



**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.

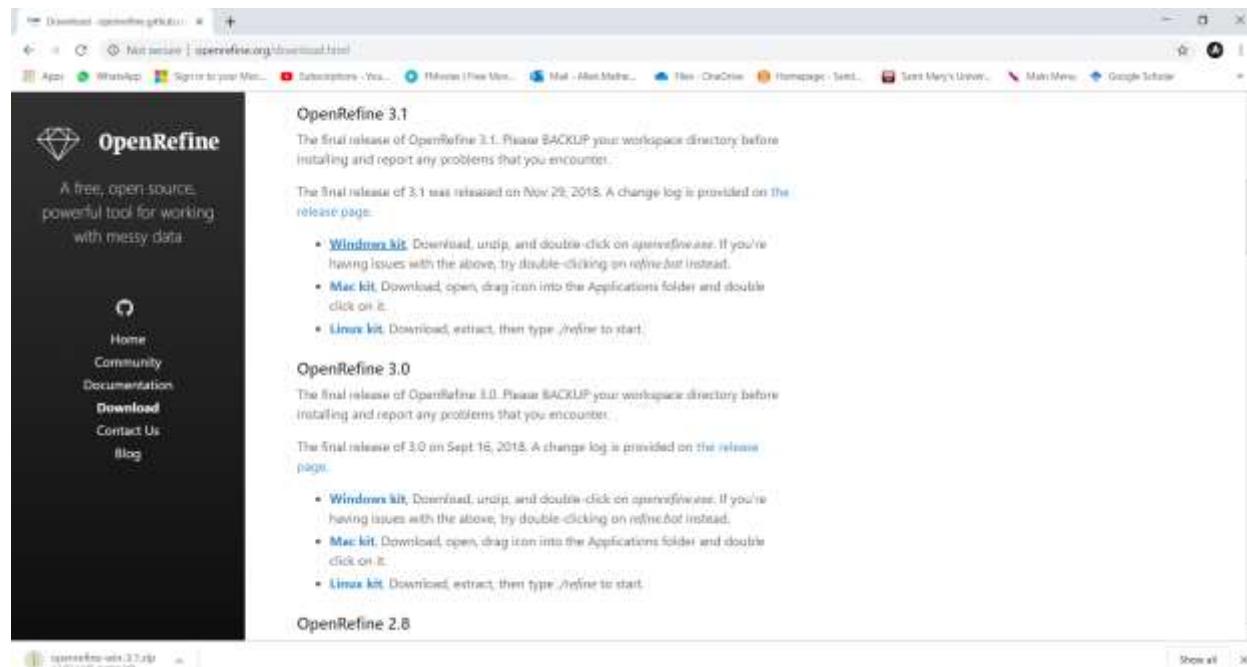


Fig 1: Open Refine Download Page

- 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.

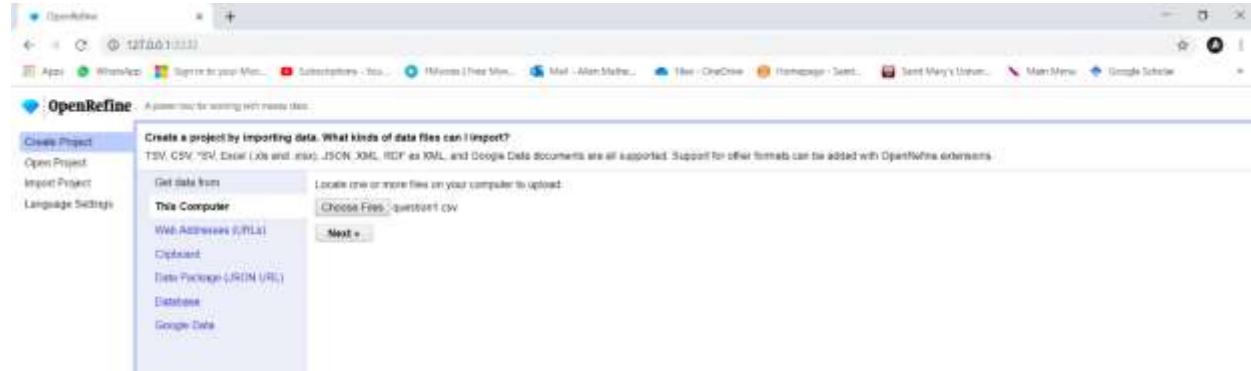


Fig 2: Choose the data file(i.e. question1.csv).

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.

This screenshot shows the main interface of OpenRefine. The title bar reads "OpenRefine - OpenRefine". The top navigation bar includes "File", "Edit", "View", "Table", "Facets", "Filter", "Help", and "Extensions". Below the title bar, there's a toolbar with icons for "Open", "Export", and "Help". The main area is titled "973 records" and displays a table with 973 rows of data. The columns are labeled: "All", "ID", "Full name", "Email", "Sex", "Birth date", "Employment", "Job", and "Img". Each row contains a unique identifier, a person's name, their email address, gender, birth date, employment status, job title, and a thumbnail image. The table has a header row and 973 data rows. On the left side of the main area, there's a sidebar with the title "Using facets and filters" and a brief description: "Use facets and filters to select subsets of your data to act on. Choose facet and filter methods from the menus at the top of each data column." It also includes links to "Not sure how to get started?" and "Watch these screencasts".

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)

This screenshot shows the same OpenRefine interface as Fig 4, but with the "Show as" operation set to "Rows". The title bar and toolbar are identical. The main area is titled "1000 rows" and displays a table with 1000 rows of data. The columns are the same: "All", "ID", "Full name", "Email", "Sex", "Birth date", "Employment", "Job", and "Img". The data is identical to the 973 records view. The left sidebar remains the same, providing instructions on using facets and filters.

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]**).

The screenshot shows the OpenRefine interface with a table containing 1000 rows. A facet has been created on the 'id' column, specifically for 'Facet by blank [null or empty string]'. The facet dropdown shows various options like 'Text facet', 'Number facet', etc., and the 'Customized facets' option is selected. The facet results show 27 rows where the 'id' column is empty. These rows are highlighted in yellow in the main data table.

id	full_name	email	sex	birth_date	credit_card	ip	http
	Wendy Gottlieb	wendy@redacted.com	Male	01.01.1990	3000000000000000000	45.5400000	14.8851158
	Marcus Flores	marcus@redacted.org	Female	24.08.1985	4000000000000000000	4.0200000	17.7018800
	Stephen Corriveau	steven@redacted.ca	Male	18.04.1991	3000000000000000000	36.3000000	29.4833256
	James	james@redacted.com	Male	01.01.1990	3000000000000000000	16.4779508	46.1771001
	David	da@redacted.com	Male	11.02.1990	3000000000000000000	22.0000000	100.3300916
	John	john@redacted.com	Male	15.12.1980	3000000000000000000	6.3881000	100.6035114
27	John	john@redacted.com	Male	08.12.1980	3000000000000000000	67.86	110.4004504
28	John	john@redacted.com	Female	28.12.1991	3000000000000000000	60.7171000	8.4254000
29	Remove all matching rows					36.4138977	28.0819427
30	John	john@redacted.com	Female	03.12.1990	3000000000000000000	41.3000277	42.0137100
31	Wendy Gottlieb	wendy@redacted.com	Male	08.08.1990	3000000000000000000	30.3000049	100.9144000
32	Lucy Terrell	lucy@redacted.ca	Female	01.03.1994	3000000000000000000	35.4400018	100.6035470
33	Pauline Whittaker	pauline@redacted.co.uk	Male	31.05.1988	3000000000000000000	10.0400008	15.7101940
34	Carmen Gossen	carmen@redacted.ca	Male	21.12.2000	3000000000000000000	67.11721000000000000	41.0000000
35	Sigrid Rosales	sigrid@redacted.ca	Male	09.09.1991	3000000000000000000	4.2070000	100.8100000
36	Fiona Duncan	fiona@redacted.com	Male	24.07.2001	3000000000000000000	44.5102	98.0210000
37	Wendy Gottlieb	wendy@redacted.ca	Female	03.04.1990	3000000000000000000	67.2000010	110.5100000
38	Caro Capwell	caro@redacted.ca	Female	23.01.1990	3000000000000000000	3.0671400	100.210001
39	Hannah Muirhead	hannah@redacted.ca	Female	31.08.1990	3000000000000000000	12.2270047	123.3002047
40	Heidi Bullock	heidib@redacted.ca	Female	05.11.1987	3000000000000000000	89.7711000	17.02.30011
41	Colleen Mac	colleen@redacted.ca	Female	27.10.1991	3000000000000000000	32.2510000	110.1545400
42	Renae Sanderson	renae@redacted.ca	Male	11.06.1988	4000000000000000000	39.0400000	10.0910000

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’).

The screenshot shows the OpenRefine interface with the same 1000 rows table. The 'Edit rows' dropdown is open, and the option 'Remove all matching rows' is selected. This action will delete the 27 rows where the 'id' column is empty.

id	full_name	email	sex	birth_date	credit_card	ip	http
	Wendy Gottlieb	wendy@redacted.com	Male	01.01.1990	3000000000000000000	45.5400000	14.8851158
	Marcus Flores	marcus@redacted.org	Female	24.08.1985	4000000000000000000	4.0200000	17.7018800
	Stephen Corriveau	steven@redacted.ca	Male	18.04.1991	3000000000000000000	36.3000000	29.4833256
	James	james@redacted.com	Male	01.01.1990	3000000000000000000	16.4779508	46.1771001
	David	da@redacted.com	Male	11.02.1990	3000000000000000000	22.0000000	100.3300916
	John	john@redacted.com	Male	15.12.1980	3000000000000000000	6.3881000	100.6035114
27	John	john@redacted.com	Male	08.12.1980	3000000000000000000	67.86	110.4004504
28	John	john@redacted.com	Female	28.12.1991	3000000000000000000	60.7171000	8.4254000
29	Remove all matching rows					36.4138977	28.0819427
30	John	john@redacted.com	Female	03.12.1990	3000000000000000000	41.3000277	42.0137100
31	Wendy Gottlieb	wendy@redacted.com	Male	08.08.1990	3000000000000000000	30.3000049	100.9144000
32	Lucy Terrell	lucy@redacted.ca	Female	01.03.1994	3000000000000000000	35.4400018	100.6035470
33	Pauline Whittaker	pauline@redacted.co.uk	Male	31.05.1988	3000000000000000000	10.0400008	15.7101940
34	Carmen Gossen	carmen@redacted.ca	Male	21.12.2000	3000000000000000000	67.11721000000000000	41.0000000
35	Sigrid Rosales	sigrid@redacted.ca	Male	09.09.1991	3000000000000000000	4.2070000	100.8100000
36	Fiona Duncan	fiona@redacted.com	Male	24.07.2001	3000000000000000000	44.5102	98.0210000
37	Wendy Gottlieb	wendy@redacted.ca	Female	03.04.1990	3000000000000000000	67.2000010	110.5100000
38	Caro Capwell	caro@redacted.ca	Female	23.01.1990	3000000000000000000	3.0671400	100.210001
39	Hannah Muirhead	hannah@redacted.ca	Female	31.08.1990	3000000000000000000	12.2270047	123.3002047
40	Heidi Bullock	heidib@redacted.ca	Female	05.11.1987	3000000000000000000	89.7711000	17.02.30011
41	Colleen Mac	colleen@redacted.ca	Female	27.10.1991	3000000000000000000	32.2510000	110.1545400
42	Renae Sanderson	renae@redacted.ca	Male	11.06.1988	4000000000000000000	39.0400000	10.0910000

Fig 7: Selecting “Remove all matching rows” within Edit rows from All dropdown, when there are blank records in the ‘id’ column

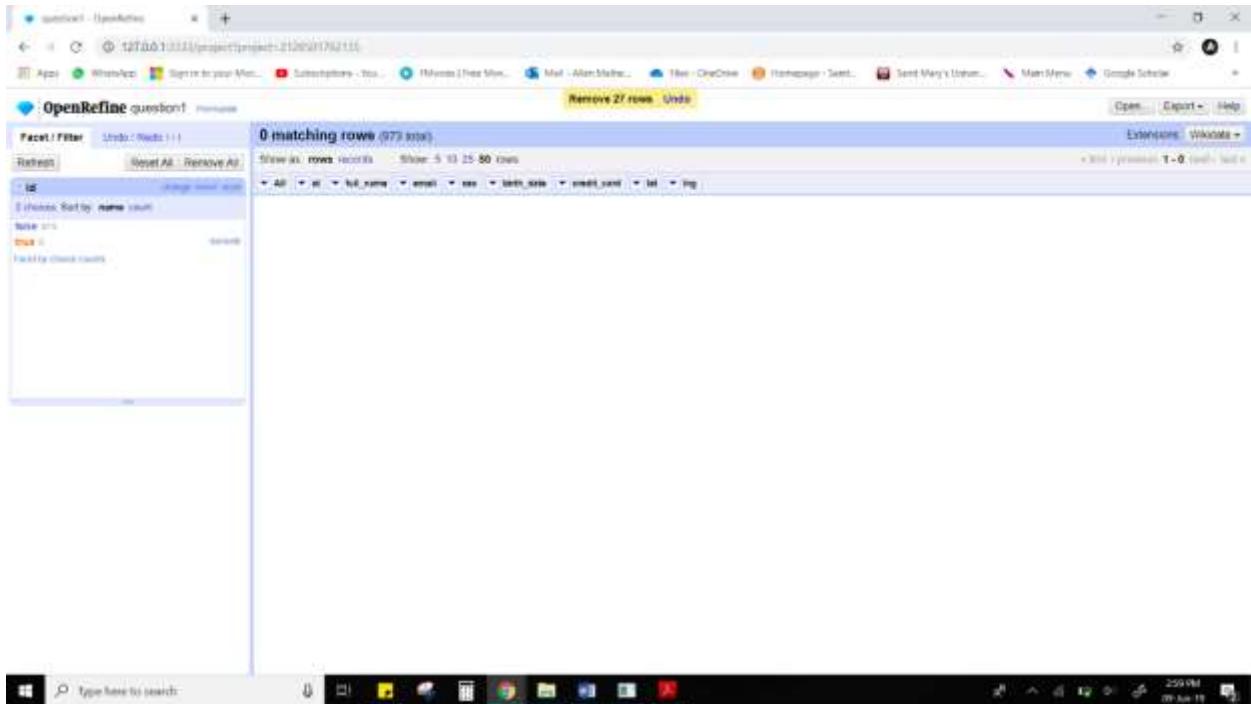


Fig 8: Confirmation pertaining to the Removal of the 27 rows with blank records in the 'id' column

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.

All	#	full_name	email	sex	birth_date	credit_card	id	img
1	1	Alice Smith	alice@query.org	Male	31/01/1992	0428982425731401	88.62039601	23.874271
2	2	Peter Johnson	peterjohnson1985.gov	Male	07/11/1988	0102141504480141	40.02001201	8.38840401
3	3	Elizabeth Williams	elizabeth.williams@business.com	Female	18/02/1992	0337774651200021	20.88170001	101.80021901
4	4	Matthew Carter	matthew.carter@connectcloud.com	Female	28/04/2000	0518055602174751	01.2	86.01003
5	5	Barbara Murray	barbara.murray@business.net.au	Male	15/08/1992	052415021480017401	58.15501010	27.7423348
6	6	Lucy Rose	lucyrose@business.edu	Female	08/02/1988	026402200000017601	21.00004401	12.770121
7	7	Mark Hatcher	markhatcher@business.com	Male	27/03/1990	004110000000017401	58.31501011	140.80021511
8	8	Charlotte Goulding	charlotte.goulding.gov	Male	14/08/1989	008817730000000173	7.23001121	76.127016
9	9	James Whitehead	jameswhitehead@business.co.uk	Male	09/08/2000	070090041961700001	28.988001	28.1971031
10	10	Grace Jackson	grace.jackson@business.co.uk	Male	19/03/1992	0365101400117100	1.8770031	4.1888344
11	11	David Ashton	david.ashton@business.co.uk	Female	08/06/1992	0311400103599802	14.4000471	79.4014601
12	12	Philip Peters	philip.peters@business.com	Female	21/02/1990	002440000000017301	21.20027701	64.400387
13	13	Kevin Clark	kevin.clark@business.net	Female	25/04/1997	000220040000000001	30.00005001	134.410102
14	14	Elisabeth Weston	elisabeth.weston@business.com	Female	27/06/1997	0007500171470001	36.1000571	116.700200
15	15	Karen Weston	karen.weston@business.com	Female	27/06/1997	0007500171470001	36.1000571	116.700200
16	16	Carrie Davies	carrie.davies@business.com	Male	23/07/2001	0084051040787001	27.7300111	13.3000008
17	17	Heidi Jones	heidi.jones@business.com	Male	06/12/1988	0124500103200001	8.17000001	112.6100211
18	18	Ronald Van Der Kolk	ronald.vanderkolk.nl	Male	22/08/1989	0010007000700111	01.0000001	17.8100001
19	19	Horatio Johnson	horatio.johnson@business.co.uk	Female	28/08/1994	0170000000000001	01.0000001	29.0010001
20	20	Olivia McVay	olivia.mcvay@business.com	Female	19/05/1991	0050000000000001	30.7100001	119.9502041
21	21	Jess Hoffmann	jess.hoffmann@business.gov	Male	23/09/1988	0000000000000001	44.1901421	125.0000071
22	22	Benedict Howell	benedict.howell@business.co.uk	Male	07/08/1990	0000000000000001	16.30000001	77.3000071
23	23	Reed Bennett	reed.bennett@business.com	Male	04/09/2000	0520000000000001	17.0000001	112.0000246
24	24	Christine Cole	christine.cole@business.com	Male	04/09/2000	0520000000000001	17.0000001	112.0000246
25	25	Hannah Fletcher	hannah.fletcher@business.com	Female	13/12/1999	0545400000000001	28.3330781	1.3910011
26	26	Natalie Army	natalie.army@business.com	Female	03/10/1991	0507500000000001	23.0000001	22.5000001
27	27	Leanne Quinn	leanne.quinn@business.com	Male	03/08/1990	0545775000000001	40.1002001	29.4250001
28	28	Sophie Marshall	sophie_marshall@google.com	Female	11/08/2003	0033781000000001	31.8700017	01.0000001

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]** ).

The screenshot shows the OpenRefine interface with the following details:

- Project Name:** question1\_OpenRefine
- Rows:** 973 rows
- Facet / Filter:** Undo / Redo (ctrl+z)
- Customized facets:** A dropdown menu where "Facet by blank [null or empty string]" is selected.
- Table Headers:** All, id, full\_name, email, sex, birth\_date, email\_domain, zip, lng.
- Data Preview:** A list of 33 rows where the 'email' column contains either null or an empty string. The rows are numbered from 1 to 33.

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’).

The screenshot shows the OpenRefine interface with the following details:

- Project Name:** question1\_OpenRefine
- Rows:** 973 rows (973 total)
- Facet / Filter:** Undo / Redo (ctrl+z)
- Customized facets:** A dropdown menu where "Facet by blank [null or empty string]" is selected.
- Table Headers:** All, id, full\_name, email, sex, birth\_date, email\_domain, zip, lng.
- Edit rows:** A dropdown menu where "Remove all matching rows" is selected.
- Data Preview:** A list of 33 rows where the 'email' column contains either null or an empty string. The rows are numbered from 1 to 33.

Fig 11: Selecting “Remove all matching rows” within Edit rows from All dropdown, when there are blank records in the ‘email’ column

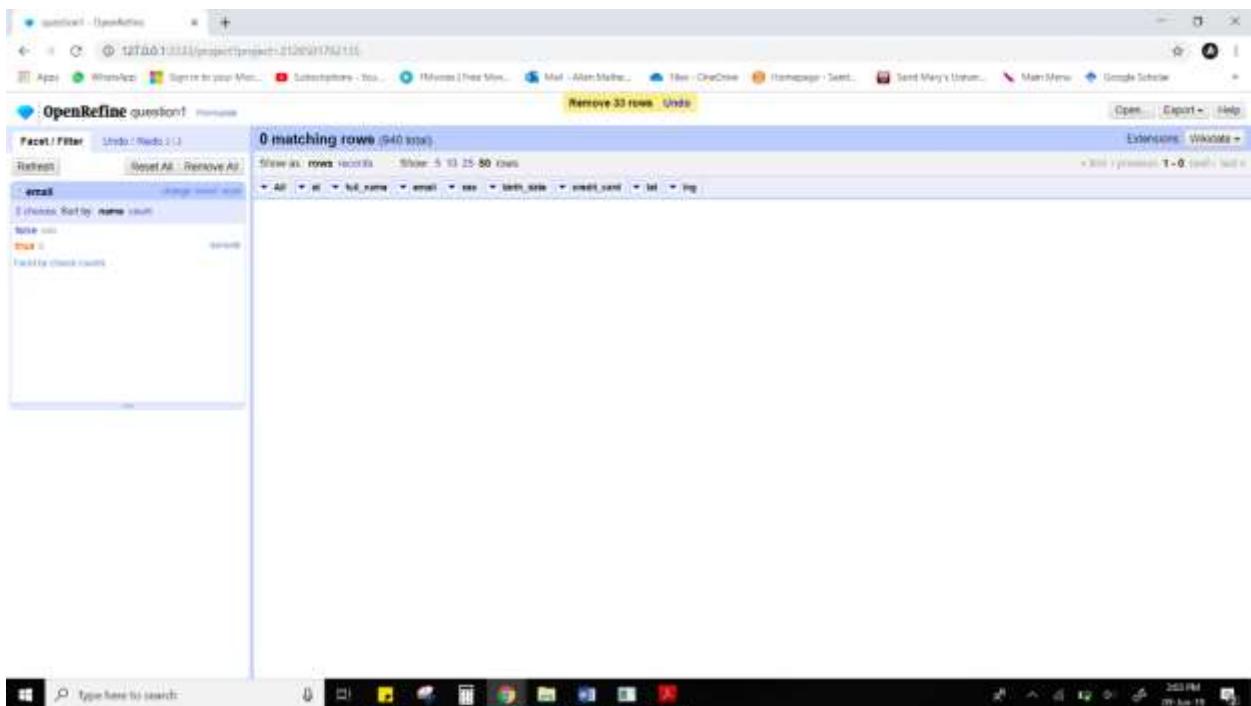


Fig 12: Confirmation pertaining to the Removal of the 33 rows with blank records in the 'email' column

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.

