initial dataset (seen in the following figures), few observations can be drawn:

1. The columns in the dataset have been given default headers while the first row in the dataset is not recognized as the appropriate headers of the respective columns.
2. There is no records with a value of not applicable(n/a).
3. The dataset has multiple duplicate records.
4. The dataset has no blank records.

Fig 84: Reviewing stock symbol and stock market via Grid Format

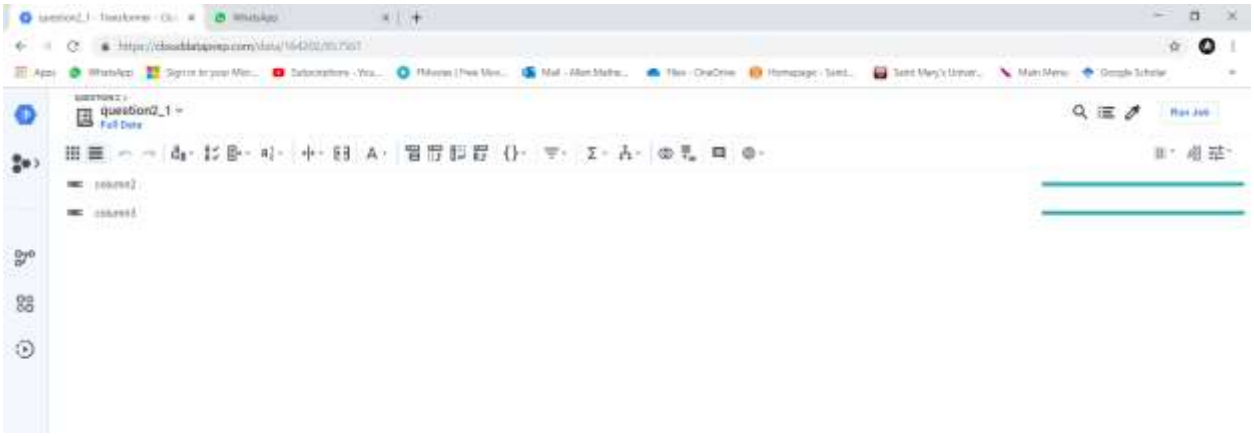

Fig 85: Reviewing stock symbol and stock market via Column Format

5.1) Requirements for question2_1 Dataset

Various steps need to be performed on the question2_1 dataset:

- The first row need to be made the header.
- The data needs to be cleaned (i.e. removing duplicate records)

Step 1: Assign the 1st row as header:

On selecting the first row in the dataset, Google Data Prep automatically provides several suggestions. The "Header" option that was suggested is added to the recipe, it converts the first row to the header.

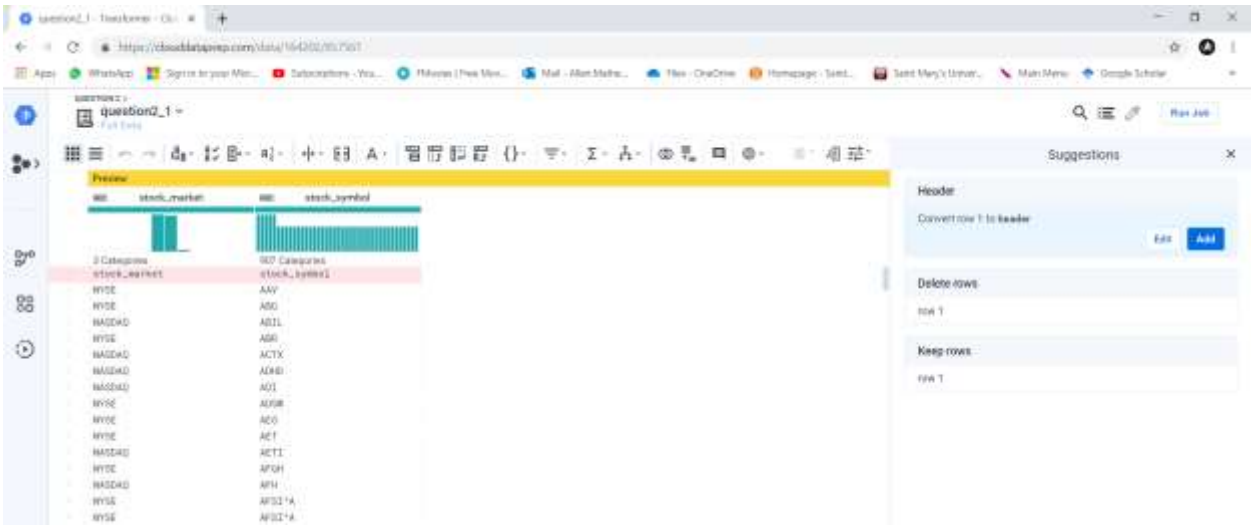

Fig 86: Assigning the 1st row of the dataset as header

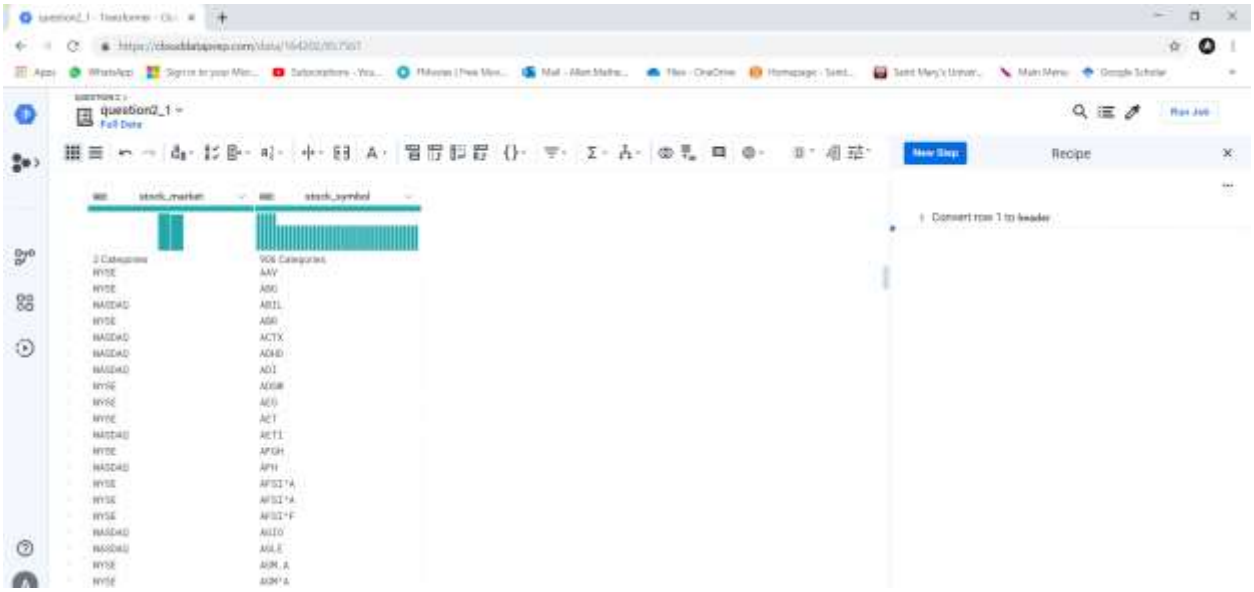

Fig 87: The first row was successfully converted as the headers of the respective columns.

Step 2: Remove rows with Duplicate values:

The searching tool located at the top right corner of the browser can be used to select the “Remove duplicate rows” option.

Fig 88: Searching for Remove Duplicates option

The “Remove duplicate rows” option is added to the recipe, it deletes all occurrences of duplicate rows (i.e. 60 rows) from the dataset.

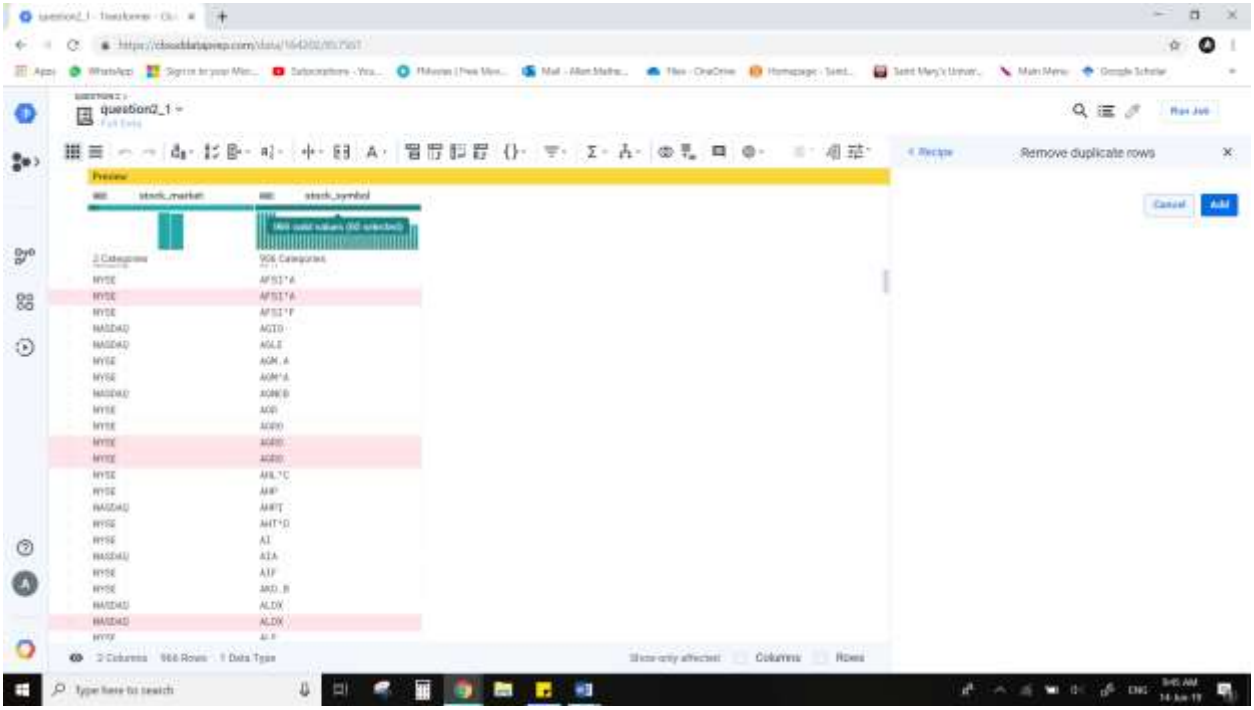

Fig 89: There are 60 rows in the dataset that consist of duplicate records.

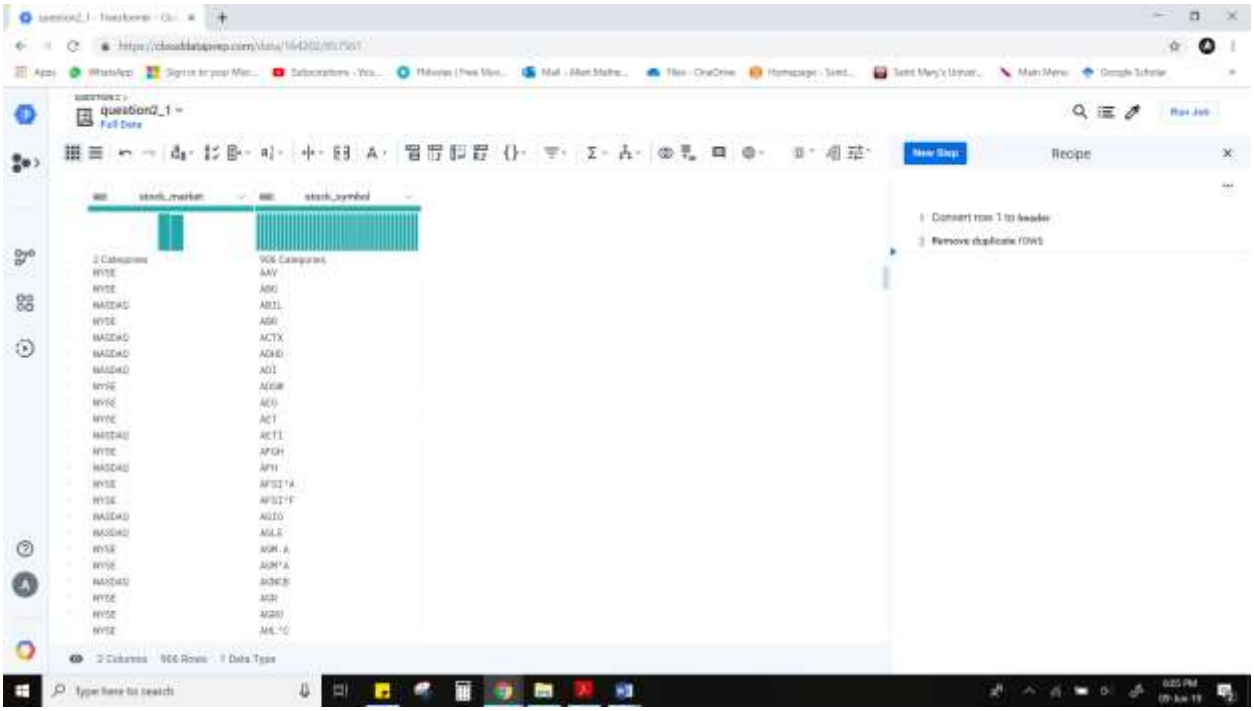

Fig 90: All duplicate rows were successfully deleted

6) Joining the Data Sets

To join all the datasets together a new recipe is added to the question2_2-2 dataset.

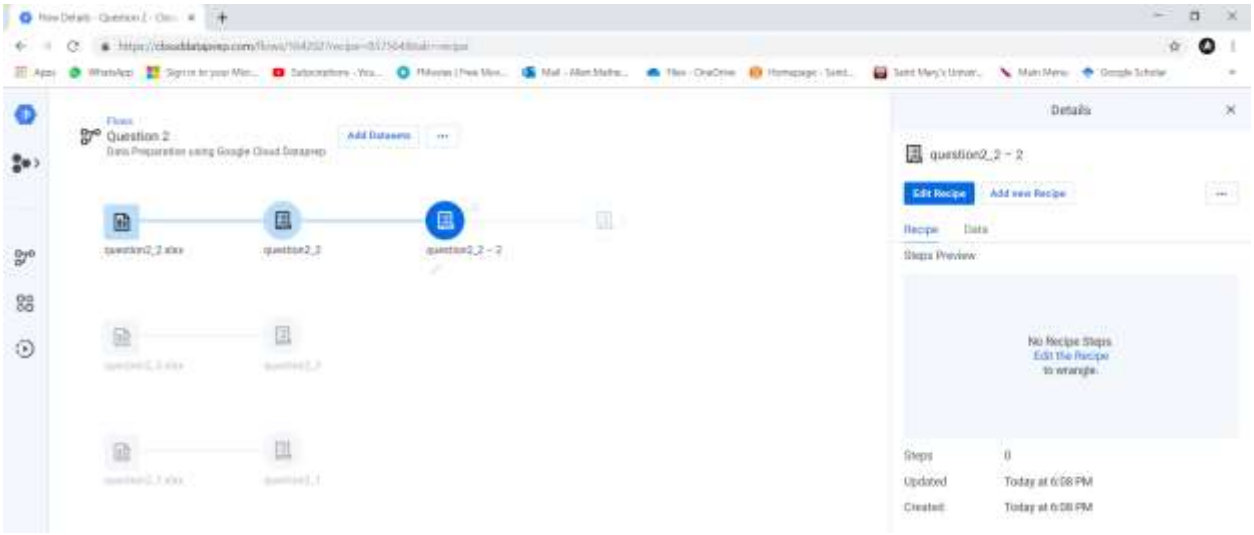

Fig 91: Adding new Recipe

The searching tool located at the top right corner of the browser can be used to select the “Join datasets” option.

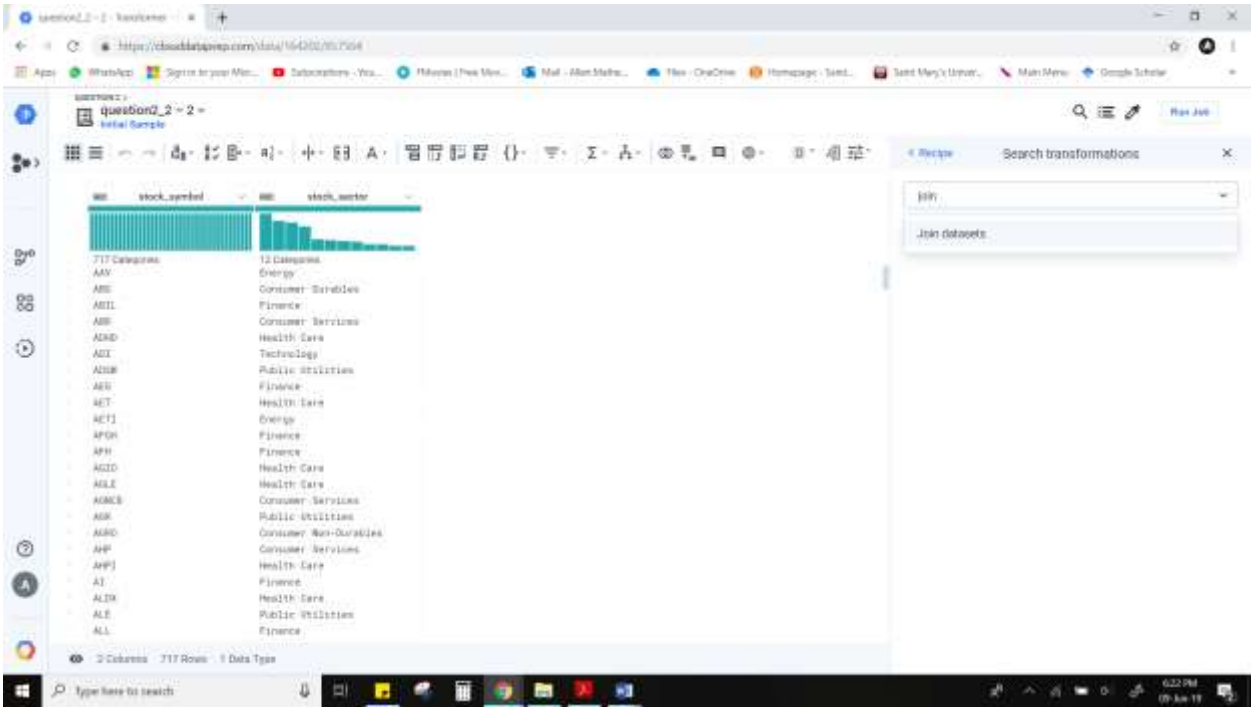

Fig 92: Searching for Join Datasets option

The question2_1 recipe is selected to join with the question2_2-2 recipe

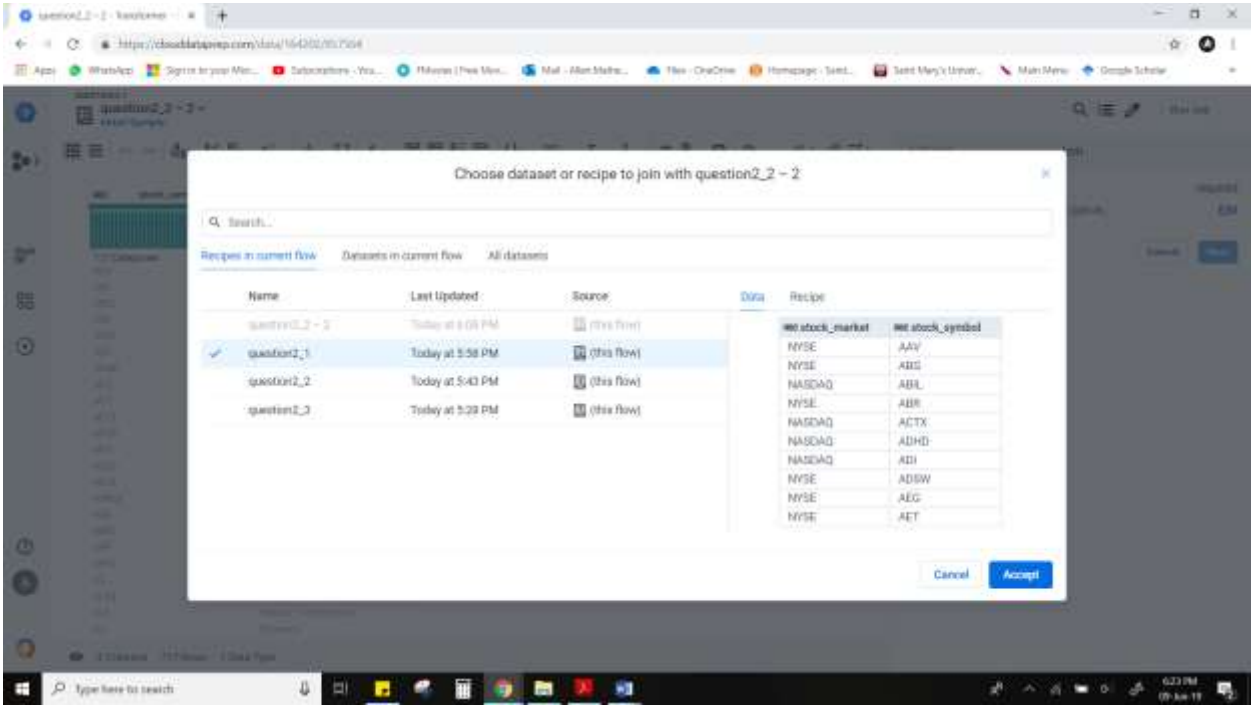

Fig 93: selecting the question2_1 recipe

To JOIN the datasets, a common column needs to be present between the 2 datasets. Google Data Prep automatically provides a suggestion for the Join keys (i.e. stock_symbol). From the summary of the results it is evident that between the 2 datasets there are 717 records with similar keys.

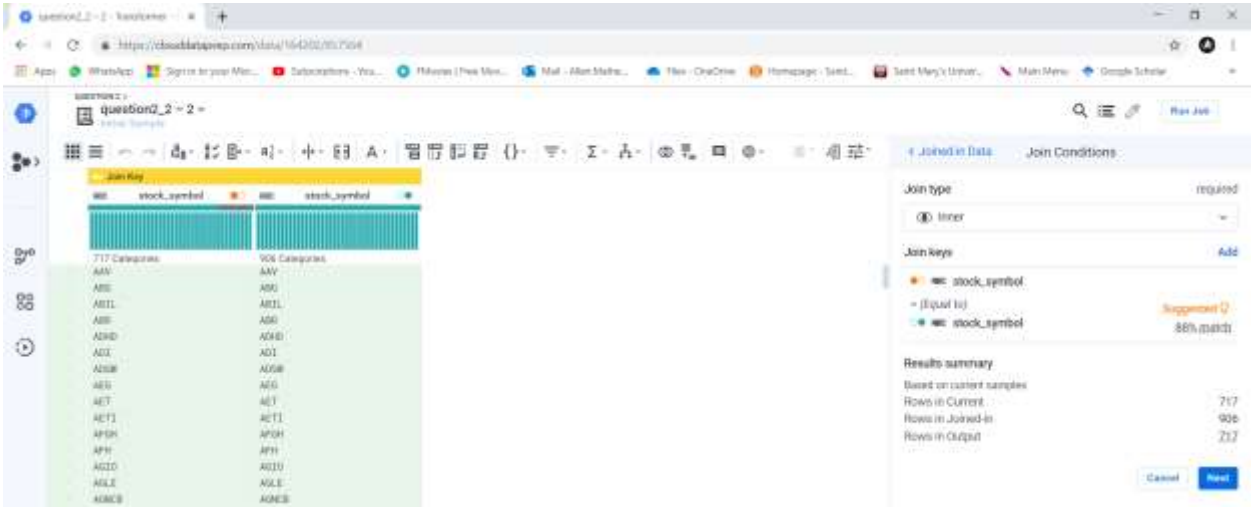

Fig 94: Google Data Prep automatically provides a suggestion for the Join keys

The necessary output columns are selected between the 2 datasets before the data sets are joined.

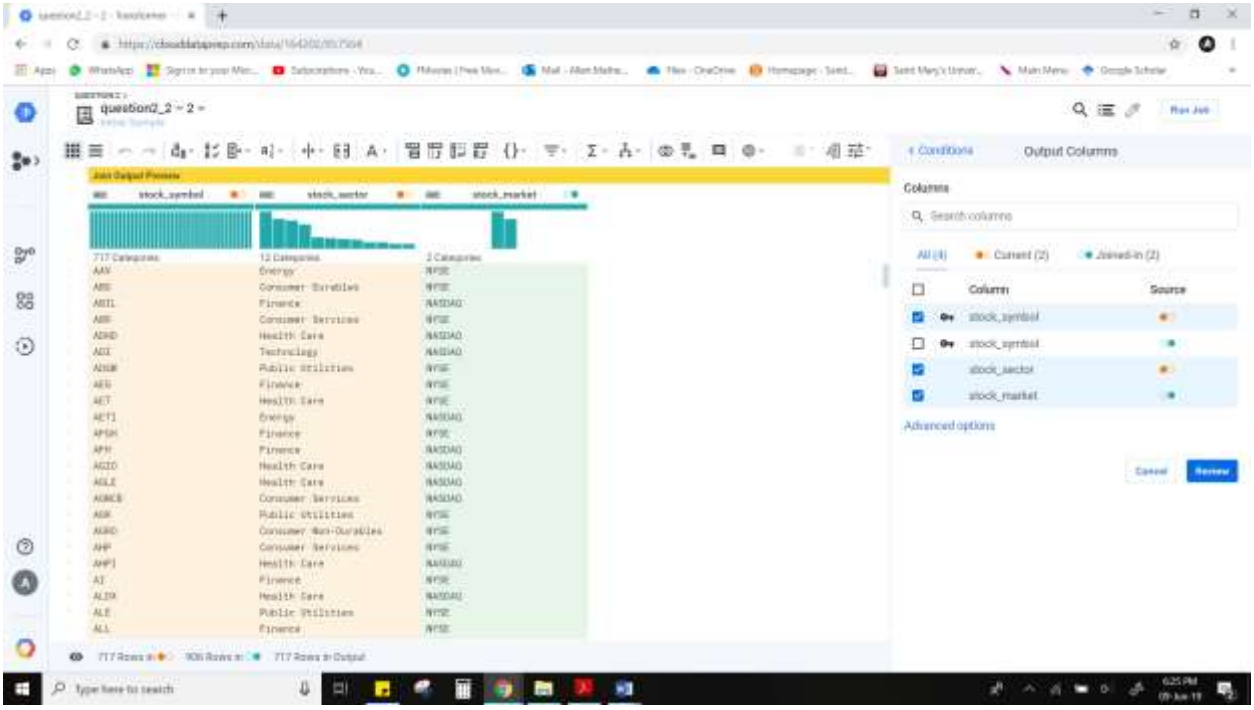

Fig 95: Select datasets Output columns

Before adding the changes to the recipe, the JOIN operation is reviewed.

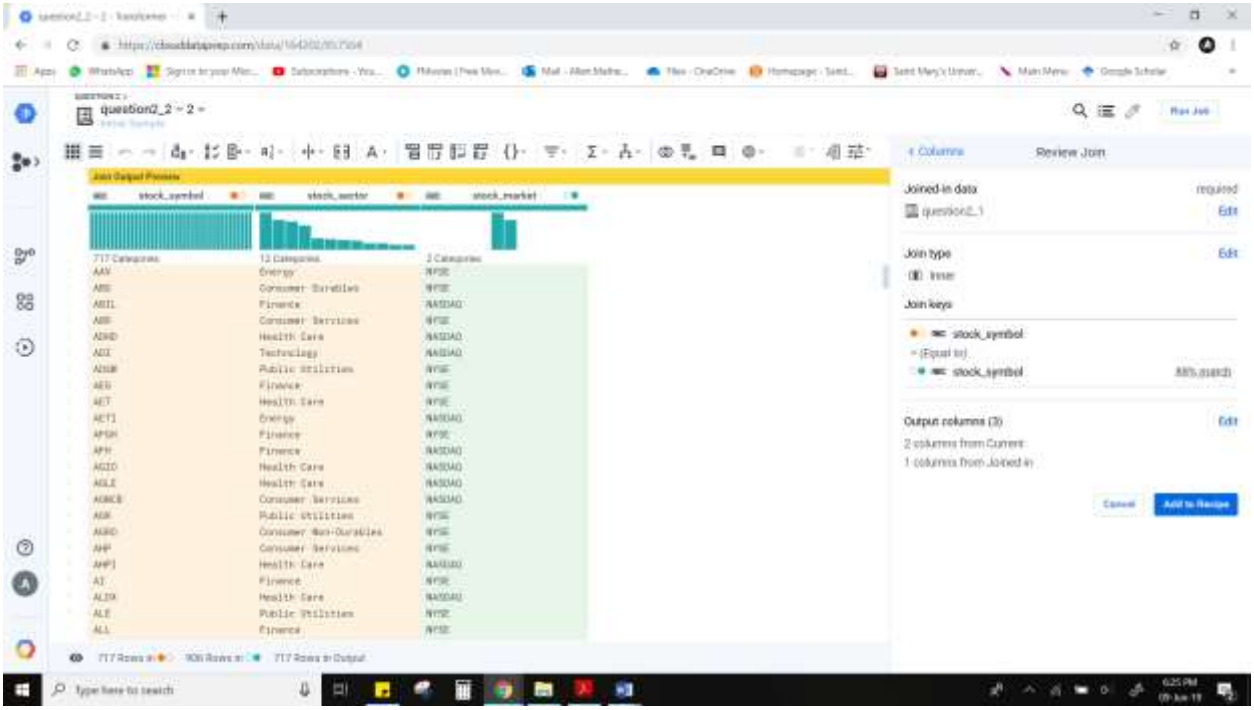

Fig 96:Revieing datasets Output columns

The JOIN operation was successfully able to join both the recipes (i.e. question2_2-2 and question2_1).

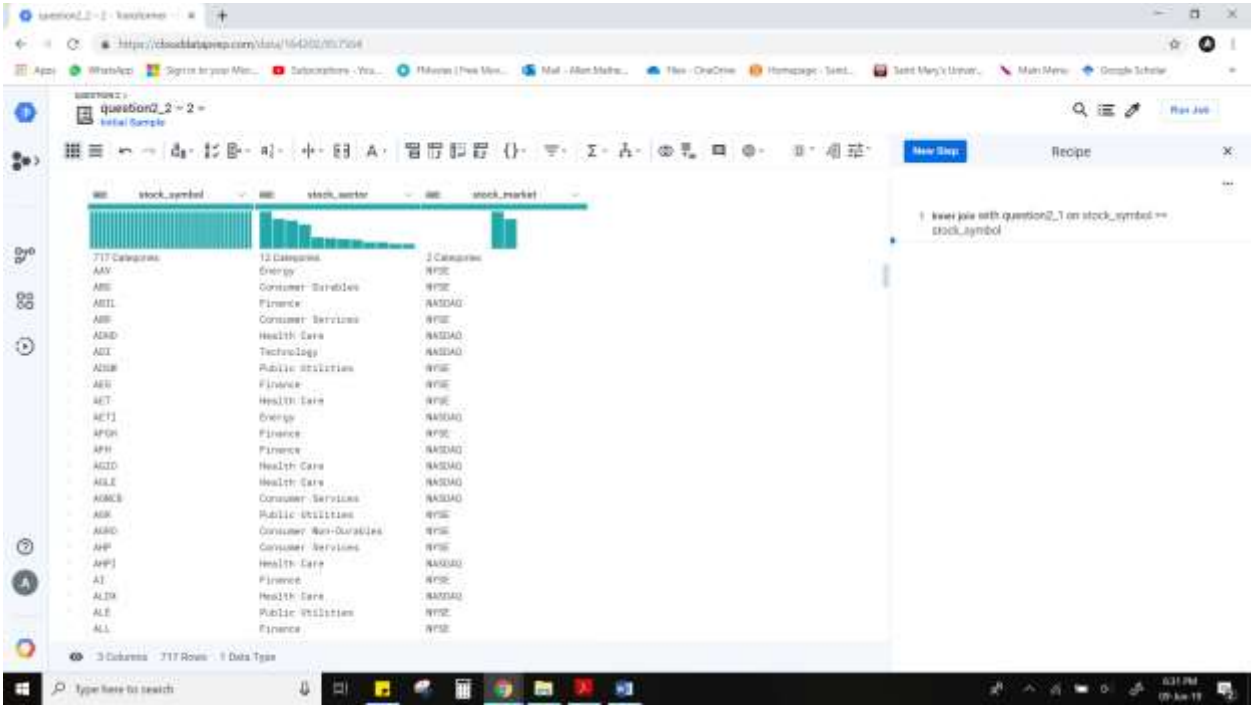

Fig 97: Inner join data sets question2_2-2 and question2_1 on stock_symbol

The Flow Diagram showing that the JOIN operation was successfully able to join both the recipes (i.e. question2_2-2 and question2_1).

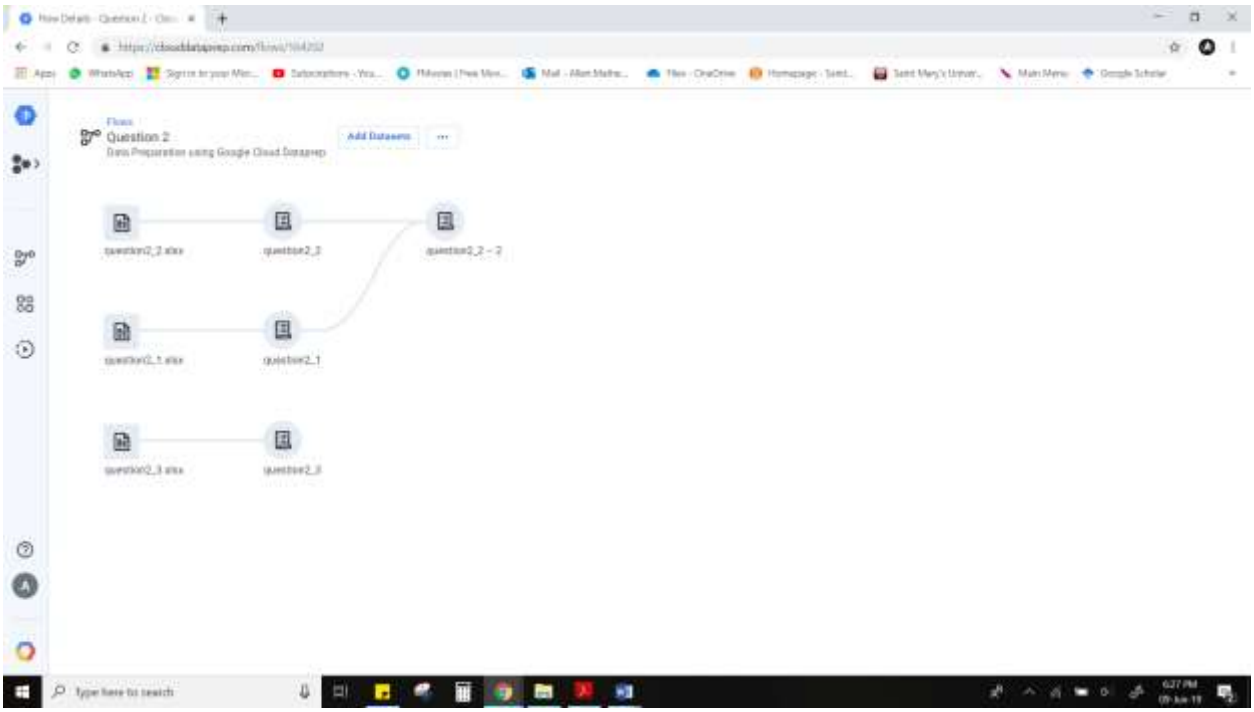

Fig 98: The flow diagram of joining both the recipes (i.e. question2_2-2 and question2_1).

A new recipe is added to the question2_2-2 dataset, to maintain a copy of the work done so far.

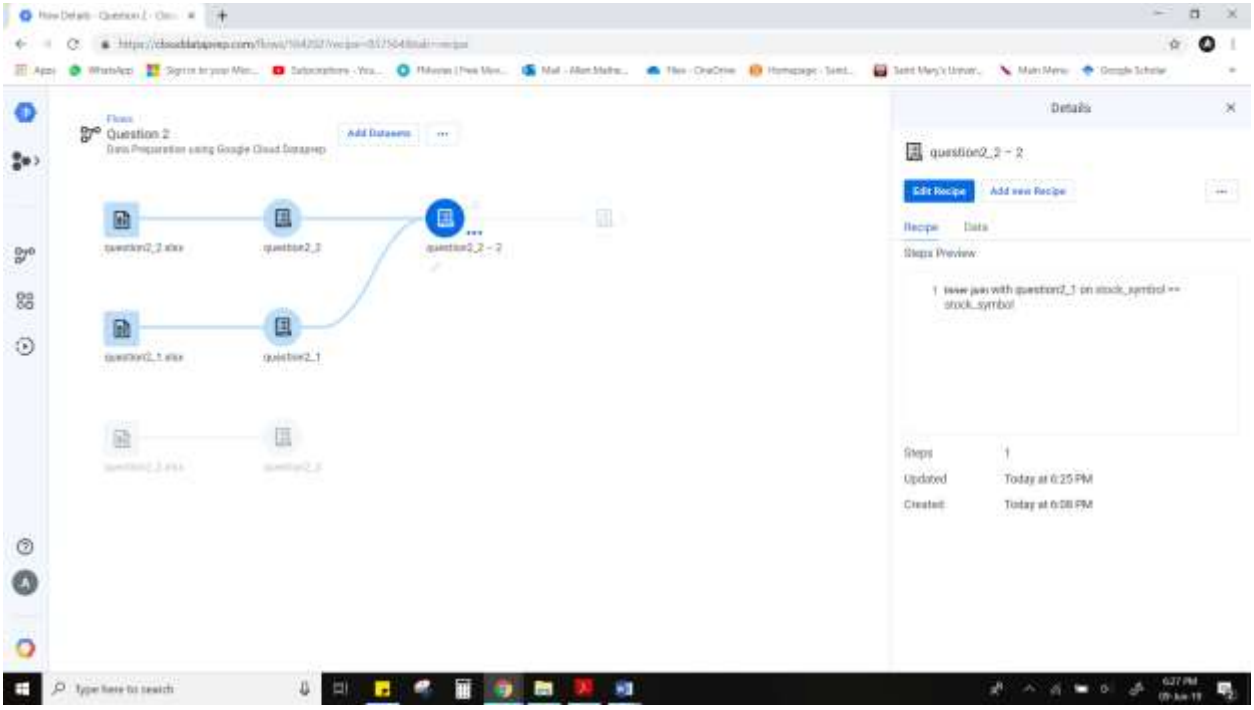

Fig 99: Adding new Recipe

To join the remaining dataset, the new question2_2-3 will be used.

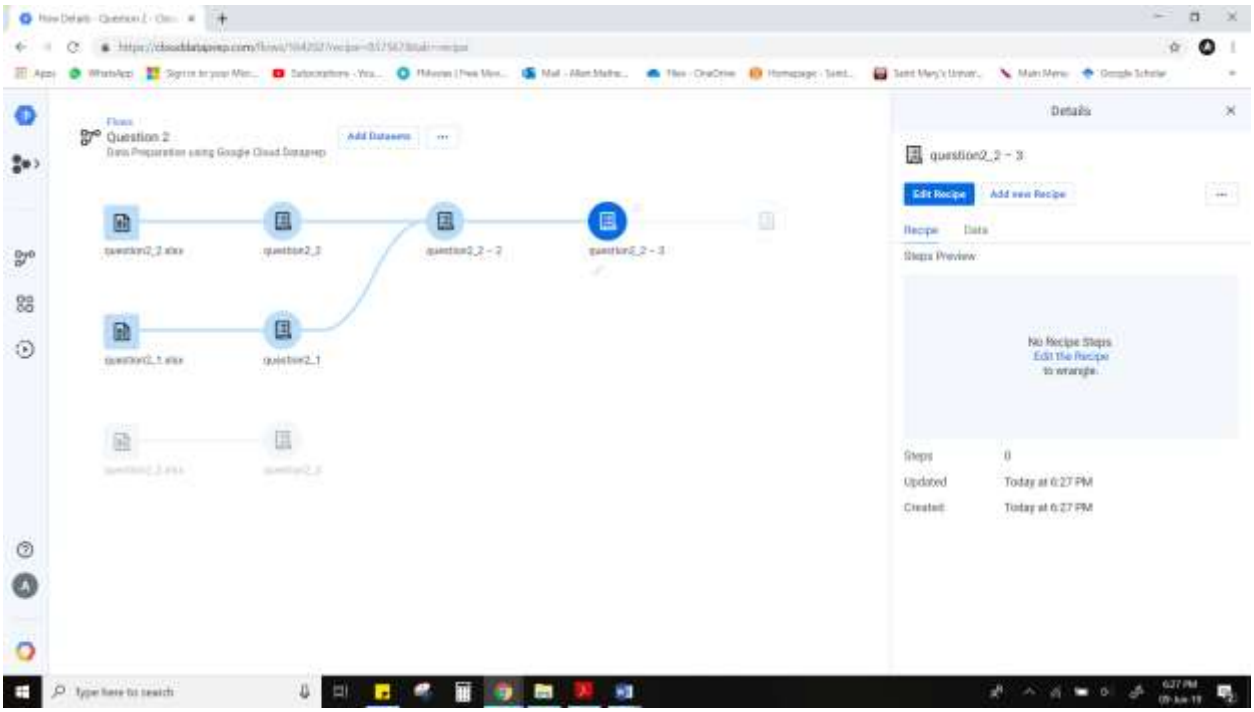

Fig 100: new Recipe was created

The searching tool located at the top right corner of the browser can be used to select the “Join datasets” option.

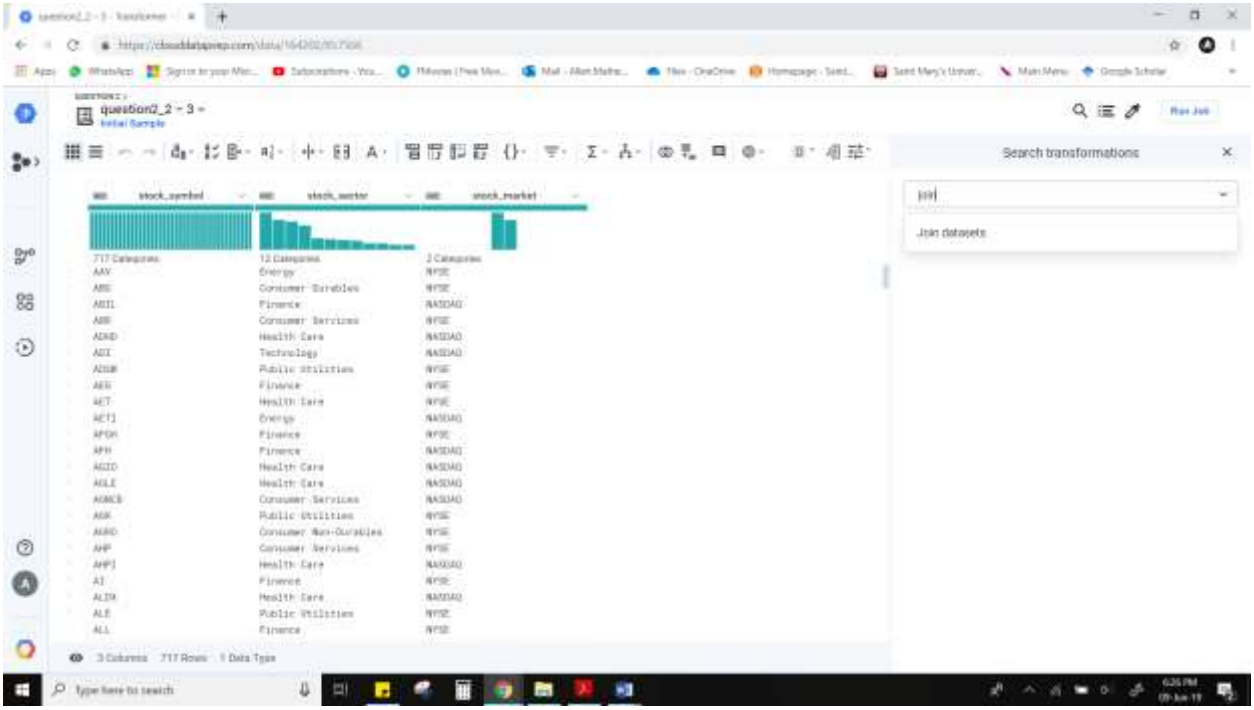

Fig 101: Searching for Join Datasets option.

The question2_3 recipe is selected to join with the question2_2-3 recipe

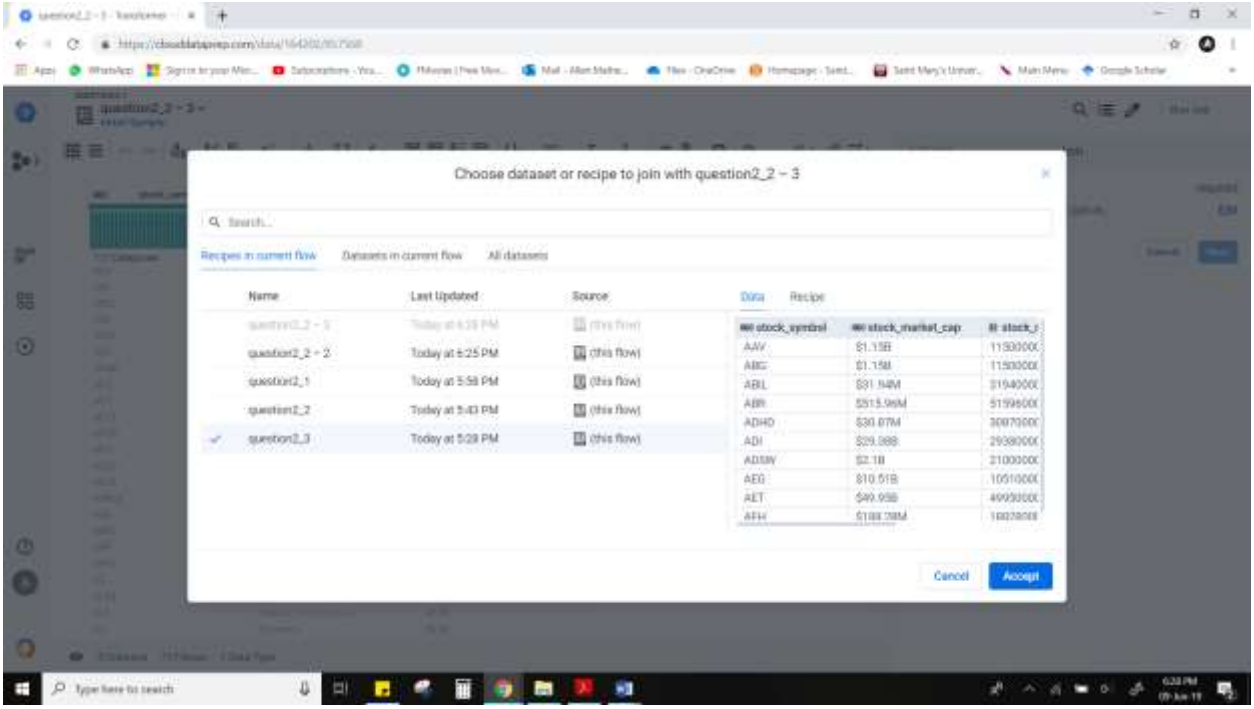

Fig 102: Selecting the question2_3 recipe

To JOIN the datasets, a common column needs to be present between the 2 datasets. Google Data Prep automatically provides a suggestion for the Join keys (i.e. stock_symbol). From the summary of the results it is evident that between the 2 datasets there are 714 records with similar keys.

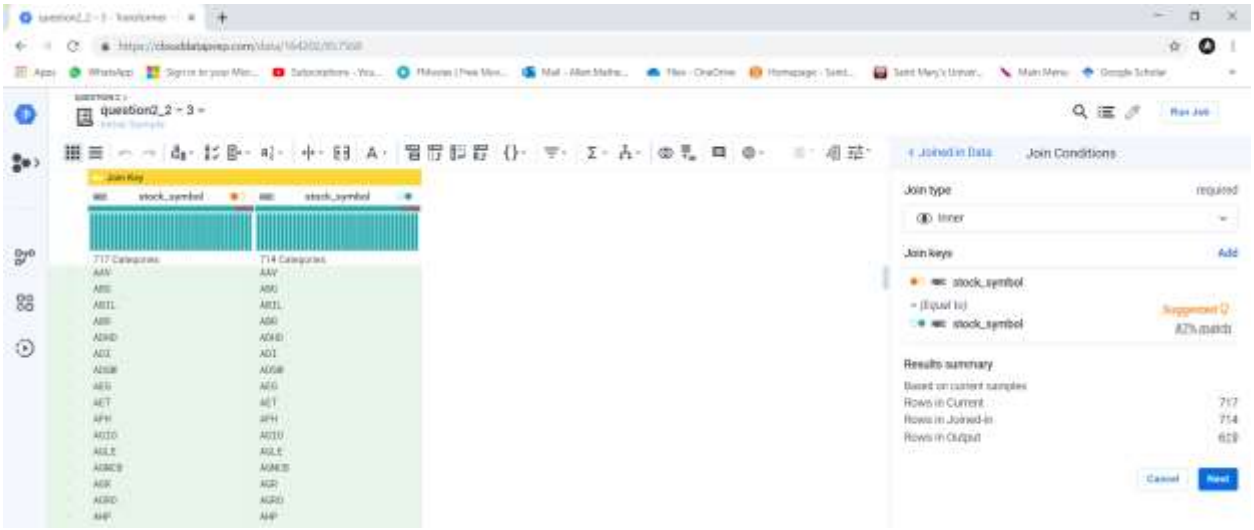

Fig 103: Google Data Prep automatically provides a suggestion for the Join keys

The necessary output columns are selected between the 2 datasets before the data sets are joined.

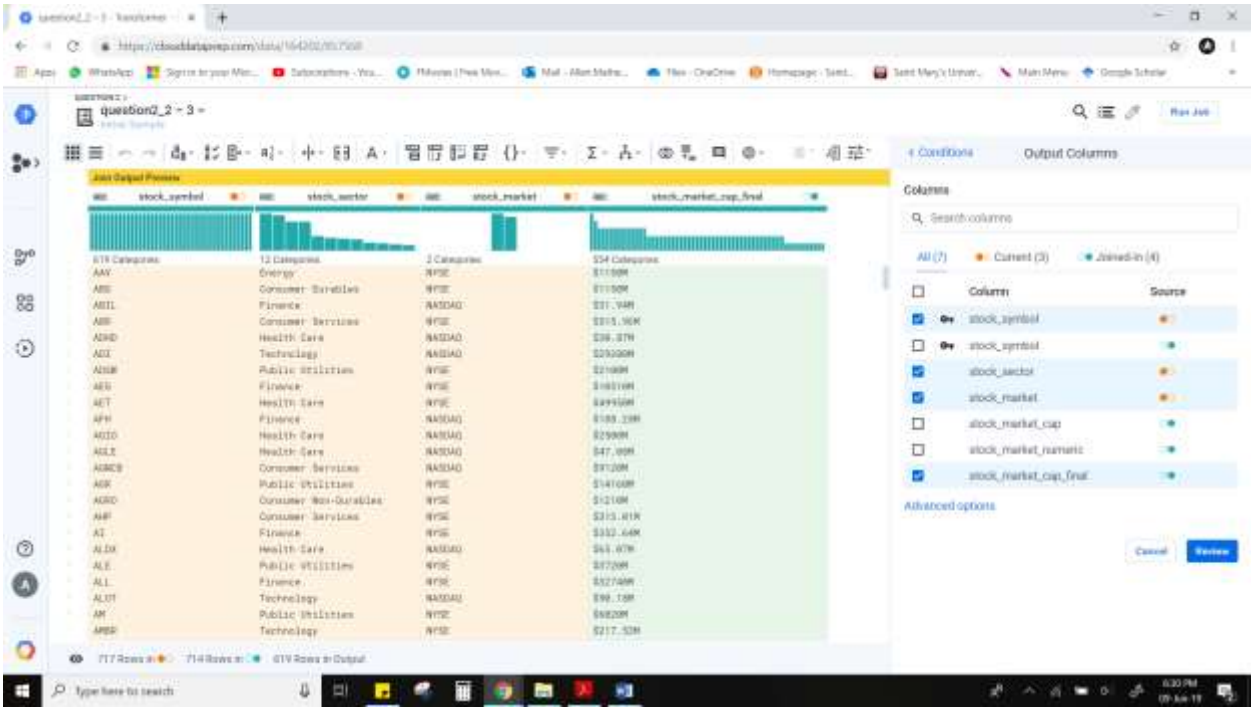

Fig 104: Select datasets Output columns

Before adding the changes to the recipe, the JOIN operation is reviewed.

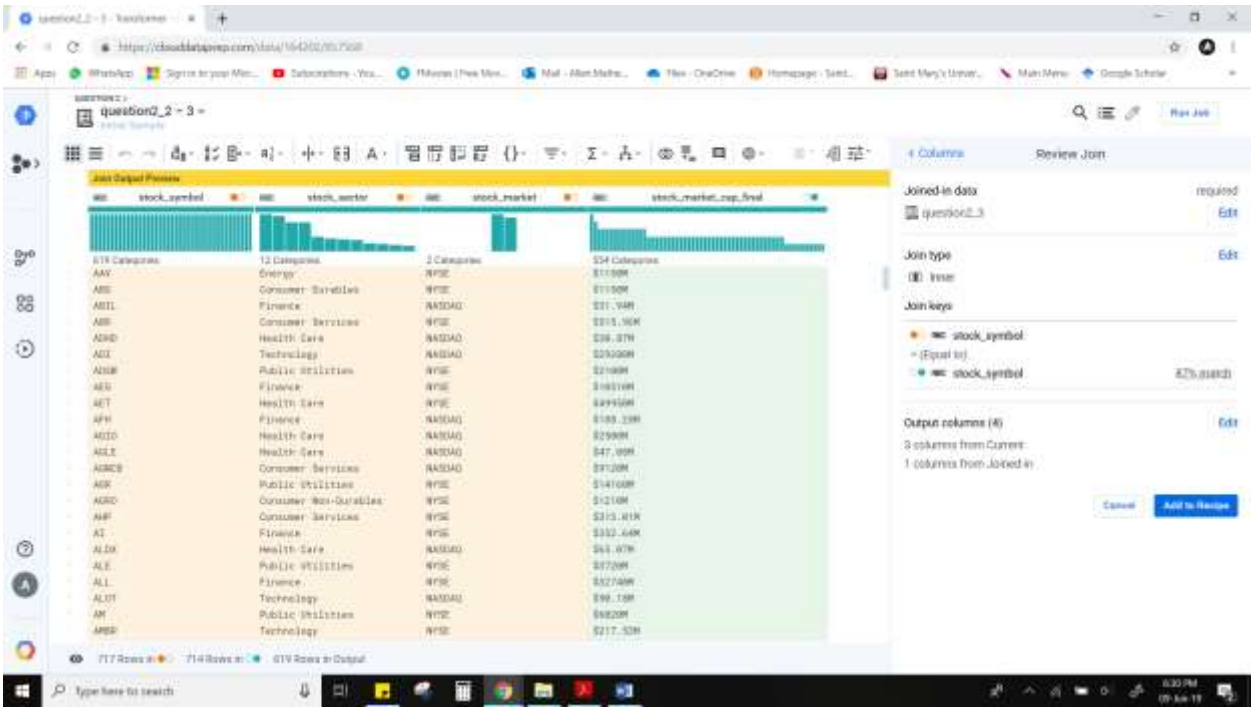

Fig 105: Reviewing datasets Output columns

The JOIN operation was successfully able to join both the recipes (i.e. question2_2-3 and question2_3).

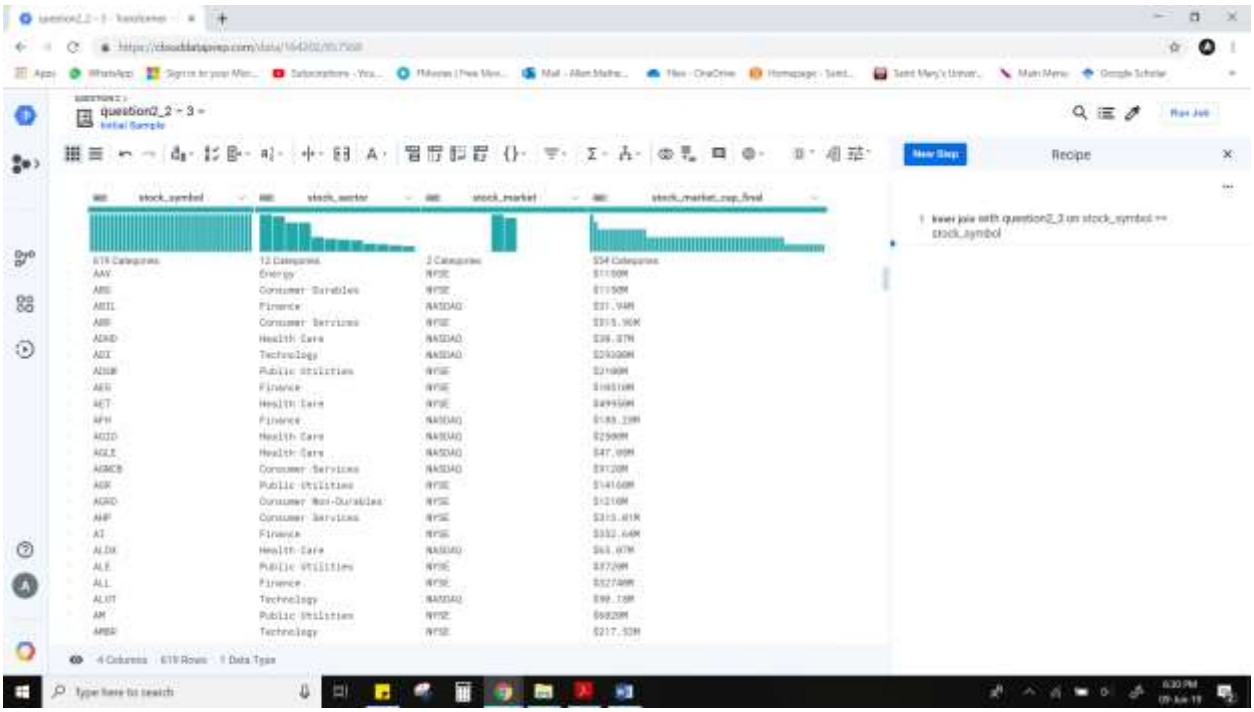

Fig 106: Inner join data sets question2_2-3 and question2_3 on stock_symbol

The Flow Diagram showing that the JOIN operation was successfully able to join both the recipes (i.e. question2_2-3 and question2_3).

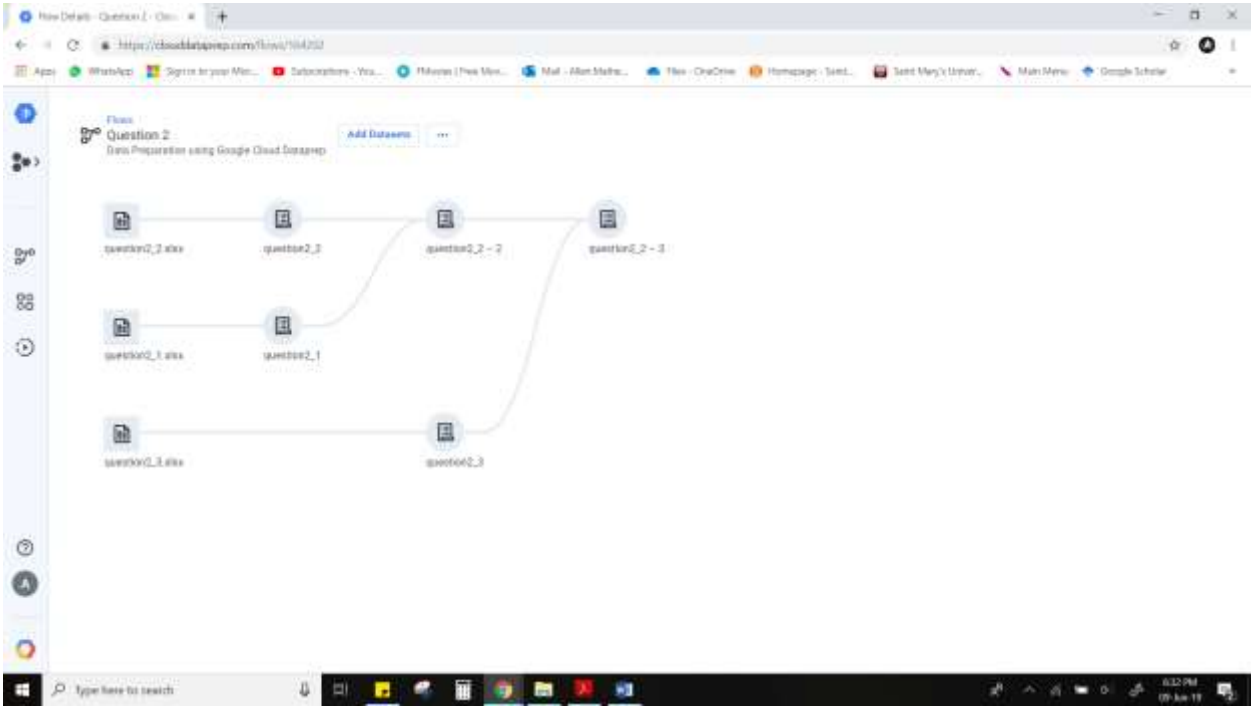

Fig 107: The flow diagram of joining both the recipes (i.e. question2_2-3 and question2_3).

A new recipe is added to the question2_2-3 dataset, to maintain a copy of the work done so far.

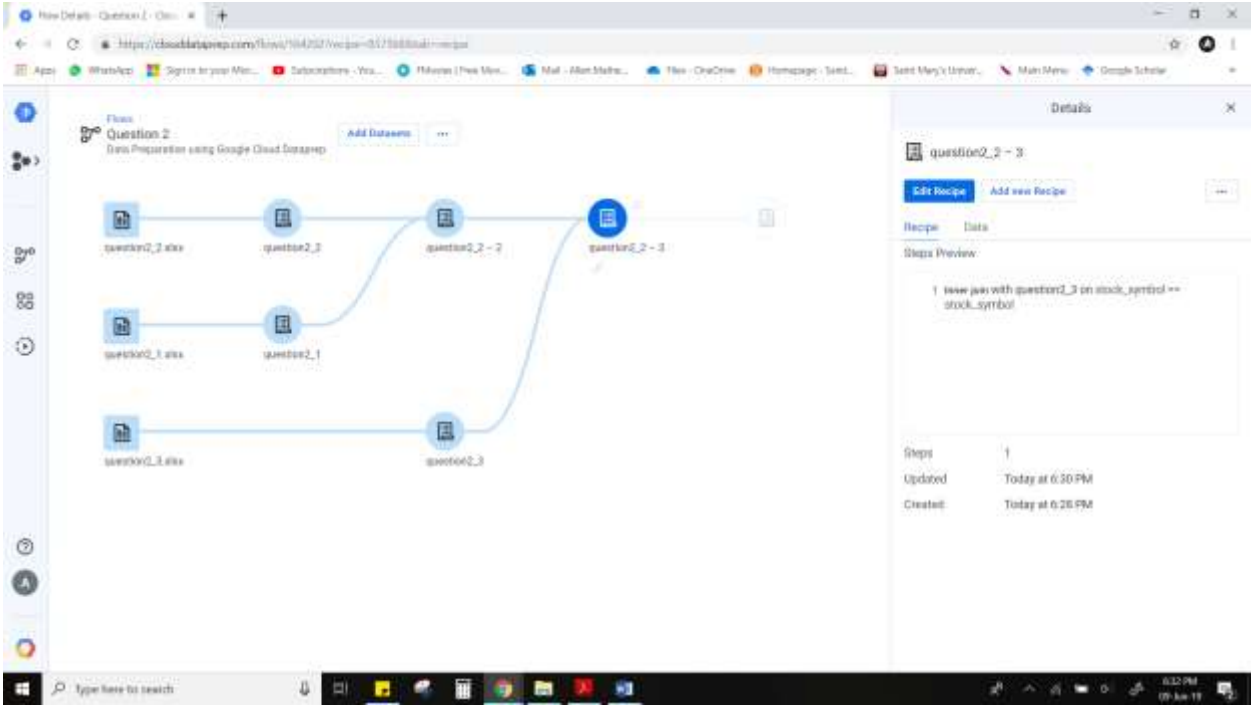

Fig 108: Adding new Recipe

To extract all records of the stocks related to **finance** sector, a new recipe is added to the question2_2-4 dataset.

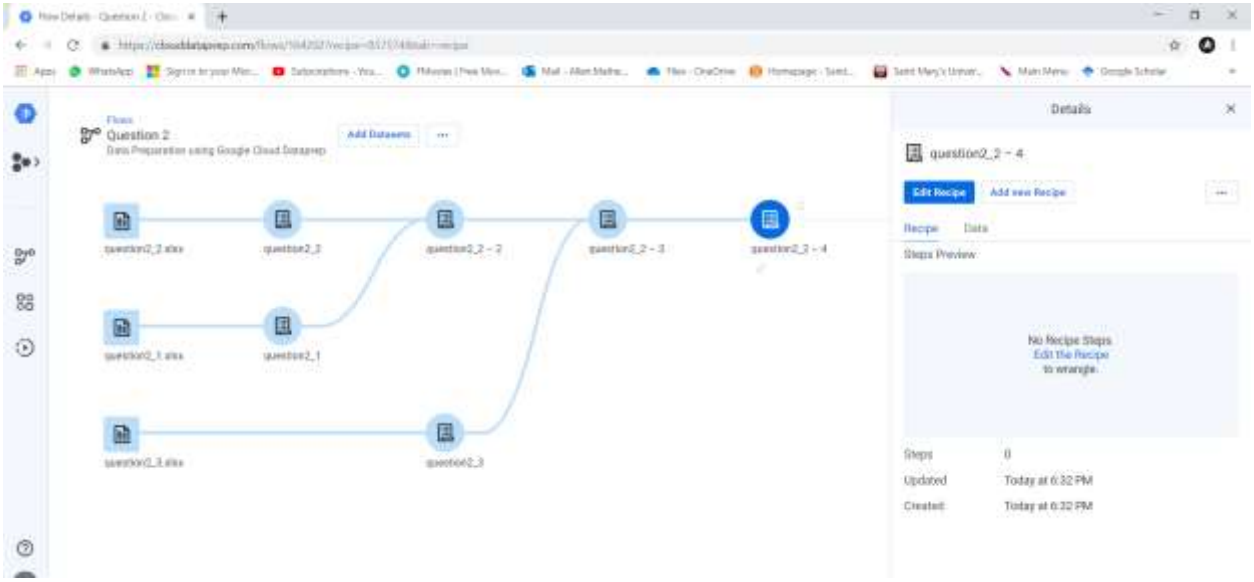

Fig 109: Adding a new Recipe.