940 rows								
id	full_name	email	sex	birth_date	credit_card	int	img	
1	Alice Smith	alice@query4it.com	Male	01/01/1992	0428982425731401	88.6239900	-21.8742470	
2	Peter Schmitt	peterm@t1000.gov	Male	07/11/1983	0329141504408145	40.0200120	-8.3884060	
3	Elizabeth Jenkins	elizabethj@sharpmind.com	Female	18/03/1992	0337771651200020	20.8887000	-10.800021007	
4	Mariano Contreras	mariano.m@connectcloud.com	Female	28/04/2000	033803560274975	43.2	36.03003	
5	Eduardo Sandoval	eduardo.s@net-mx.mx	Male	15/08/1992	033215021480021401	58.1550100	-77.7423348	
6	Lacey Rose	lacey@connectcloud.co	Female	08/02/1998	032642209000070001	24.0000440	-12.7162124	
7	Marion Fletcher	marionf@groupgurus.com	Male	27/03/1990	0341930000074001	38.3151541	-130.8007517	
8	Esther Goulding	estherg@t1000.gov	Male	14/08/1990	033881773000000001	7.2300112	-76.1297604	
9	Elaine Swanson	elaine.s@sharpmind.com	Female	13/08/2000	030990241967000001	28.98800	-28.1971031	
10	Grace Aschenbrenner	graceaschenbrenner@t1000.gov	Male	19/03/1992	036531014009117001	18.779031	-4.8888344	
11	Darly Aschenbrenner	darlyaschenbrenner@t1000.gov	Female	08/06/1992	0311490103588952	14.4803471	-57.4034607	
12	Polly Petree	polly@connectcloud.com	Female	21/02/1992	032440000000000001	21.2827730	-64.400387	
13	Kevin Cook	kevinco@t1000.gov	Female	25/04/1997	030222004000000001	3.3010550	-134.4310122	
14	Eliseous Henriquez	eliseous.h@connectcloud.com	Female	27/06/1997	0305160217400001	36.165271	-16.7620308	
15	Karen Harriet	karenharriet@sharpmind.com	Male	23/07/2001	036845104070000001	27.7340011	-13.3000008	
16	Carmen Davis	carmend@t1000.gov	Male	06/12/1998	034500000000000001	8.1760003	-112.6358211	
17	Hector Valdez Ybarra	hector.v@sharpmind.com	Male	23/08/1999	031600000000000001	0.0370007	-17.7130001	
18	Horacio Hernandez	horacio.h@connectcloud.co	Female	28/08/1994	031917000000000001	33.8900000	-29.0810001	
19	Olivia McNear	oliviame@t1000.gov	Female	19/05/1991	030700000000000001	32.7100007	-119.9520044	
20	Jessie Hoffmann	jessiehoffmann@t1000.gov	Male	23/09/1998	030000000000000001	44.191142	-125.8000071	
21	Brielle Ramey	brielle.r@connectcloud.com	Male	07/03/1999	030000000000000001	16.3000000	-71.3000074	
22	Christine Cole	christine.c@t1000.gov	Male	04/09/2000	032300000000000001	17.1800011	-112.8000246	
23	Hannah Fletcher	hannah.f@sharpmind.com	Female	13/12/1999	034540000000000001	28.333781	-81.3912713	
24	Isabel Arroyo	isabelarroyo@t1000.gov	Female	02/11/1991	032970000000000001	23.0000001	-22.5000281	
25	Leanne Qualls	leanneq@connectcloud.com	Male	23/05/1990	034277000000000001	60.1902000	-29.4253300	
26	Ross Marquardt	rossmarquardt@sharpmind.com	Female	11/08/2003	032310000000000001	51.8700017	-61.0000001	
27	Natalie Southwick	nataliesouthwick@t1000.gov	Female	11/03/1993	032400000000000001	80.7300000	-113.0000042	
28	Tasha Farago	tashafarago@sharpmind.com	Female	28/08/1993	030942230000000001	34.92000	-58.00000	
29	Julian Fralick	juanfralick@t1000.gov	Female	16/11/1999	030000000000000001	43.8971000	-64.3000011	
30	Monica Ulrich	monicau@sharpmind.com	Female	27/04/1993	030000000000000001	30.4700000	-1.3000001	

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).

The screenshot shows the OpenRefine interface with a dataset containing 940 rows. The 'full\_name' column is highlighted, and a context menu is open, showing options like 'Add column based on this column'. This option is being selected to create new columns 'f\_name' and 'l\_name'.

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.

The screenshot shows the 'Add column based on column full\_name' dialog box. The 'New column name' field is set to 'f\_name'. The 'Expression' field contains the code `value.split("")[0]`. The preview table shows the first few rows of the dataset, and the 'Preview' tab is selected.

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 **I\_name** column.

The screenshot shows the OpenRefine interface with a dialog box titled "Add column based on column full\_name". The dialog box contains the following fields:

- New column name: I\_name
- Store view/stylesheet:  set to blank  store error  copy value from original column
- Language: General Refine Expression Language (GREL)
- Expression: value.split("").slice(1).join(" ")

The preview section shows the transformation of the "full\_name" column into the "I\_name" column:

row_index	row_value	value.split("").slice(1).join(" ")
1	Arne Burde	Burde
2	Pete Burkholtz	Burkholtz
3	Elizabeth Ferraro	Ferraro
4	Malvina Coonman	Coonman
5	Banner Duley	Duley
6	Dacey Flory	Flory
7	Marken F-Hausler	F-Hausler

Fig 16: Expression to derive last name from the full\_name column

The **f\_name** and **I\_name** column were successfully created.

This screenshot shows the OpenRefine interface with a project titled "OpenRefine question!". A yellow banner at the top reads "Create new column f\_name based on column full\_name by filling 940 rows with great value.apf[\"Label(1)\"].f\_name". The main table has 940 rows and 11 columns: #, f\_name, I\_name, email, sex, birth\_date, credit\_card, #, and img. The "f\_name" column contains names like "Avery Smith", "Peter Burkhart", etc. The "I\_name" column is empty. The "email" column contains various email addresses. The "sex" column has values like "Male" and "Female". The "birth\_date" column shows dates from 19.01.1992 to 27.02.1998. The "credit\_card" column has values ranging from 10.000.000.000.000.000 to 10.000.000.000.000.000. The "#, #, and img" columns are also present. The interface includes facets and filters on the left and various toolbars at the top and bottom.

Fig 17: *f\_name* Column was Created Successfully.

This screenshot shows the OpenRefine interface with a project titled "OpenRefine question!". A yellow banner at the top reads "Create new column I\_name based on column full\_name by filling 940 rows with great value.apf[\"Label(1)\"].I\_name". The main table has 940 rows and 11 columns: #, f\_name, I\_name, email, sex, birth\_date, credit\_card, #, and img. The "f\_name" column contains names like "Avery Smith", "Peter Burkhart", etc. The "I\_name" column now contains names like "Smith", "Burkhart", etc. The "email" column contains various email addresses. The "sex" column has values like "Male" and "Female". The "birth\_date" column shows dates from 19.01.1992 to 27.02.1998. The "credit\_card" column has values ranging from 10.000.000.000.000.000 to 10.000.000.000.000.000. The "#, #, and img" columns are also present. The interface includes facets and filters on the left and various toolbars at the top and bottom.

Fig 18: *I\_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 **I\_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...**).

The screenshot shows the OpenRefine interface with a dataset containing 940 rows. A context menu is open over the 'sex' column, with the 'Transform...' option highlighted. The menu also includes other options like 'Edit column', 'Common transforms', 'Fill down', 'Mark down', 'Split', 'View', 'Self multi-valued cells', 'Join multi-valued cells', 'Cluster and edit...', and 'Replace'.

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.

The screenshot shows the 'Custom text transform on column sex' dialog box. The 'Expression' field contains the code 'value.replace("Male","M").replace("Female","F")'. The 'Language' dropdown is set to 'General Refine Expression Language (OREL)'. Below the expression, a preview table shows the transformation of gender values from 'Male' and 'Female' to 'M' and 'F'. At the bottom, there are options for 'On error' (keep original, set to blank, ignore error), a checkbox for 'Re-transform up to 10 times until no change', and buttons for 'OK' and 'Cancel'.

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

All	#	id	first_name	last_name	email	sex	birth_date	credit_card	age	img
1	1	Anna Burts	Anna	Burts	aburts@parny.net	M	31.01.1990	3418630-02721160	81.0203608	33.0724272
2	2	Peter Burkharts	Peter	Burkharts	pburkharts1@qrs.gov	M	07.11.1968	340147808484140	40.0320118	8.380406
3	3	Elizabeth Fentie	Elizabeth	Fentie	elizabeth@fenties.com	F	19.03.1992	3322771000192460	90.0307308	142.8821702
4	4	Mariah Coonan	Mariah	Coonan	mcooan@mariahscoona.com	F	28.08.2000	3308600001114370	43.6	92.6233
5	5	Barney Dally	Barney	Dally	barneydally@net.au	M	19.08.1992	3302733100000180	91.1697131	77.765344
6	6	Daisy Foss	Daisy	Foss	dfoss@causing.edu	F	06.02.1998	337584996570500	94.0590405	17.7703129
7	7	Mariam Edmonde	Mariam	Edmonde	medmonde1@google.com	M	07.02.1996	3341900000000400	103.1703941	194.8221447
8	8	Valerie Cooling	Valerie	Cooling	vcooling@lars.pms	M	14.06.1999	330047084620510	7.2550112	79.129784
9	9	Karen Garside	Karen	Garside	karen.garside1@qrs.com	F	13.08.2000	3376601545700000	23.08060	84.1919101
10	10	Landy Jassawill	Landy	Jassawill	l.jassawill@jss.com	M	15.03.1990	330221000011160	5.277621	6.380644
11	11	Phyllis Foss	Phyllis	Foss	phyllis.foss@phyliss.com	F	09.06.1998	3317440000000000	14.080415	79.0304805
12	12	Kaylyn Jacob	Kaylyn	Jacob	kylynn.jacob@kylynn.com	F	23.02.1993	3304280000000001	37.2627170	44.400387
13	13	Esteban Revere	Esteban	Revere	esteban.revere@estebans.cool	F	29.04.1997	3302230000000000	93.0389997	134.4910135
14	14	Karen Howard	Karen	Howard	kaearen@howards.com	F	27.06.1997	3307902311470000	90.1607071	116.765398
15	15	Delois Derry	Delois	Derry	delois.derry@delois.com	M	32.07.2001	3344012194700100	17.2344811	133.0900689
16	16	Isoma Sampi	Isoma	Sampi	isoma.sampi@isoma.com	M	06.11.1996	334200042260450	6.1700003	142.026244
17	17	Ianita Van der Kist	Ianita	Van der Kist	ianita.vander.kist@vander.kist	M	22.06.1998	33067800730115	90.0370007	17.9146644
18	18	Hector Hockman	Hector	Hockman	hector.hockman@hectors.co.uk	F	28.06.1994	33161875025130	52.0564308	23.0873044
19	19	Isidore Hixson	Isidore	Hixson	isidore.hixson@isidore.com	F	18.02.1993	3326160047000000	4.7176407	116.3620448
20	20	John Hobbsen	John	Hobbsen	johobbsen@johns.org	M	23.06.1996	3308610710000000	44.1011747	131.0306701
21	21	Heath Hobell	Heath	Hobell	heath.hobell@heath.co.jp	M	07.03.1998	3308992300000000	78.0011998	21.3559774
22	22	Christine Cade	Christine	Cade	christine.cade@christine.cade	M	04.06.2002	33291901000002041	7.3700718	142.0007246
23	23	Marilyn Michael	Marilyn	Michael	marilyn.michael@marilyn.michael	F	13.12.1995	3304030000001200	26.022791	81.3910772
24	24	Helen Jappi	Helen	Jappi	helen.jappi@helen.jappi.com	F	03.11.1997	33072900000001720	85.0000401	27.050281
25	25	Leanne Champs	Leanne	Champs	leanne.champs@leanne.champs	M	28.05.1993	33077305031780	60.1962805	28.0210004

Fig 21: The sex column was successfully transformed from "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).

All	#	id	first_name	last_name	email	sex	birth_date	credit_card	age	img
1	1	Anna Burts	Anna	Burts	aburts@parny.net	M	24.07.1990	3302000000000000	33.0724272	
2	2	Peter Burkharts	Peter	Burkharts	pburkharts1@qrs.gov	M	08.04.1968	3305010000000000	40.0320118	
3	3	Elizabeth Fentie	Elizabeth	Fentie	elizabeth@fenties.com	F	19.03.1992	3322771000000000	90.0307308	
4	4	Mariah Coonan	Mariah	Coonan	mcooan@mariahscoona.com	F	28.08.2000	3308600001114370	43.6	
5	5	Barney Dally	Barney	Dally	barneydally@net.au	M	19.08.1992	3302733100000000	91.1697131	
6	6	Daisy Foss	Daisy	Foss	dfoss@causing.edu	F	06.02.1998	337584996570500	94.0590405	
7	7	Mariam Edmonde	Mariam	Edmonde	medmonde1@google.com	M	07.02.1996	3341900000000400	103.1703941	
8	8	Valerie Cooling	Valerie	Cooling	vcooling@lars.pms	M	14.06.1999	330047084620510	7.2550112	
9	9	Karen Garside	Karen	Garside	karen.garside1@qrs.com	F	13.08.2000	3376601545700000	23.08060	
10	10	Landy Jassawill	Landy	Jassawill	l.jassawill@jss.com	M	15.03.1990	330221000011160	5.277621	
11	11	Phyllis Foss	Phyllis	Foss	phyllis.foss@phyliss.com	F	09.06.1998	3317440000000000	14.080415	
12	12	Kaylyn Jacob	Kaylyn	Jacob	kylynn.jacob@kylynn.com	F	23.02.1993	3304280000000001	37.2627170	
13	13	Esteban Revere	Esteban	Revere	esteban.revere@estebans.cool	F	29.04.1997	3302230000000000	93.0389997	
14	14	Karen Howard	Karen	Howard	kaearen@howards.com	F	27.06.1997	3307902311470000	90.1607071	
15	15	Delois Derry	Delois	Derry	delois.derry@delois.com	M	32.07.2001	3344012194700100	17.2344811	
16	16	Isoma Sampi	Isoma	Sampi	isoma.sampi@isoma.com	M	06.11.1996	334200042260450	6.1700003	
17	17	Ianita Van der Kist	Ianita	Van der Kist	ianita.vander.kist@vander.kist	M	22.06.1998	33067800730115	90.0370007	
18	18	Hector Hockman	Hector	Hockman	hector.hockman@hectors.co.uk	F	28.06.1994	33161875025130	52.0564308	
19	19	Isidore Hixson	Isidore	Hixson	isidore.hixson@isidore.com	F	18.02.1993	3326160047000000	4.7176407	
20	20	John Hobbsen	John	Hobbsen	johobbsen@johns.org	M	23.06.1996	3308610710000000	44.1011747	

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.

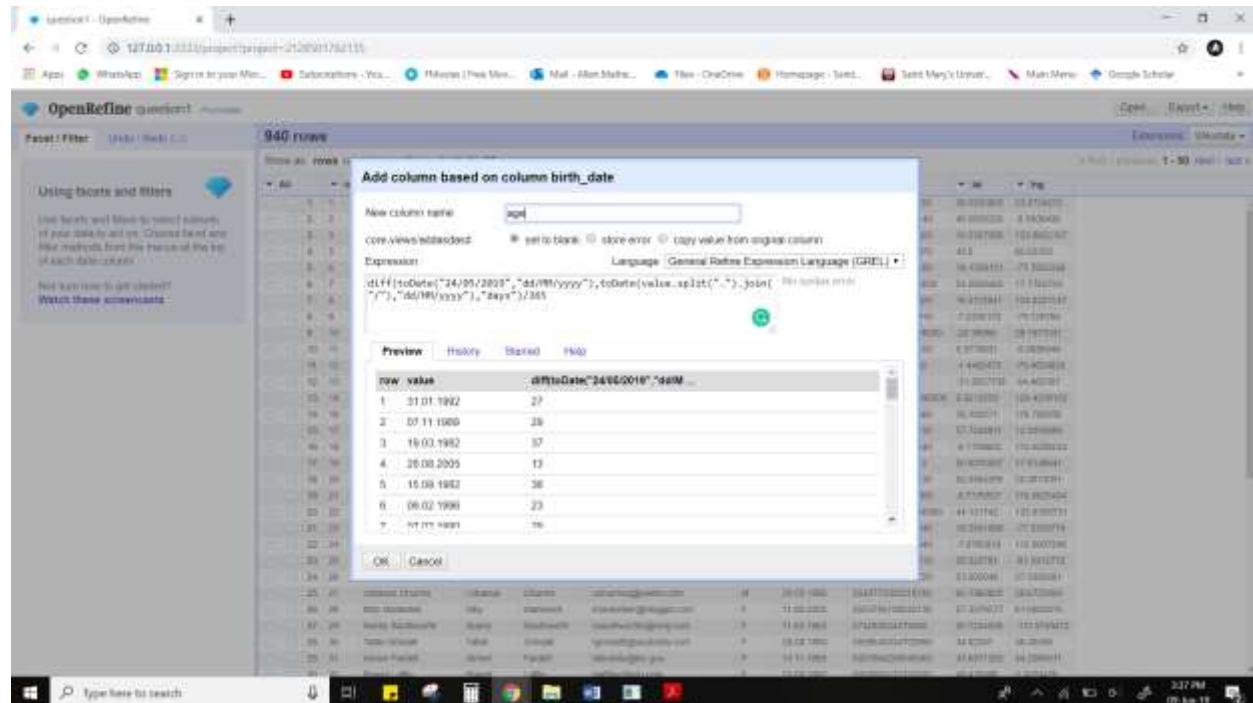


Fig 23: Expression to derive age from birthdate

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** column.

### 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 4<sup>th</sup> 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.

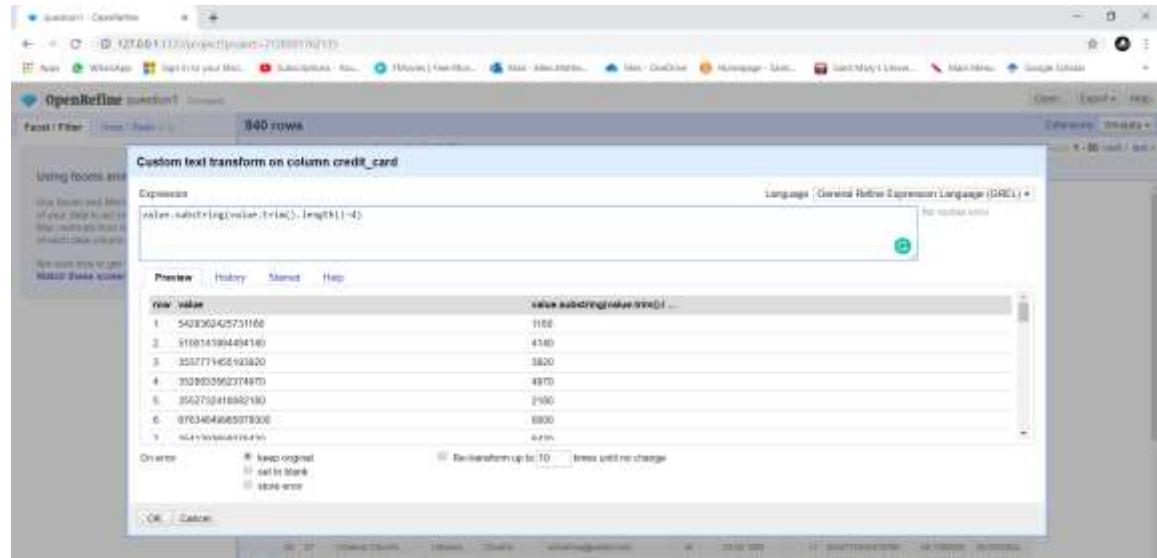


Fig 26: Expression to transform credit card string to the last 4 digits of the credit card

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**).

The screenshot shows the OpenRefine interface with a table containing 940 rows. The 'full\_name' column is highlighted in blue, indicating it is selected for modification. A context menu is open over this column, with the option 'Remove the column' highlighted in red. Other visible options in the menu include 'Add column based on this column', 'Add column based on previous column', 'Add column by filtering USA...', 'Add column from associated pages', 'Remove this column', and 'Drop this column'.

Fig 28: Selecting Remove the column within Edit Column from full\_name column.

The screenshot shows the OpenRefine interface after the 'full\_name' column has been successfully removed. The table now contains 940 rows and 11 columns. The previously selected 'full\_name' column is no longer present in the header or body of the table.

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**).

The screenshot shows the OpenRefine interface with a table containing 940 rows. A context menu is open over the 'birth\_date' column header, with the 'Remove the column' option highlighted in blue. Other options visible in the menu include 'Split into several columns', 'Add column based on this column', 'Add column by filtering URLs', 'Add column by selecting values...', 'Rename this column', and 'Remove this column'.

Fig 30: Selecting Remove the column within Edit Column from birth\_date column.

The screenshot shows the same OpenRefine interface after the 'birth\_date' column has been successfully deleted. The table now has 940 rows and the columns listed are 'id', 'name', 'title', 'Lname', 'email', 'sex', 'age', 'credit\_card', 'ret', and 'mg'. The 'birth\_date' column is no longer present in the list.

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.

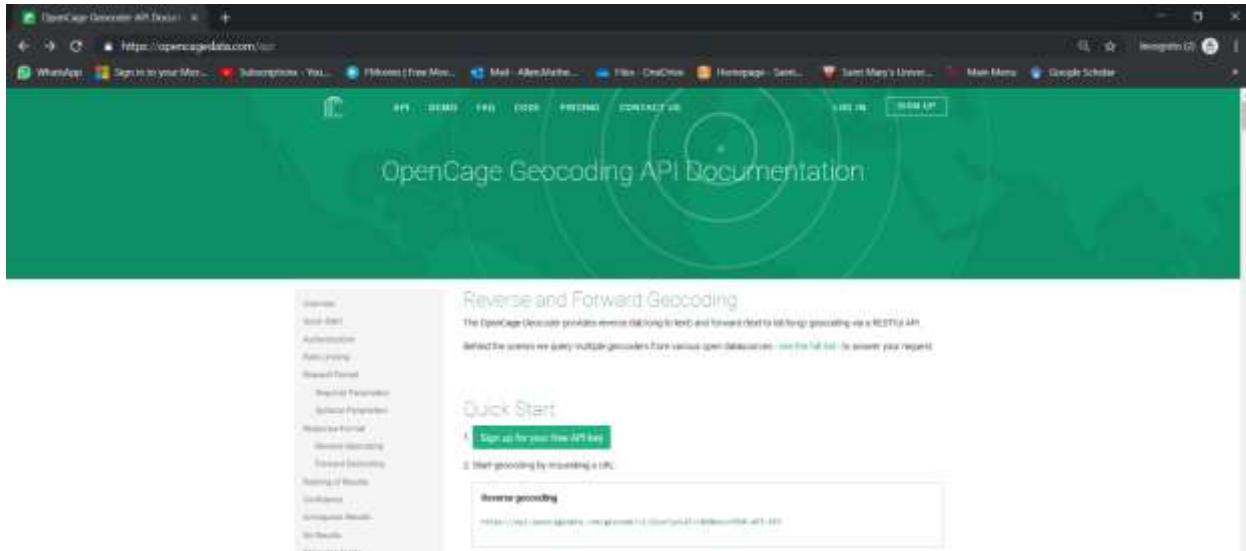


Fig 32: Open Cage Geocoding API Account/ API Key.

## 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).

A screenshot of the OpenRefine interface. A table with 17 rows and columns for 'id', 'name', 'lat', 'long', etc. is displayed. A context menu is open over the 'lat' column, specifically at the row for 'Derek'. The menu options include 'Edit cell', 'Edit column', 'Transpose', 'Sort...', 'Value', 'RecordId', and several submenu items: 'Split into several columns', 'Add column based on the column...', 'Add column by fetching URLs...', 'Add column from recorded values', 'Rename this column', 'Remove this column', 'Move column to beginning', 'Move column to end', 'Move column left', and 'Move column right'. The 'Add column by fetching URLs...' option is highlighted in red.

Fig 33: Add column by fetching URL within Edit Column from **lat** 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 **Ing** 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.

The screenshot shows the OpenRefine interface with the 'Add column by fetching URLs based on column Ing' dialog open. The dialog contains the following configuration:

- New column name: raw\_data
- Throttle delay: 2000 milliseconds
- On error: set to blank, store error
- HTTP Headers to be used when fetching URLs: None
- Cache responses: checked
- Expression: `'https://api.opencagedata.com/geocode/v1/json?q='+cells['lat'].value+'+'+value+'&key=512728198e3f4ebf869f382852c8c949'`
- Language: General Refine Expression Language (GREL)

The preview section displays the first four rows of the raw\_data column, which are JSON objects representing geographical location details. The raw\_data column is then shown in the main workspace.

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

The screenshot shows the OpenRefine interface with the raw\_data column successfully created. The raw\_data column contains JSON data, such as:

```
[{"raw_value": "https://api.opencagedata.com/geocode/v1/json?lat=23.07240272&lon=88.5608436", "value": {"status": "OK", "code": 200, "text": "Geocoding successful", "lat": 23.07240272, "lon": 88.5608436, "label": "Kolkata, West Bengal, India", "country": "India", "country_code": "IN", "locality": "Kolkata", "neighborhood": "Ballygunge", "street": "Ballygunge", "house_number": "10", "city": "Kolkata", "state": "West Bengal", "state_code": "WB", "zip": "700019", "lat": 23.07240272, "lon": 88.5608436}, {"raw_value": "https://api.opencagedata.com/geocode/v1/json?lat=-8.5808436&lon=41.005328", "value": {"status": "OK", "code": 200, "text": "Geocoding successful", "lat": -8.5808436, "lon": 41.005328, "label": "Mysore, Karnataka, India", "country": "India", "country_code": "IN", "locality": "Mysore", "neighborhood": "Mysore", "street": "Mysore", "house_number": "10", "city": "Mysore", "state": "Karnataka", "state_code": "KA", "zip": "560001", "lat": -8.5808436, "lon": 41.005328}, {"raw_value": "https://api.opencagedata.com/geocode/v1/json?lat=123.8822107&lon=9.3307394", "value": {"status": "OK", "code": 200, "text": "Geocoding successful", "lat": 123.8822107, "lon": 9.3307394, "label": "Kochi, Kerala, India", "country": "India", "country_code": "IN", "locality": "Kochi", "neighborhood": "Kochi", "street": "Kochi", "house_number": "10", "city": "Kochi", "state": "Kerala", "state_code": "KL", "zip": "682001", "lat": 123.8822107, "lon": 9.3307394}, {"raw_value": "https://api.opencagedata.com/geocode/v1/json?lat=62.633353&lon=43.215208", "value": {"status": "OK", "code": 200, "text": "Geocoding successful", "lat": 62.633353, "lon": 43.215208, "label": "Tartu, Tartu County, Estonia", "country": "Estonia", "country_code": "EE", "locality": "Tartu", "neighborhood": "Tartu", "street": "Tartu", "house_number": "10", "city": "Tartu", "state": "Tartu County", "state_code": "TT", "zip": "5000", "lat": 62.633353, "lon": 43.215208}], ...]
```

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**).

The screenshot shows the OpenRefine interface with the following details:

- Facet / Filter:** Under 'Rows' tab.
- Using facets and filters:** A sidebar with instructions on how to use facets and filters.
- Facets:** A list of facets including 'Facet', 'Text filter', 'Edit cells', 'Edit column', 'Split into several columns', 'Transpose', 'Sort', 'View', and 'Records'.
- Add column:** A dropdown menu with 'Add column based on this column' selected.
- Records:** A list of rows with their raw data values.
- Preview:** A preview window showing the extracted 'continent' and 'country' values for the first few rows.

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.

The screenshot shows the OpenRefine interface with the following details:

- Facet / Filter:** Under 'Rows' tab.
- Using facets and filters:** A sidebar with instructions on how to use facets and filters.
- Add column based on column raw\_data:** A dialog box for creating a new column 'country'.
- New column name:** 'country'.
- com.views.html.added:** 'set to blank' (radio button selected).
- Expression:** 'value.parseJson().results[0].components.country'.
- Language:** 'General Refine Expression Language (GREL)'.
- Preview:** A preview window showing the extracted 'country' values for the first few rows.

Fig 37: Expression to extract country data from JSON data

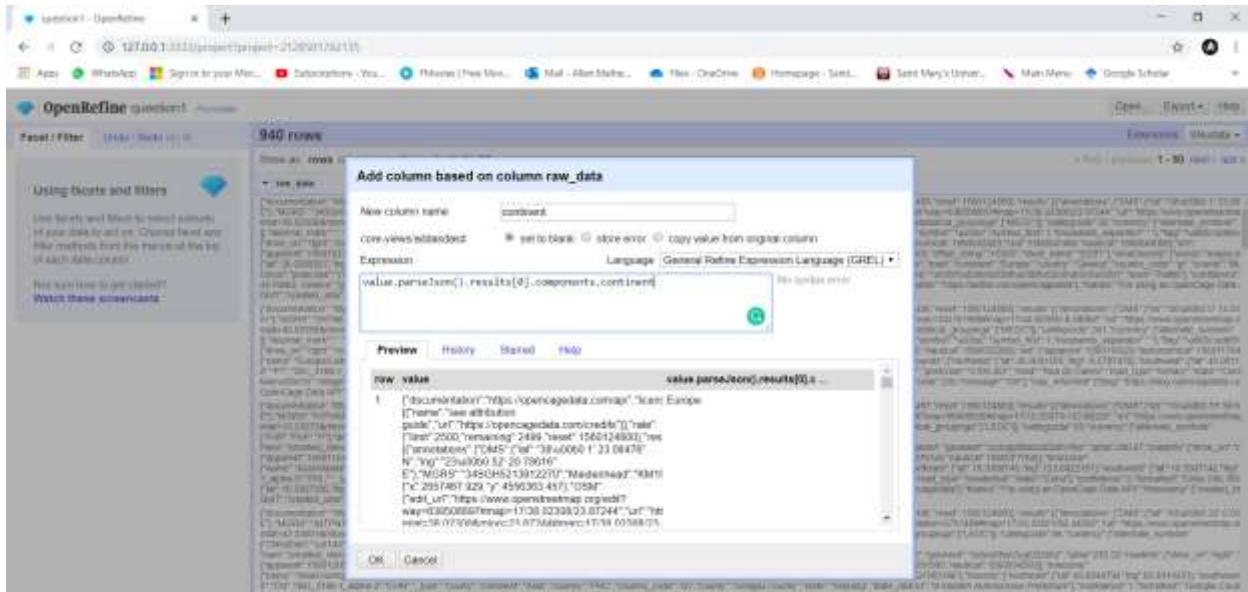


Fig 38: Expression to extract continent data from JSON data

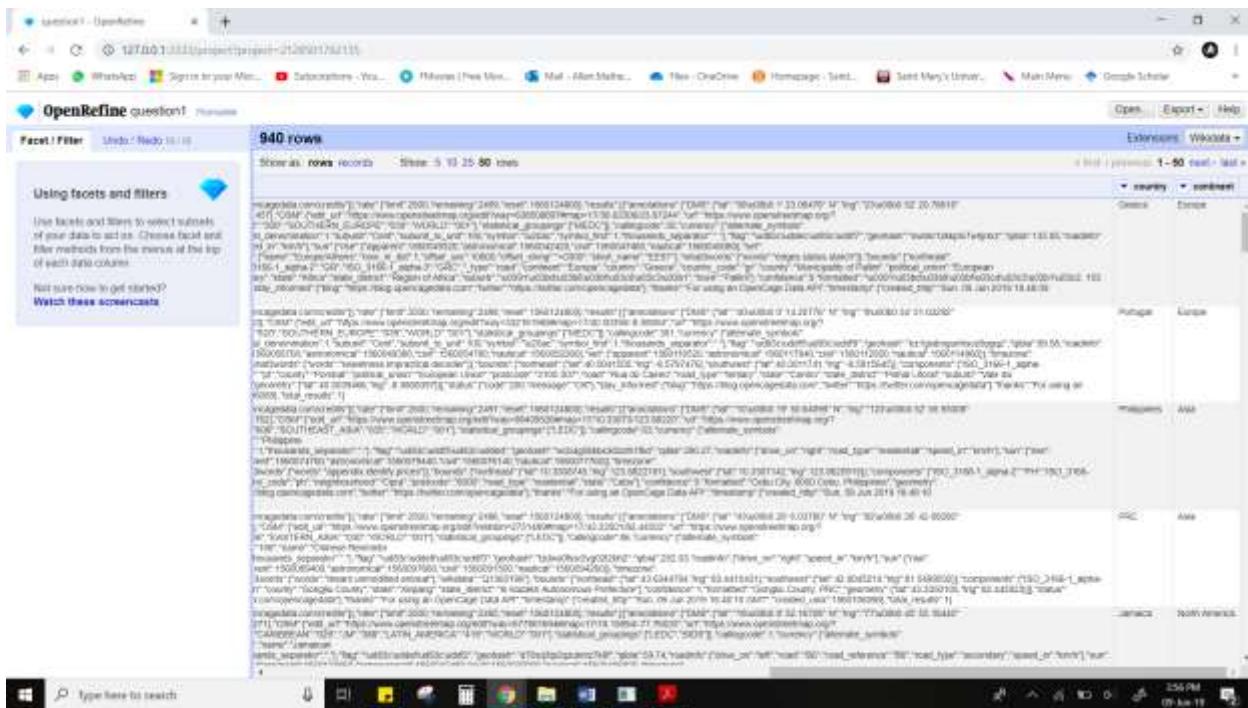


Fig 39: The continent and country column were successfully created

After performing a quick overview, it is evident that the country and continent 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**).

The screenshot shows the OpenRefine interface with the 'Edit Column' menu open. The menu path is 'Edit > Remove the column'. The main workspace displays 940 rows of data with columns labeled 'id', 'name', 'age', 'sex', 'email', 'ssn', 'zip', 'country', and 'continent'. A tooltip from the 'Edit Column' menu provides a detailed description of the 'Remove the column' option.

**Remove the column**

Removes the selected column from the dataset. This operation is irreversible. If you want to keep a copy of the data, consider creating a backup or cloning the dataset before proceeding.

Fig 40: Selecting Remove the column within Edit Column from raw\_data column.

The screenshot shows the OpenRefine interface after the 'raw\_data' column has been successfully removed. The 'Edit Column' menu is no longer open. The main workspace now displays 940 rows of data with columns labeled 'id', 'name', 'age', 'sex', 'email', 'ssn', 'zip', 'country', and 'continent'. The data rows are identical to the previous state, but the column headers have been updated.

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**).

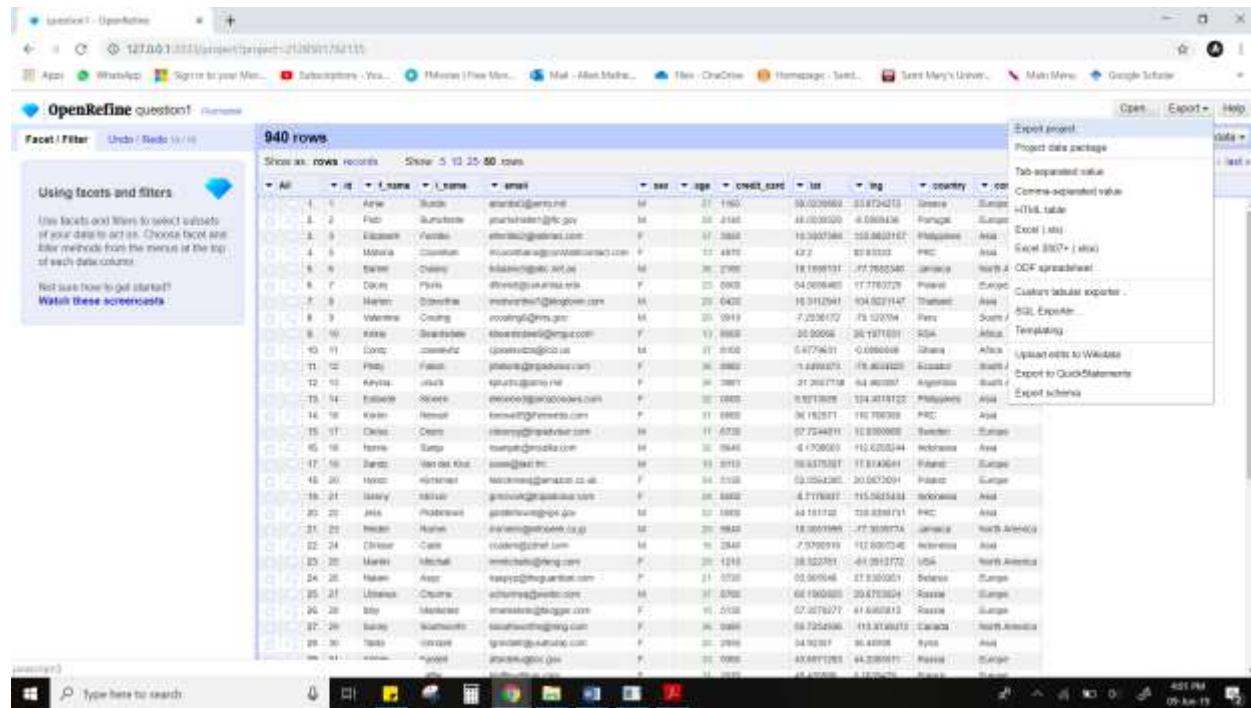


Fig 42:Exporting the OpenRefine Project as tar.gz file

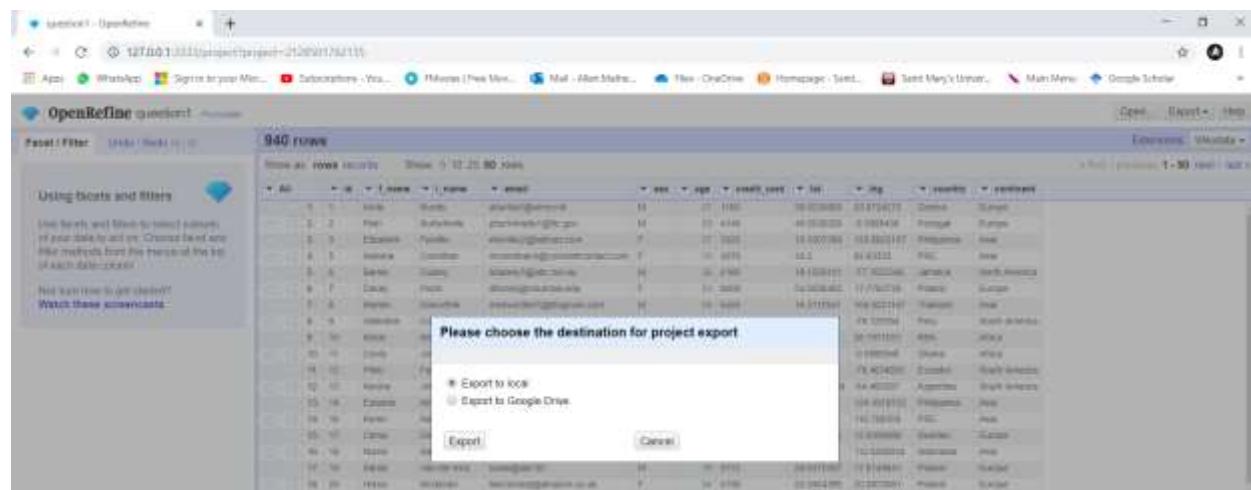


Fig 43: Exporting project to local device

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

The screenshot shows the OpenRefine interface with a dataset containing 940 rows. The columns include: id, first name, last name, email, sex, age, credit score, tel, lang, country, and zip. The interface features a sidebar for facets and filters, a main table view with sorting and filtering options, and a top navigation bar with various export and project management buttons.

Fig 44:Export to CSV file as .csv

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

The screenshot shows the Windows File Explorer interface with the 'Downloads' folder open. Inside the folder are two files: 'solution1.tar.gz' and 'solution1.csv'. The 'solution1.csv' file is highlighted, indicating it has been recently selected or is the current item.

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

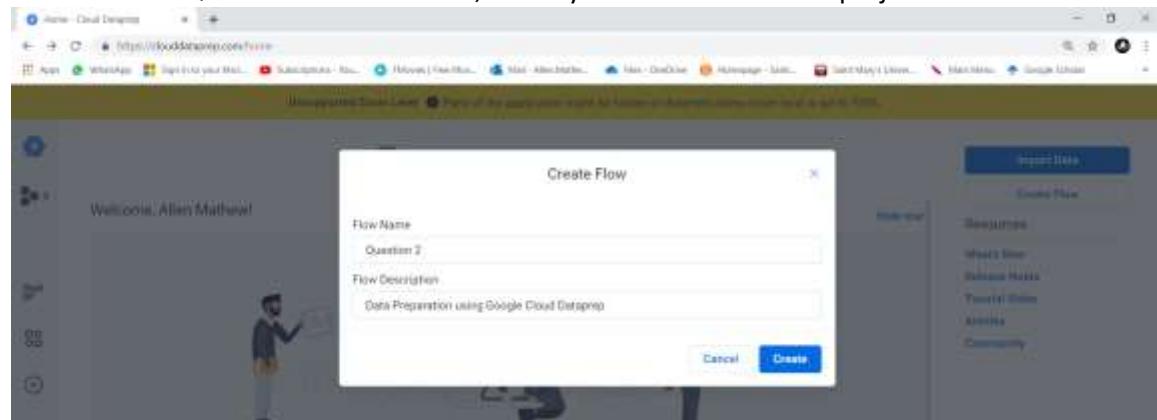


Fig 47: A flow is created with title as Question 2.