The “Filter rows using custom formula” option is added to the recipe, it keeps all rows (i.e 110 rows) where the records in the stock_sector column is “Finance”. The following is the expression (Google Cloud, 2019) used to filter the rows in the dataset:

Condition

stock_sector == ‘Finance’

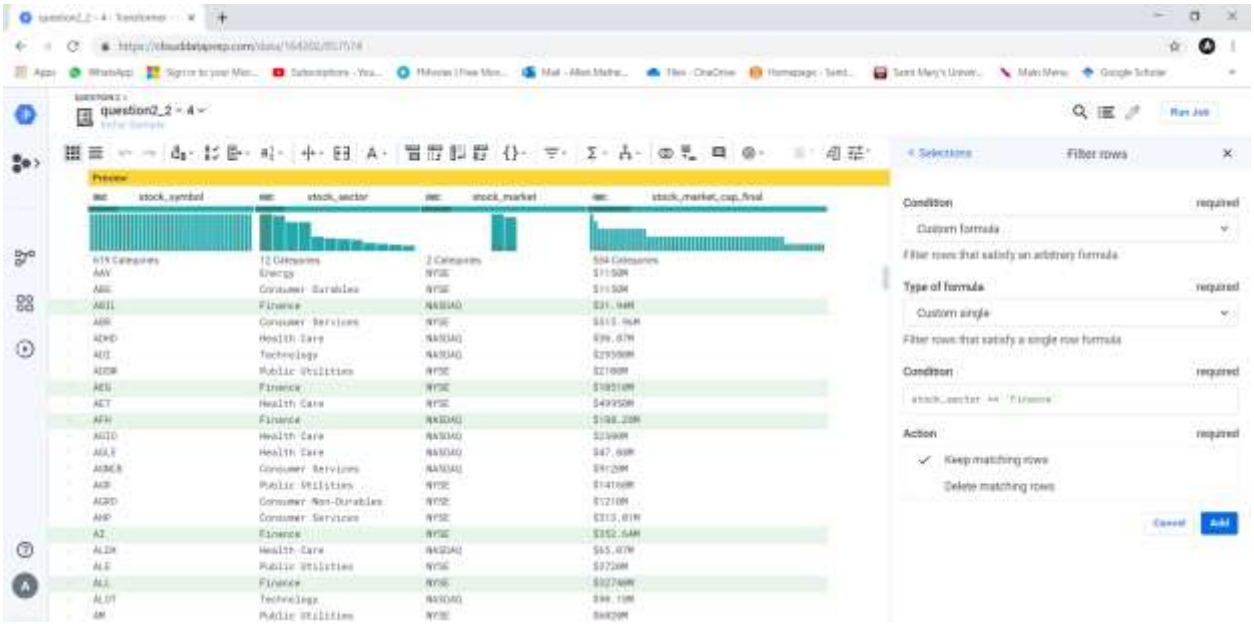

Fig 110: Select rows where stock_sector is FINANCE

On applying the filter, 110 rows were successfully retrieved where the records in the stock_sector column is “Finance”.

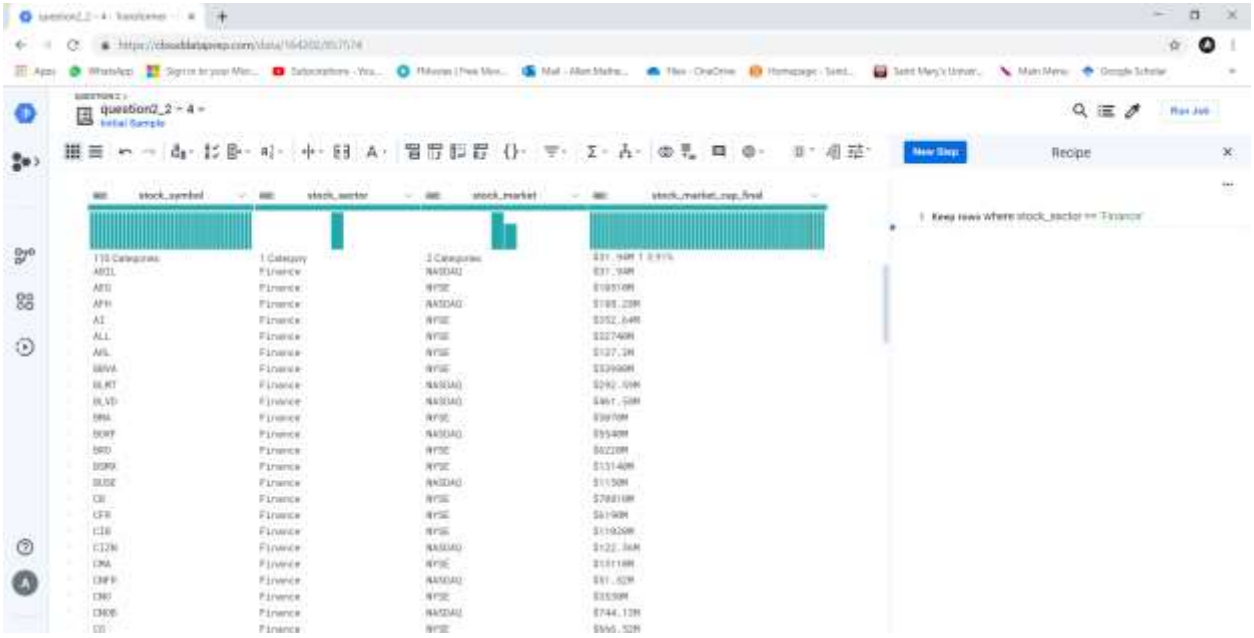

Fig 111: Successfully retrieved where the records in the stock_sector column is “Finance”..

The stock_sector column is then deleted. Since the column is not required when exporting the output.

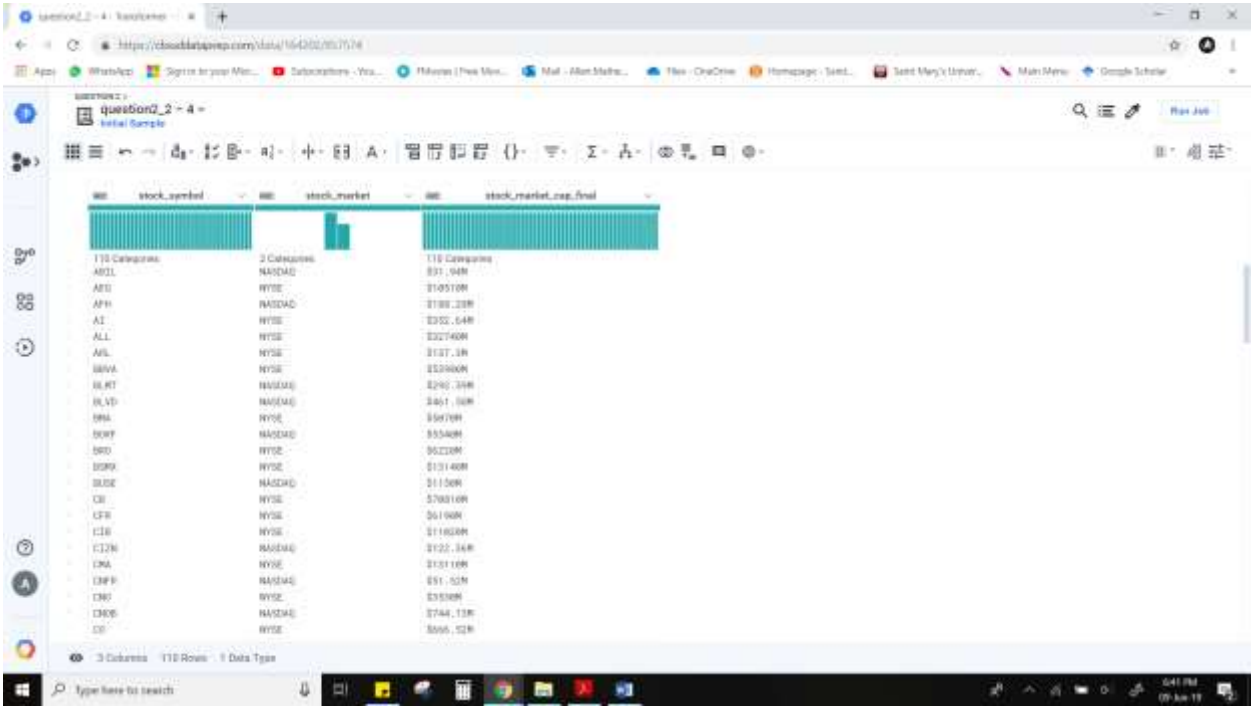

Fig 112: Successfully Deleted stock_sector column

To extract all records of the stocks related to **health care sector**, a new recipe is added to the question2_2-5 dataset.

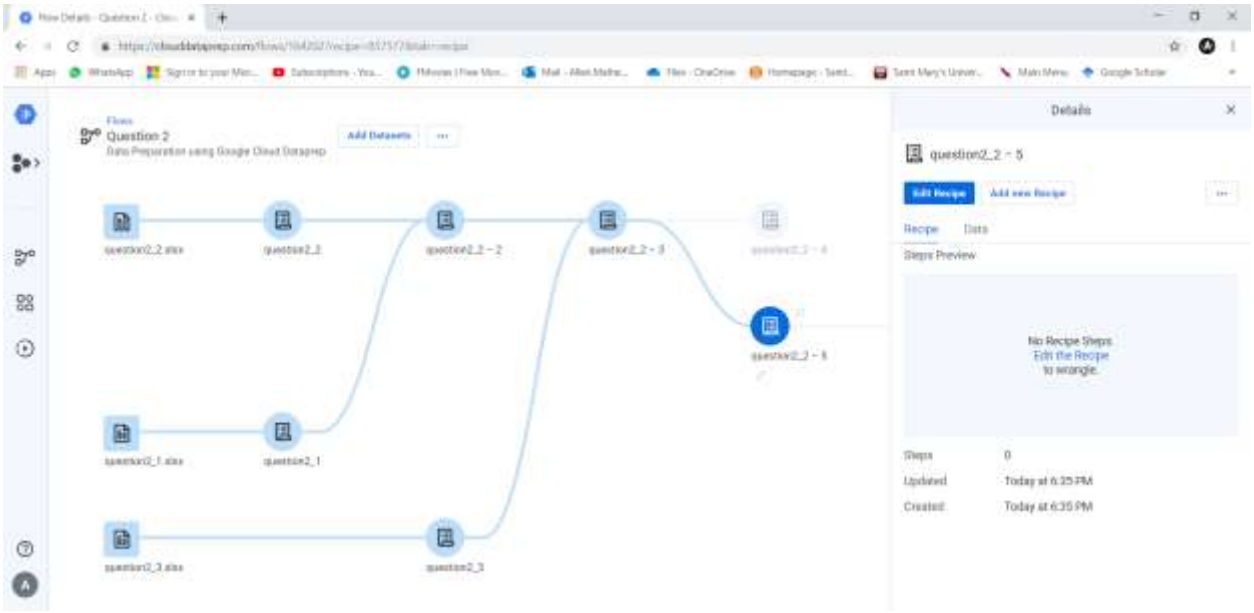

Fig 113: Adding a new Recipe.

The "Filter rows using custom formula" option is added to the recipe, it keeps all rows (i.e 86 rows) where the records in the stock_sector column is "Health Care'". The following is the expression (Google Cloud, 2019) used to filter the rows in the dataset:

Condition

stock_sector == 'Health Care'

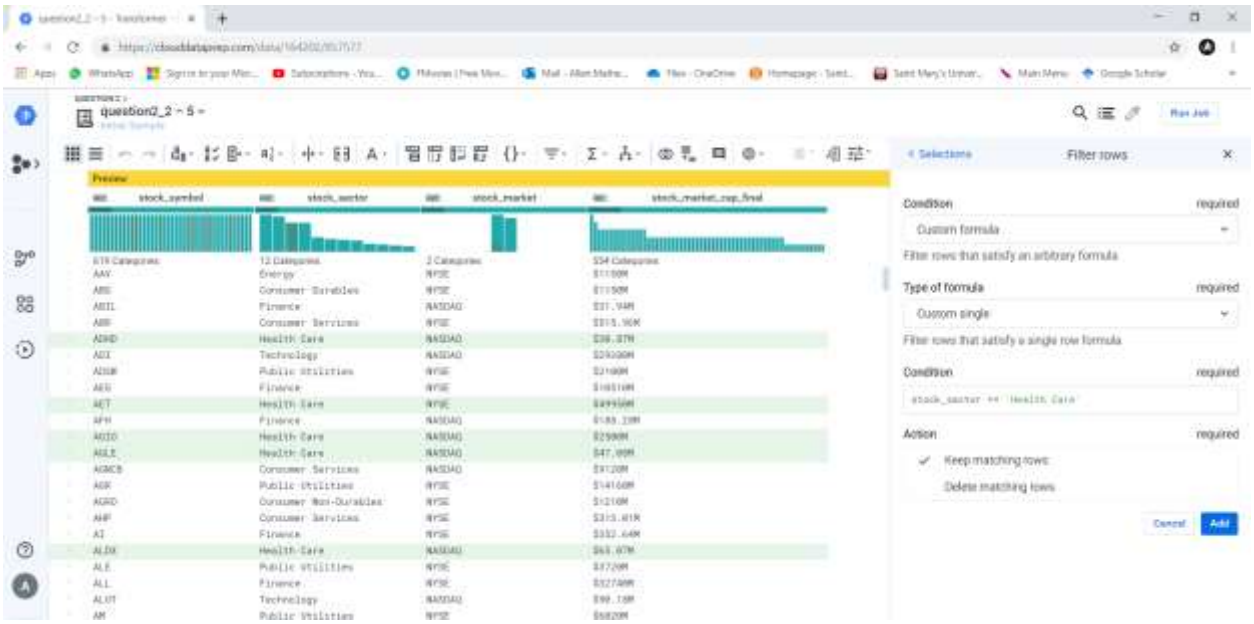

Fig 114: Select rows where stock_sector is Health Care

On applying the filter, 86 rows were successfully retrieved where the records in the stock_sector column is “Health Care’”.

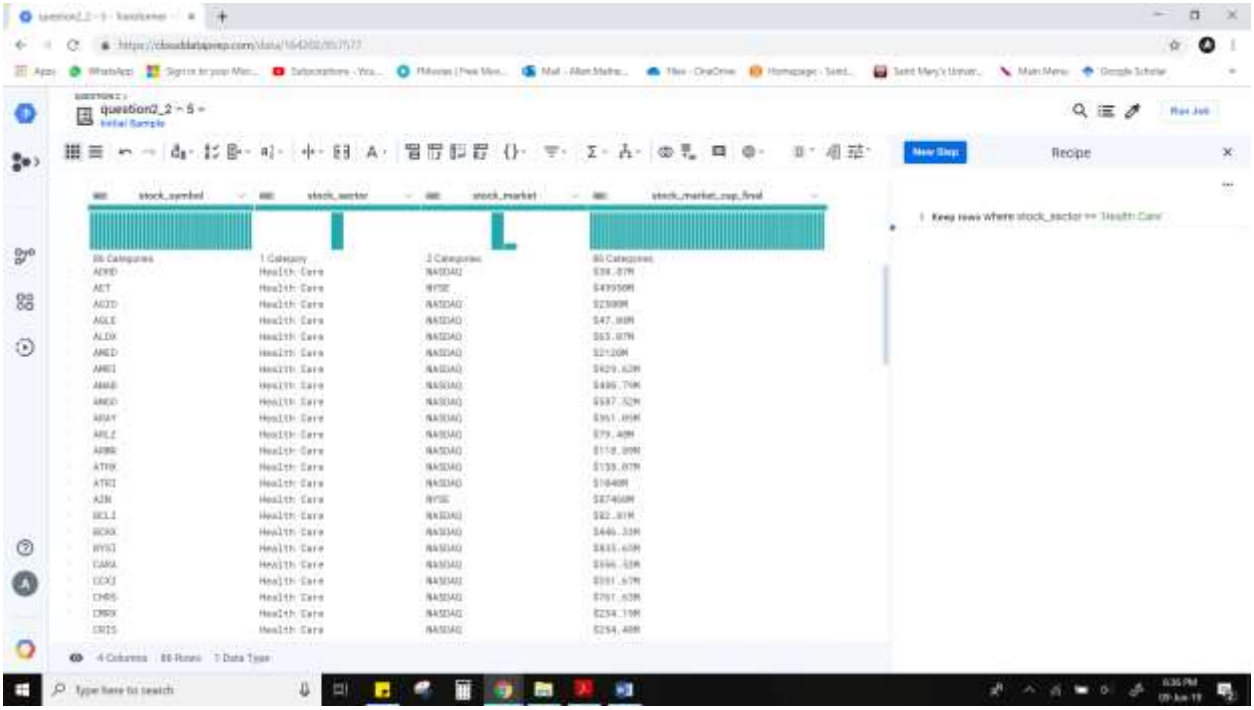

Fig 115: Successfully retrieved where the records in the stock_sector column is Health Care

The stock_sector column is then deleted. Since the column is not required when exporting the output.

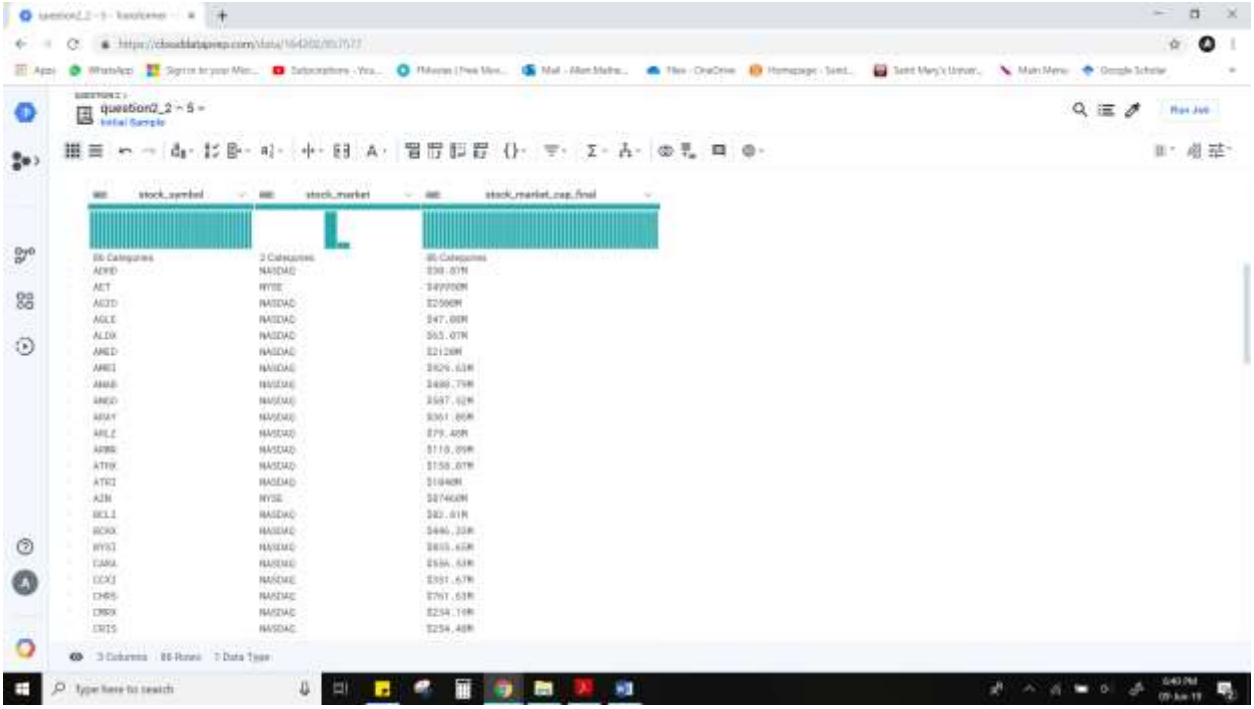

Fig 116: Successfully Deleted stock_sector column

To extract all records of the stocks related to **technology sector**, a new recipe is added to the question2_2-6 dataset.

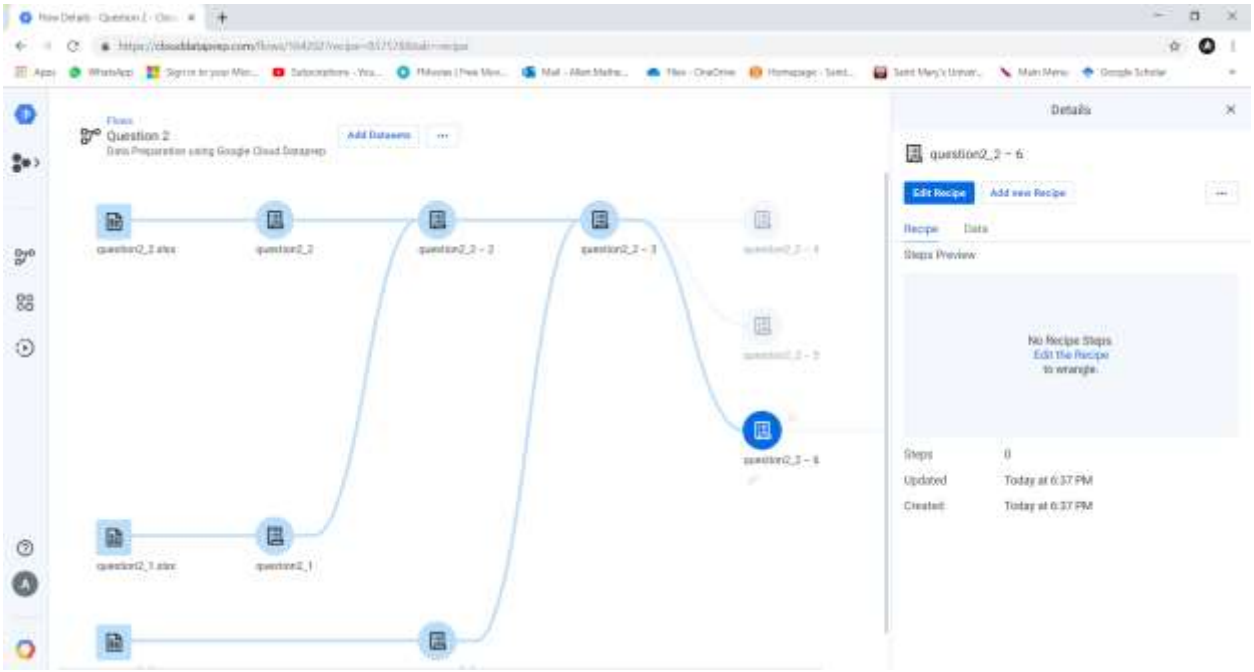

Fig 117: Adding a new Recipe.

The “Filter rows using custom formula” option is added to the recipe, it keeps all rows (i.e 86 rows) where the records in the stock_sector column is “Technology”. The following is the expression (Google Cloud, 2019) used to filter the rows in the dataset:

Condition

stock_sector == ‘Technology’

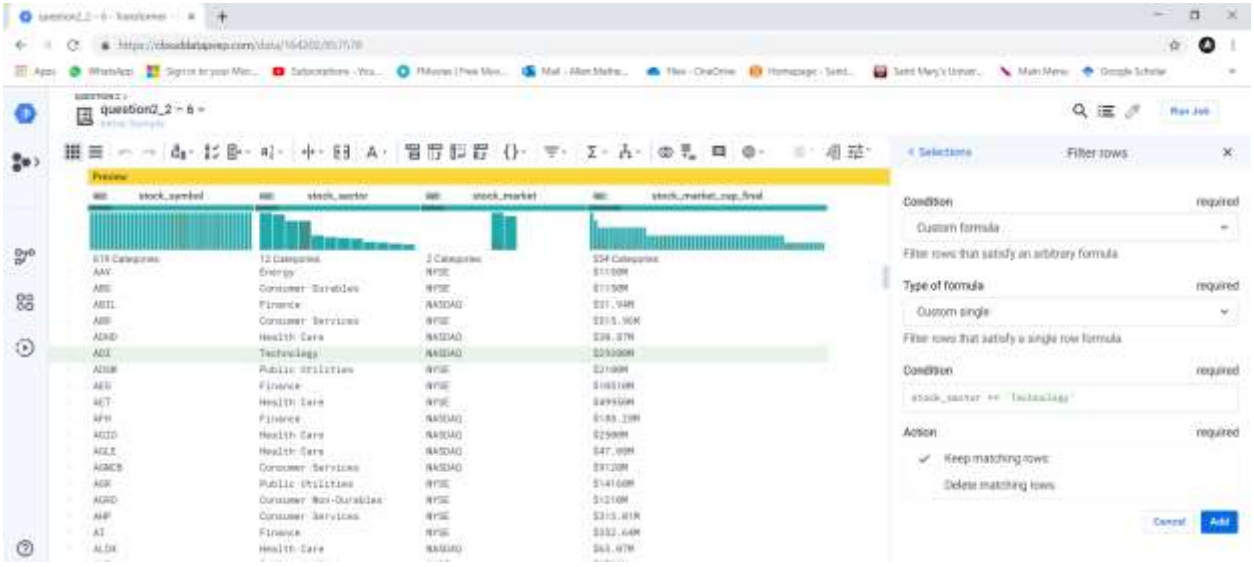

Fig 118: Select rows where stock_sector is Technology column

On applying the filter, 86 rows were successfully retrieved where the records in the stock_sector column is “Technology”.

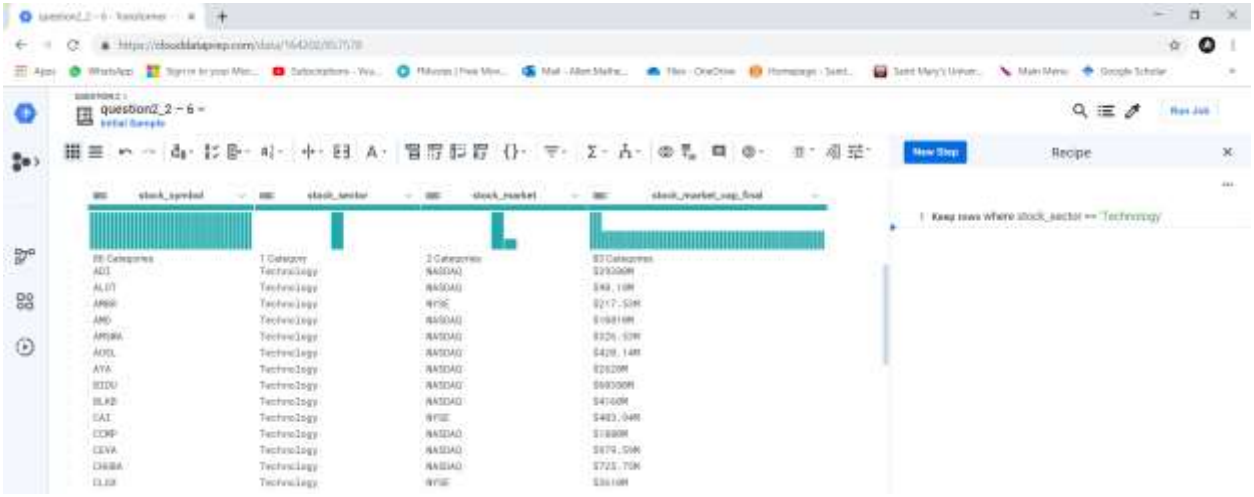

Fig 119: Successfully retrieved where the records in the stock_sector column is Technology

The stock_sector column is then deleted. Since the column is not required when exporting the output.

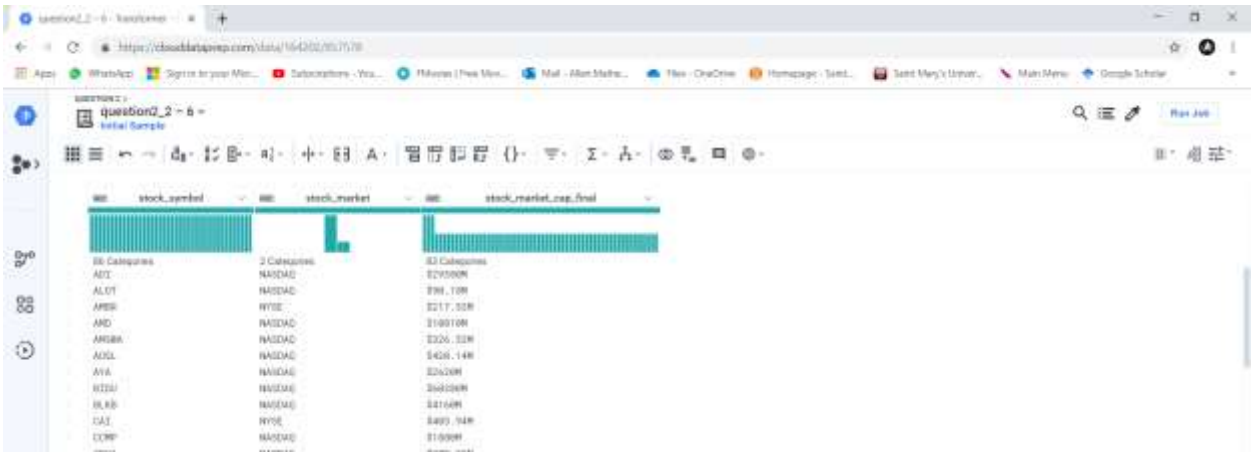

Fig 120: Successfully Deleted stock_sector column

7) Creating Output to Run

Rename the following recipes from:

- question2_2-4 to solution2_1
- question2_2-5 to solution2_2
- question2_2-6 to solution2_3

After renaming the recipes “Create the Output to Run” for each recipe. Then click on the “Run Job” button for each recipe.

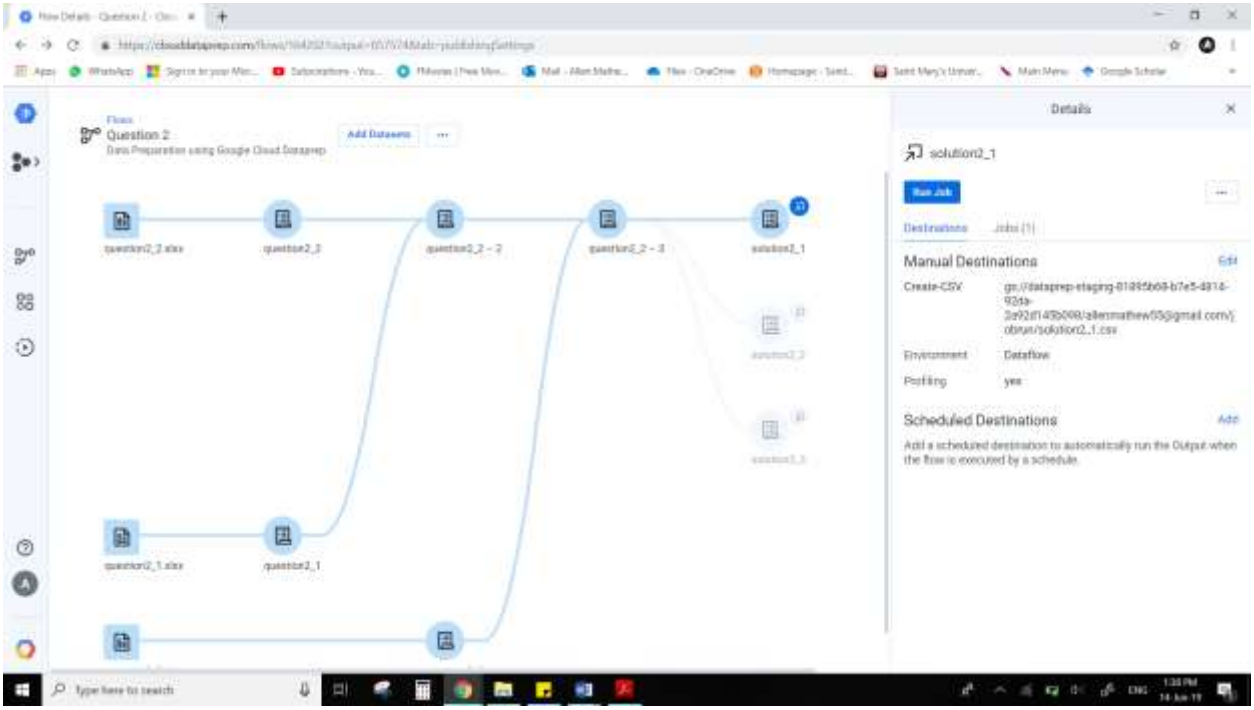

Fig 121: Create the Output to Run

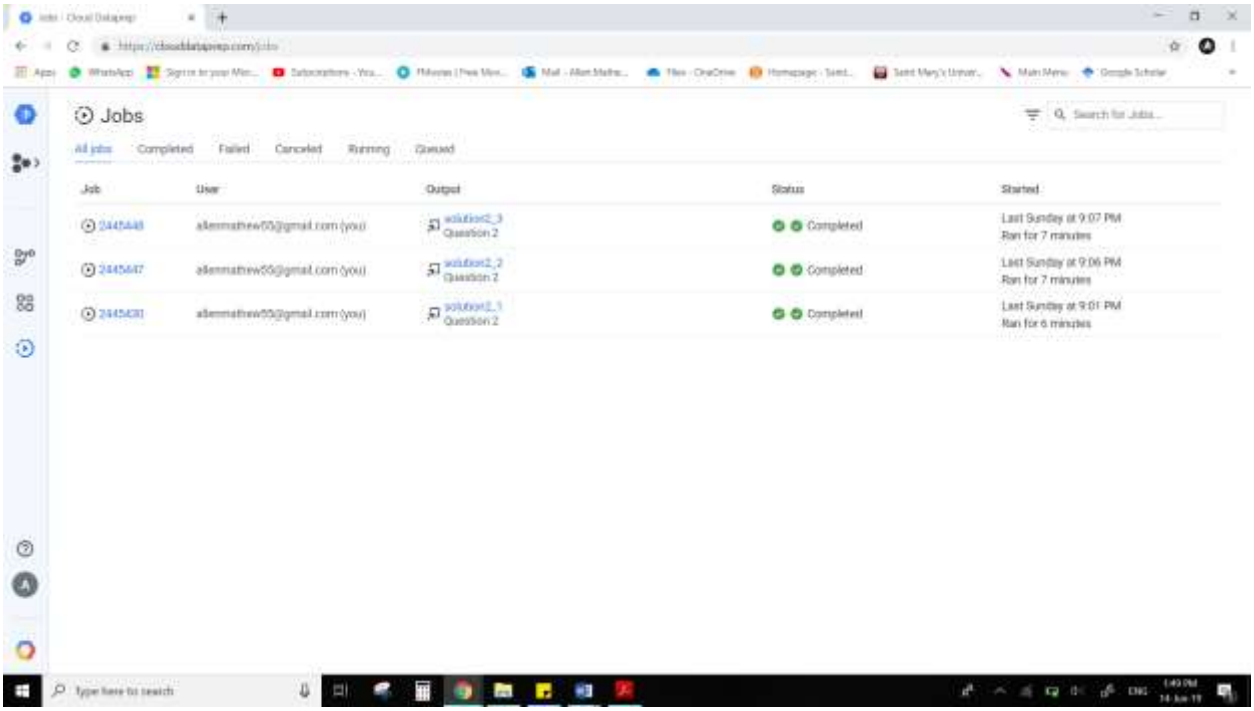

Fig 122: All jobs ran successfully.