## 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.

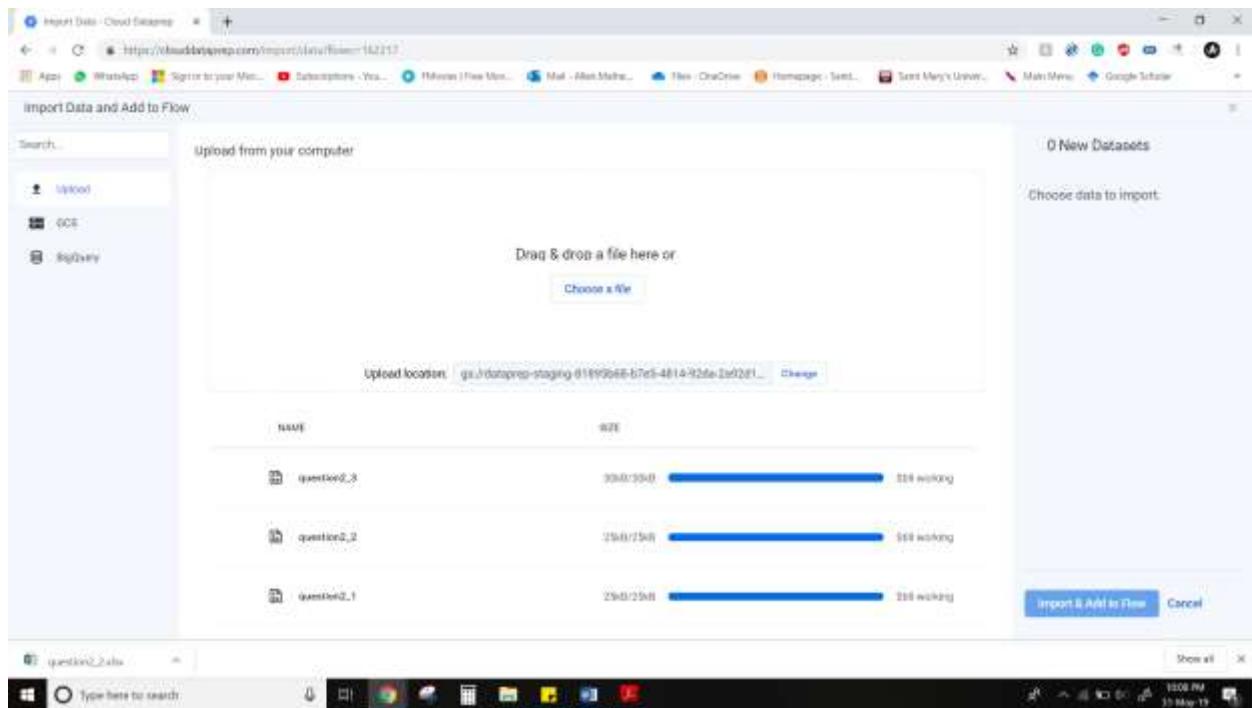


Fig 48: Imported Files.

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

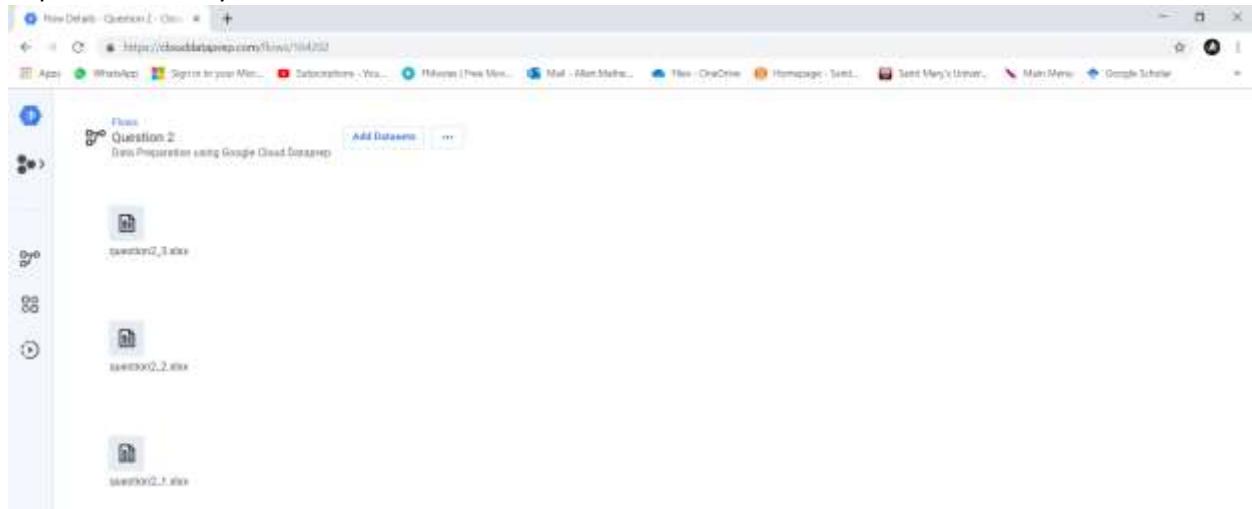


Fig 49: Dataset is uploaded to the flow

### 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.

The screenshot shows the Google Cloud Dataflow interface. On the left, there's a sidebar with a tree view of datasets: 'question2\_3' (selected), 'question2\_3.xlsx', and 'question2\_3.csv'. The main area shows a flow diagram with two nodes: 'question2\_3.xlsx' and 'question2\_3'. A 'Details' panel is open for 'question2\_3.xlsx', displaying its schema:

Column	Type
stock_symbol	string
stock_market_cap	string

The data preview shows the following rows:

stock_symbol	stock_market_cap
AAV	\$1.15B
ABG	\$1.15B
ABA	\$31.94M
ABR	\$31.98M
ACTA	n/a
ADH-AT	\$31.97M

Below the schema, the 'Location' is listed as 'gs://dataprep-staging-81195b68-b7e5-4814-02da-3a93f1a2d090/alexmatthew20@gmail.com/upload/7951d908-5fed-4c72-8e01-e30451e8f38/176db792-6334-483d-9442-1999fa0f61e' and the 'File Size' is '15.19KB'.

Fig 50: Reviewing Column Details for dataset question2\_3.

To clean the dataset a new recipe is added to the question2\_3 dataset.

The screenshot shows the Google Cloud Dataflow interface. The 'question2\_3' dataset now has a blue circular icon next to it, indicating it has a recipe. The 'Details' panel for 'question2\_3.xlsx' shows a 'New Recipe' button. The 'Recipe' tab is selected, showing a 'Steps Preview' section with the message 'No Recipe Steps. Edit the Recipe to wrangle.' Below this, the 'Steps' count is '0', and the 'Updated' and 'Created' times are 'Today at 4:40 PM'.

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.

A screenshot of a data grid interface. The top navigation bar shows 'question2\_3' and 'Full Data'. The grid has two columns: 'stock\_symbol' and 'stock\_market\_cap'. The data rows include: NCV Categories, stock\_symbol, stock\_market\_cap; AAV, \$1.12B; AMB, \$1.12B; AETL, \$21.94B; AEW, \$21.94B; ACTB, n/a; ADHD, \$20.37B; ADT, \$29.38B; ALGM, \$3.16B; AEG, \$16.31B; AET, \$49.43B; AETL, \$12.31B; AFH, n/a; AFH, \$188.28B; AFH, n/a; AFH, n/a; AFH, n/a; AGD, \$2.58B; AGLF, \$47.88B; AGR-B, n/a; AGR-A, n/a. The 'stock\_symbol' column is bolded.

stock_symbol	stock_market_cap
NCV Categories	
stock_symbol	
AAV	\$1.12B
AMB	\$1.12B
AETL	\$21.94B
AEW	\$21.94B
ACTB	n/a
ADHD	\$20.37B
ADT	\$29.38B
ALGM	\$3.16B
AEG	\$16.31B
AET	\$49.43B
AETL	\$12.31B
AFH	n/a
AFH	\$188.28B
AFH	n/a
AFH	n/a
AGD	\$2.58B
AGLF	\$47.88B
AGR-B	n/a
AGR-A	n/a

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

A screenshot of a data grid interface. The top navigation bar shows 'question2\_3' and 'Full Data'. The grid has two columns: 'stock\_symbol' and 'stock\_market\_cap'. The data rows are identical to Fig 52. The 'stock\_symbol' column is bolded.

stock_symbol	stock_market_cap
NCV Categories	
stock_symbol	
AAV	\$1.12B
AMB	\$1.12B
AETL	\$21.94B
AEW	\$21.94B
ACTB	n/a
ADHD	\$20.37B
ADT	\$29.38B
ALGM	\$3.16B
AEG	\$16.31B
AET	\$49.43B
AETL	\$12.31B
AFH	n/a
AFH	\$188.28B
AFH	n/a
AFH	n/a
AGD	\$2.58B
AGLF	\$47.88B
AGR-B	n/a
AGR-A	n/a

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

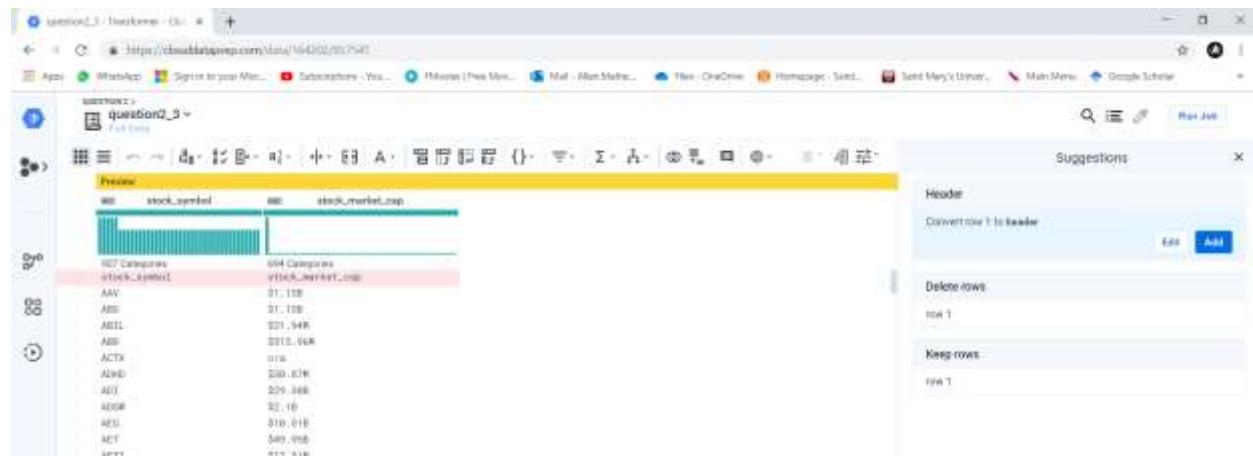
### 3.1) Requirements for question2\_3 Dataset :

Various steps need to be performed on the question2\_3 dataset:

- The first row needs 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

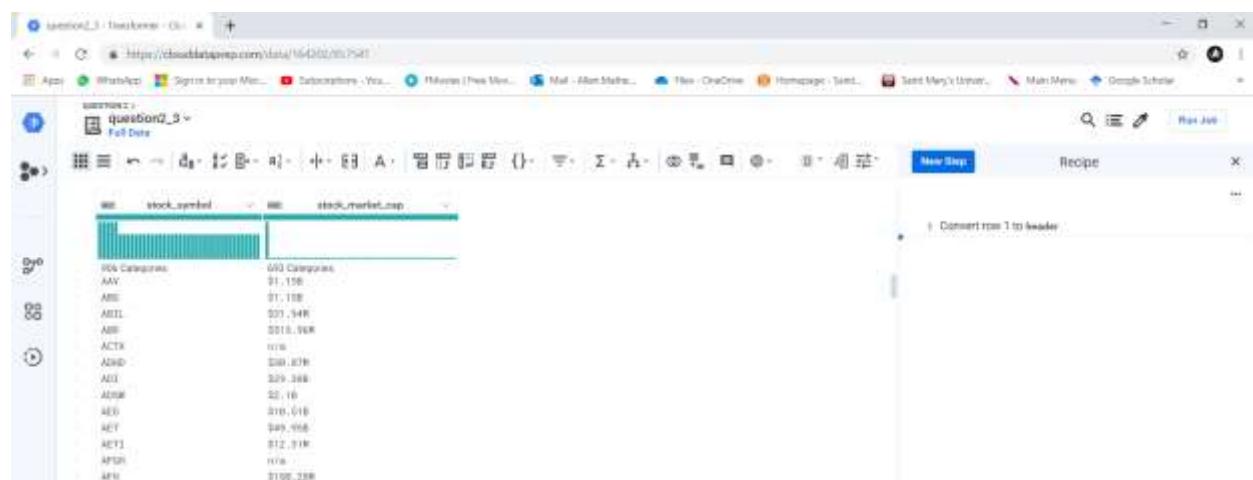
On selecting the first row in the dataset, Google Data Prep automatically provides several suggestions. The “Header” option that was suggested is added to the recipe, it converts the first row to the header.



The screenshot shows the Google Data Prep interface with a dataset named 'question2\_3'. The 'Header' step is selected on the right side of the screen. A tooltip 'Convert row 1 to header' is visible above the 'Edit' button. The main area displays the dataset with two columns: 'stock\_symbol' and 'stock\_market\_cap'. The first row, which contains the values 'MSO' and '51.15B', is highlighted in pink and serves as the header. The subsequent rows show various stock symbols and their market caps in billions.

stock_symbol	stock_market_cap
MSO	51.15B
AAP	31.15B
ABIL	321.94K
ABIF	321.94K
ACTX	0.16
ADHD	228.82B
AEG	229.34B
AEMB	52.16
AESI	310.01B
AET	549.95B
AFT	312.51B

Fig 54: Assigning the 1<sup>st</sup> row of the dataset as header



The screenshot shows the Google Data Prep interface with the 'Header' step completed. The 'Header' step is now listed under 'Recent Steps' at the top. The main area shows the dataset with the first row ('MSO' and '51.15B') now serving as the header for the 'stock\_symbol' and 'stock\_market\_cap' columns respectively. The subsequent data rows are displayed below.

stock_symbol	stock_market_cap
MSO	51.15B
AAP	31.15B
ABIL	321.94K
ABIF	321.94K
ACTX	0.16
ADHD	228.82B
AEG	229.34B
AEMB	52.16
AESI	310.01B
AET	549.95B
AFT	312.51B

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.

The screenshot shows a Google Data Prep interface with a dataset named 'question2\_3'. A row containing 'n/a' in the 'stock\_market\_cap' column is selected, highlighted with a pink background. A 'Suggestions' card is open on the right side of the screen, listing several options:

- 'n/a starting after (start) ending before (end)' (with a preview of the row)
- 'n/a starting after (start) ending before (end)' (with a preview of the row)
- 'Split on value matching' (with a preview of the row)
- 'Set' (with a preview of the row)
- 'Keep rows' (with a preview of the row)
- 'Delete rows' (with a preview of the row)

The status bar at the bottom indicates 2 Columns, 986 Rows, and 1 Data Type.

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.

The screenshot shows the same Google Data Prep interface as Fig 56, but now the 'Delete rows' action has been added to the recipe. The 'Rows' section of the recipe editor shows a condition: 'IF ISNULL([stock\_market\_cap]) = TRUE'. The 'Action' dropdown is set to 'Delete matching rows'.

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

The screenshot shows a data transformation interface. A tooltip on the right side of the screen says:

- 1 Convert from 1 to header
- 2 Delete rows where MATCHES@stock\_market\_cap = 'n/a'

stock_symbol	stock_market_cap
TSLA	692 Categories
AMZN	31.15B
GOOGL	31.14B
GOOGL	31.13B
AAPL	31.13B
ADBE	32.67B
MSFT	32.36B
ADBE	32.1B
AMZN	310.61B
ACT	303.76B
ACT	312.21B
AFH	316.23B
AGD	32.5B
AGLE	347.08B
AGRE	39.12B
AGF	314.16B
AGRO	31.21B
AMRO	31.21B
AMRO	31.21B
AHP	3212.61B
AHP	38.91B
AZ	3352.44B
AIA	3305.98B
AIF	3236.63B

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.

The screenshot shows a search results panel for 'Remove' transformations. The 'Remove duplicate rows' option is highlighted.

Remove
Remove duplicate rows
Remove whitespace from text
Remove accents from text
Remove symbols from text
Function REMOVEWHITESPACE
Function REMOVESYMBOLS
Trim whitespace
Filter rows when value ends with
Filter rows when value starts with

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.

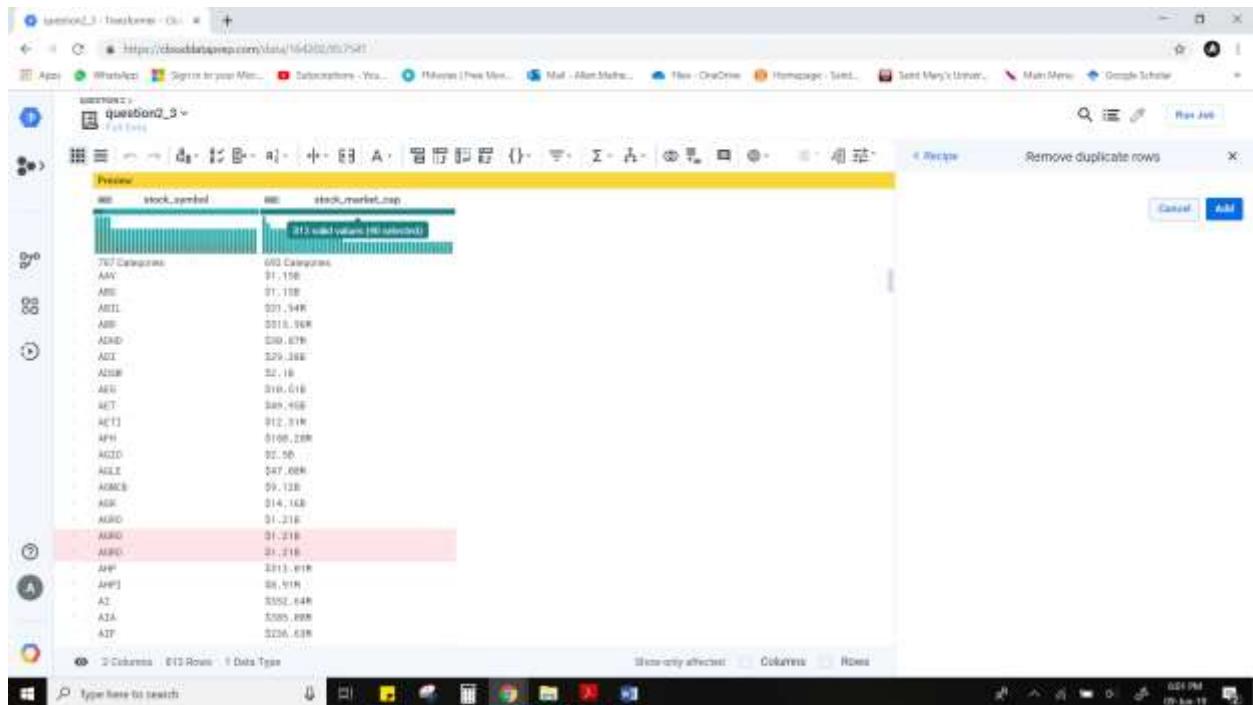


Fig 60: There are 46 rows in the dataset that consist of duplicate records.

All duplicate rows were successfully deleted.

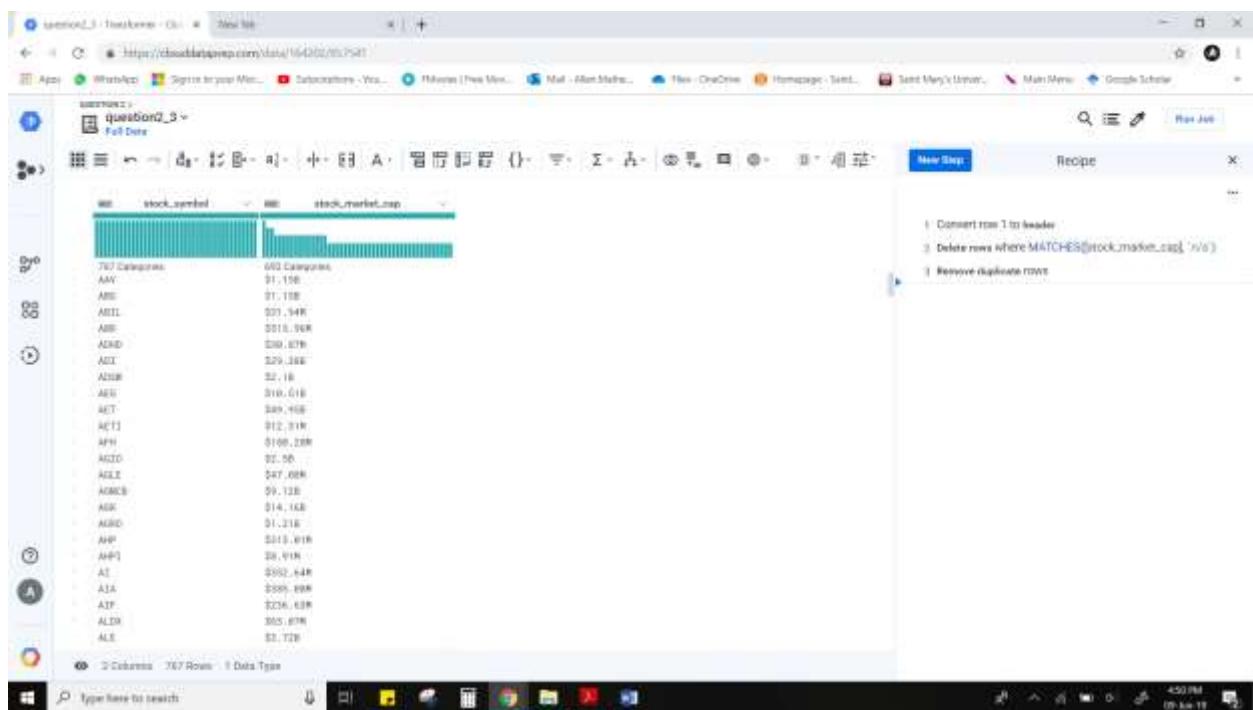


Fig 61: All rows with duplicate records were removed form the dataset

#### 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 complete 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.

The screenshot shows a data processing interface with a preview of a table. The table has three columns: stock\_symbol, stock\_market\_cap, and stock\_market\_numeric. The stock\_symbol column lists various stock symbols. The stock\_market\_cap column contains values like '\$1.15B' and '\$1.15M'. The stock\_market\_numeric column contains the corresponding numeric values: 1150000000.0 and 1150000.0 respectively. To the right, a 'Conditions' panel is open, showing the logic for generating the stock\_market\_numeric column. It uses an 'If... Then...' statement where the condition is 'MATCHES(stock\_market\_cap, 'B')'. The 'Then' part of the statement is 'ROUND(SUBSTRING(stock\_market\_cap, 1, LEN(stock\_market\_cap) - 1) \* 1000000000, 0)'. The 'Else' part is 'ROUND(SUBSTRING(stock\_market\_cap, 1, LEN(stock\_market\_cap) - 1) \* 1000000, 0)'. A 'New columnname' field is set to 'stock\_market\_numeric'.