The output CSV file can be exported by selecting each “Job”, then navigating to the “Output Destinations” tab and then click on link under the “Location” section.

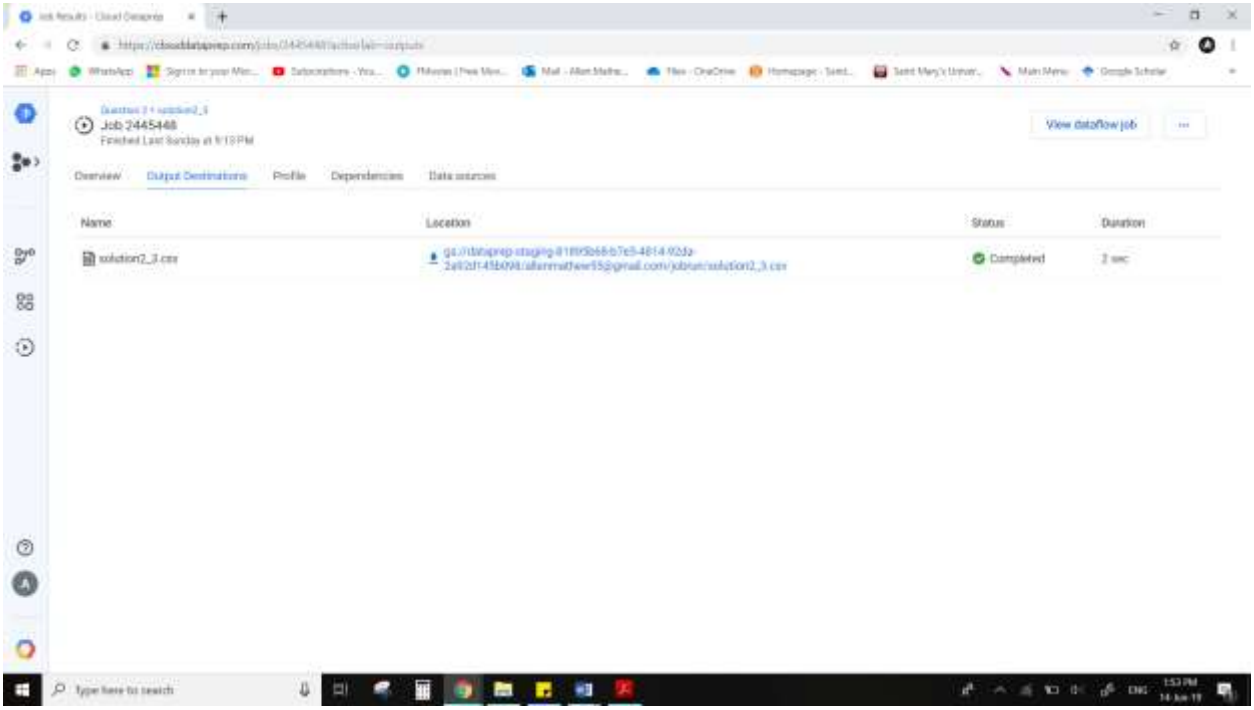

Fig 123: Download the document

All the CSV files were exported successfully.

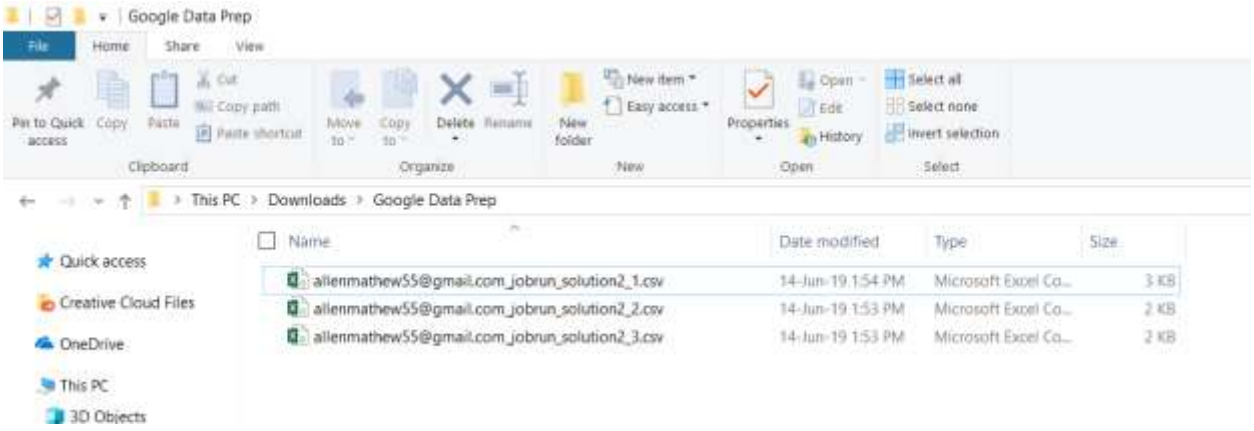

Fig 124: The Exported CSV files

B. Brief on Work Done on Google Data Prep:

Google Cloud Dataprep is a cloud-based data preparation tool. To use the tool an account was created on Google Cloud Dataprep, then a new project was built where the 3 datasets that was provided was loaded into the project.

A set of tasks were carried out on this dataset such as removing duplicate or n/a records, removing records where the stock market capital is lesser than 30 million, to maintain consistency records for the stock market capital were transformed to represent the data in million (i.e. billion were converted to millions). All the cleaned datasets are then joined to create a single combined dataset, from which multiple output files were created wherein the dataset is further filtered based on the respective market sector.

SSIS / Data Preparation (ETL Pipeline)

Microsoft SSIS can be used for Data preparation by using ETL pipeline process. We have also used MS SQL Server for creating the database Schema.

A. Dataset Used:

We have used NYC rolling sales Dataset for this Assignment. The data is about sales of properties in New York city, United states. The dataset provides us information such as:

1. Sale date
2. Year built
3. Building category
4. Building Class
5. Tax Class
6. Zip code
7. Sale Price

B. Schema:

For ETL Pipeline process we have created a Star Database schema which consists of seven Dimension Table and one Fact table. All the tables in this schema are in third normalized form.

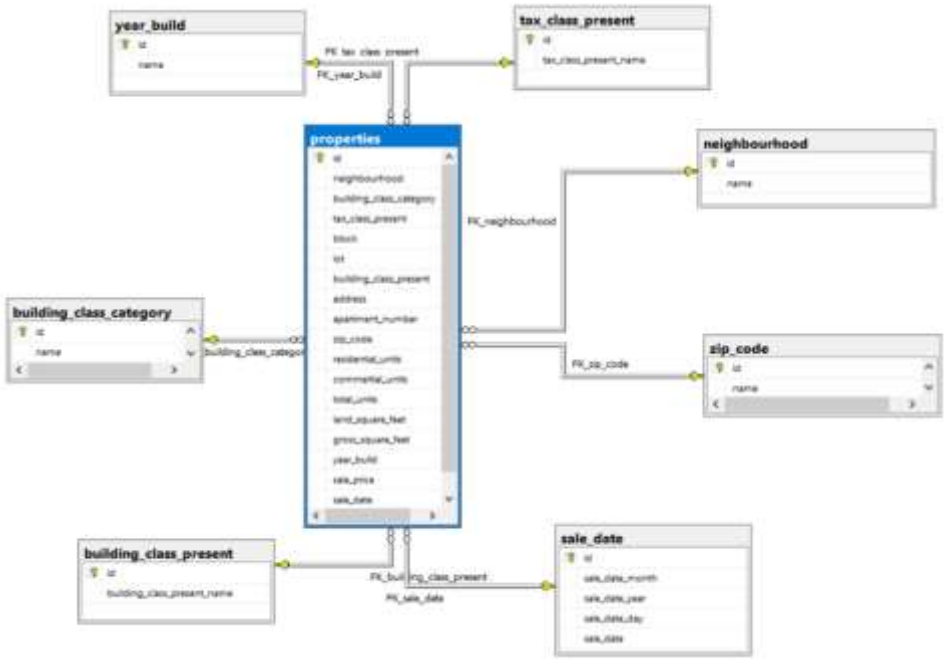

Fig 125:Star schema implementation of our NYCPROPERTYSALES Database

The tables created in this Star schema are given below: -

Dimension Tables:

1. NEIGHBORHOOD
2. ZIP_CODE
3. TAX_CLASS_PRESENT
4. BUILDING_CLASS_PRESENT
5. SALE_DATE
6. BUILDING_CLASS_CATEGORY
7. YEAR_BUILD

Fact Tables:

1. PROPERTIES

Query for creating the database: Appendix

C. ETL Pipeline Process:

There are four major steps of ETL pipeline process as given below:

1. Empty all the tables of database to store new Data.
2. Load new Data into Dimension Tables.
3. Load new Data into fact table.
4. Save prepared data as per required format.

During this Assignment we have performed all the four major steps as shown in the image below-

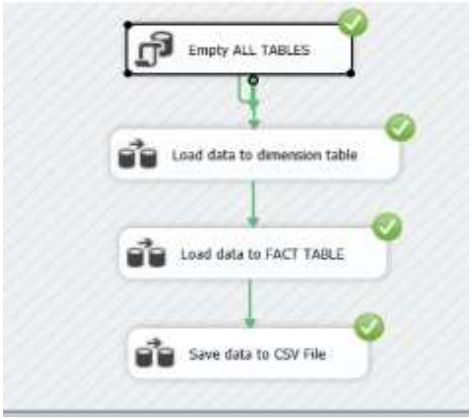

Fig 126:Four major steps for ETL Pipelining

For better understanding lets, discuss the above four tasks. The step involved in this are given below: -
We have connected SSIS tool and MS SQL so that we can manipulate our Data tables by using SQL language and created a new project

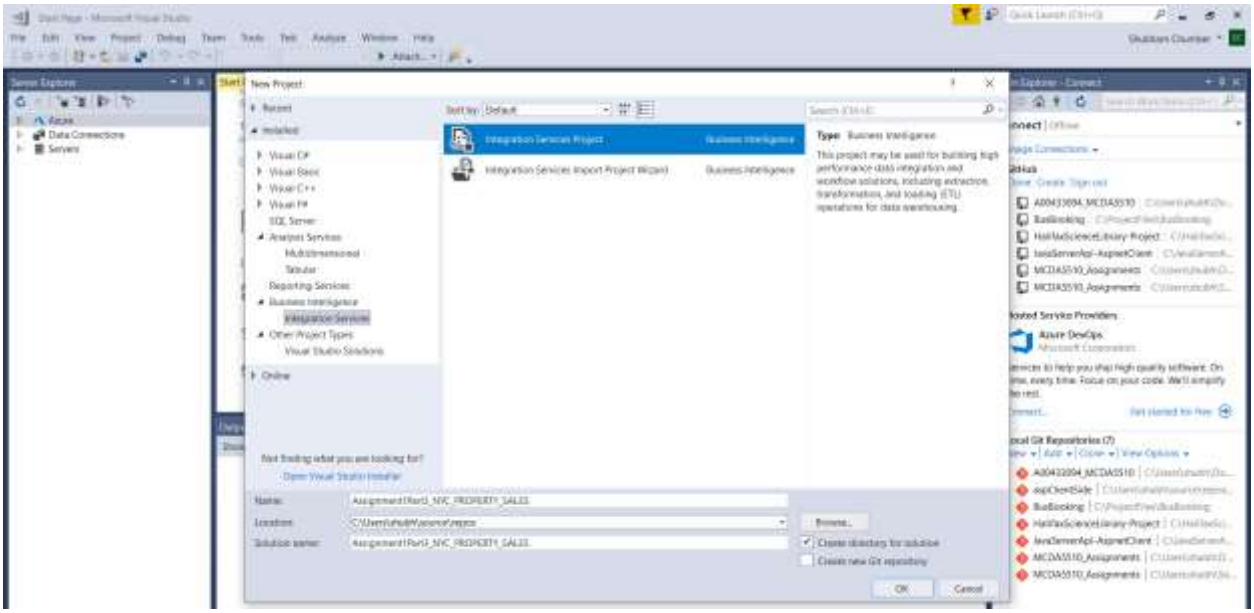

Fig 127:Creating a new project

We have created an Execute SQL task as shown in the below image:

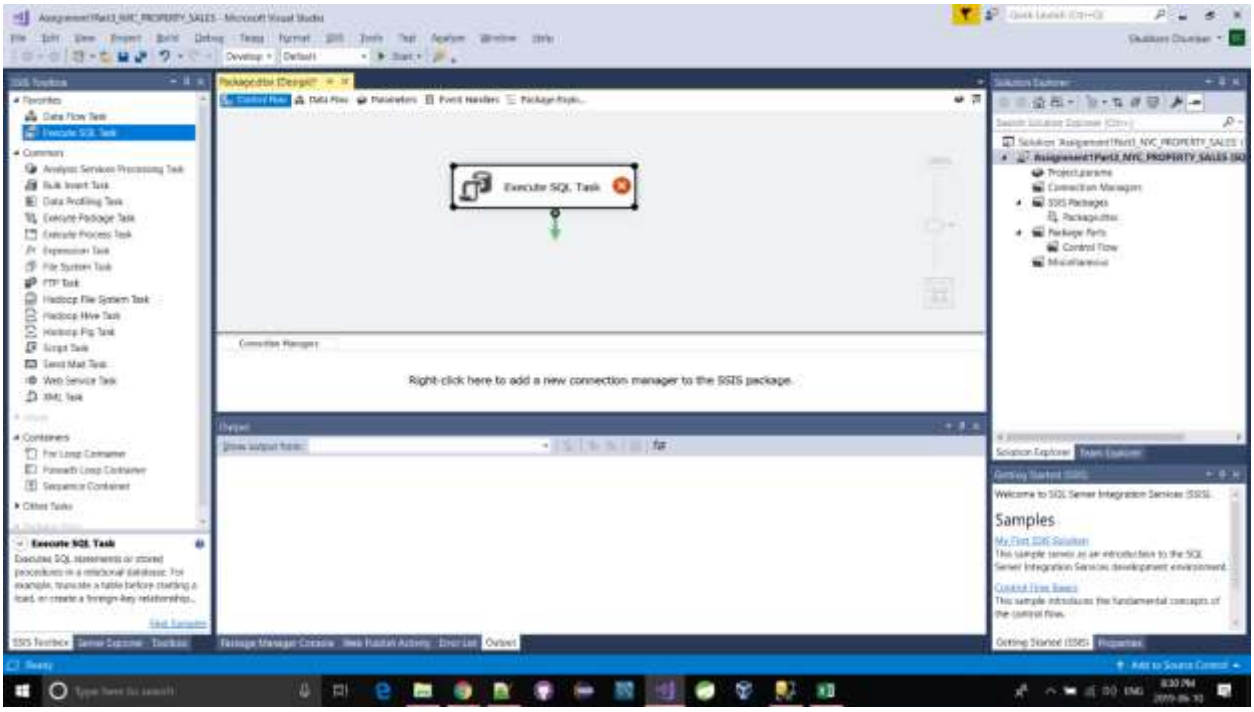

Fig 128:First task of control flow

We have named the created Execute SQL task as “EMPTY ALL TABLES” and added a Data flow task as shown in the below image to load data into dimension tables created earlier.

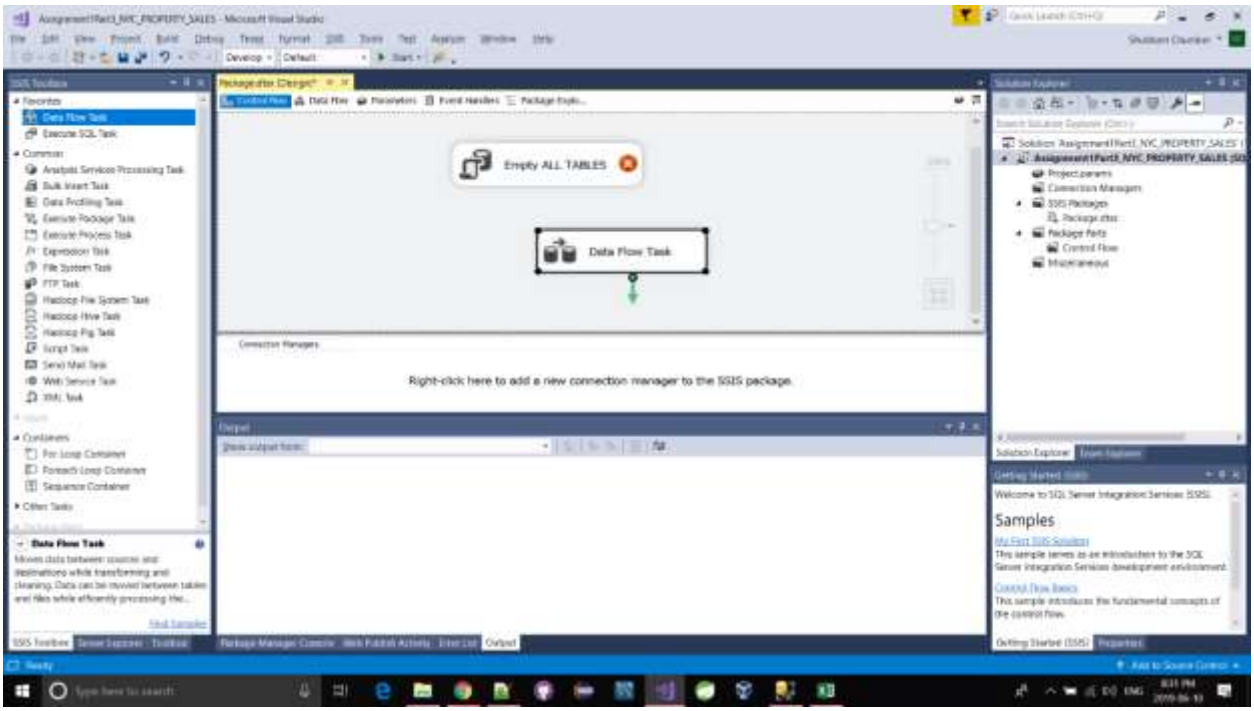

Fig 129:Drag a data flow task

We have added two more Data flow tasks to load data into Fact table and store the output in csv as our final step of the ETL pipeline for backup.

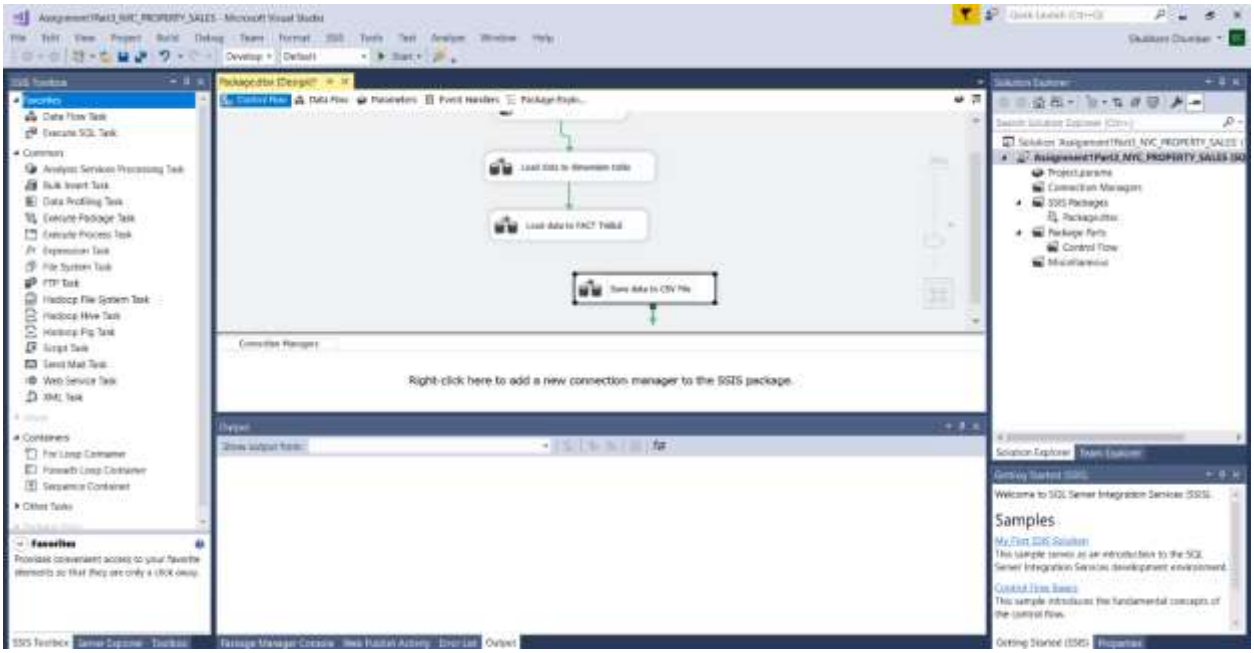

Fig 130:Creating task for saving CSV file

After Creating all the Tasks, we have worked on each task one by one. For better understanding lets discuss on each task in depth.

1. Empty All Tables:

To empty all the Dimension tables and fact table we have used used SQL query. Thus, we are connecting MS SQL and SSIS tool as shown in the below image.

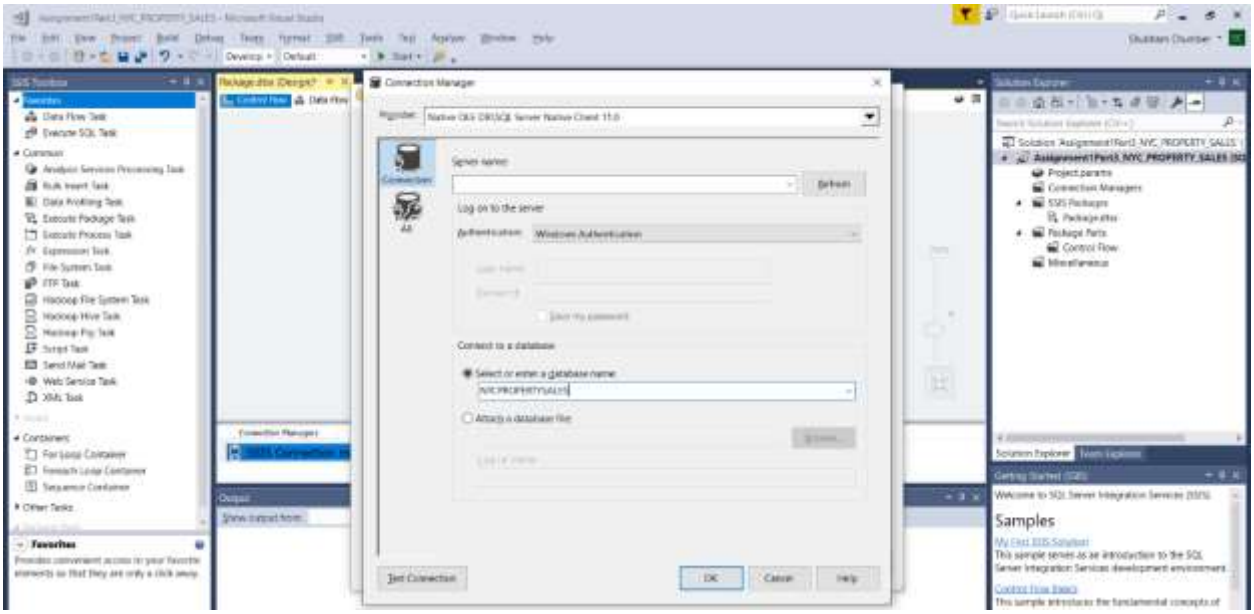

Fig 131:Making connection with the server

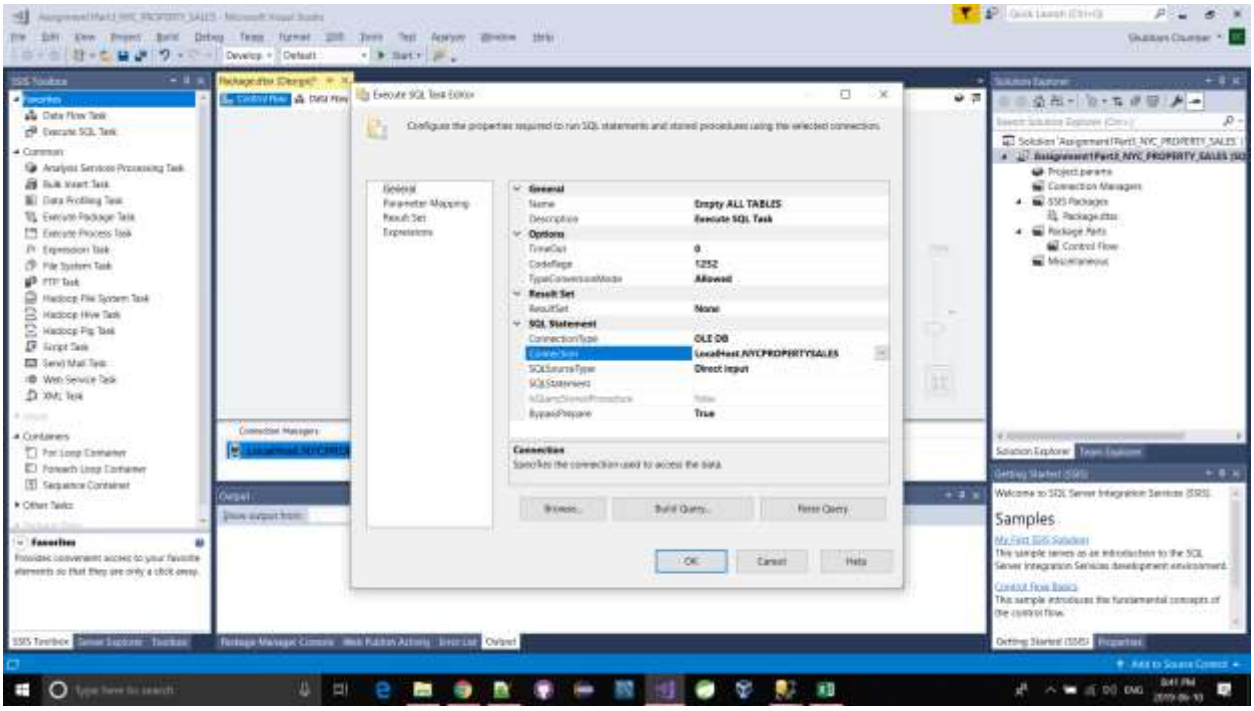

Fig 132:Configuring the properties required to run SQL statements

Then we add this query in the SSIS Window as shown in the below image

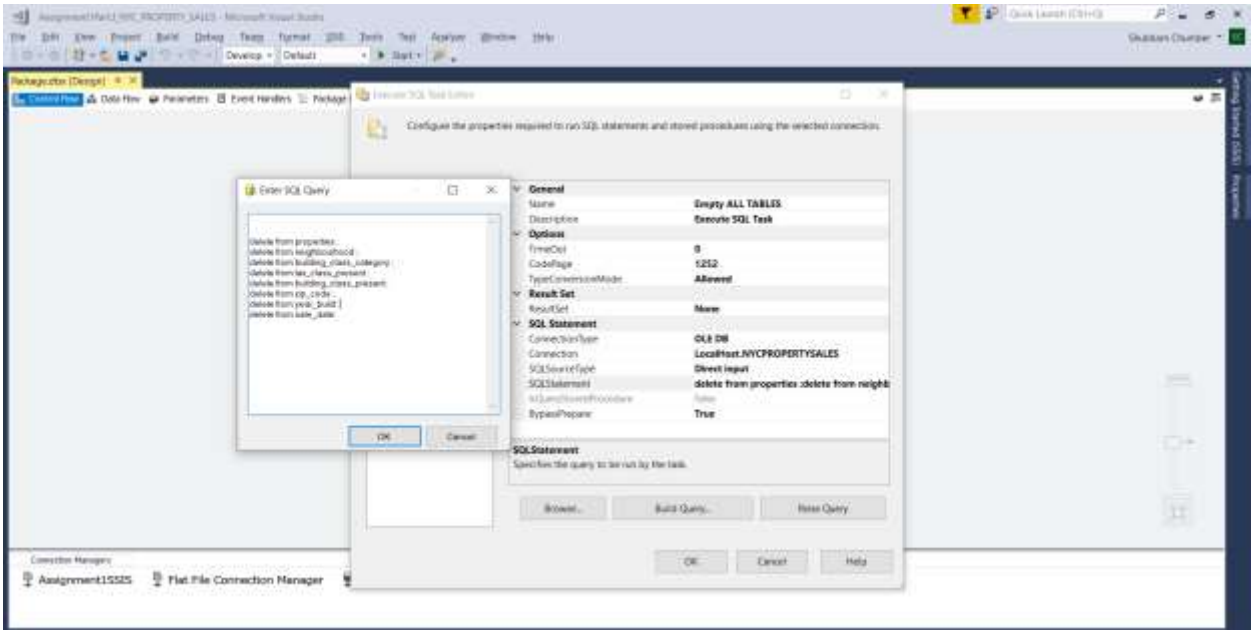

Fig 133:Executing SQL statements to empty the tables

After executing the query, we got successful emptying the tables as shown in the below image

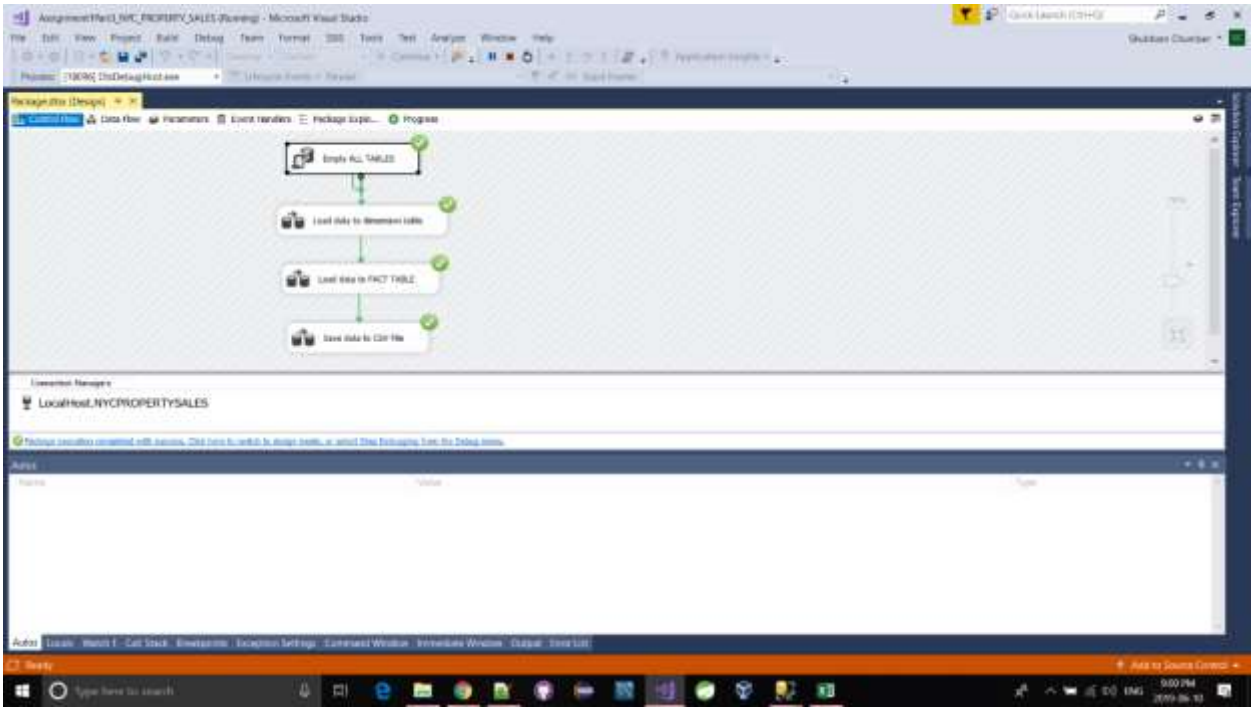

Fig 134:Screen after successful in emptying all tables

2. Load new Data into Dimension Tables:

This is our second major step to load new data into Dimension tables. We inserted Flat file source component to load data into dimension tables from dataset “nyc-rolling-sales” as shown in the below image.

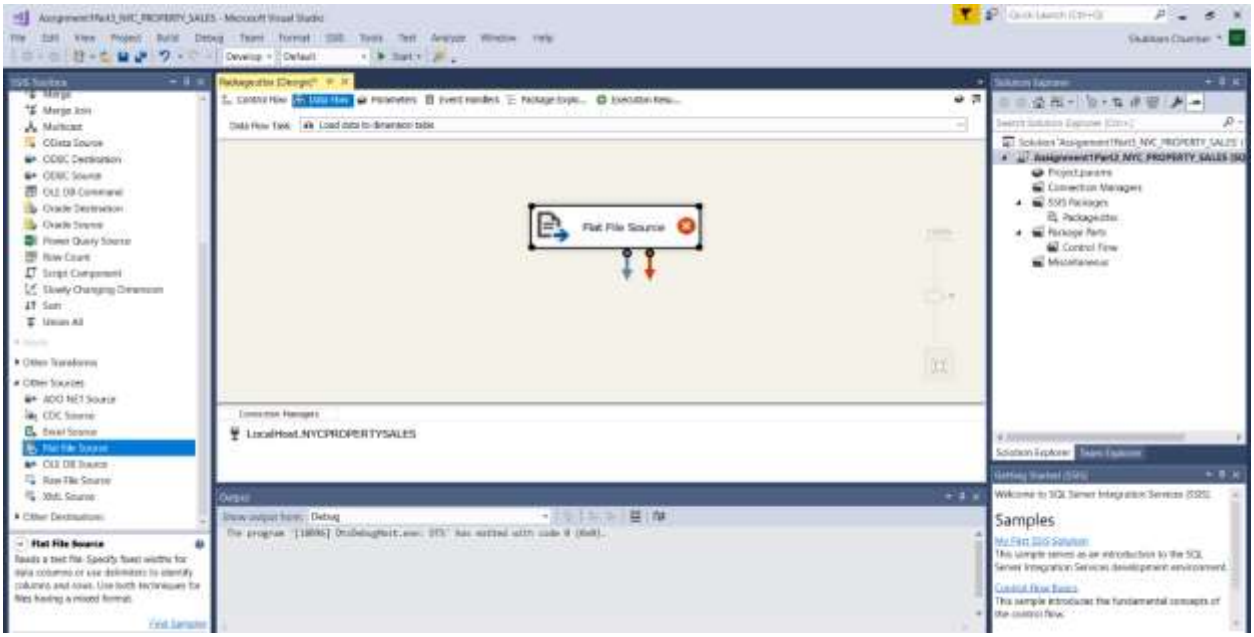

Fig 135:Created a flat file source to fetch data from dataset

Selecting the file from local storage.

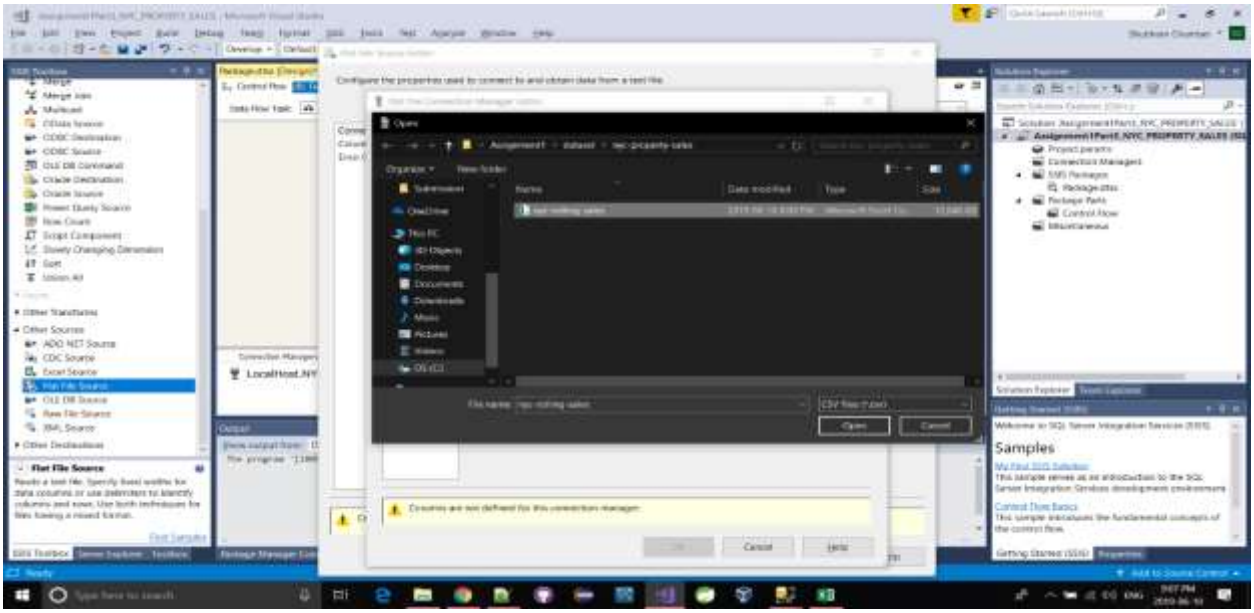

Fig 136:Uploading Dataset “NYC rolling sales”

Then we selected the columns from the dataset as per requirement in the initial stages. i.e

We have removed following columns for reasons such as:

1. Column 0 – just a serial number of records
2. BOROUGH – Not required as address is there already.
3. EASE-MENT – Null in most of records. Also, not useful.
4. BUILDING CLASS AT TIME OF SALE – Duplicate to another column
5. TAX CLASS AT TIME OF SALE – Duplicate to another column

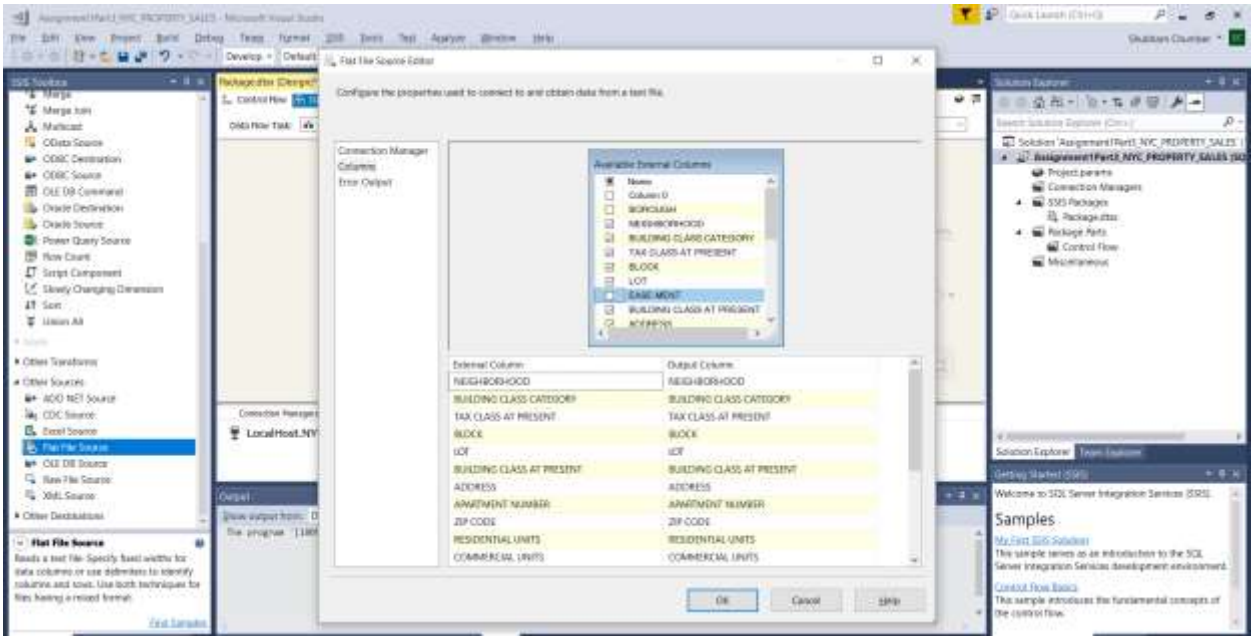

Fig 137:Filtering uploaded dataset

Then we changed the truncation of the column from ''Fail component" to "Redirect Now" so that the flow of data should not stop due to some records having error. By changing the truncation to redirect now all such records having error will be automatically transferred to Catch error space.

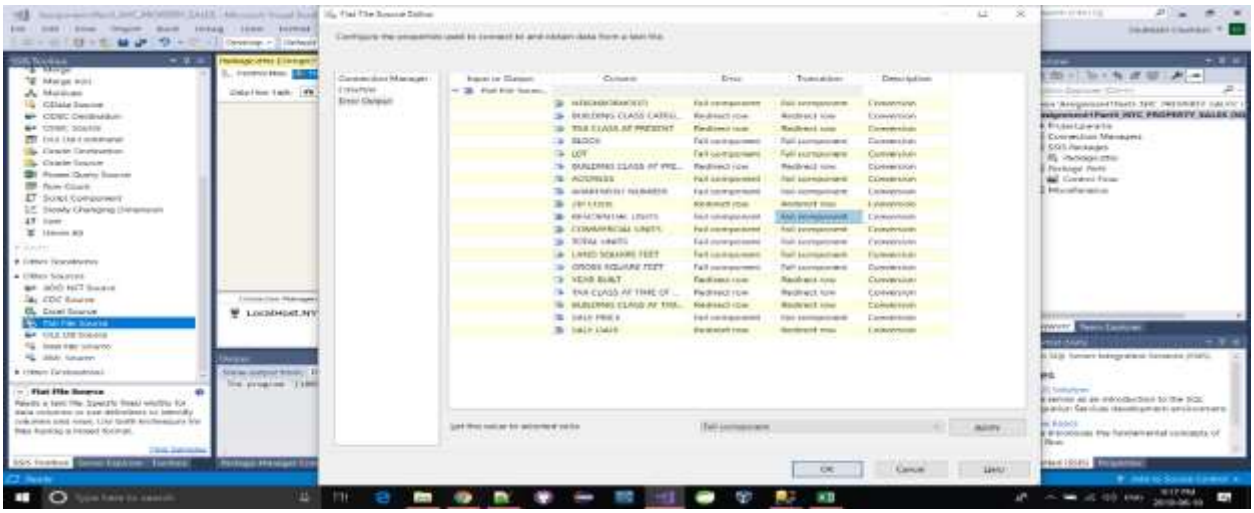

Fig 138:Changing the truncation of attributes

Now Flat file source is ready to use as shown in the below image

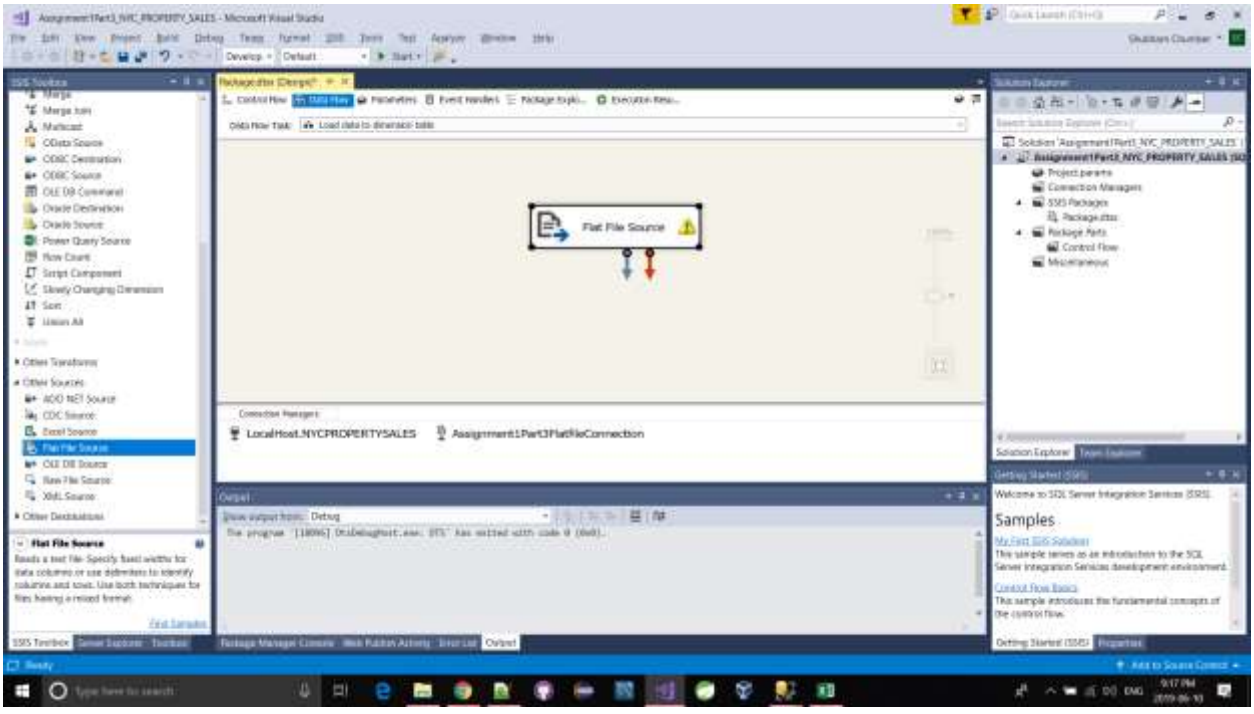

Fig 139:Successfully load the data from dataset

A multicast is added to where rows can be redirected if any error occurs. Also, renamed it as “Catch error” block as shown in the below image

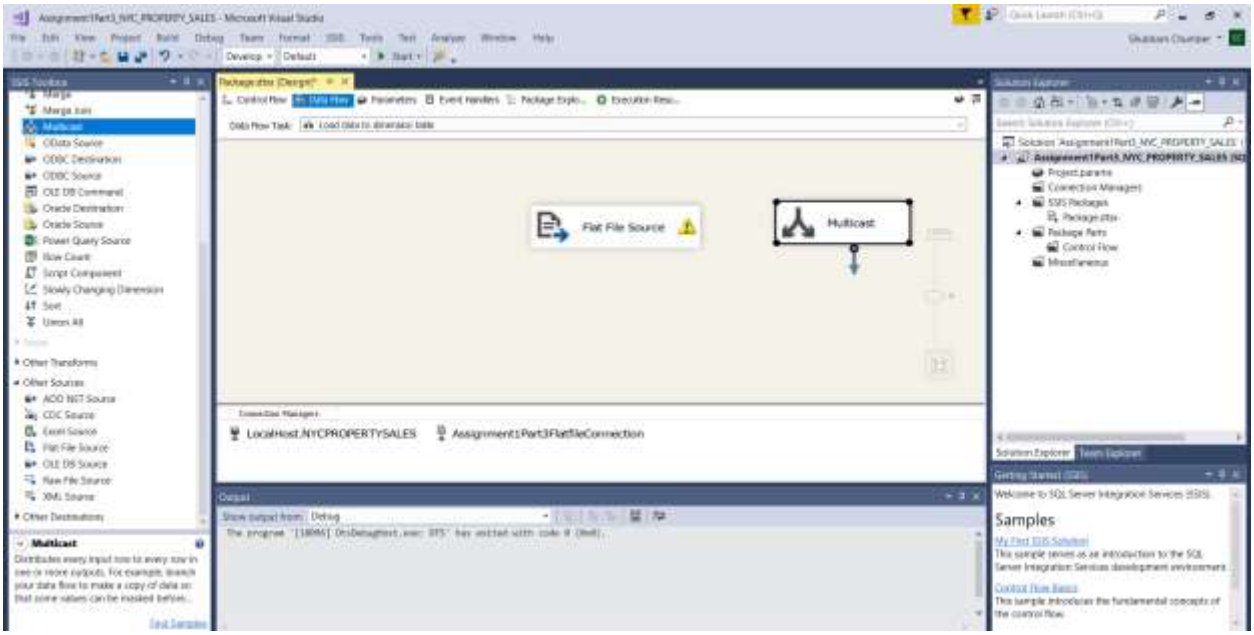

Fig 140:Adding new Multicast block

All the records with error being transferred to Catch error block as shown in the below image

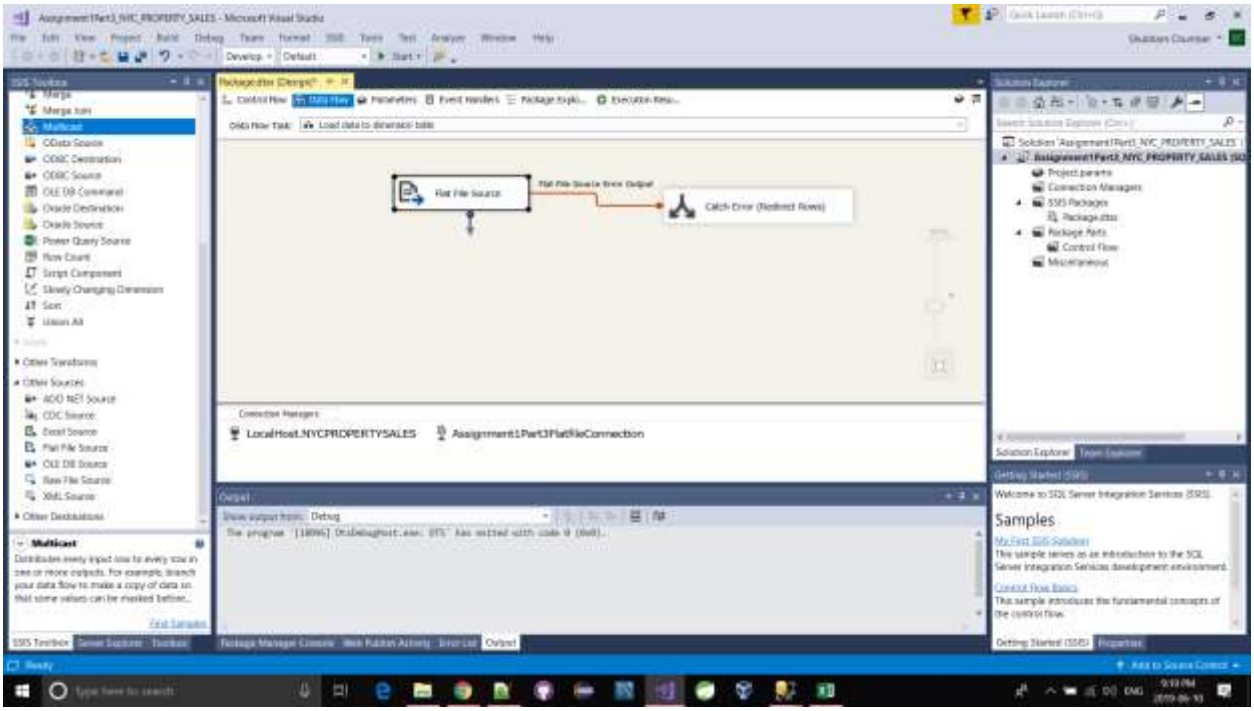

Fig 141:Assigning a task to multicast of catching error

Then we have added a multicast to create and load data into Dimension table as shown in the below image

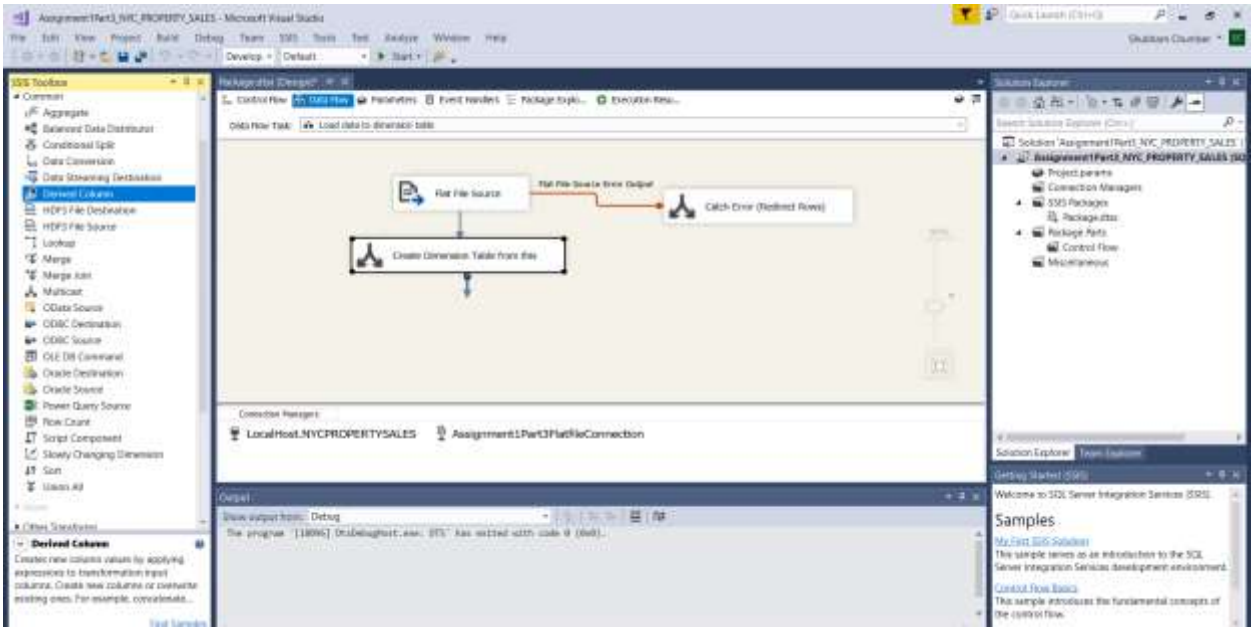

Fig 142:Adding a new multitask block to load dimension tables

Then we have derived three columns i.e Sales_month, Sales_day, Sales_year from original Sales Date column to form a dimension table as shown in the below image

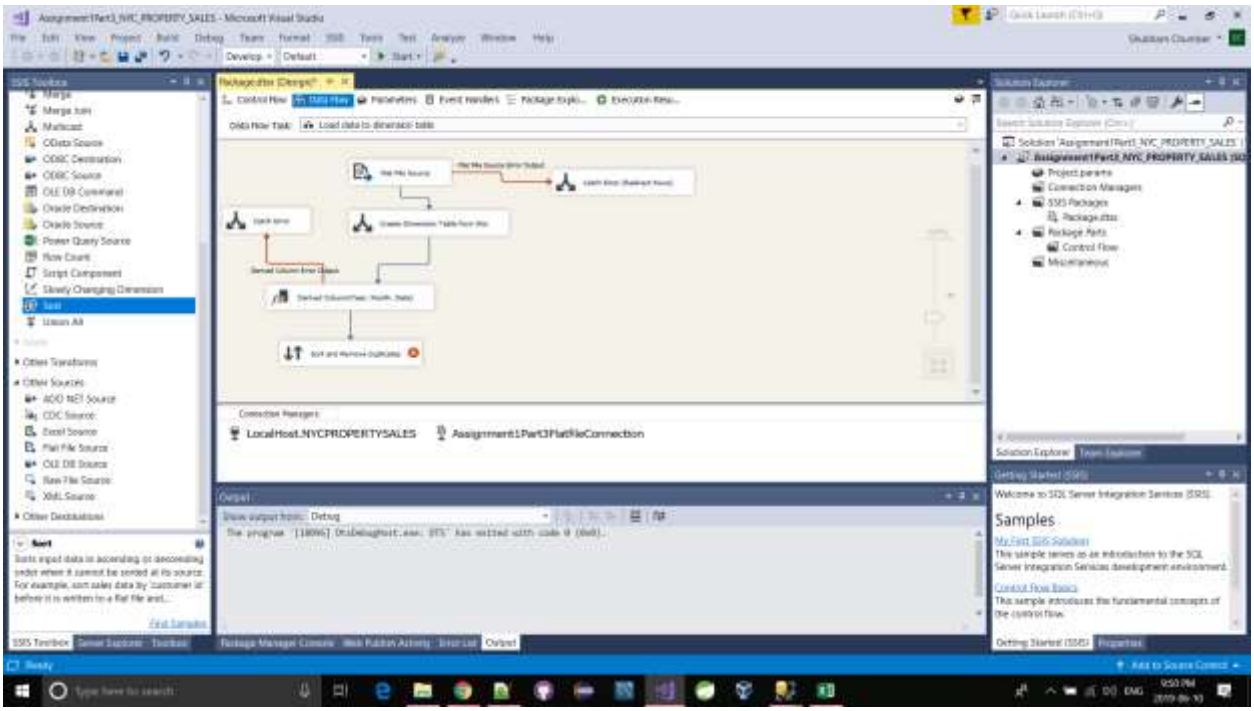

Fig 143:Deriving attributes from the parent attribute

Sorted the rows and remove duplicates while the records with error got saved in the Catch error file as shown in the below image. Also, records reduced to 364 in number after sorting and removing duplicates.

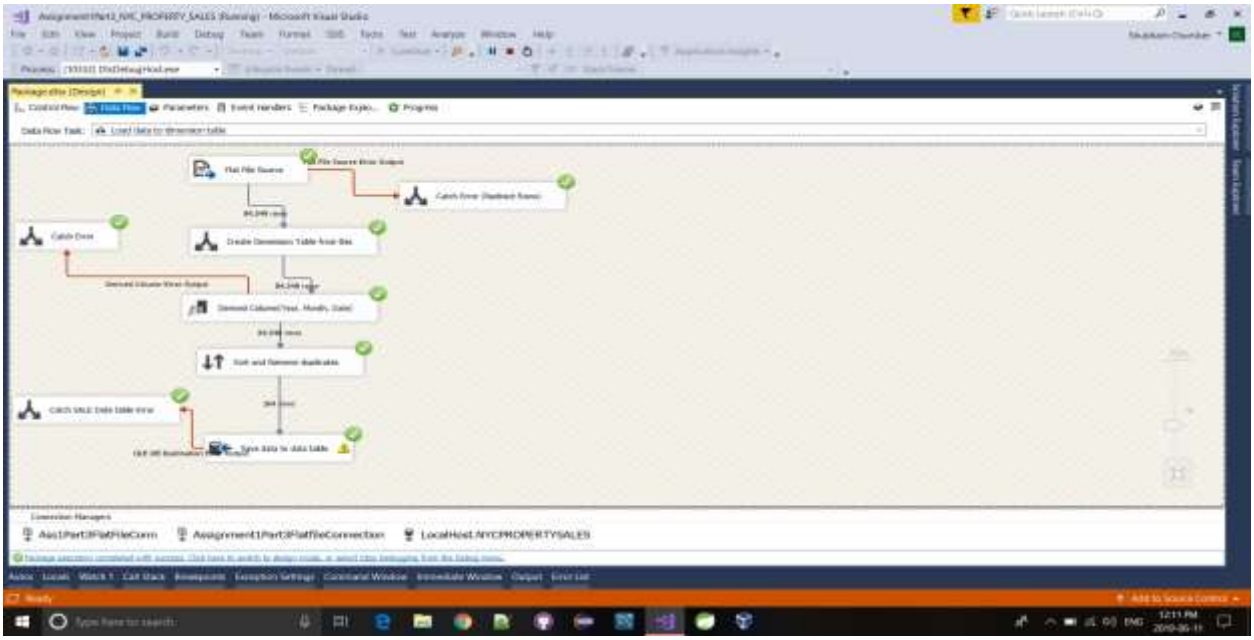

Fig 144:Sorting, removing duplicates and catching records having error

Afterwards, we have loaded data in rest of the dimension tables as shown in the below images by following a similar approach.

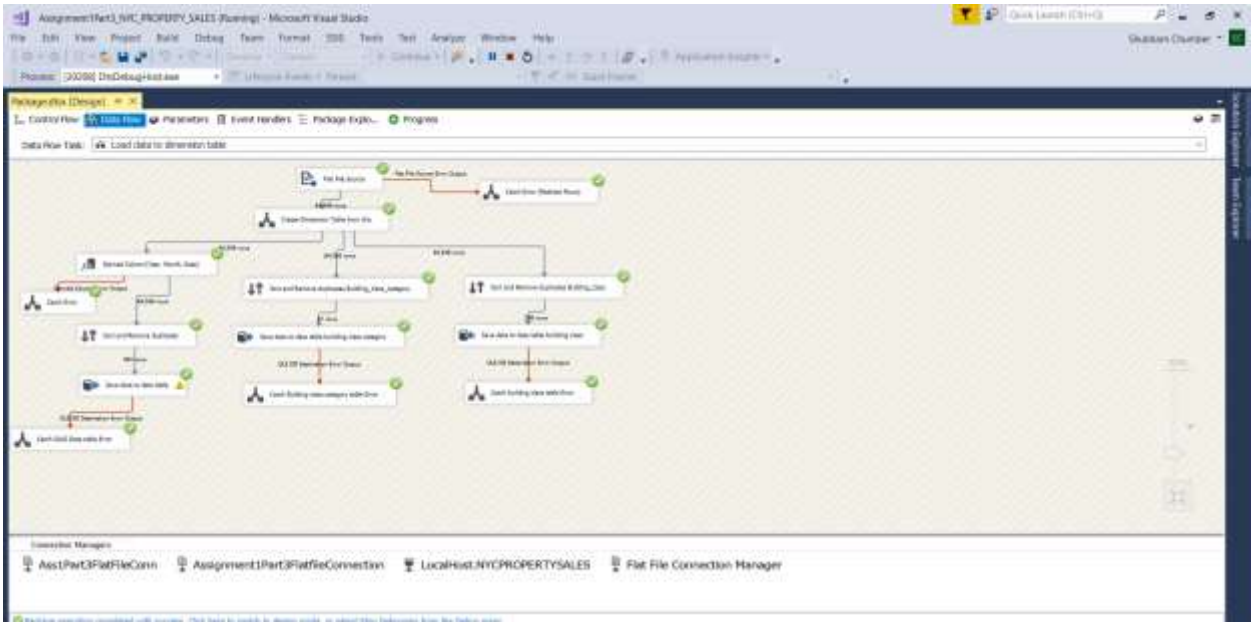

Fig 145:Successful added three dimension tables

Complete Load dimension table data flow task of our ETL pipeline. Changed position of components to fit them for a clearer picture.

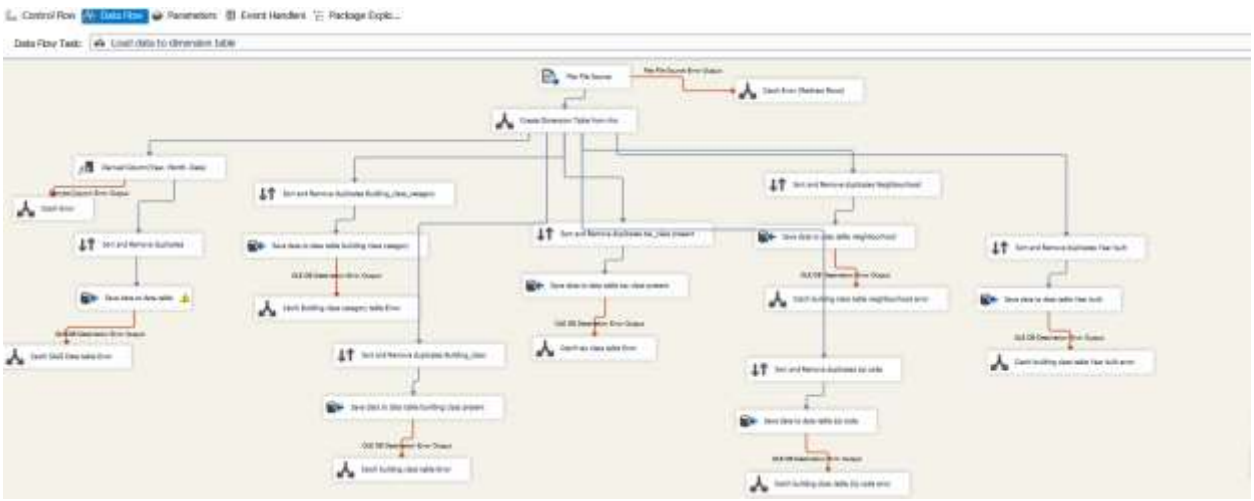

Fig 146:Successfully loaded data into all Dimension tables

Working - data flow task “LOAD DATA INTO DIMENSION TABLE”

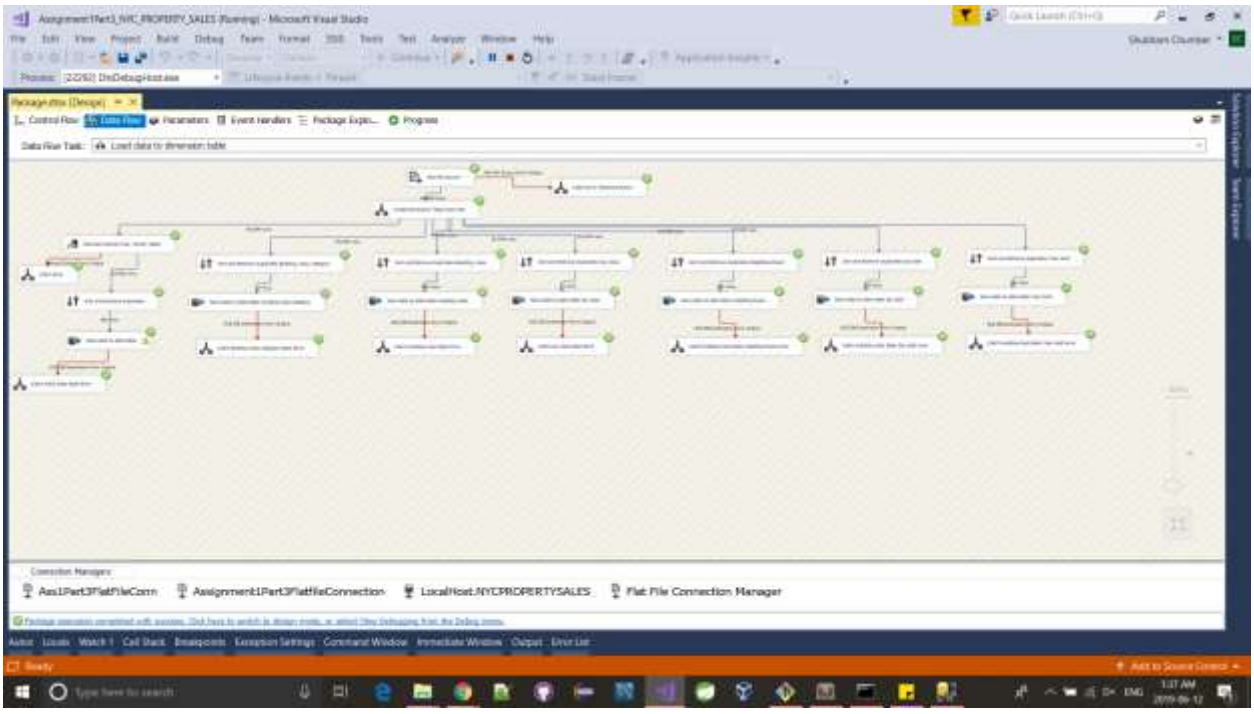

Fig 147:Successfully loaded data into all Dimension tables

3. Load new Data into fact table:

Now on to our next major step i.e Loading data into Fact table.
Added Flat file Source and Lookup component to look for ID of dim. tables we created earlier.