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

The stock\_market\_numeric column was successfully created, and the string was converted to its numeric equivalent.

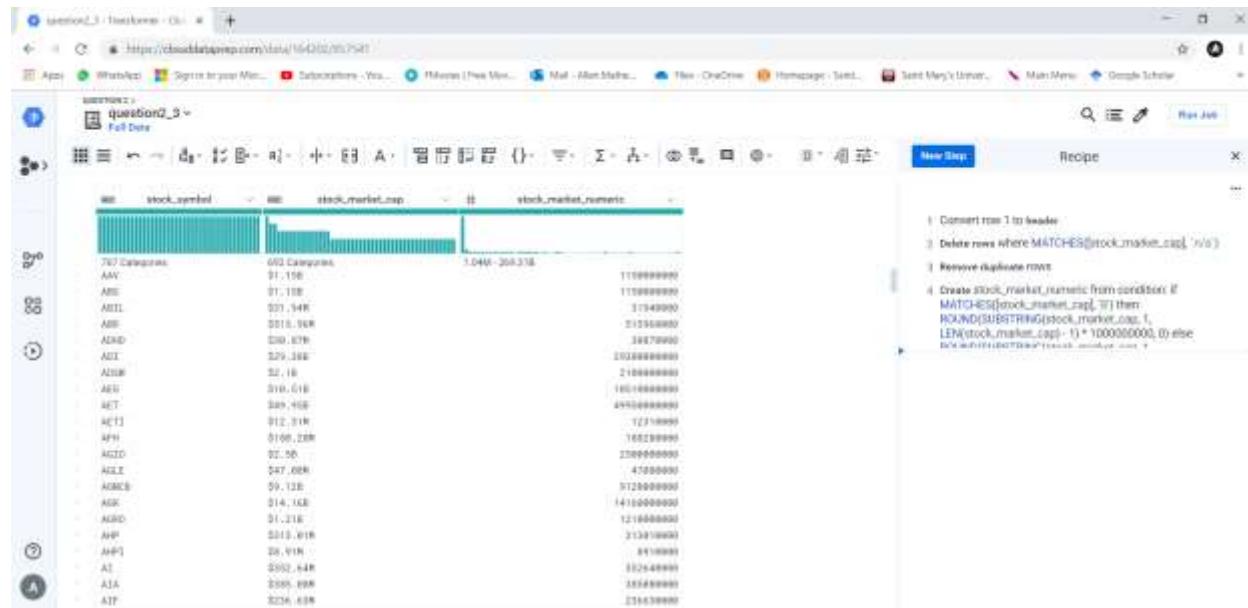


Fig 63: stock\_market\_numeric column was successfully created.

## 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.

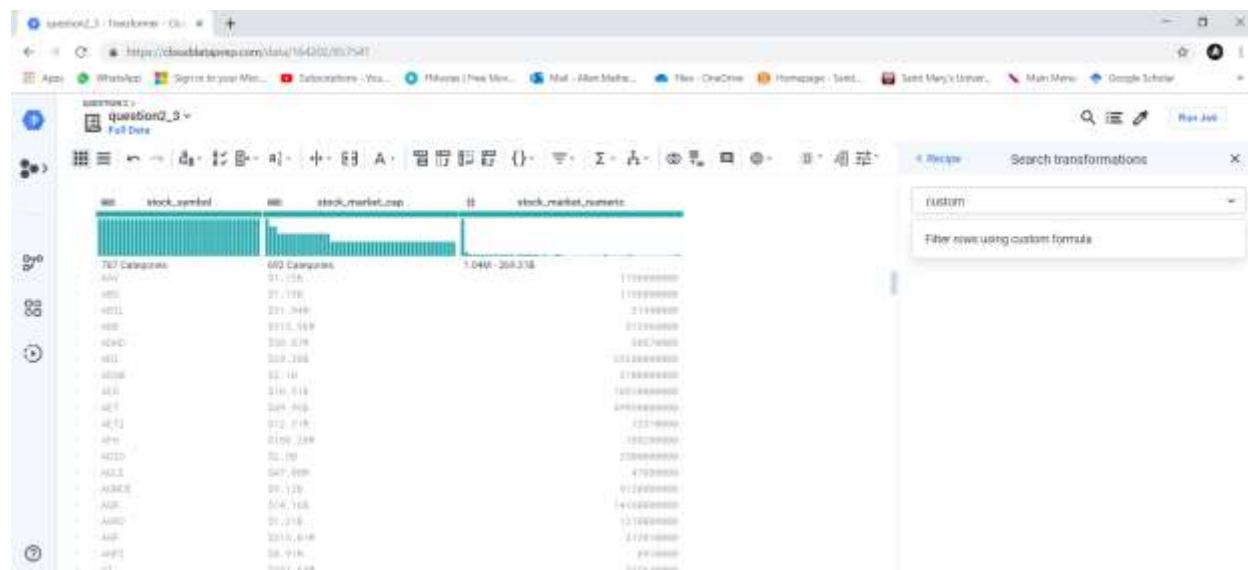


Fig 64: Searching for Filter rows using custom formula option

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

The screenshot shows a Google Cloud Data Studio dashboard titled "question2\_3". The main area displays a table with three columns: "stock\_symbol", "stock\_market\_cap", and "stock\_market\_numeric". The "stock\_market\_numeric" column contains values such as 1159999999, 1159999999, 31949999, etc. A sidebar on the right is titled "Recipe" and contains the following configuration:

- Condition:** Custom formula required
- Filter rows that satisfy an arbitrary formula.**
- Type of formula:** Custom single required
- Filter rows that satisfy a single row formula.**
- Condition:** stock\_market\_numeric < 30000000 required
- Action:** Keep matching rows required
- Delete matching rows**

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

The screenshot shows the same Google Cloud Data Studio dashboard as Fig 65. The table data is identical. The Recipe panel now lists the following steps:

- Convert row 1 to header
- Delete rows where MATCHES@stock\_market\_cap, 'n/a'
- Remove duplicate rows
- Create stock\_market\_numeric from condition if MATCHES@stock\_market\_cap, 'n/a' then ROUND(SUBSTRING(stock\_market\_cap, 1, LEN(stock\_market\_cap)-1)\*1000000000, 0) else null when stock\_market\_numeric is null end
- Delete rows where stock\_market\_numeric < 30000000

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.

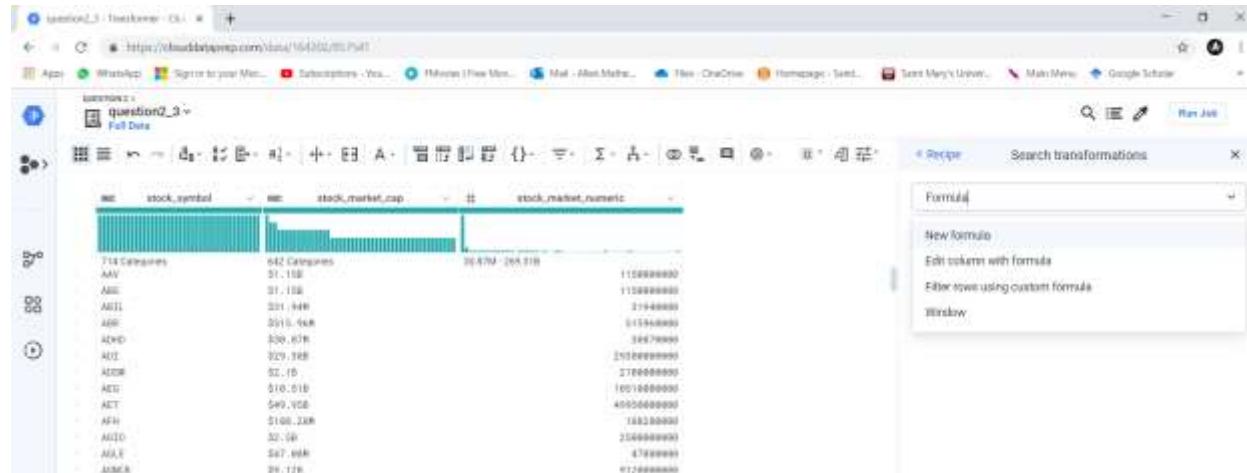


Fig 67: Searching for New formula option

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 consistency 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'])

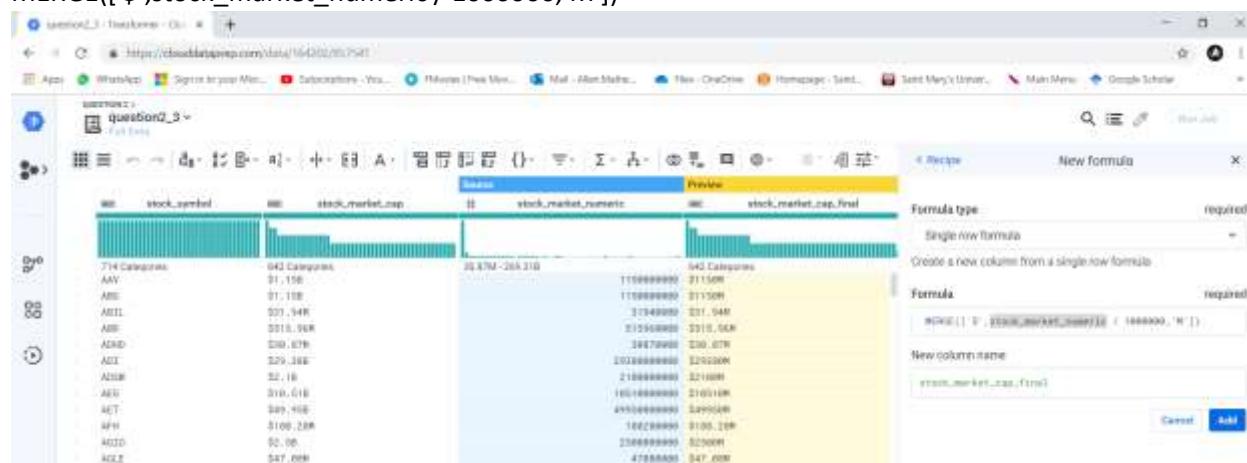


Fig 68:stock market cap final

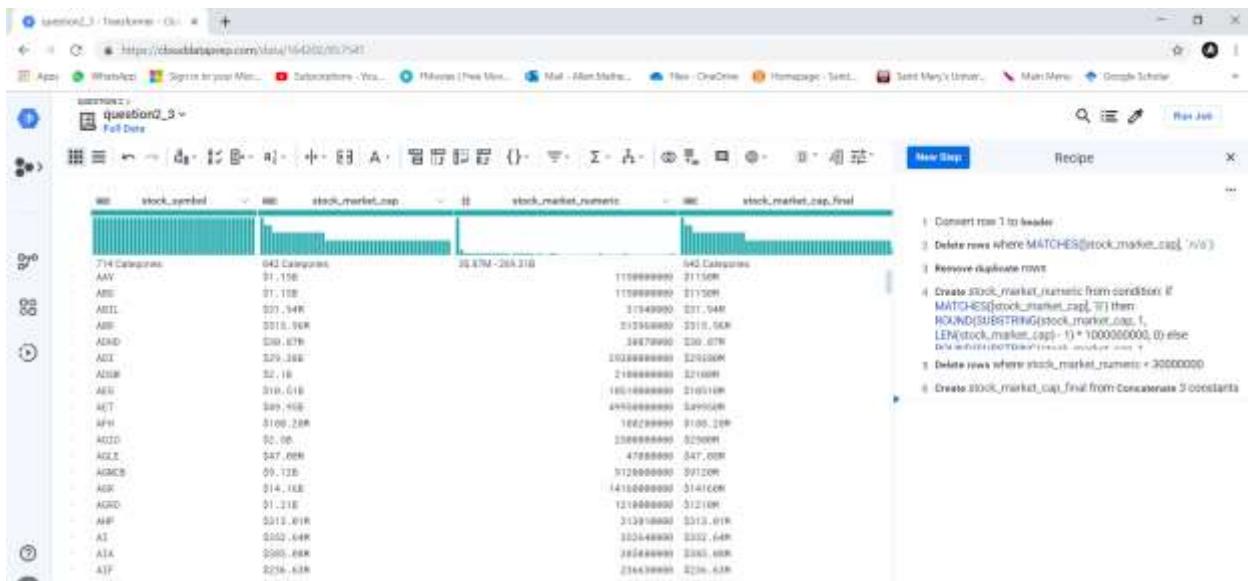


Fig 69: The new column stock\_market\_cap\_final was created successfully

#### 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.

The screenshot shows the Google Cloud Storage interface. On the left, there's a sidebar with navigation icons. The main area displays a list of files:

- question2\_3.csv
- question2\_2.xlsx
- question2\_1.xlsx

Each file has a preview icon and a download icon. To the right, there's a detailed view for "question2\_2.xlsx":

**Details**

**File** question2\_2.xlsx

**Add new Recipe**

**Data Preview**

row	column1	column2	column3
1	stock_symbol	stock_sector	
2	AMV	Energy	
3	ABG	Consumer Durables	
4	ABL	Finance	
5	ABR	Consumer Services	
6	ACTX	n/a	
7	ADHD	Health Care	

**Type** GCS

**Location** gs://f9stprep-staging-81895b69-b7e5-4818-93d9-2a92d1495068@karenmatthew55@gmail.com/v2/17758ee87d054977be6-7221be657007342c0037107-4443-aesa-2db1ca5d74db

**Filesize** 18.72B

*Fig 70: Reviewing Column Details for dataset question2*

To clean the dataset a new recipe is added to the question2\_2 dataset.

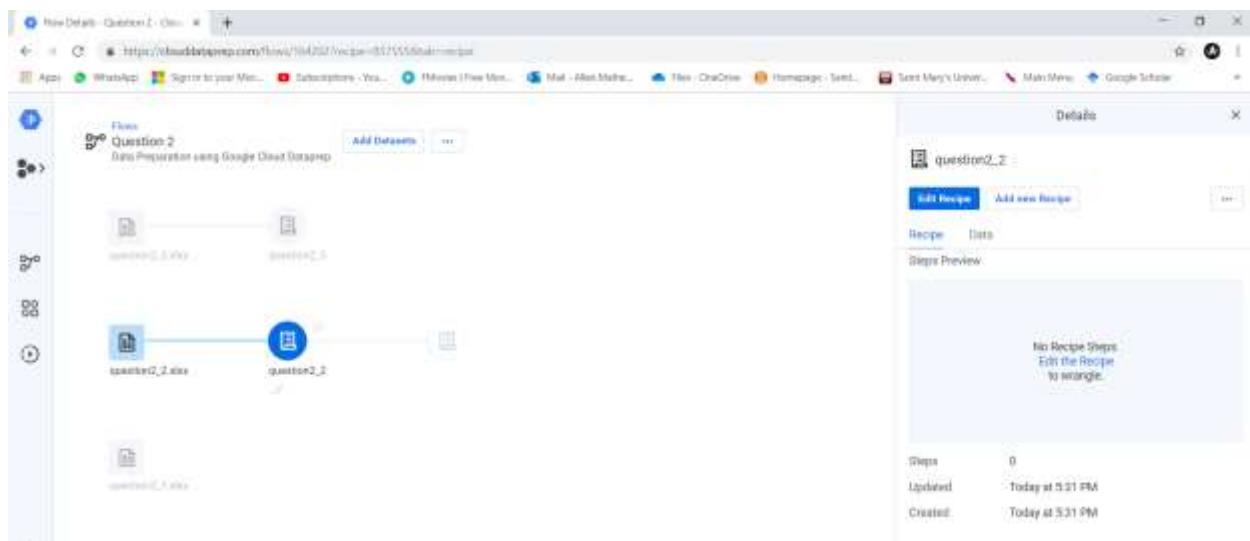


Fig 71: Adding new Recipe

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 1<sup>st</sup> 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.

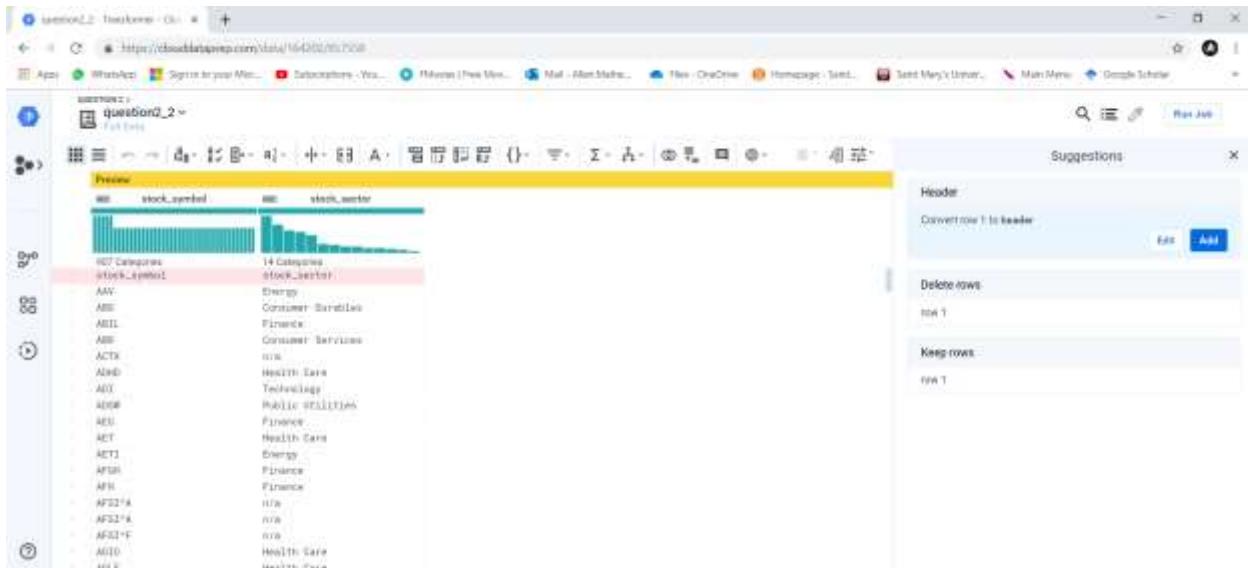


Fig 74: Assigning the 1<sup>st</sup> row of the dataset as header

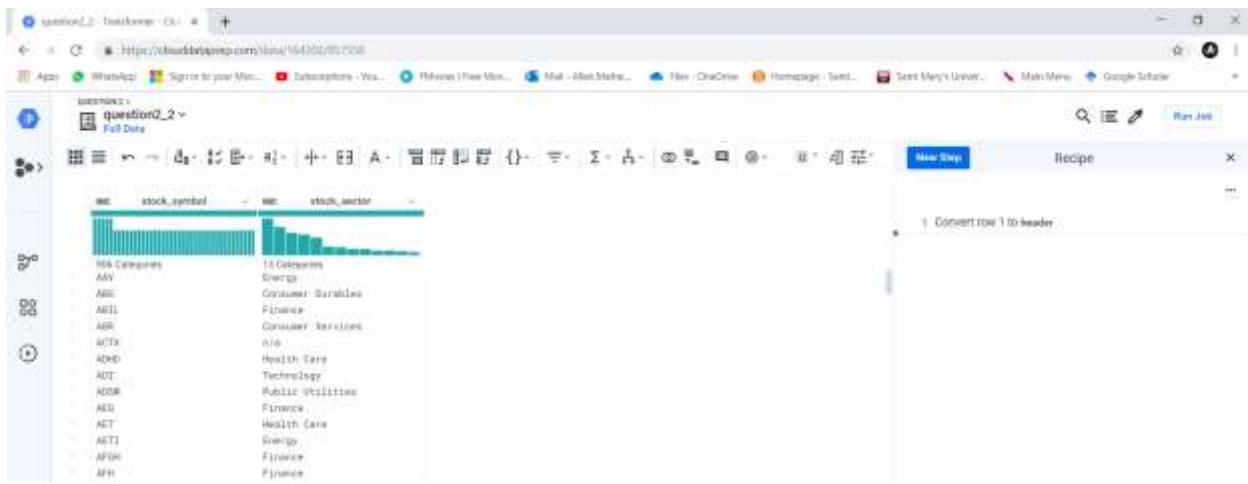


Fig 75: The first row was successfully converted as the headers of the respective columns