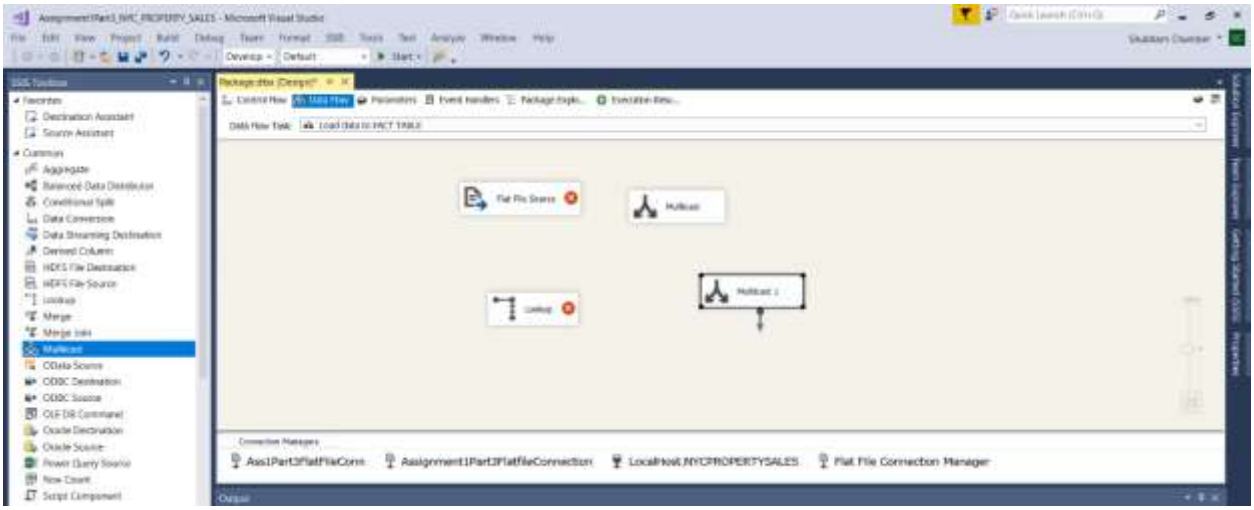

Fig 148:Adding source and multitask block to load data into Fact table

Setting up the Flat file Source and its mappings

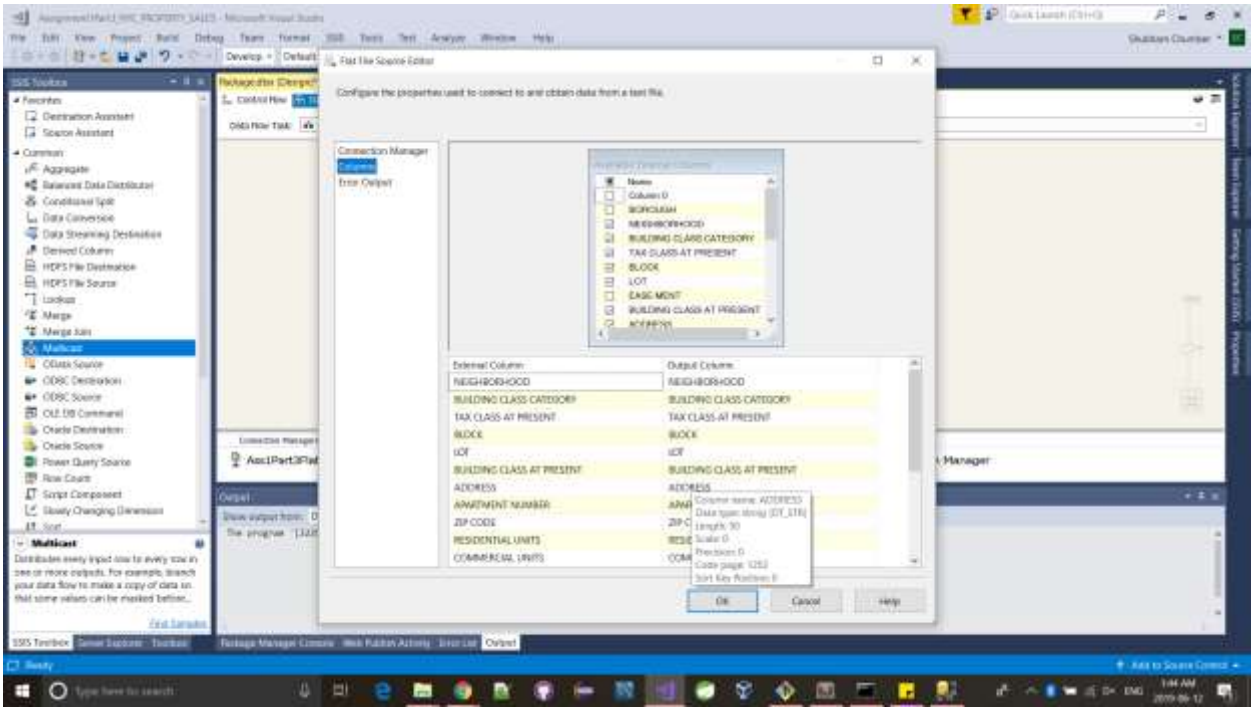

Fig 149: Selecting the required columns

After setting up the Flat file soure and redirecting the errors to multicast. Let's start with Sale date id LOOKUP

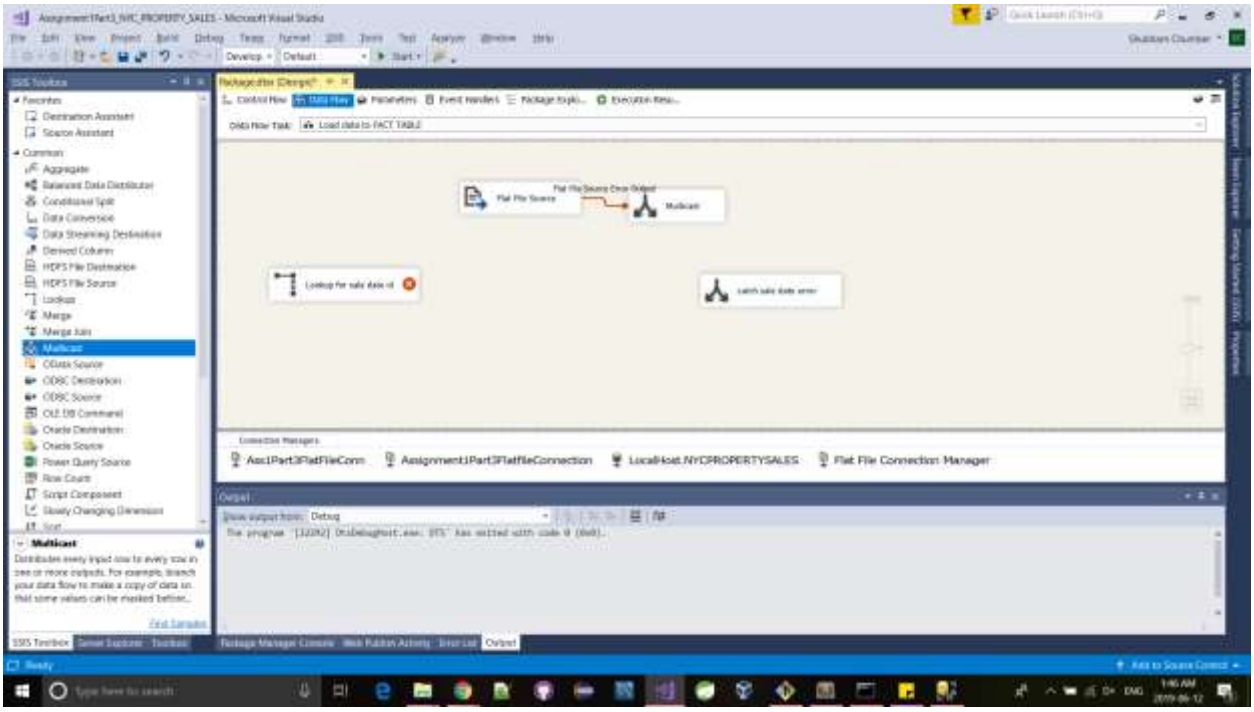

Fig 150:Redirecting the errors into catch error block

Setting up the Lookup for sale_date_id by mapping the component to sale_date table

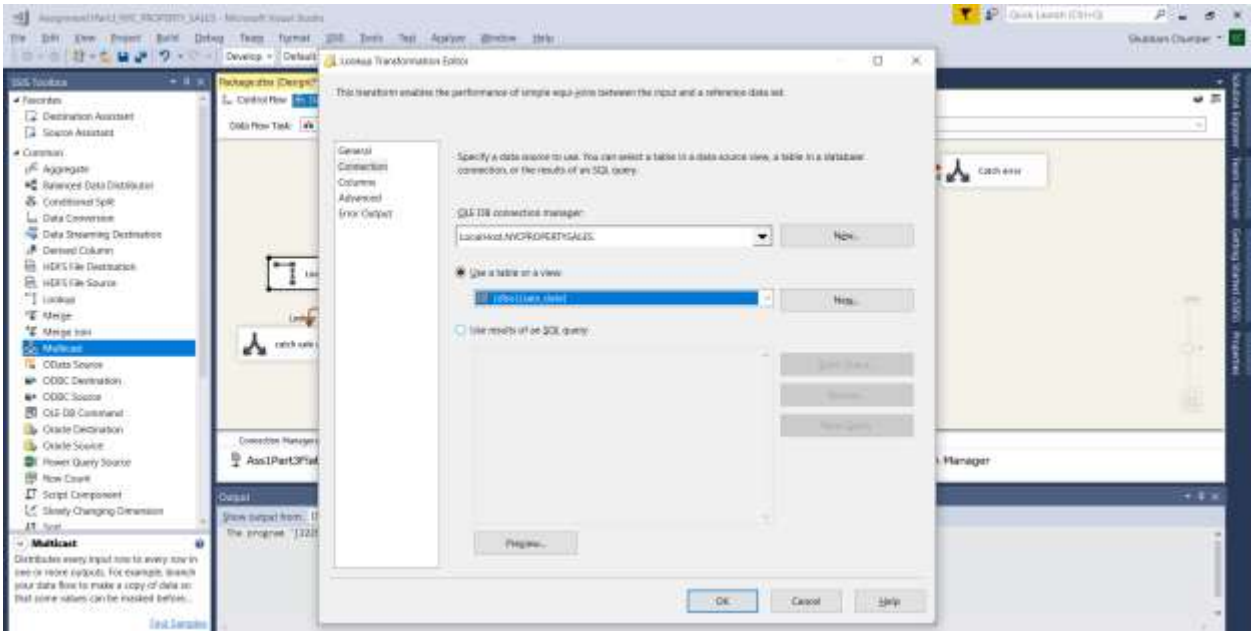

Fig 151:Lookup transformation editor

Mapping the Input SALE DATE column to sale_date of our sale_date dimension table to keep the original values of dates as well.

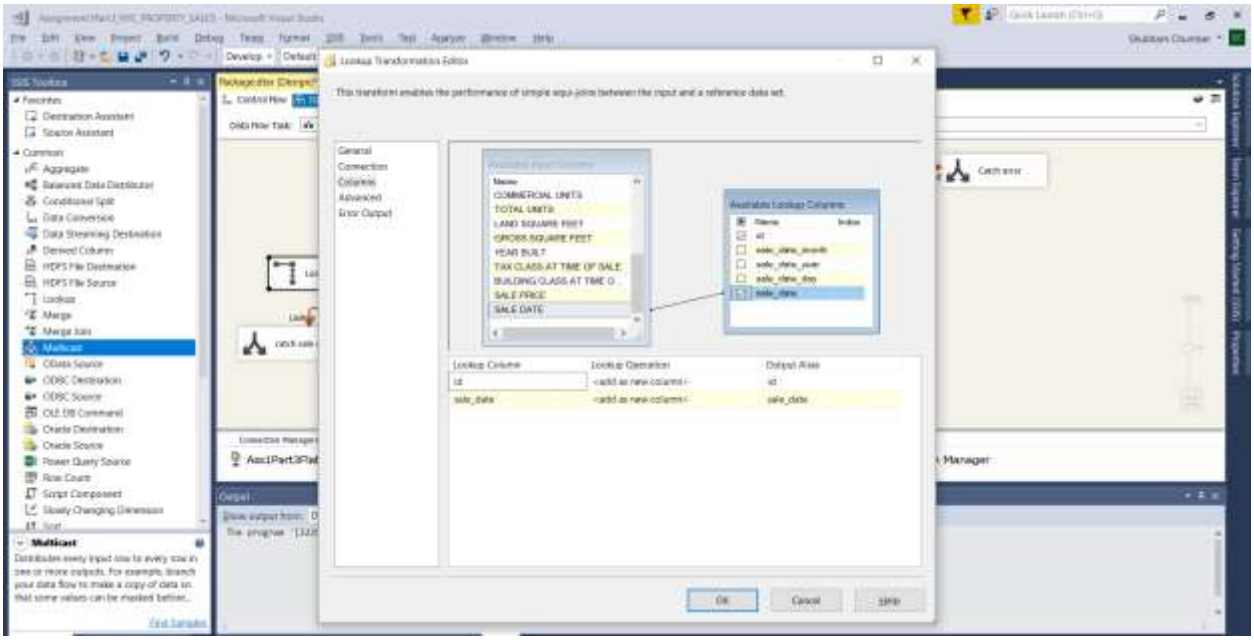

Fig 152:Mapping Dimension table attribute

Similarly, following the same approach. We have added lookup for other dimension tables primary keys.

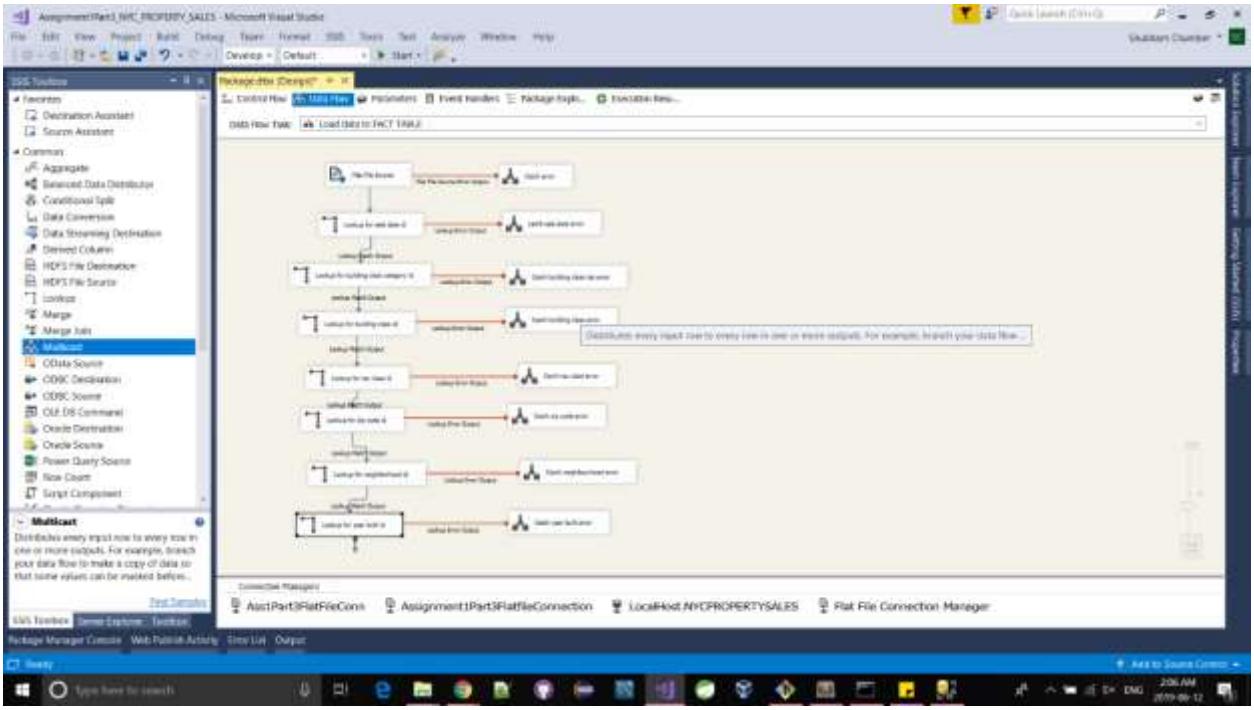

Fig 153:Lookups for Dimension tables

To store the data into fact table, we have added an OLE DB Destination component.

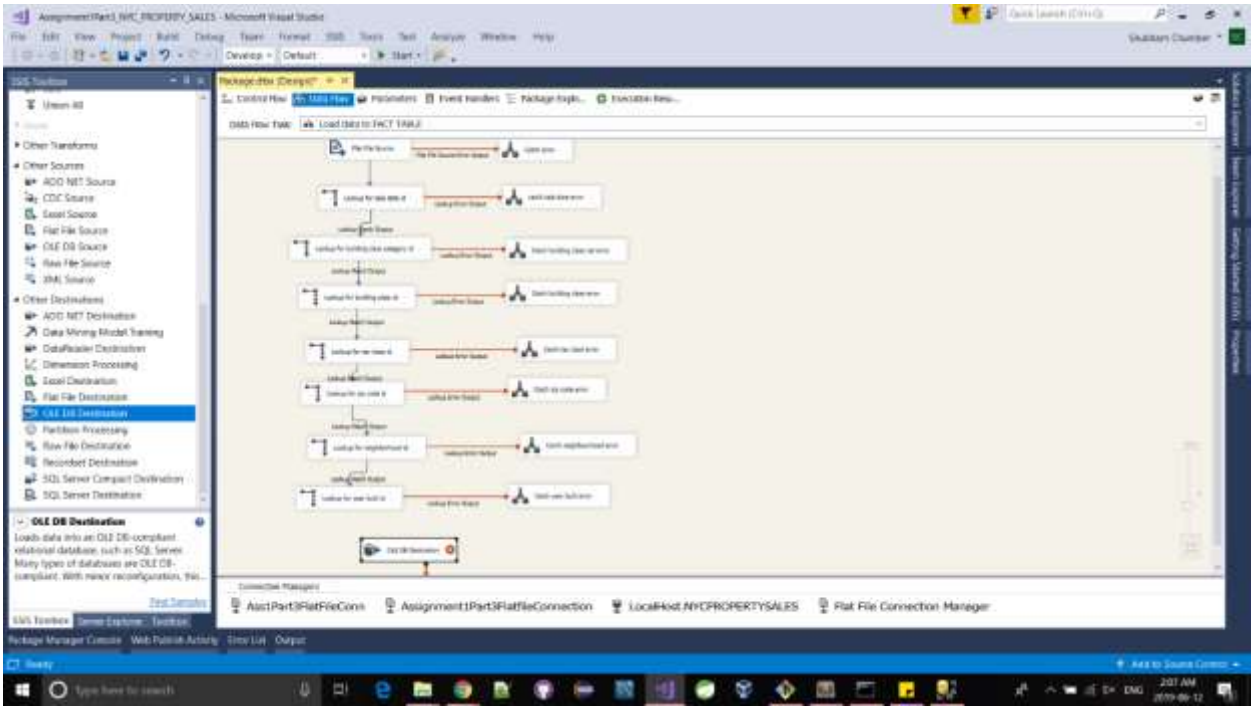

Fig 154:Successfully added OLE DB Destination component

Setting up the OLE DB Destination component connection as Fact table Properties.

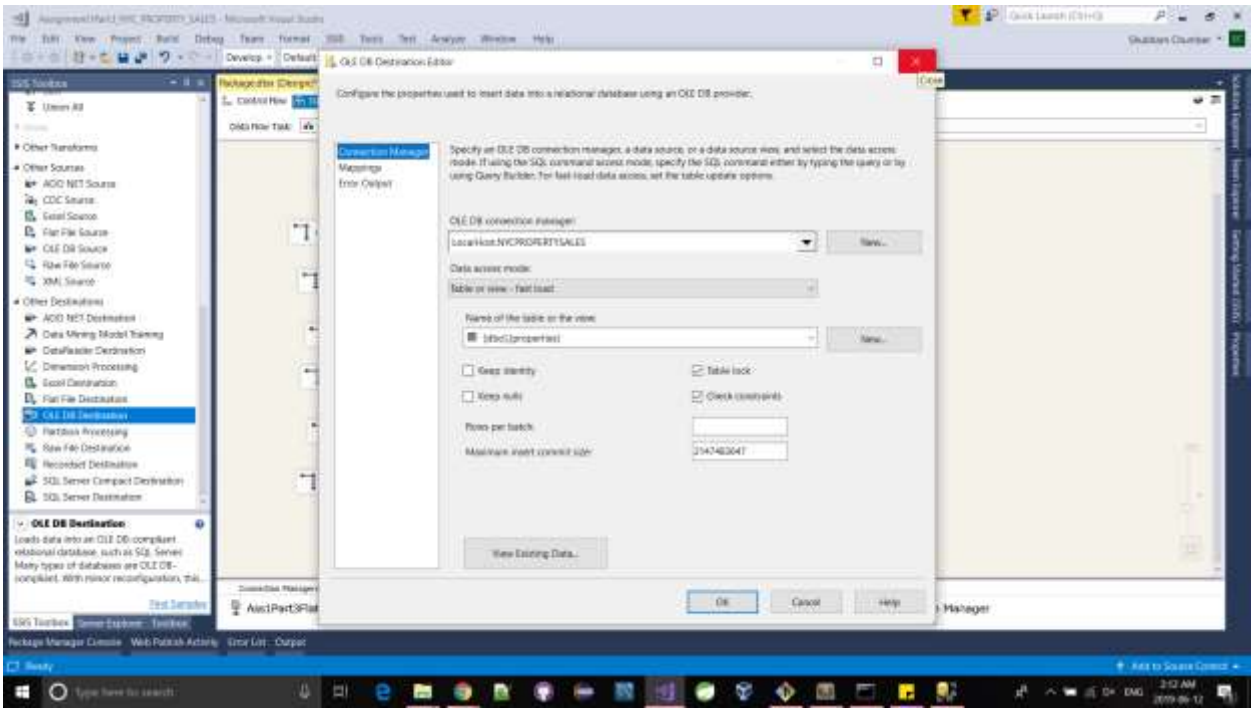

Fig 155:OLE DB connection

Setting up the OLE DB Destination component to map the required fields as per requirement.

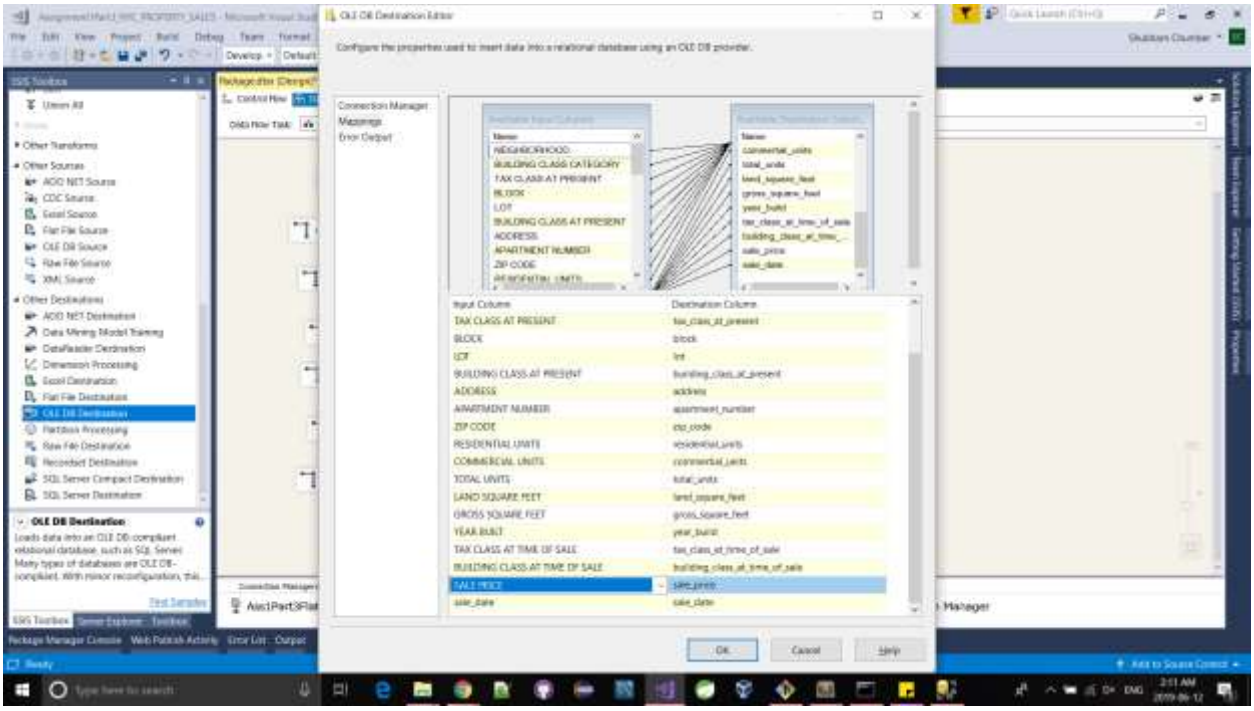

Fig 156:Mapping the OBE DB component with required fields

Complete data flow task of 'LOAD DATA INTO FACT TABLES'

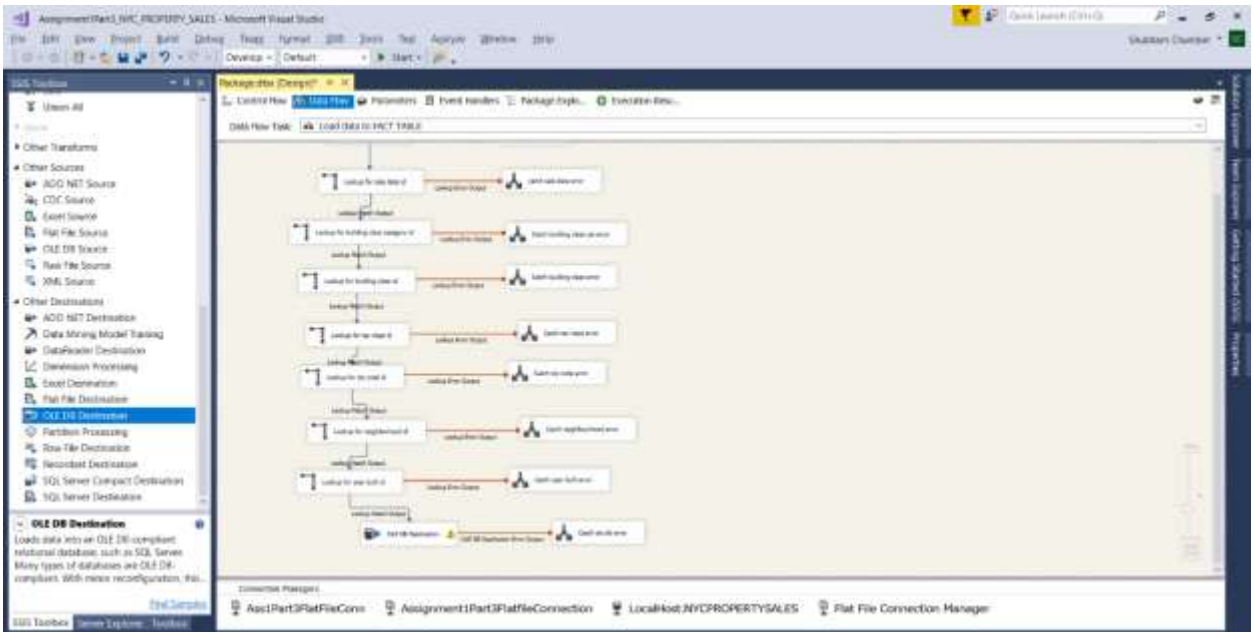

Fig 157:Successfully loaded data into all Fact table

Working – Load data into fact table. However, as you can see all 84,548 records are redirected to multicast component (Catch ole db error), which is not right. Let's finish the pipeline first and come back to solve this problem.

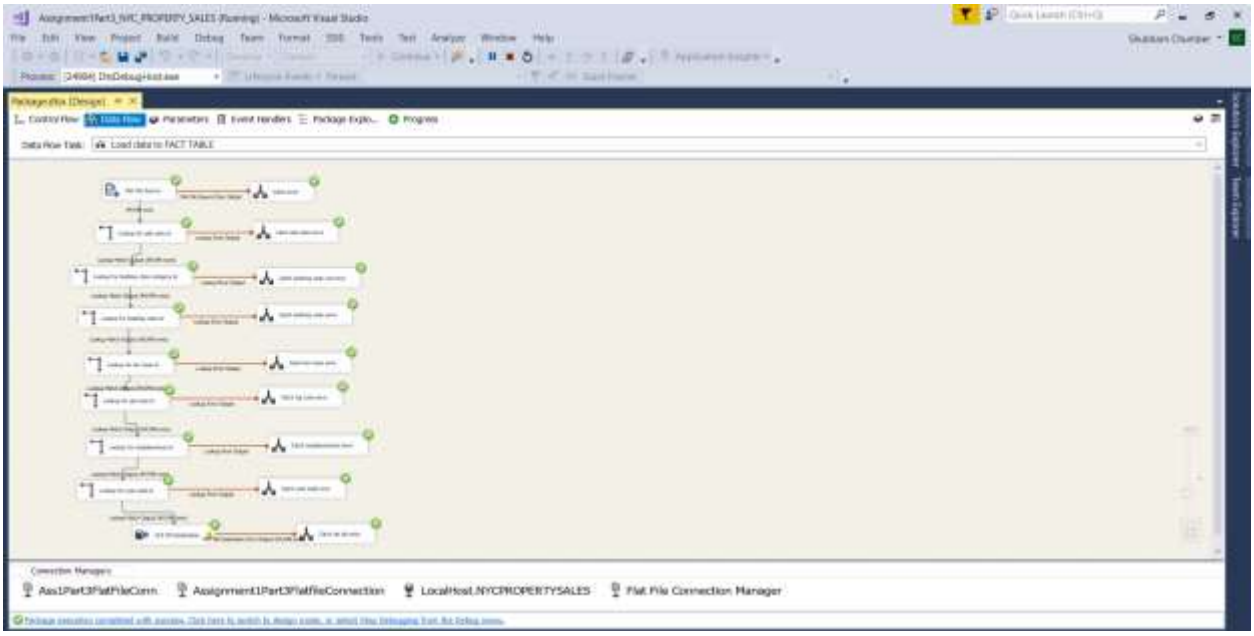

Fig 158:observing the error records transferred to the Catch error block

4. Save data to CSV file:

Added an OLE DB Source because we must export the data that we just loaded in database in previous tasks to csv file now.