## 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.

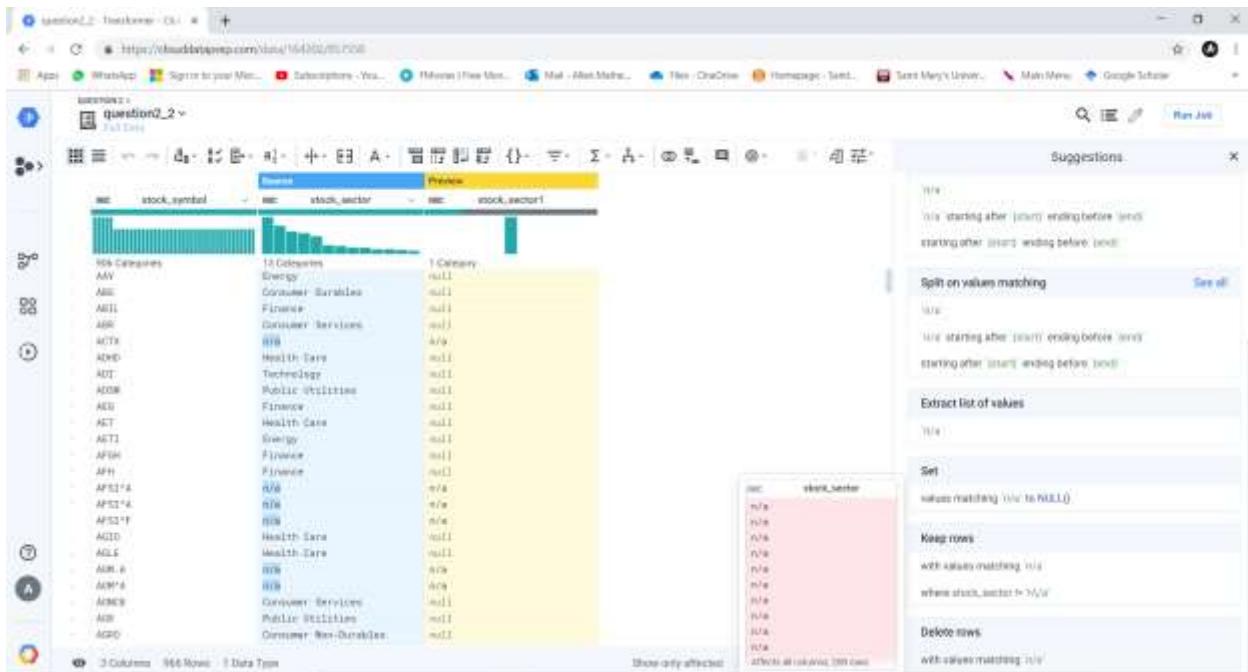


Fig 76: Removing Rows for every occurrence of NA's in stock\_sector column

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.

question2\_2

question2\_2 ~ Last Edit

https://public.tableau.com/views/164302/102088

Ajax WhatsApp Signer to your Mac... Subscriptions - You... iPhone (Free Mail...) Mail - Mail.Mac... OneDrive Homepage - Sam... Sam... Main Menu Google Scholar

Search Run Job

Rows: 1 1000 10000 100000 1000000 10000000

Filter rows

Condition

Custom formula

Filter rows that satisfy an arbitrary formula

Type of formula

Custom single

Filter rows that satisfy a single row formula

Action

Keep matching rows

Delete matching rows

Cancel Add

Preview

Stock Symbol Stock Sector

1000 Rows Selected (2000 Selected)

Stock Symbol	Stock Sector
AMZN	Technology
AMC	Energy
AMBR	Consumer Services
AMCI	Finance
AMER	Consumer Services
AMET	n/a
AMD	Health Care
ADT	Technology
ADOB	Public Utilities
ADBE	Finance
AEAT	Health Care
AETI	Energy
AFGH	Finance
AFH	Finance
AFS21A	n/a
AFS21E	n/a
AFS21F	n/a
AGDQ	Health Care
AGLS	Health Care
AMR_R	n/a
AMPPA	n/a
ANRR	Consumer Services
ASR	Public Utilities
ACPO	Consumer Non-Durables

2 Columns 966 Rows 8 Data Types

Show only affected Board

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

lesson2\_2 - Data Studio - Google Chrome

https://datastudio.google.com/data/16201707208

Apps Sheets Sign-in to your Sheets Subscriptions - YouTube Photos (New) Mail - Alan Morris OneDrive Homepage - Gmail Saint Mary's University Main Menu Google Scholar

question02\_2 - Full Data

New Step Recipe Run Job

stock\_symbol stock\_sector

717 Companies 12 Categories

stock_symbol	stock_sector
AAM	Energy
ABE	Consumer Services
ABIL	Finance
ABR	Consumer Services
ABRD	Health Care
ABX	Technology
ABNB	Public Utilities
ABE	Finance
ACT	Health Care
ACTJ	Energy
AFGR	Finance
AFH	Finance
AGDZ	Health Care
AGLE	Health Care
AGMB	Consumer Services
AKK	Public Utilities
AKD	Consumer Non-Durables
AKD	Consumer Non-Durables
AKRO	Consumer Non-Durables
AKP	Consumer Services
AKP	Health Care
AT	Finance
ALDN	Health Care

2 Columns 718 Rows 0 Data Type

Type here to search

241 PM 10 Aug 19

**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.

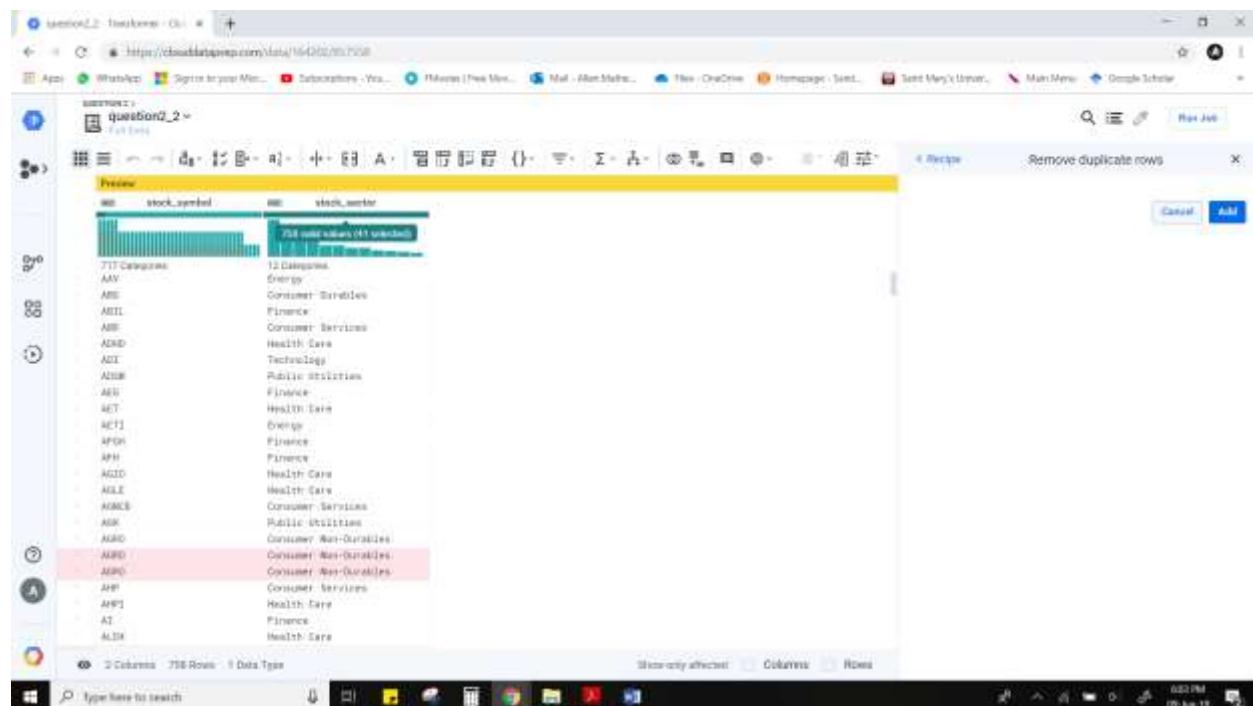


Fig 80: There are 41 rows in the dataset that consist of duplicate records.

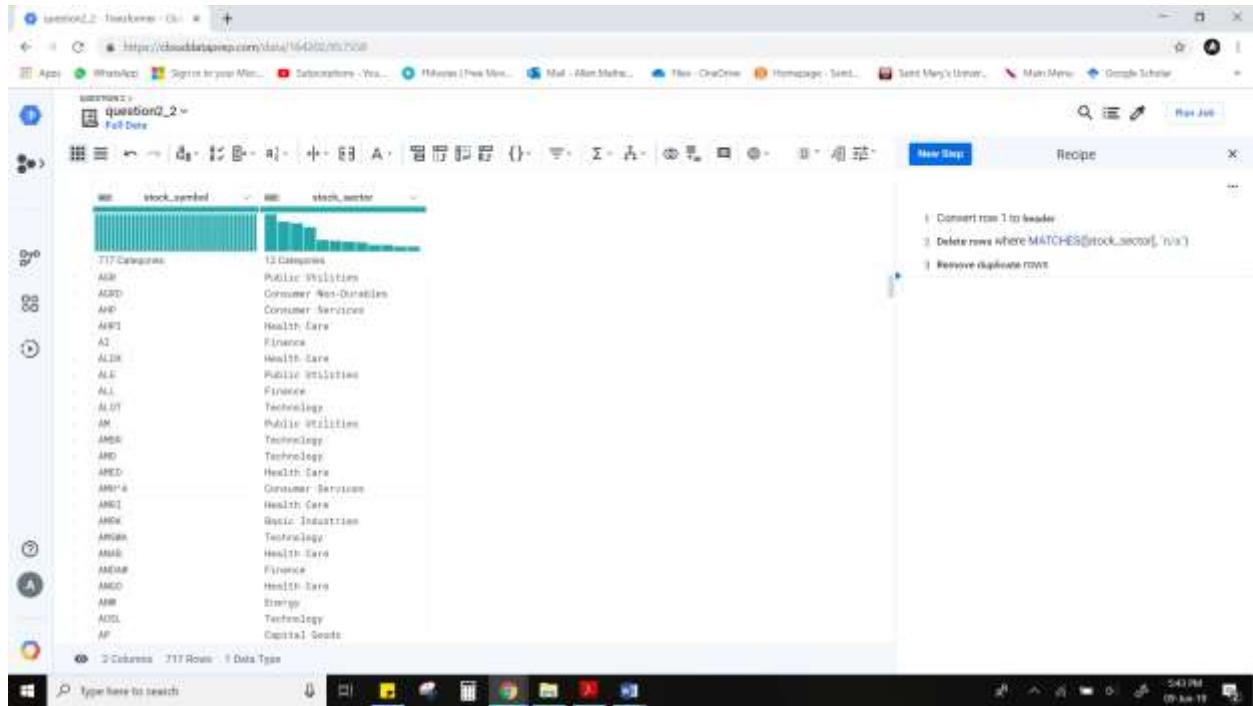


Fig 81: All duplicate rows were successfully deleted.

## 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)

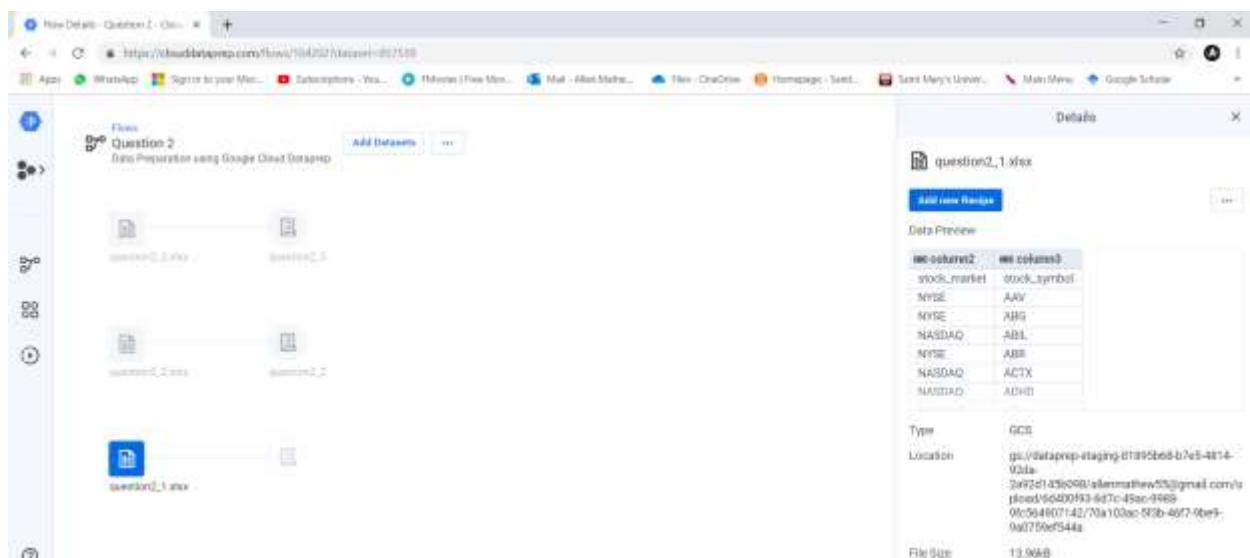


Fig 82: Reviewing Column Details for dataset question2\_2

To clean the dataset a new recipe is added to the question2\_3 dataset.

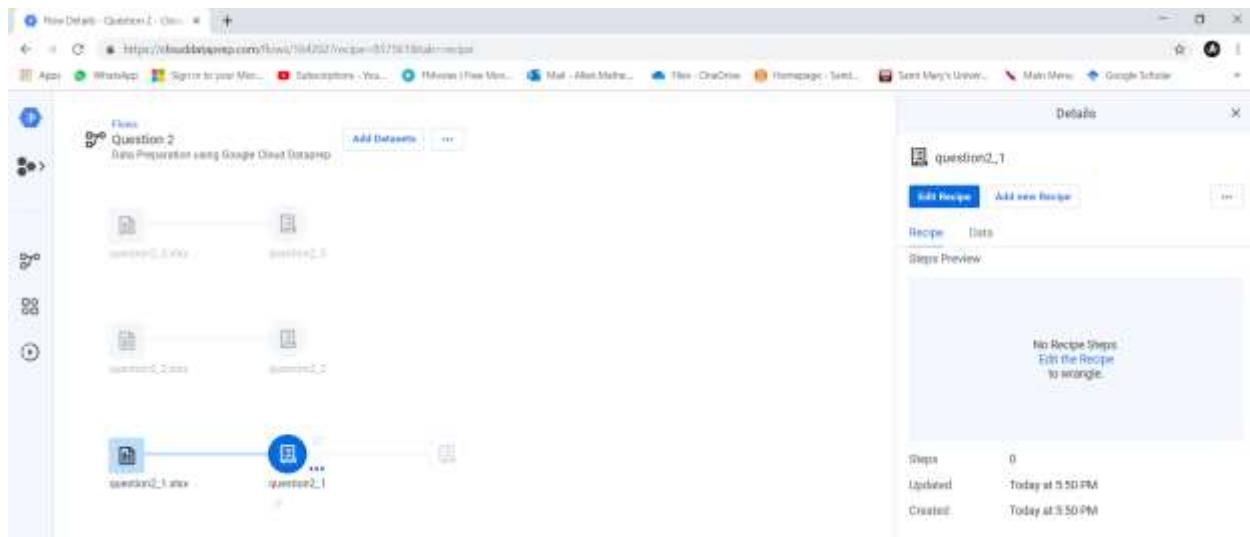


Fig 83: Adding new Recipe

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

stock_market	stock_symbol
NYSE	AAV
NYSE	ABG
NASDAQ	ABIL
NYSE	ABR
NASDAQ	ACTX
NASDAQ	ADBI
NASDAQ	AOL
NYSE	AUDR
NYSE	AVG
NASDAQ	AET
NASDAQ	AKTI
NYSE	APCH
NASDAQ	APIH
NYSE	APSI
NYSE	APSI-A

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 1<sup>st</sup> 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.

stock_market	stock_symbol
NYSE	AAV
NYSE	ABG
NASDAQ	ABIL
NYSE	ABR
NASDAQ	ACTX
NASDAQ	ADBI
NASDAQ	AOL
NYSE	AUDR
NYSE	AVG
NASDAQ	AET
NASDAQ	AKTI
NYSE	APCH
NASDAQ	APIH
NYSE	APSI
NYSE	APSI-A

Fig 86: Assigning the 1<sup>st</sup> row of the dataset as header

stock_market	stock_symbol
2 Categories	AVG Categories
NYSE	AVV
NYSE	AGU
NASDAQ	AGUL
NYSE	AGU
NASDAQ	ACTX
NASDAQ	ADBD
NASDAQ	ADL
NYSE	ADOB
NYSE	AEU
NYSE	AET
NASDAQ	AETI
NYSE	AFGH
NASDAQ	AFH
NYSE	AFSII'A
NYSE	AFSII'A
NASDAQ	AFSII'F
NASDAQ	AGIO
NYSE	AGLE
NYSE	AGM'A
NASDAQ	AGM'A

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.

stock_market	stock_symbol
2 Categories	AVG Categories
NYSE	AVV
NYSE	AGU
NASDAQ	AGUL
NYSE	AGU
NASDAQ	ACTX
NASDAQ	ADBD
NASDAQ	ADL
NYSE	ADOB
NYSE	AEU
NYSE	AET
NASDAQ	AETI
NYSE	AFGH
NASDAQ	AFH
NYSE	AFSII'A
NYSE	AFSII'A
NASDAQ	AFSII'F
NASDAQ	AGIO
NYSE	AGLE
NYSE	AGM'A
NASDAQ	AGM'A

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.

A screenshot of a data processing interface. The main area shows a table with two columns: 'stock\_market' and 'stock\_symbol'. A tooltip at the top right of the table says '60 rows selected (60 duplicates)'. The table contains 906 rows in total, with many rows having identical values in both columns. The interface includes a toolbar with various icons and a status bar at the bottom.

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

A screenshot of the same data processing interface after the 'Remove duplicate rows' step was run. The table now shows 846 rows. The 'Recipe' pane on the right lists the steps: 'Convert from 1 to header' and 'Remove duplicate rows'. The status bar at the bottom indicates the operation was completed at 6:25 PM on 10-Jun-19.

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.

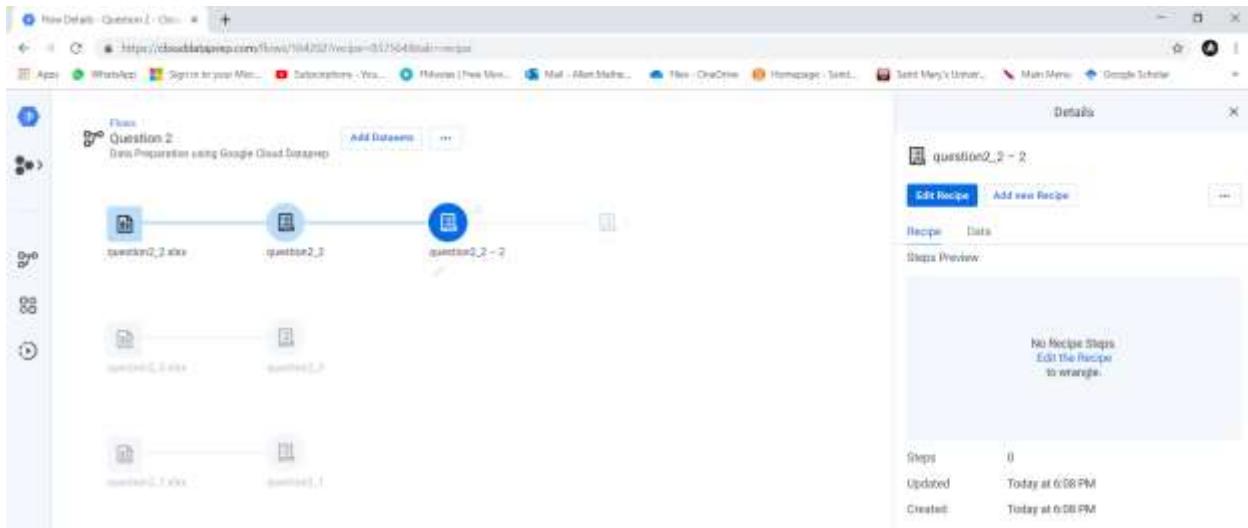


Fig 91: Adding new Recipe

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

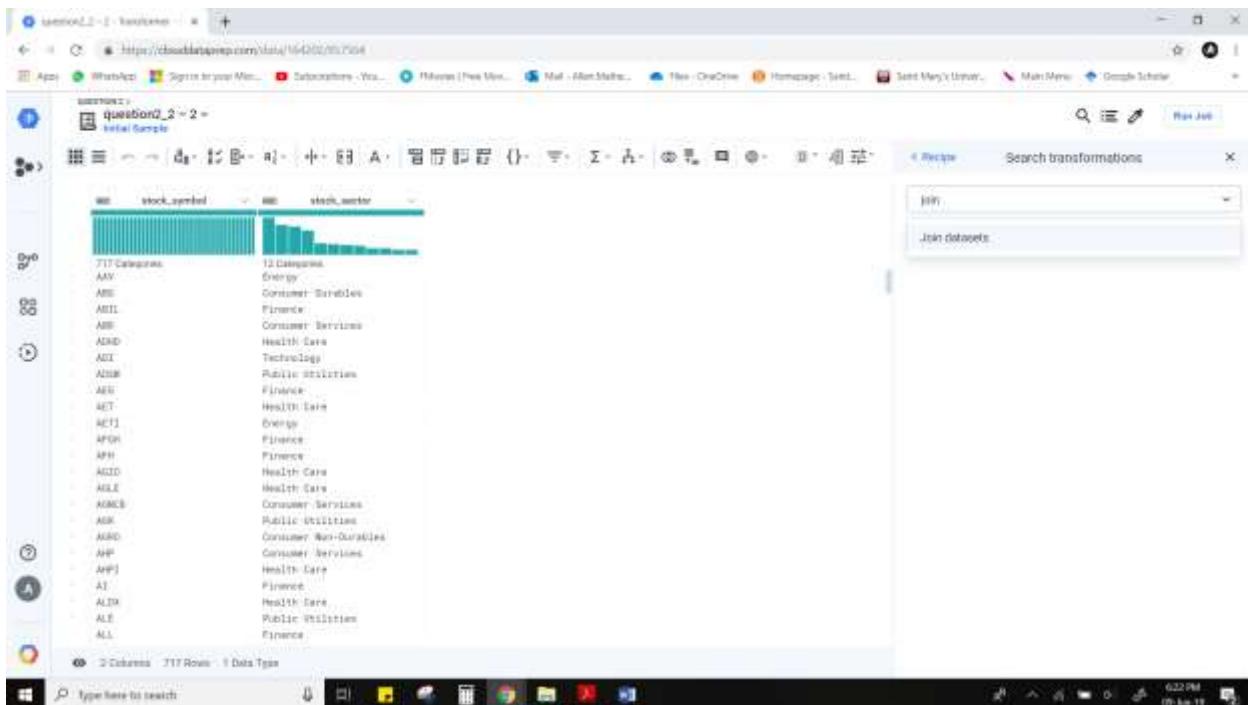


Fig 92: Searching for Join Datasets option

The question2\_1 recipe is selected to join with the question2\_2-2 recipe

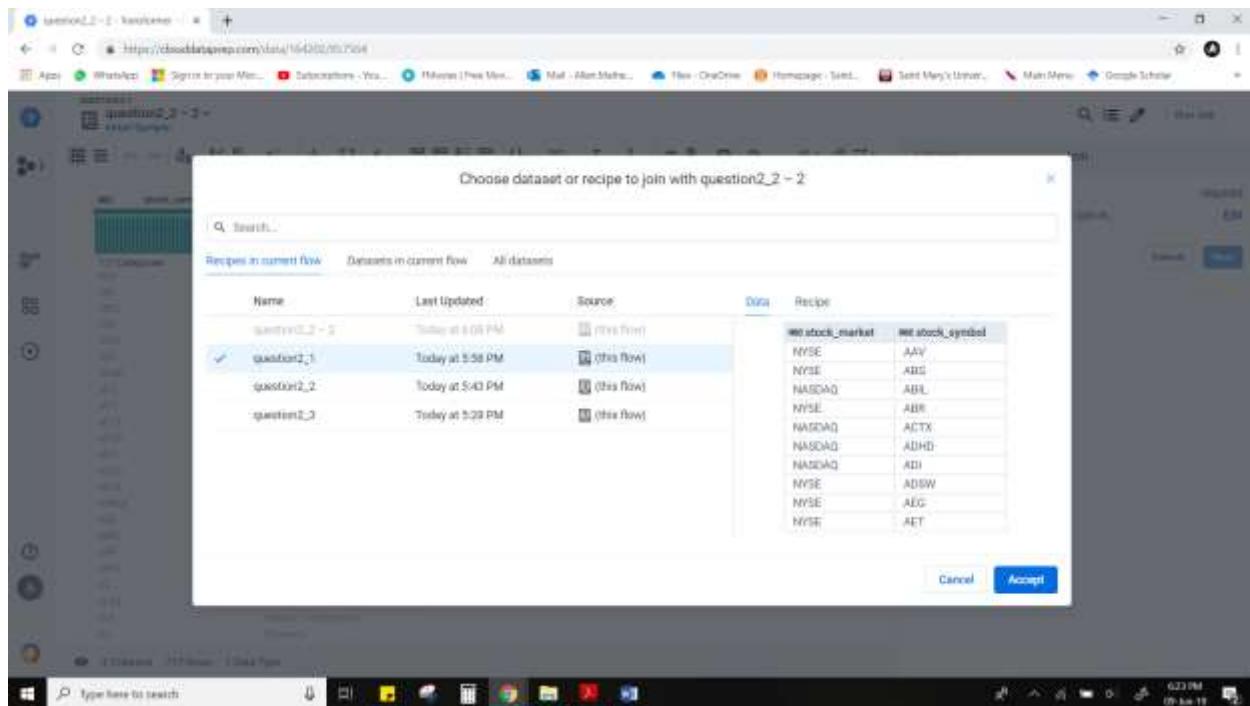


Fig 93: selecting the question2\_1 recipe

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.

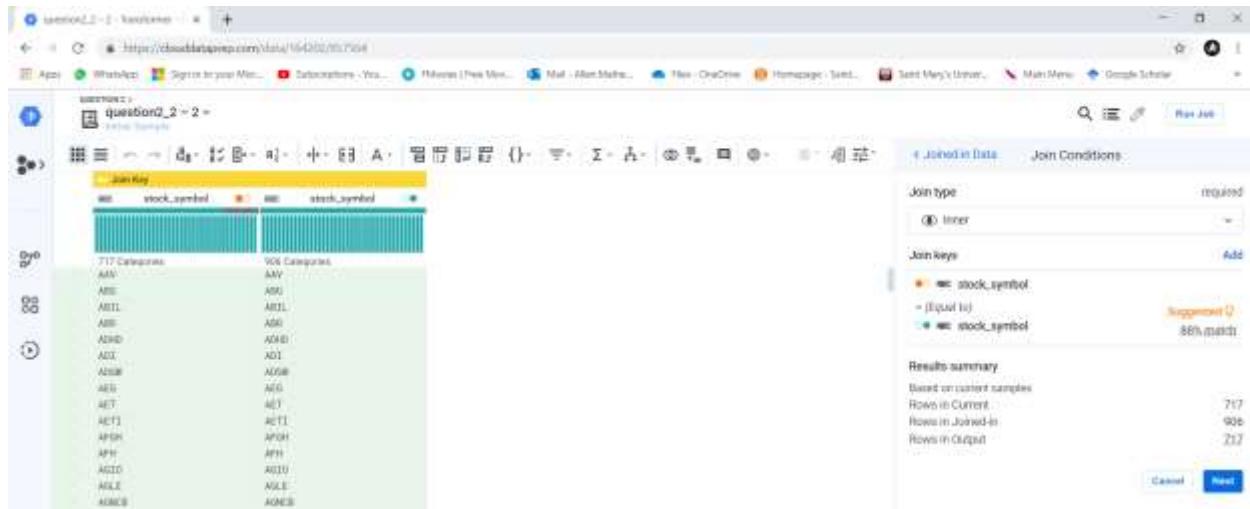


Fig 94: Google Data Prep automatically provides a suggestion for the Join keys

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

The screenshot shows the Data Studio interface with the 'question2\_2 - 2' project open. In the center, there's a preview of the 'stock\_symbol' dataset showing stock symbols, categories, and markets. To the right, the 'Output Columns' panel is displayed, listing three columns: 'stock\_symbol' (Current), 'stock\_sector' (Joined-in), and 'stock\_market' (Joined-in). The 'Advanced options' section is visible at the bottom of the panel.

Fig 95: Select datasets Output columns

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

The screenshot shows the Data Studio interface with the 'question2\_2 - 2' project open. The 'Review Join' panel is active on the right, showing the 'Join type' as 'Inner' and 'Join keys' as 'stock\_symbol'. It also lists 'Output columns (3)' which include 'stock\_symbol' (from Current), 'stock\_sector' (from Joined-in), and 'stock\_market' (from Joined-in). The 'Edit' button is visible at the bottom right of the panel.

Fig 96:Reviewing datasets Output columns

The JOIN operation was successfully able to join both the recipes (i.e. question2\_2-2 and question2\_1).

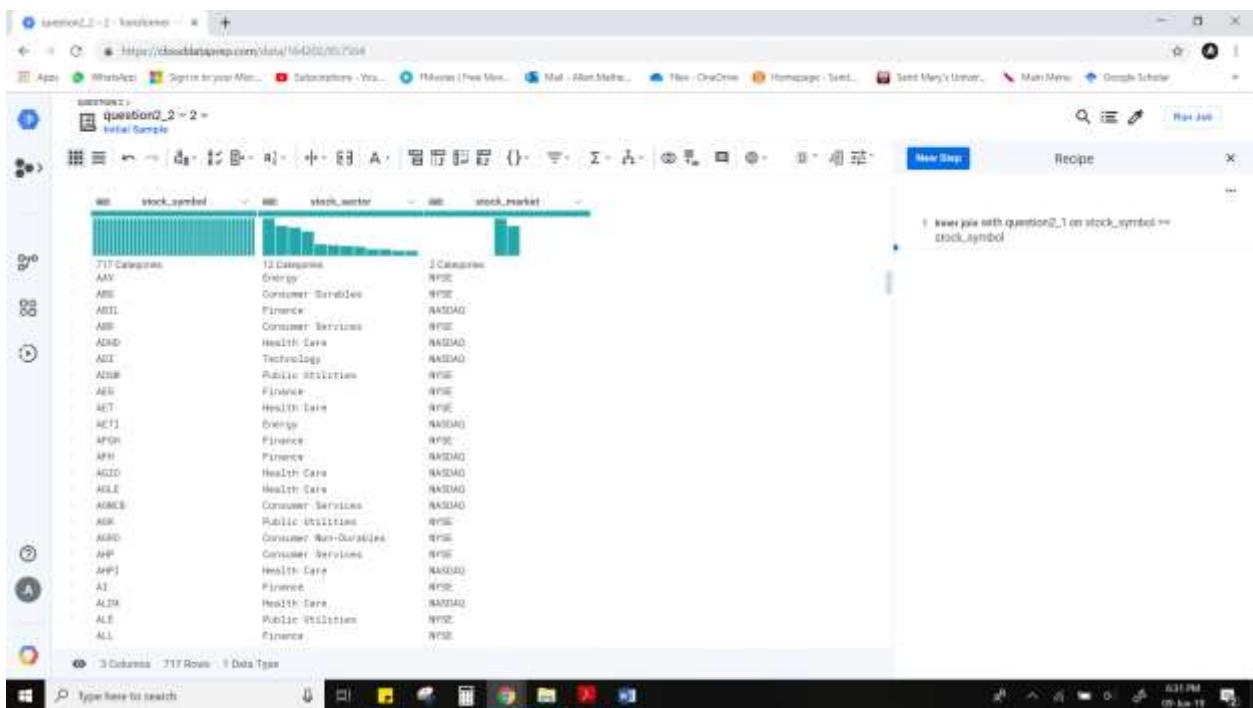


Fig 97: Inner join data sets question2\_2-2 and question2\_1 on stock\_symbol

The Flow Diagram showing that the JOIN operation was successfully able to join both the recipes (i.e. question2\_2-2 and question2\_1).

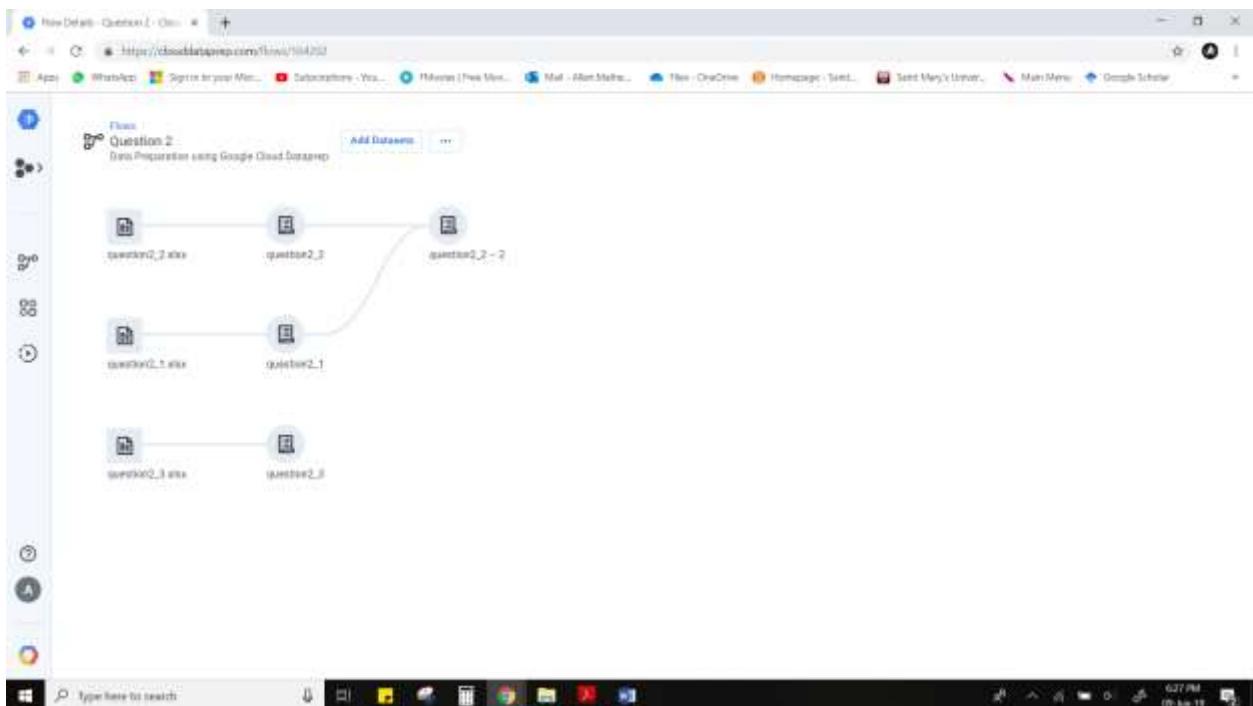


Fig 98: The flow diagram of joining both the recipes (i.e. question2\_2-2 and question2\_1).

A new recipe is added to the question2\_2-2 dataset, to maintain a copy of the work done so far.

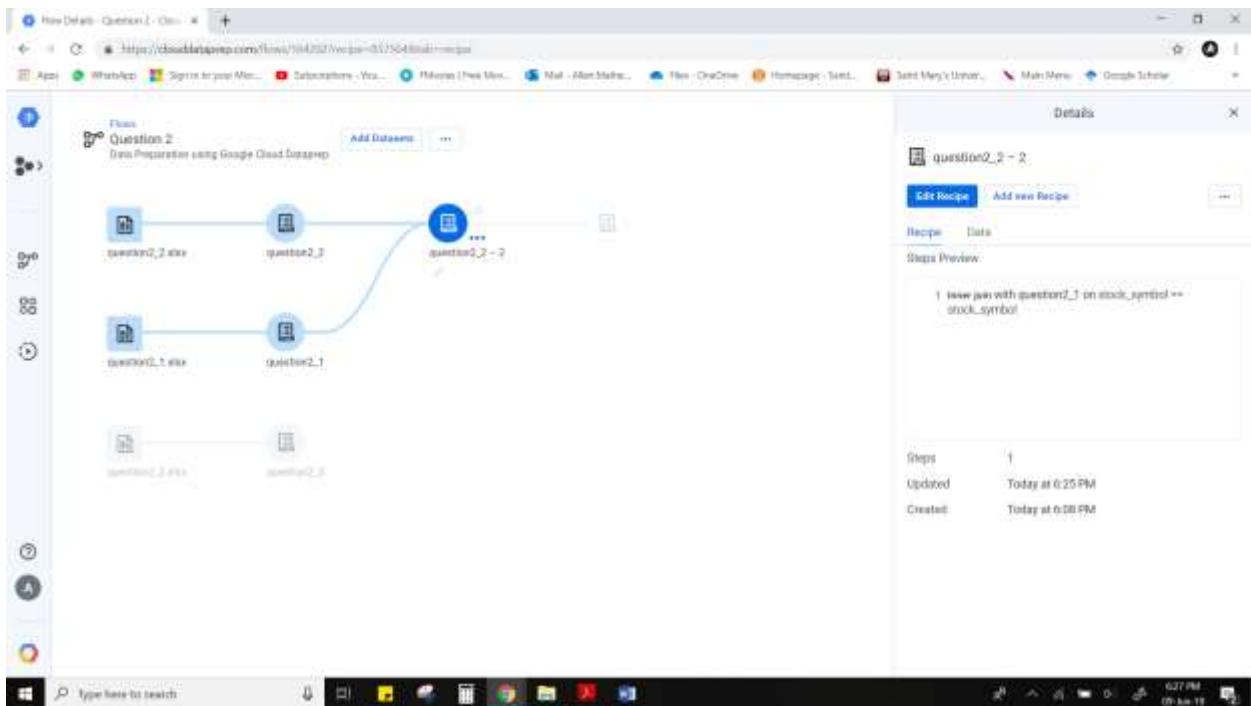


Fig 99: Adding new Recipe

To join the remaining dataset, the new question2\_2-3 will be used.

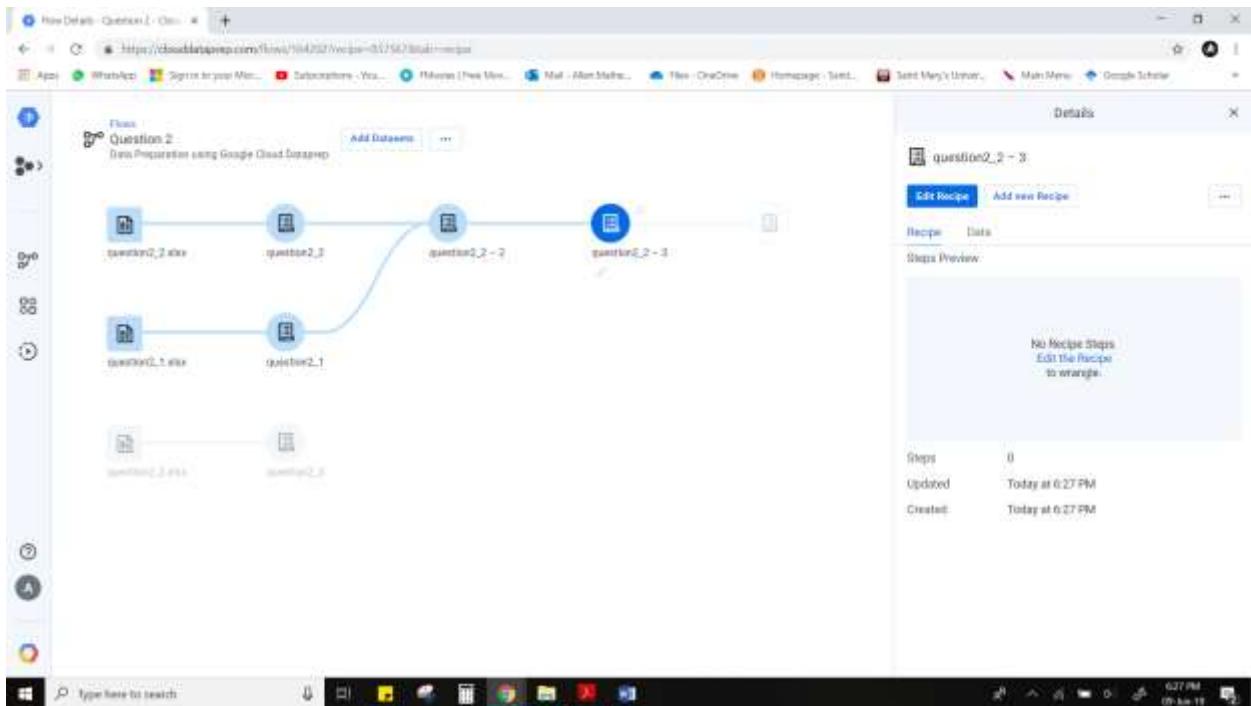


Fig 100: new Recipe was created

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

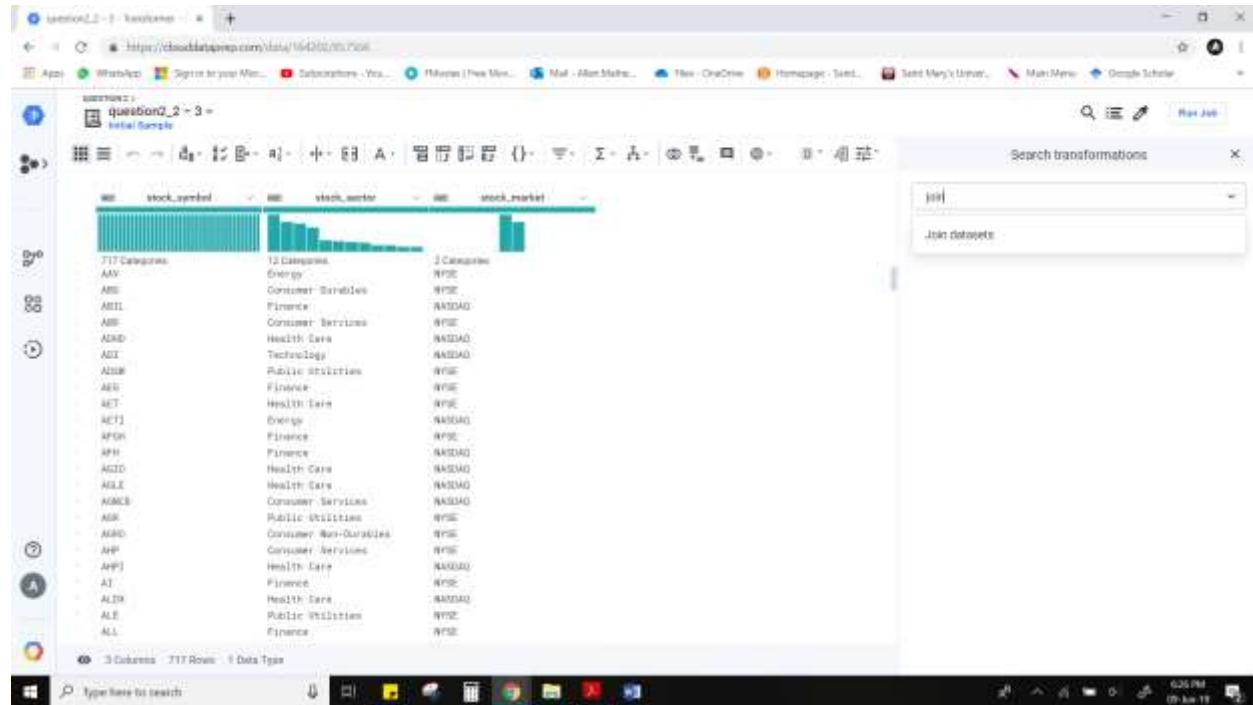


Fig 101: Searching for Join Datasets option.