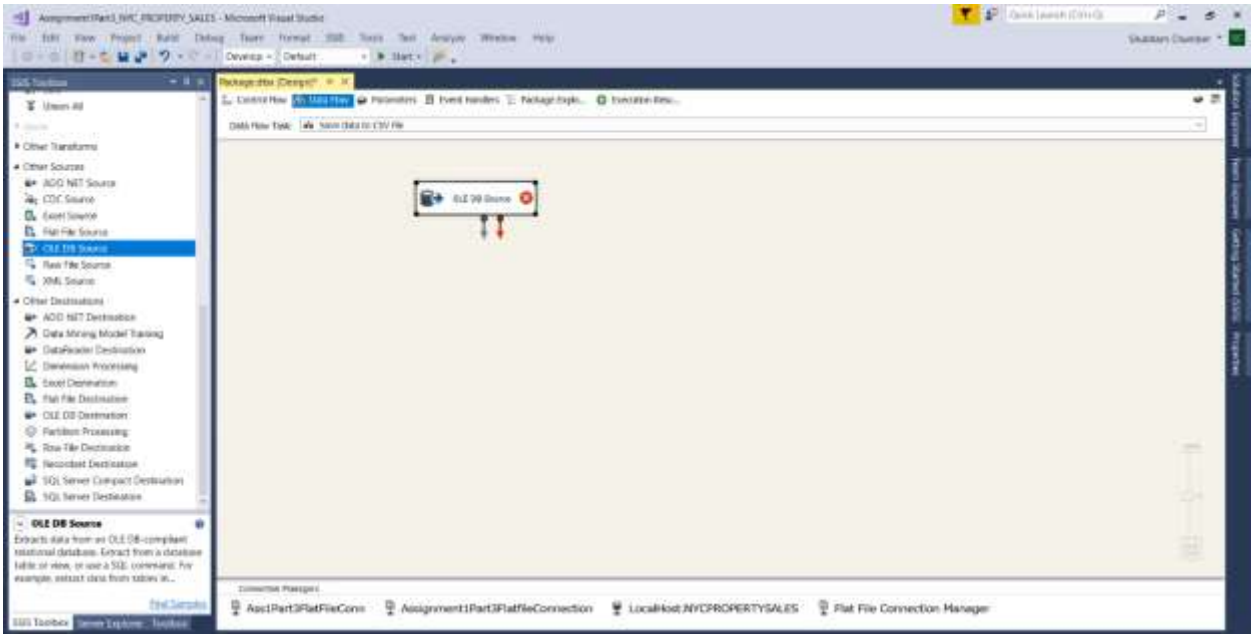

Fig 159:Successfully added OLE DB source block

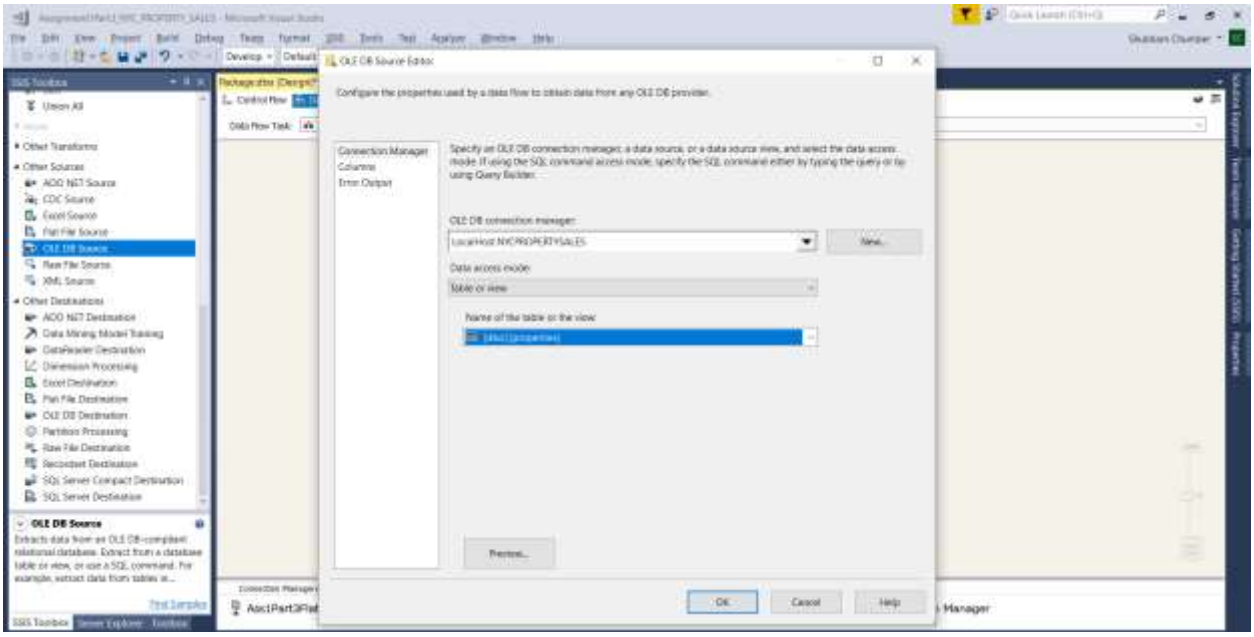

Fig 160:OLE DB connection manager

Setting up the DB source component. Selected the Fact table Properties and mapped its columns to output columns.

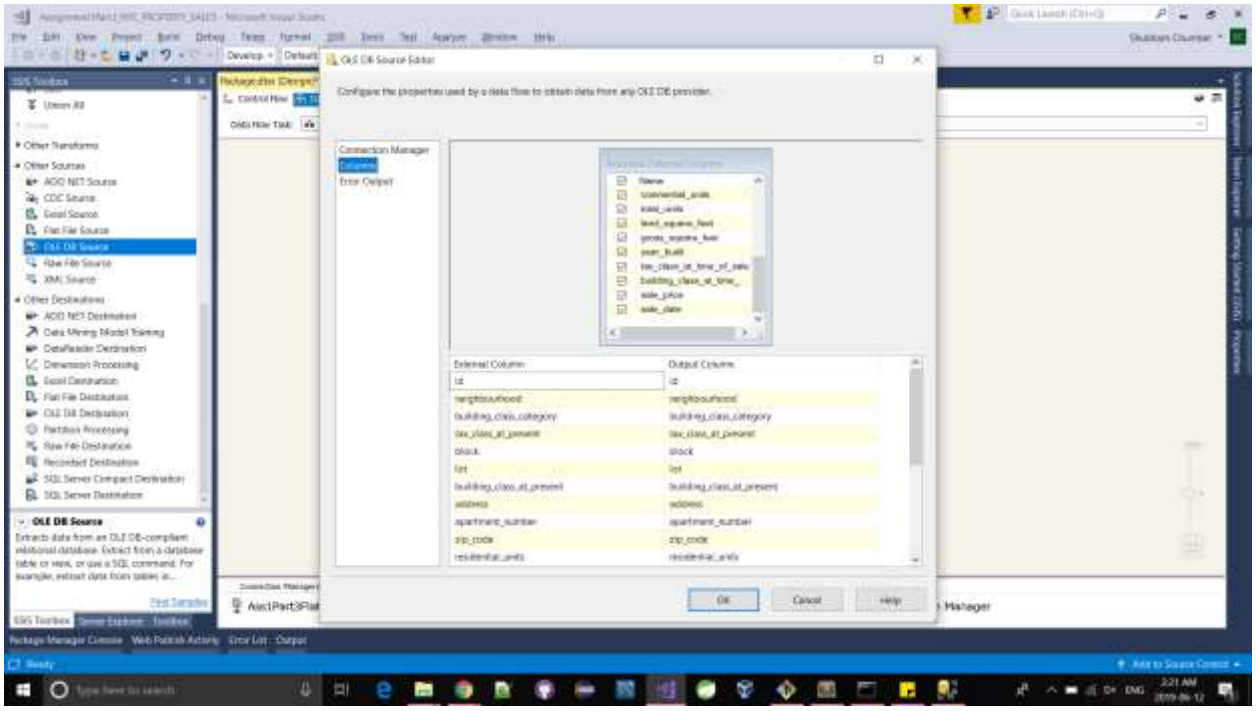

Fig 161:Setting DB source component

Setting up the DB source component. Redirecting the errors.

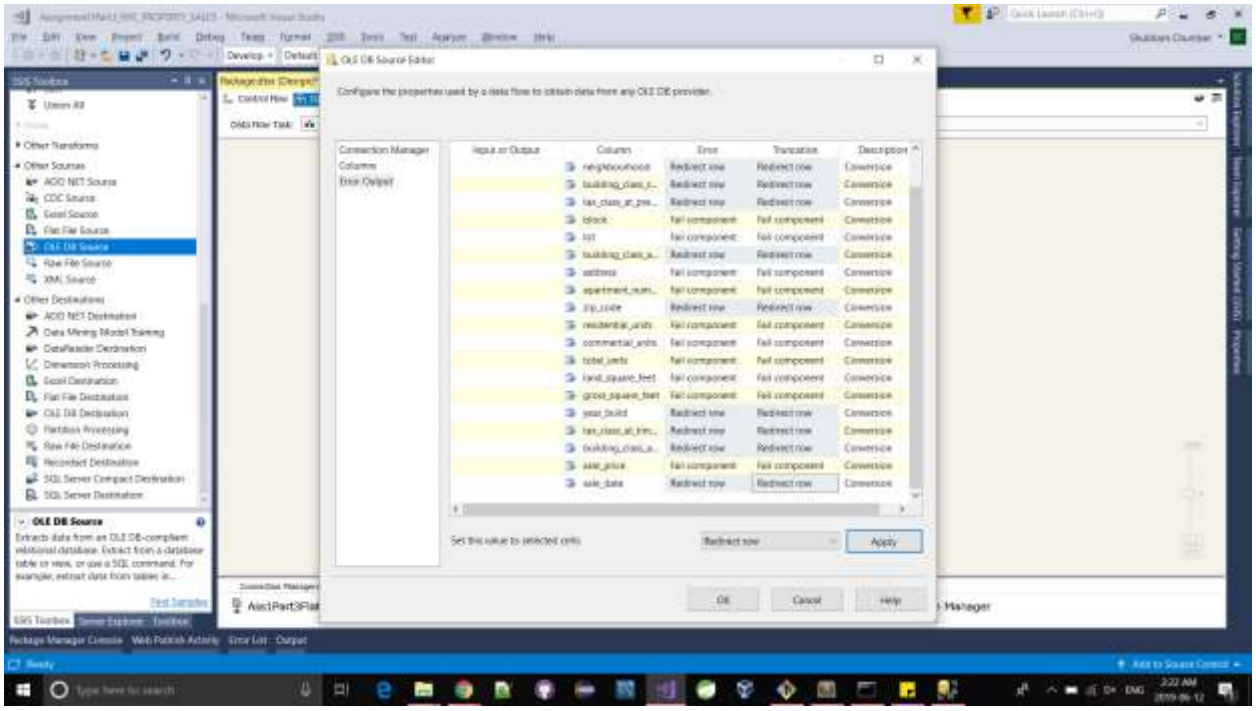

Fig 162:Changing the status of the truncation

Added the lookup the get the values from the columns corresponding to dimension table primary key which is maintained in Database using Foreign key relationship

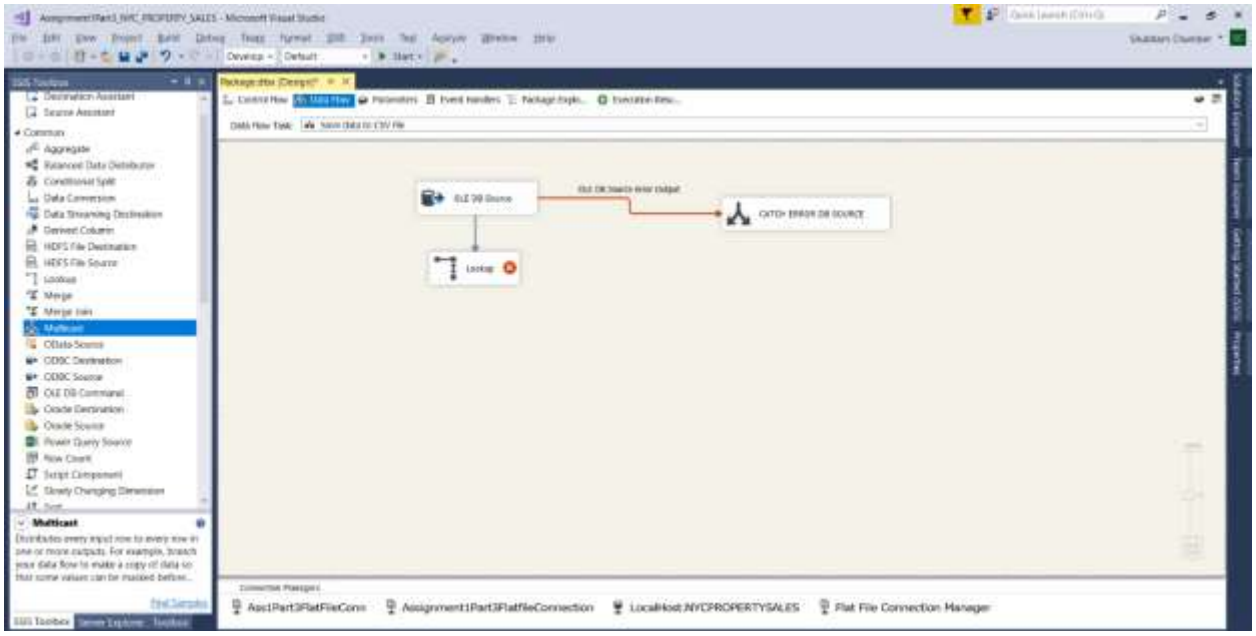

Fig 163:Adding Lookup and Catch error block

Setting up the Sale date lookup

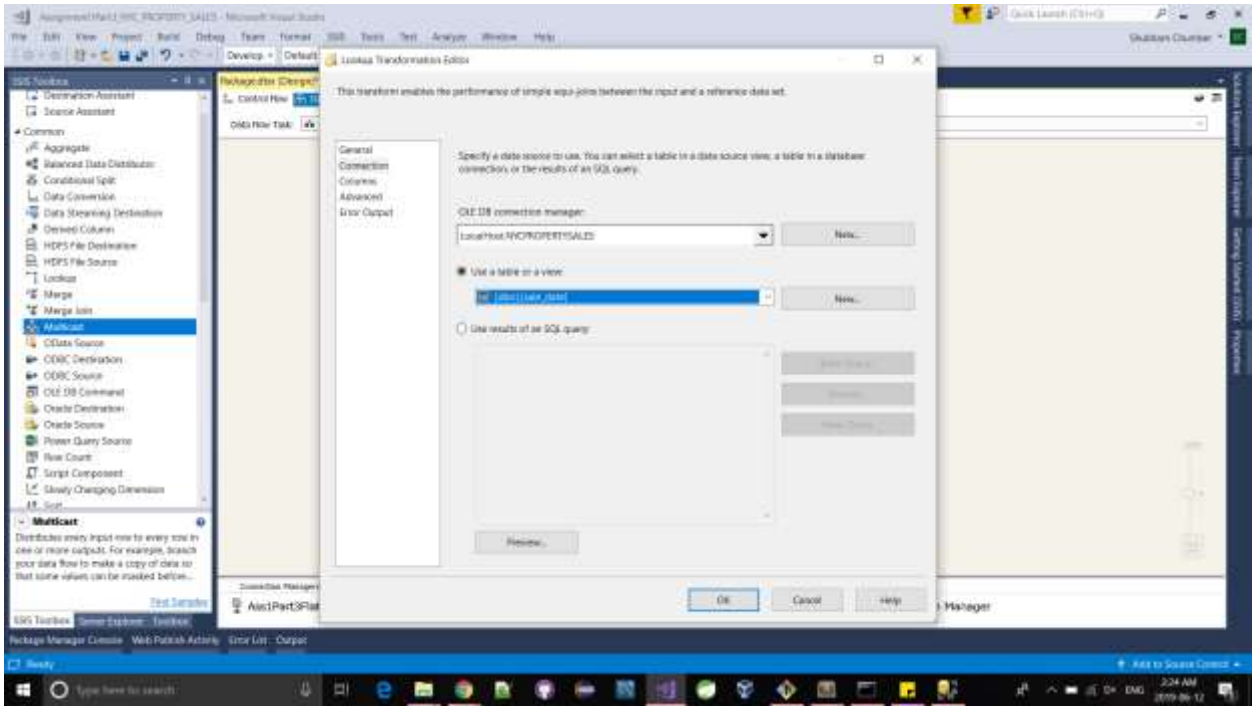

Fig 164:Sales Date Lookup

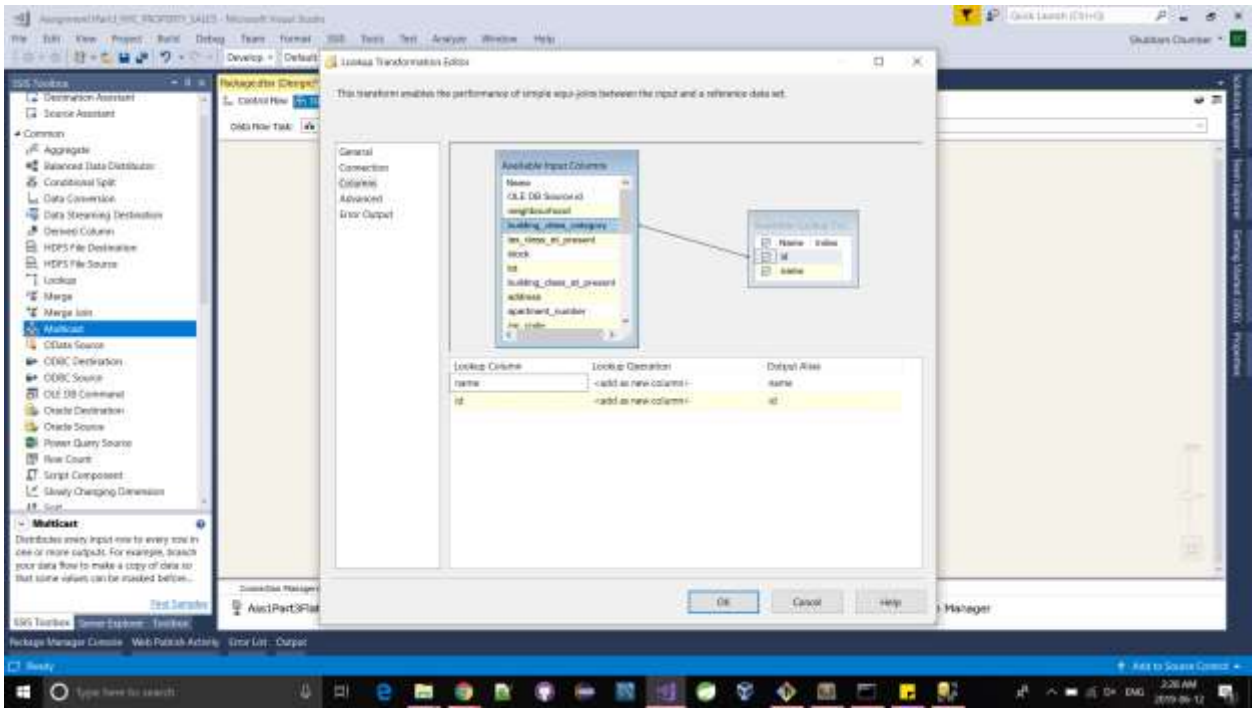

Fig 165:Mapping the Building class category similarly

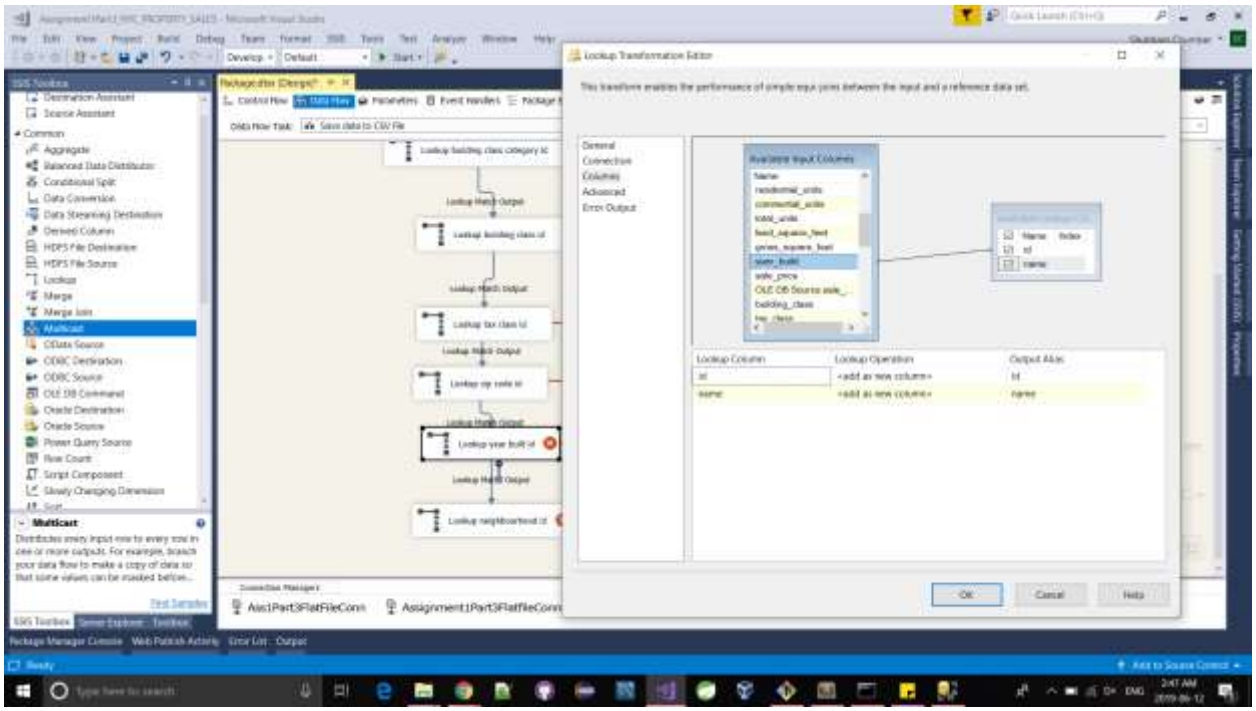

Fig 166:Mapping the Year built similarly

Added all the dimension table columns following same approach

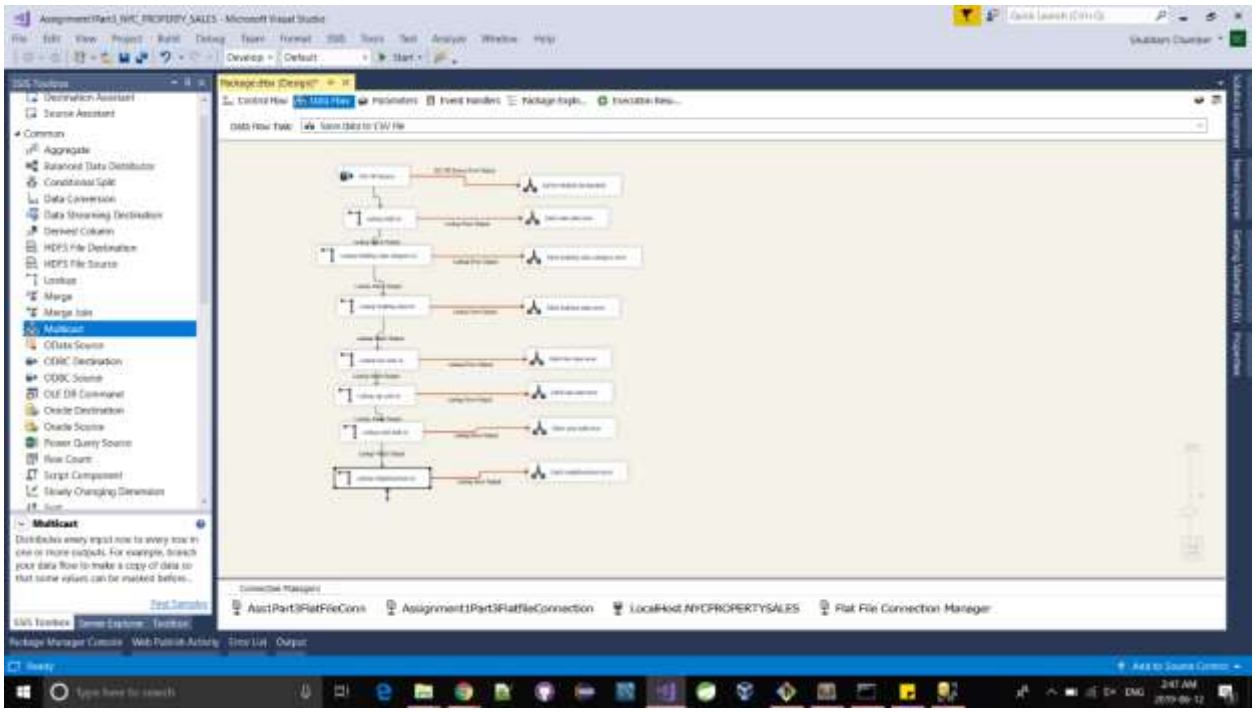

Fig 167:Adding Dimension tables

Added the Flat file destination component to direct the output to CSV file.

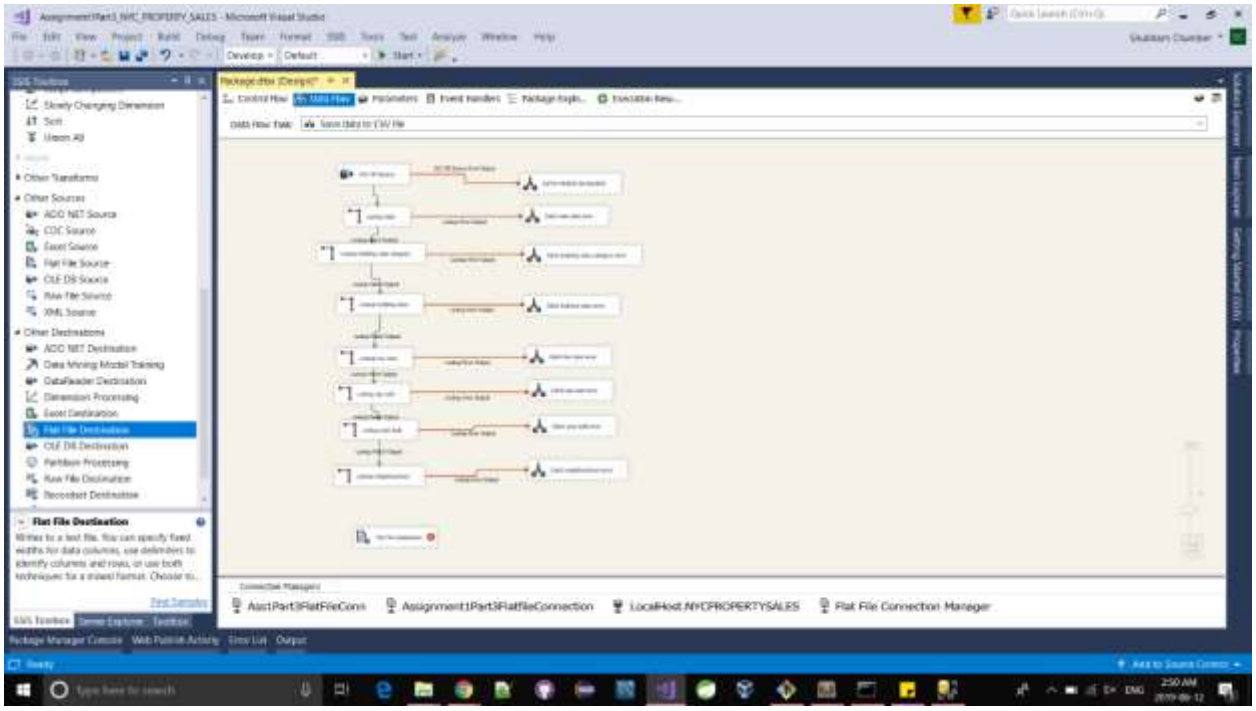

Fig 168:Destination component to csv file

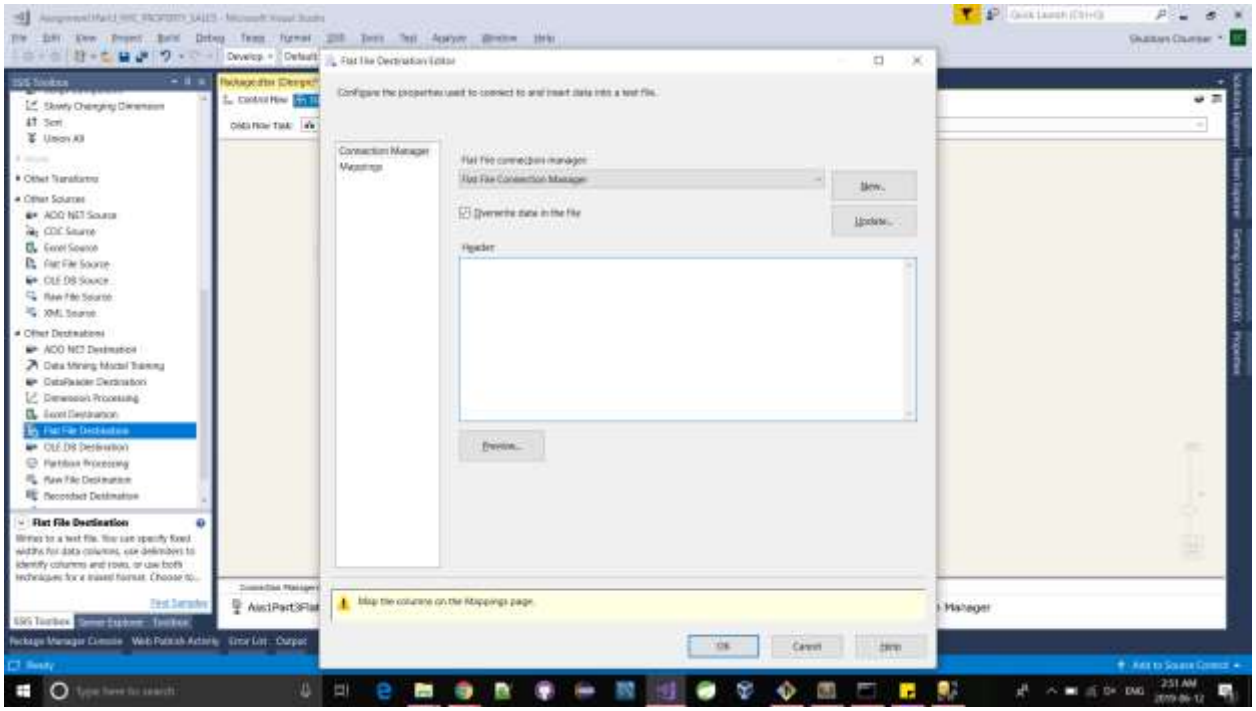

Fig 169;Setting up the Flat file component.

Working – Data flow of task ‘Save to CSV File’

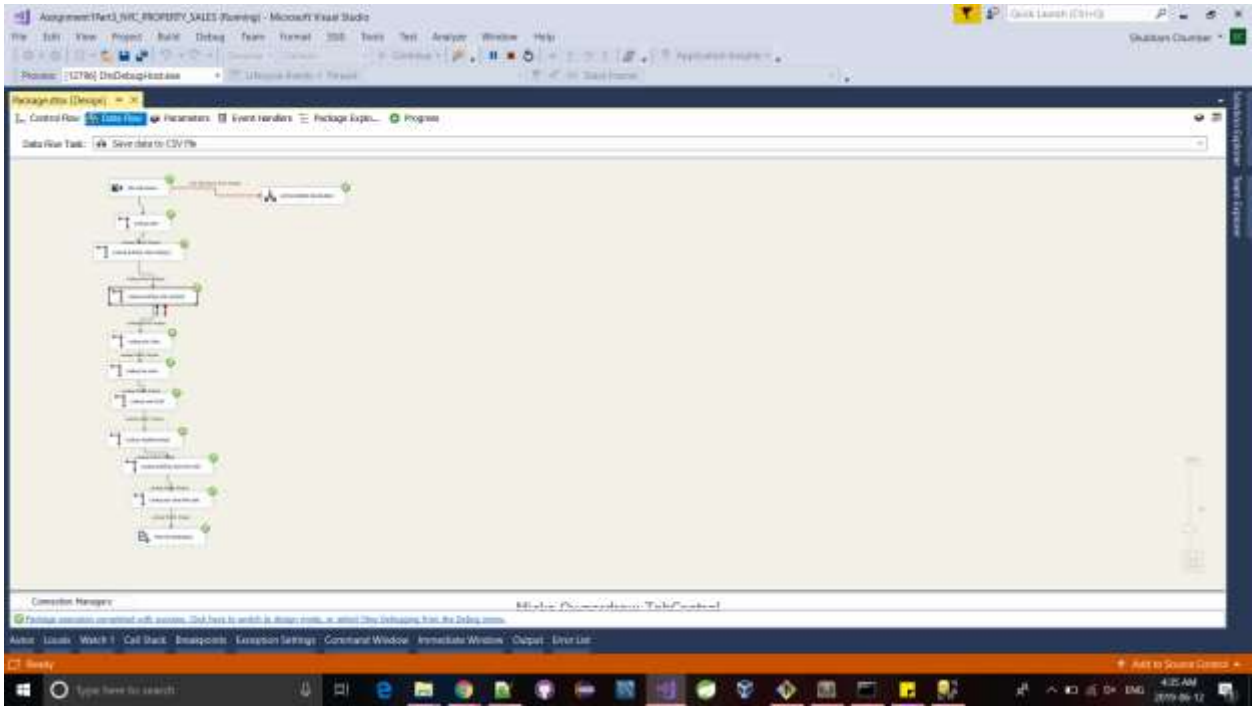

Fig 170: Working Data flow

Complete Working ETL pipeline of NYCPROPERTYSALES dataset

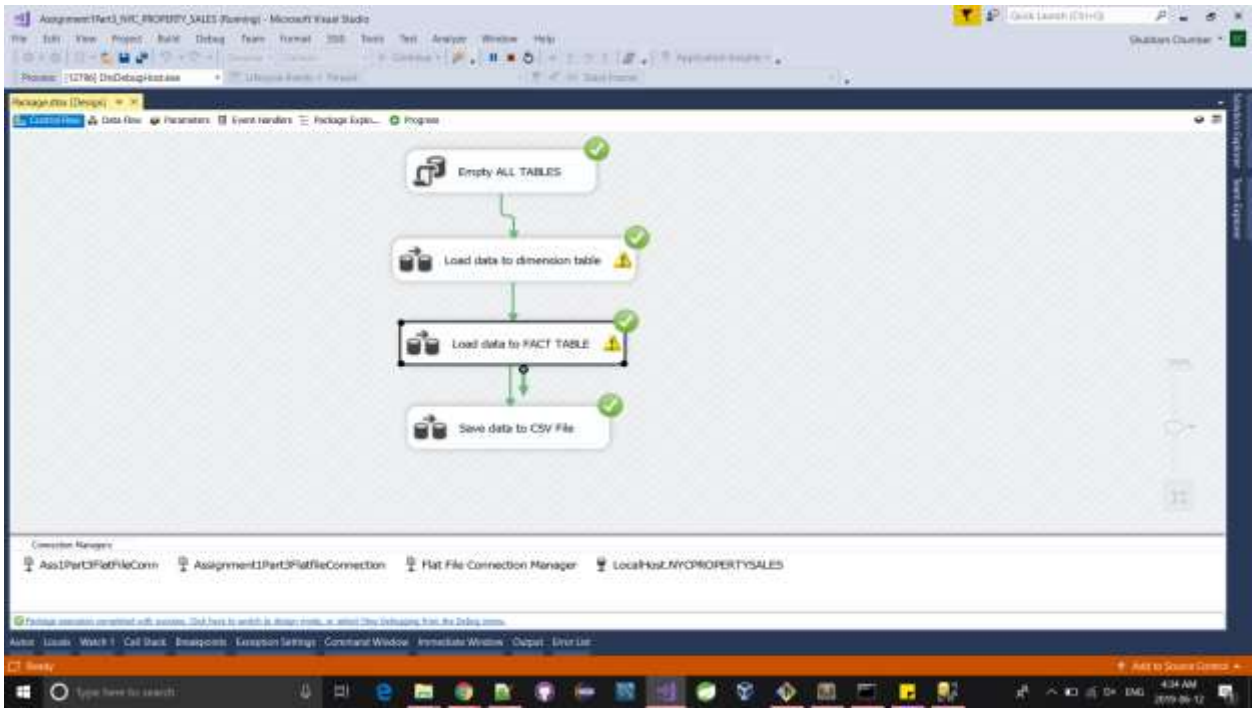

Fig 171:Status of complete ETL Pipeline process

D. Testing of the ETL pipeline:

So, we have successfully completed the ETL pipeline of given dataset NYCPROPERTYSALES. But, did it work? Let's find out!

Let's look at NYCPROPERTYSALES database and check if the data was loaded correctly or not.

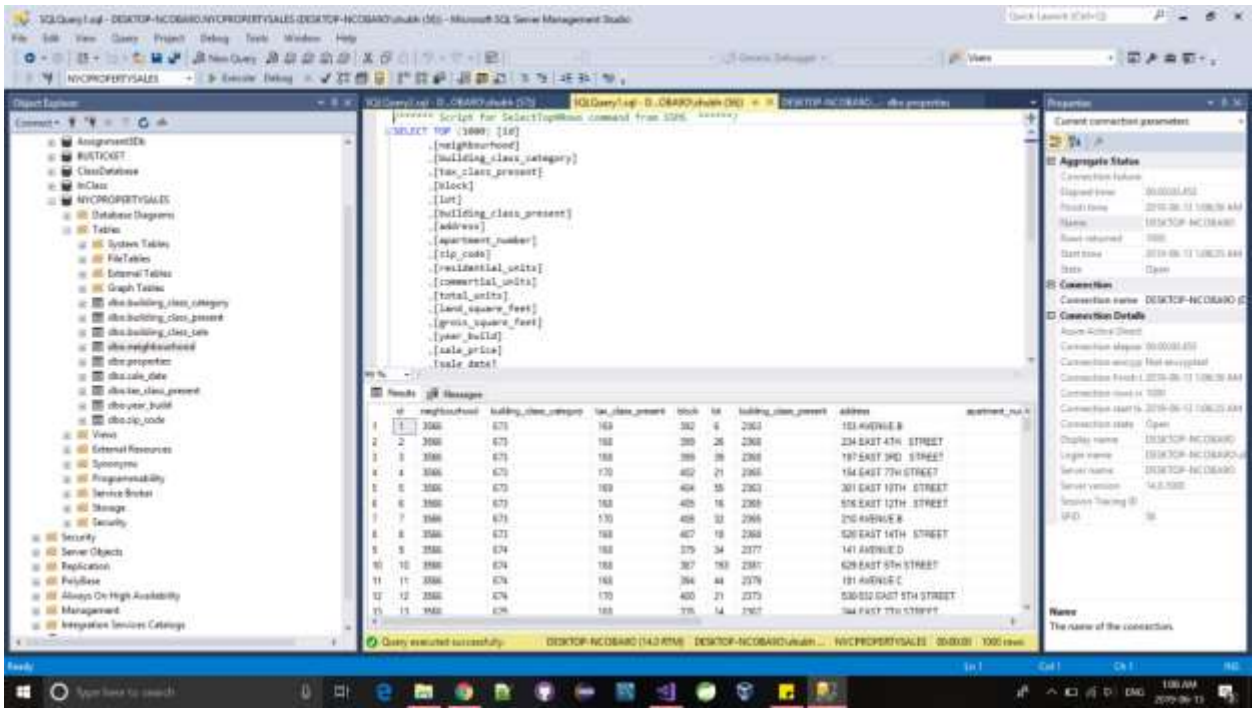

Fig 172:Here, this is a select query of top 1000 rows in 'Properties' Fact table.

PROPERTIES fact table is filled with meaningful data as you can see in Fig. i.e primary key of dimension tables in columns like Neighborhood, building_class_category, tax_class_present, building_class_present, sale_date, zip_code and year_build. Hence, which means the ETL pipeline was successfully executed. Let's further check our Dimension tables for further confirmation and validation.

Dimension Tables:

➔ Neighborhood

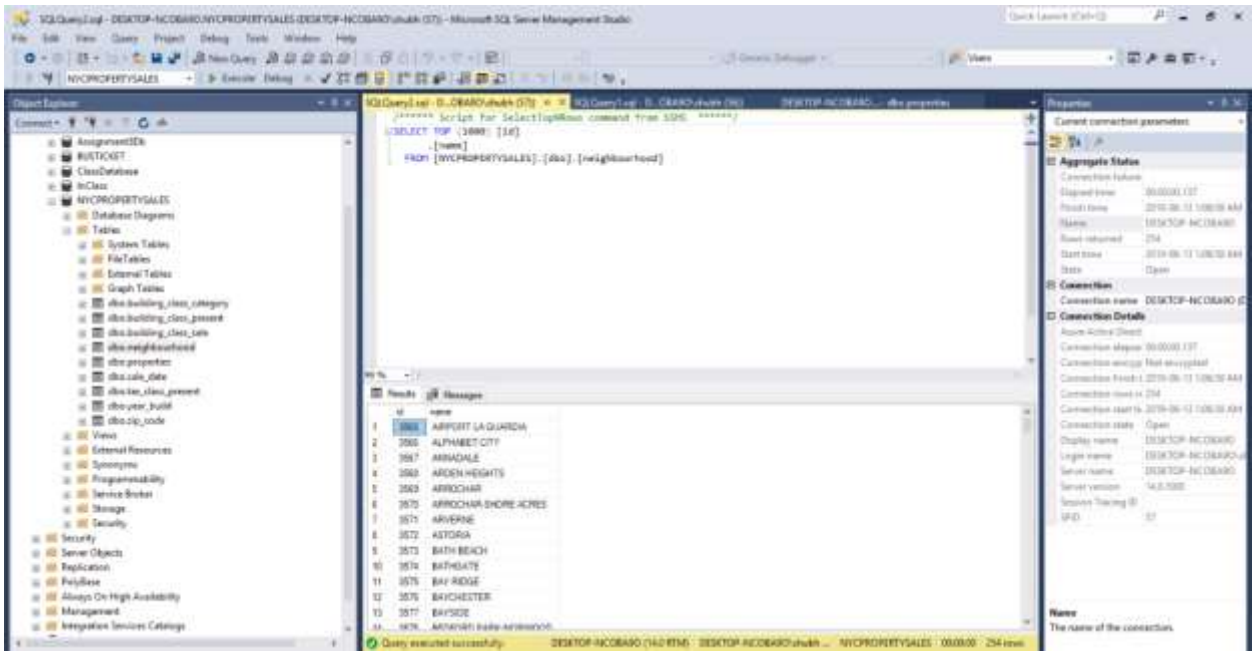

Fig 173: Neighborhood

➔ Building_class_present

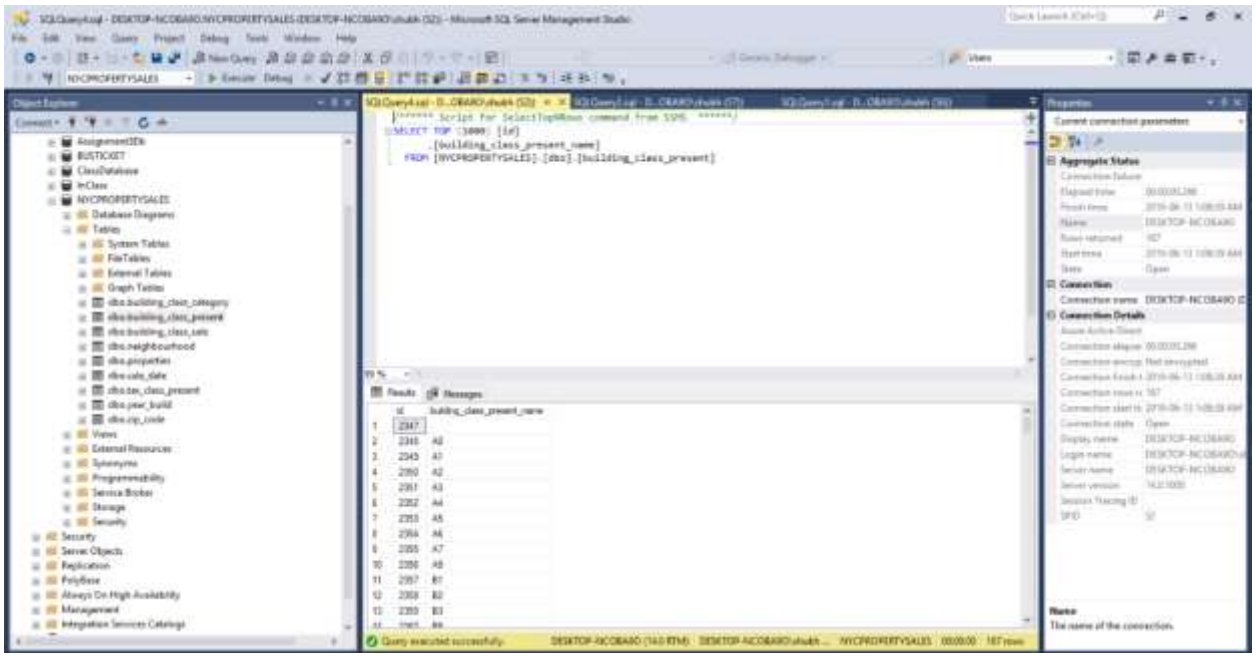

Fig 174: Build class present

➔ Building_class_category

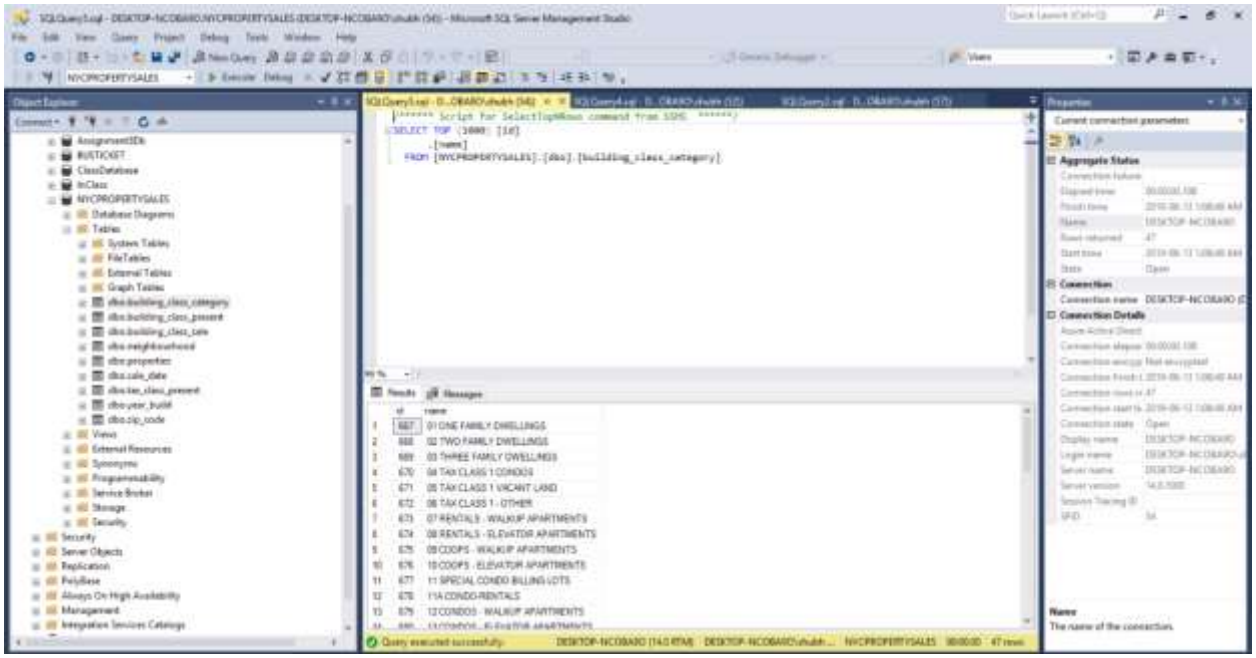

Fig 175: Building class category

➔ Sale_date

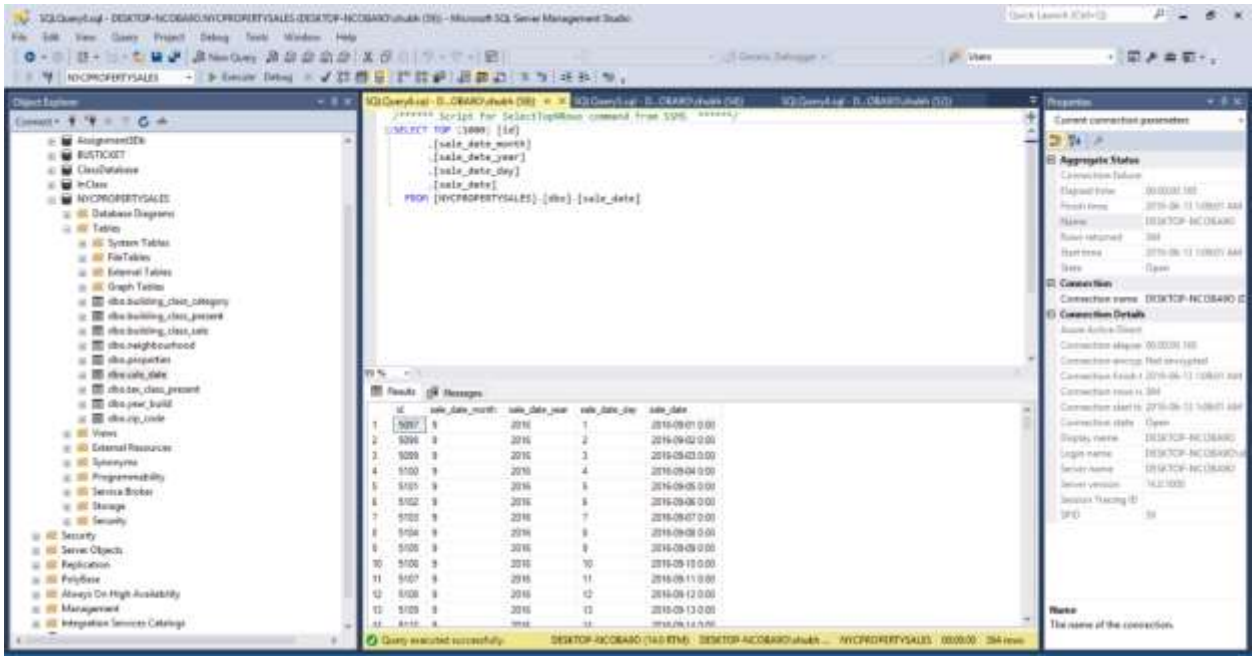

Fig 176: Sale date

➔ Tax_class_present

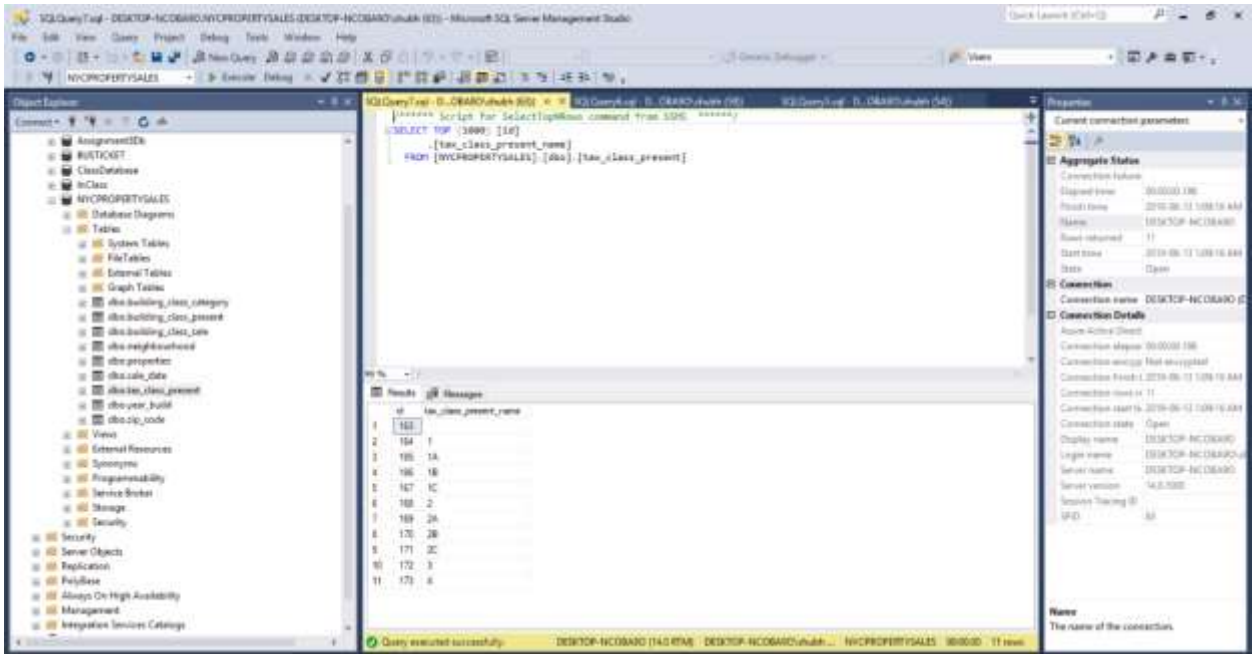

Fig 177: Tax class present

➔ Year_build

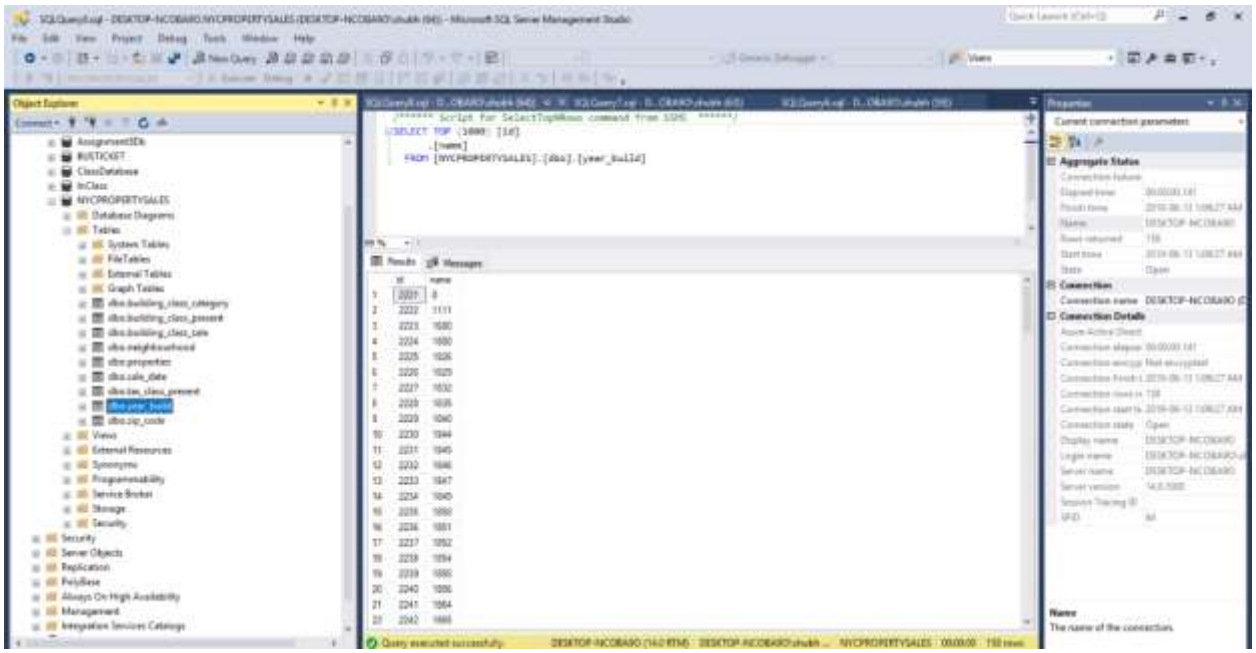

Fig 178: Year built

➔ Zip_code

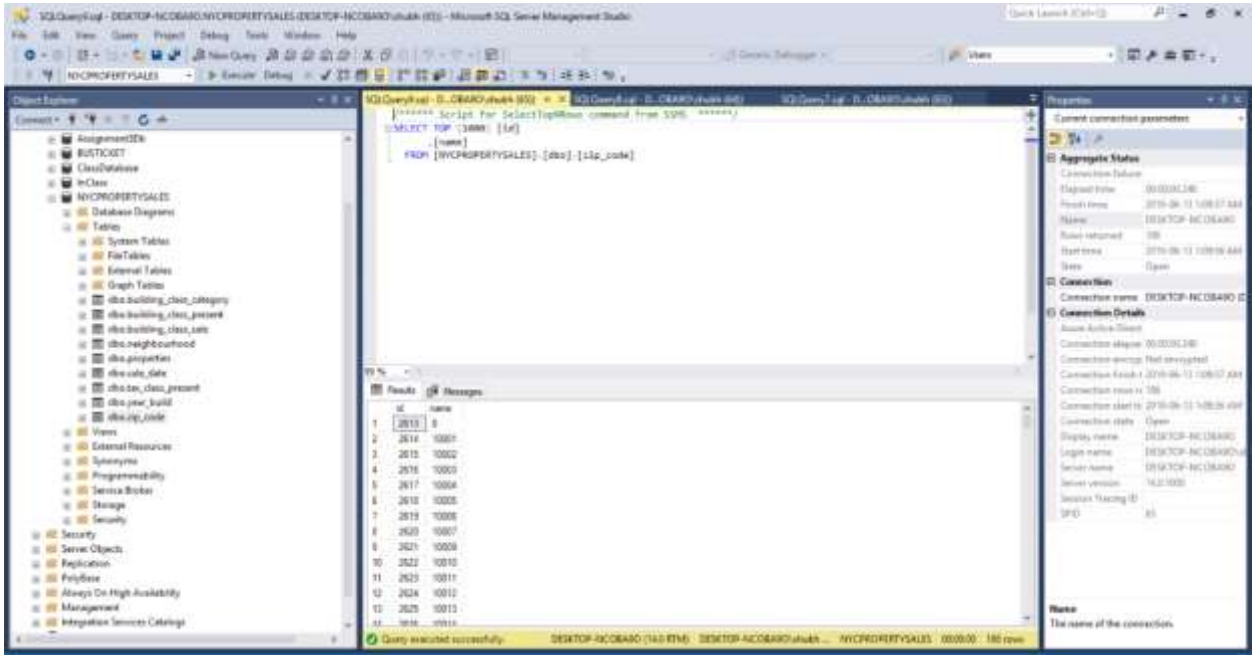

Fig 179: Zip code

E. CSV File generated:

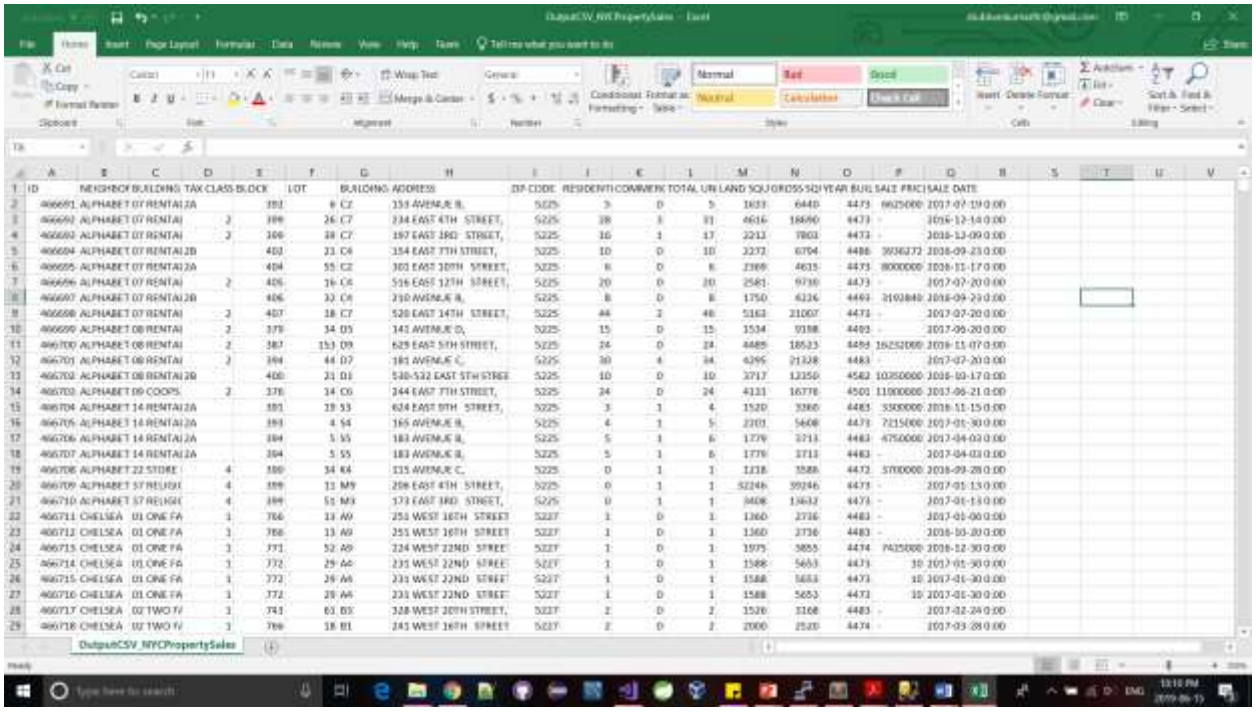

Fig 180: CSV file

Fig. Final CSV file generated

F. Brief on Work Done on SSIS:

Microsoft SSIS, is a tool used for data warehousing and extract, transform and load operations. As part of our Business intelligence assignment, we had to perform a task using this tool.
The task on our hand was to create an automated end to end ETL pipeline to extract data from a source, clean it and load it into the data warehouse by creating a control flow of ideal tasks.
Part of the task was to find a suitable dataset that can be used for this purpose. Therefore, we chose a New York city Property sales dataset from Kaggle. [NycPropertySales]

To accomplish this task successfully, we had to follow certain steps as following:

- Choose a schema design such as snow flake schema or star schema to lay foundation for our data warehouse
- Create dimension tables and fact able as per the analysis and understanding or business requirement in real scenario and make sure all tables are in 3NF OR BCNF form
- Create integration service project in SSIS to implement the ETL pipeline and perform following tasks:
    1. Empty all tables
    2. Load data into dimension tables
    3. Load data into fact tables
    4. Save data into CSV
- After performing above tasks successfully, the data should be exported to CSV file using Fact table and its foreign key relationship with dimension tables.
- Next step would be to use the loaded tables for OLAP queries to analyze the data.

Upon successfully completing the task, we were able to understand how the ETL tasks are carried out in real life and automated to keep consistency and data integrity.

Comparison of the Data Preparation Software:

The Data Preparation Software that was used to complete the tasks were fairly new to the members in the team, but in spite that it was easy to learn how to use the software to perform various operation due to well documented reports (GitHub, 2019) (Google Cloud, 2019) (Docs.microsoft.com, 2019) made by the community and developers.

OpenRefine and Google Data Prep are easy to start up since one run via an *EXE* file and the other run on the cloud which can be accessed by signing into the account. Both provide an easy to read and use UI, but Google Data Prep can provide accurate perdition/ suggestion to carry out a task. But Google Data Prep is a paid software where the cost depends on the amount of data being transformed while OpenRefine is open source and additional modules could be created and added to it.

For Microsoft SSIS, the installation process was time consuming and a bit complex to carry out. It takes up a lot of space on the desktop system. But the uses of Data structures permits it to be more scalable and its GUI makes it easy to carry out processes and error handling can be done using try catch block.

The following table provides a concise comparison between the 3 Data Preparation Tools for different aspects/parameters:

OPEN REFINE	GOOGLE DATA PREP	SSIS
Sleek tool to data clean using different constructs of conditions for filtering	Equally useful for cleaning the data as open refine.	Equally useful for cleaning data.
Data structure is not required.	Data structure is not required.	Preparation of tables is required.
Desktop lighter version.	Cloud version.	Desktop version.
Easy to write conditions and evaluate and view data.	Easy to design data flow as it supports huge data.	Enterprise application which has error handling.
Debugging not possible	Debugging possible to some extent.	Debugging is available but complicated.
Is best for data cleaning.	Is best for big data.	Best for any kind of data, but the product has instability.
Easy to work and execute.	Easy to work and execute.	Complex to work and execute.

Table 1: Comparison of the different Data Preparation Software

Conclusion:

On completing the tasks, we were able to get a better understanding of the Data Preparation concepts, ETL procedure as well as gain experience from using the various Data Preparation Tools. The results of the tasks were recorded and documented with screenshots. The Data cleaning, Data evaluation and Data manipulation is carefully done in different steps and debugging is possible at every stage with different evaluation tools. Using google data prep, Open Refine and Enterprise Solutions like SSIS, dealing with large amounts of data has become easy for processing and storing in the data warehouse. Later, the cleaned data from the warehouse can be exported to data cubes to perform OLAP operations or other operation.

References:

Docs.microsoft.com. (2019). *SQL Server Integration Services - SQL Server Integration Services (SSIS)*. [online] Available at: https://docs.microsoft.com/en-us/sql/integration-services/sql-server-integration-services?view=sql-server-2017 [Accessed 15 Jun. 2019].

GitHub. (2019). *OpenRefine/OpenRefine*. [online] Available at: https://github.com/OpenRefine/OpenRefine/wiki [Accessed 15 Jun. 2019].

Google Cloud. (2019). *Cloud Dataprep by Trifacta documentation | Google Cloud Dataprep Documentation | Google Cloud*. [online] Available at: https://cloud.google.com/dataprep/docs/ [Accessed 15 Jun. 2019].

Williamson, E. (2019). *Fetching and Parsing Data from the Web with OpenRefine*. [online] Programminghistorian.org. Available at: https://programminghistorian.org/en/lessons/fetch-and-parse-data-with-openrefine [Accessed 15 Jun. 2019].

Appendix:

- SQL scripts for SSIS – Microsoft Sql server

```
use master;
IF EXISTS (SELECT * FROM sys.databases WHERE name = N'NYCPROPERTYSALES') BEGIN DROP
DATABASE NYCPROPERTYSALES END;
GO
IF NOT EXISTS (SELECT * FROM sys.databases WHERE name = N'NYCPROPERTYSALES') BEGIN
CREATE DATABASE NYCPROPERTYSALES END;
GO
USE NYCPROPERTYSALES
GO

create table properties(
id int primary key identity(1,1),
neighbourhood int null,
building_class_category int null,
tax_class_present int null,
[block] int null,
lot int null,
building_class_present int null,
[address] varchar(50) null,
apartment_number varchar(50) null,
zip_code int null,
residential_units int null,
commertial_units int null,
total_units int null,
land_square_feet int null,
gross_square_feet int null,
year_build int null,
sale_price varchar(50) null,
sale_date int null
)
GO
create table neighbourhood(
id int primary key identity(1,1),
[name] varchar(50))
GO
create table sale_date(
id int primary key identity(1,1),
sale_date_month int null,
sale_date_year int null,
sale_date_day int null,
sale_date varchar(20) null)
GO
create table building_class_category(
id int primary key identity(1,1),
[name] varchar(50))
GO
create table tax_class_present(
id int primary key identity(1,1),
tax_class_present_name varchar(50))
GO
create table building_class_present(
```

```
id int primary key identity(1,1),
building_class_present_name varchar(50) null)
GO
create table zip_code(
id int primary key identity(1,1),
[name] varchar(50) null)
GO
create table year_build(
id int primary key identity(1,1),
[name] varchar(50) null)
GO

Alter table properties
add constraint FK_neighbourhood foreign key (neighbourhood)
references neighbourhood(id);
Alter table properties
add constraint FK_building_class_category foreign key (building_class_category)
references building_class_category(id);
Alter table properties
add constraint FK_sale_date foreign key (sale_date)
references sale_date(id);
Alter table properties
add constraint FK_tax_class_present foreign key (tax_class_present)
references tax_class_present(id);
Alter table properties
add constraint FK_building_class_present foreign key (building_class_present)
references building_class_present(id);
Alter table properties
add constraint FK_year_build foreign key (year_build)
references year_build(id);
Alter table properties
add constraint FK_zip_code foreign key (zip_code)
references zip_code(id);
```

- Expression used to derive date into day, month and year
    A. YEAR(((DT_DATE) [SALE DATE]))
    B. MONTH(((DT_DATE) [SALE DATE]))
    C. DAY(((DT_DATE) [SALE DATE]))