The question2\_3 recipe is selected to join with the question2\_2-3 recipe

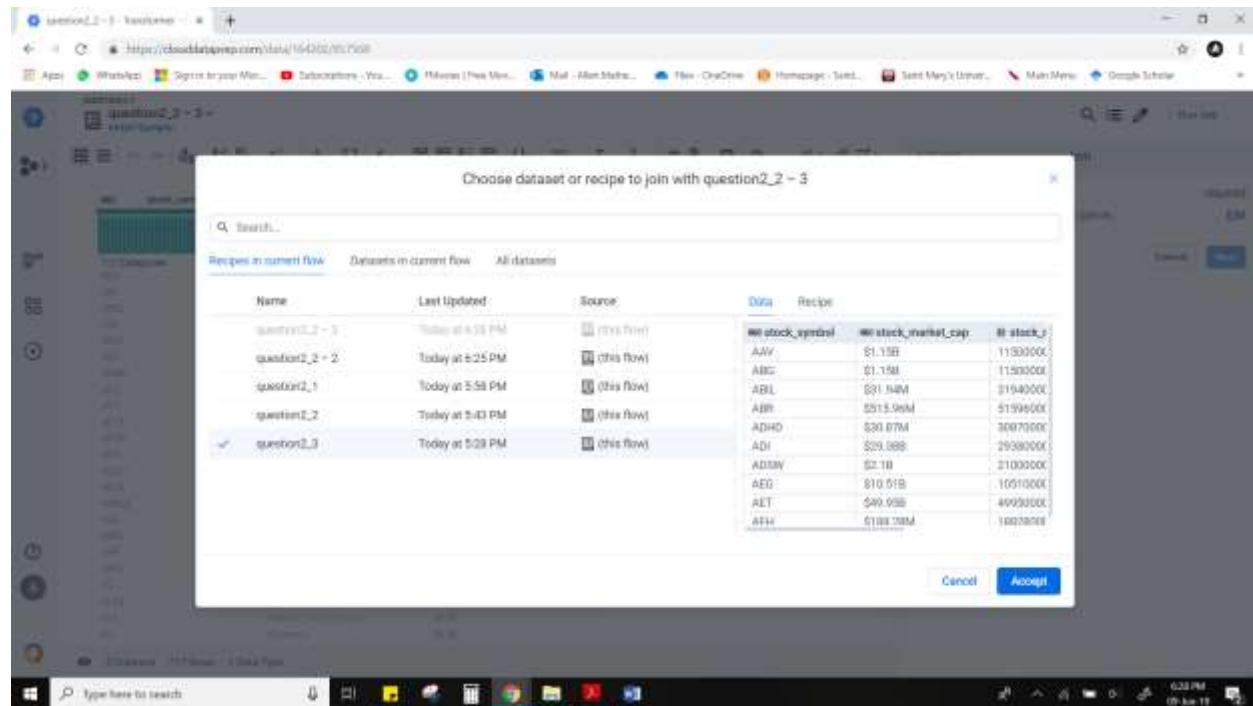


Fig 102: Selecting the question2\_3 recipe

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.

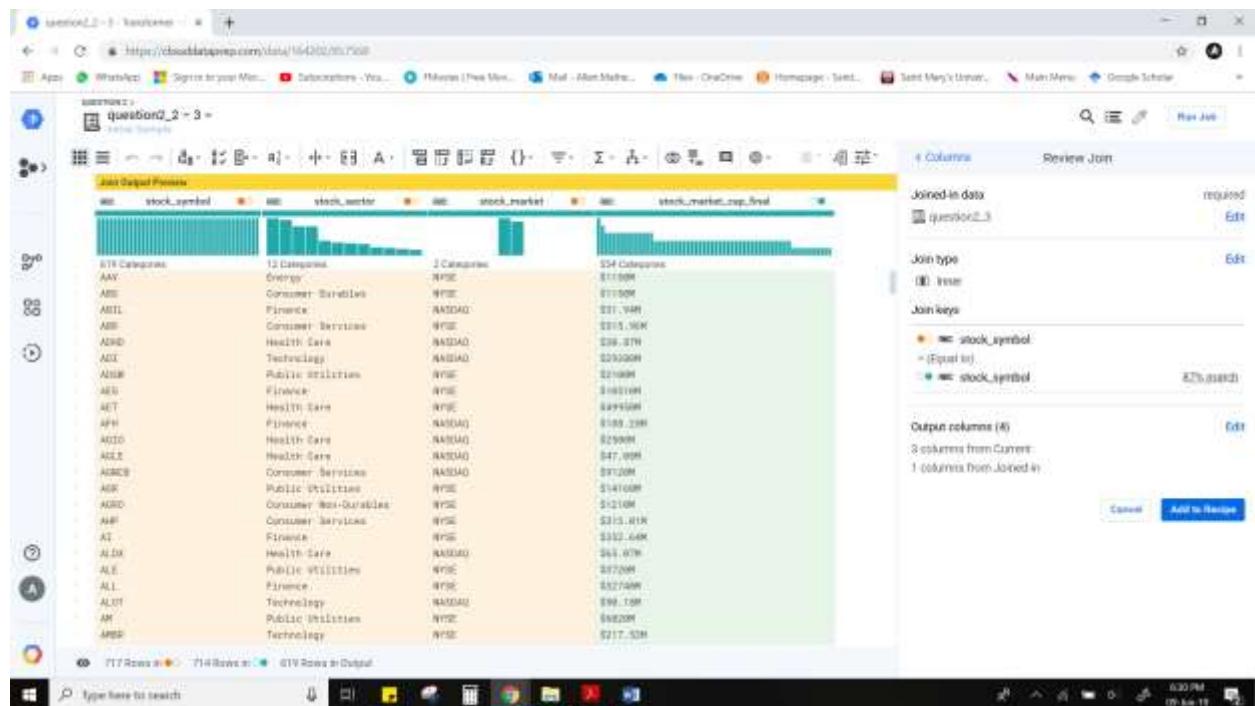


Fig 105: Reviewing datasets Output columns

The JOIN operation was successfully able to join both the recipes (i.e. question2\_2-3 and question2\_3).

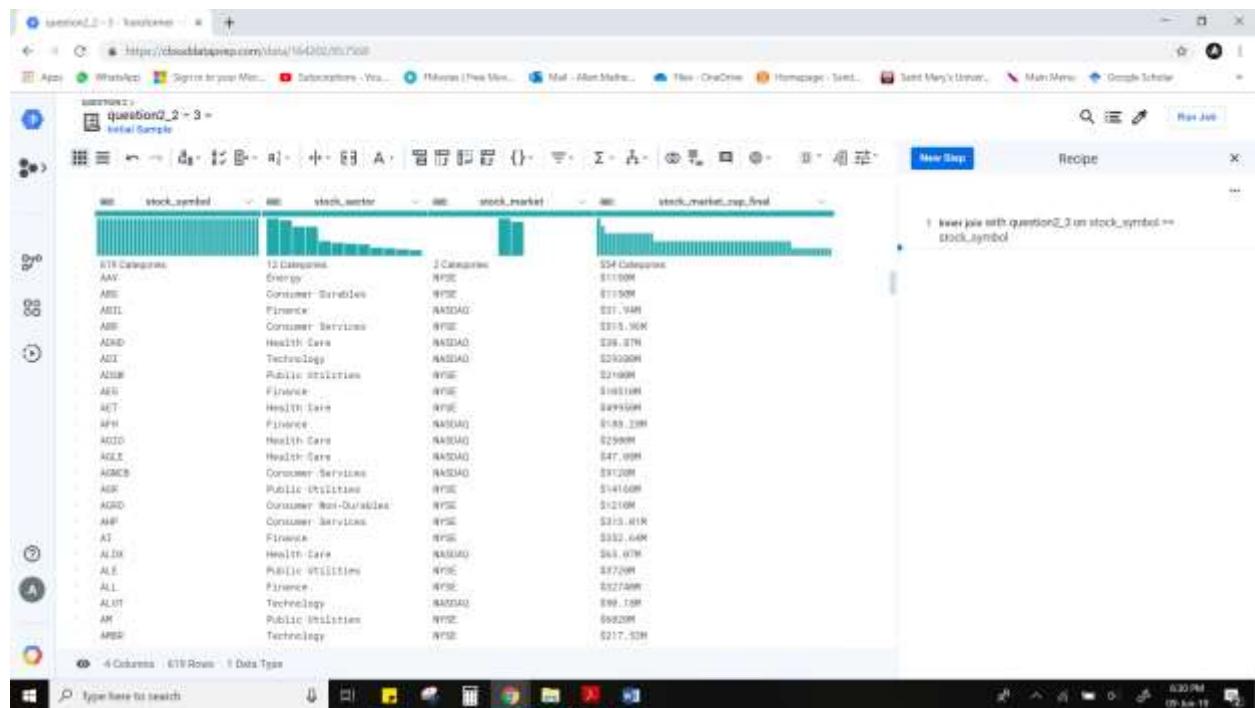


Fig 106: Inner join data sets question2\_2-3 and question2\_3 on stock\_symbol

The Flow Diagram showing that the JOIN operation was successfully able to join both the recipes (i.e. question2\_2-3 and question2\_3).

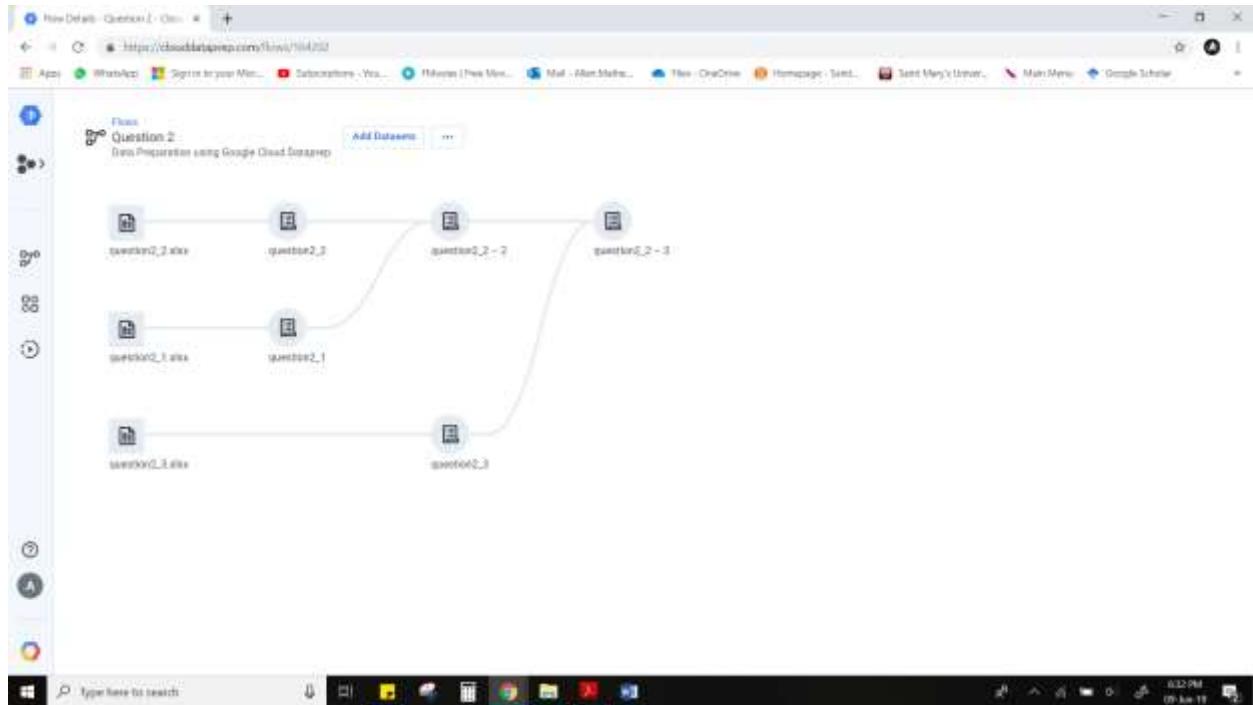


Fig 107: The flow diagram of joining both the recipes (i.e. question2\_2-3 and question2\_3).

A new recipe is added to the question2\_2-3 dataset, to maintain a copy of the work done so far.

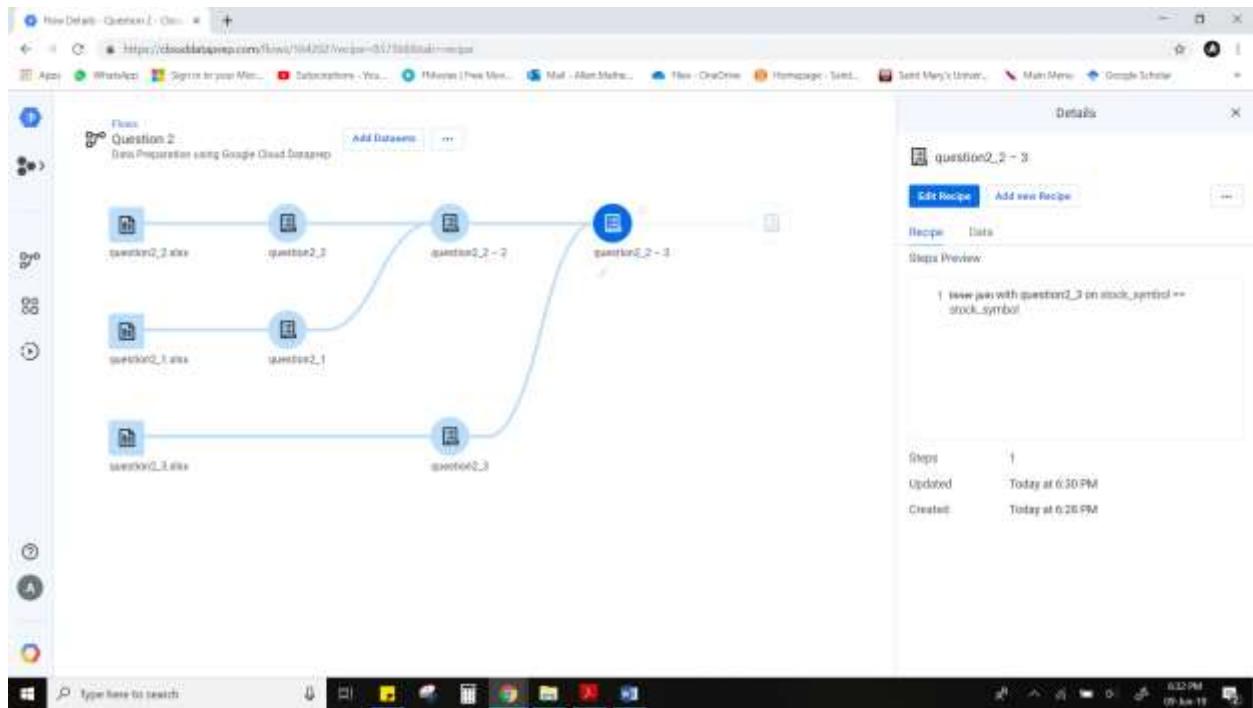


Fig 108: Adding new Recipe

To extract all records of the stocks related to **finance** sector, a new recipe is added to the question2\_2-4 dataset.

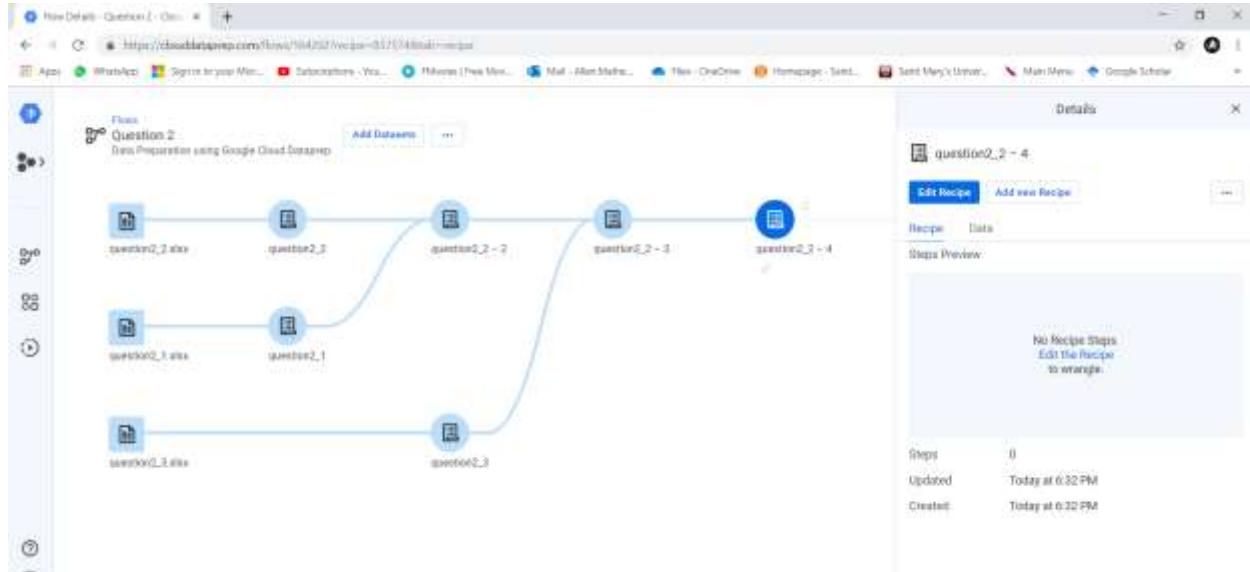


Fig 109: Adding a new Recipe.

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’

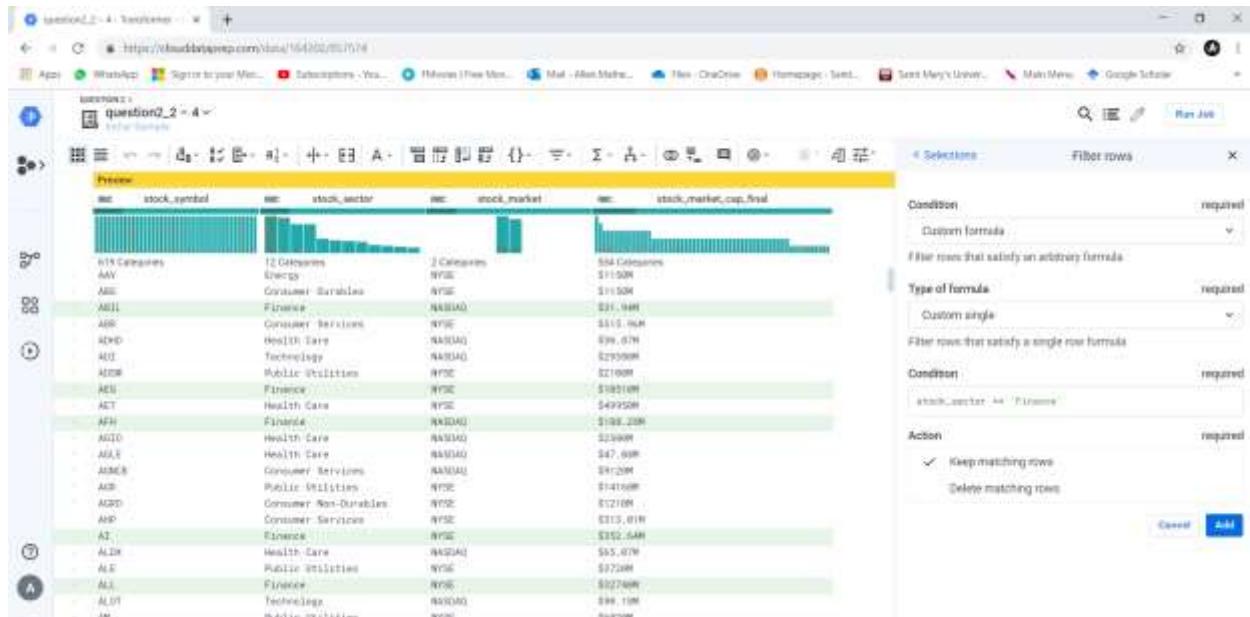


Fig 110: Select rows where stock\_sector is FINANCE

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

	stock_symbol	stock_sector	stock_market	stock_market_cap_final
0	110 Companies	Finance	NASDAQ	\$31,948 1.81%
1	ABLI	Finance	NYSE	\$105108
2	AET	Finance	NASDAQ	\$188,238
3	APH	Finance	NYSE	\$352,648
4	A1	Finance	NYSE	\$227408
5	ALL	Finance	NYSE	\$127,38
6	AGL	Finance	NYSE	\$139608
7	ABVA	Finance	NASDAQ	\$29,598
8	ABRT	Finance	NASDAQ	\$467,588
9	BLVD	Finance	NYSE	\$99708
10	BNA	Finance	NASDAQ	\$55408
11	BKF	Finance	NYSE	\$22208
12	BDO	Finance	NYSE	\$111408
13	BDP	Finance	NASDAQ	\$11,508
14	BSE	Finance	NYSE	\$702108
15	CII	Finance	NYSE	\$61908
16	CFR	Finance	NYSE	\$116208
17	CIB	Finance	NASDAQ	\$122,368
18	CIBK	Finance	NYSE	\$111108
19	CNA	Finance	NASDAQ	\$51,528
20	CNP	Finance	NYSE	\$35308
21	CNO	Finance	NASDAQ	\$744,138
22	CNOB	Finance	NYSE	\$966,528
23	CO	Finance	NYSE	

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.

	stock_symbol	stock_market	stock_market_cap_final
0	110 Companies	NASDAQ	\$31,948
1	ABLI	NYSE	\$105108
2	APH	NASDAQ	\$188,238
3	A1	NYSE	\$352,648
4	ALL	NYSE	\$227408
5	AGL	NYSE	\$127,38
6	ABVA	NYSE	\$139608
7	ABRT	NASDAQ	\$29,598
8	BLVD	NASDAQ	\$467,588
9	BNA	NYSE	\$99708
10	BKF	NASDAQ	\$55408
11	BDO	NYSE	\$22208
12	BDP	NYSE	\$111408
13	BSE	NASDAQ	\$11,508
14	CII	NYSE	\$702108
15	CFR	NYSE	\$61908
16	CIB	NYSE	\$116208
17	CIBK	NASDAQ	\$122,368
18	CNA	NYSE	\$111108
19	CNP	NASDAQ	\$51,528
20	CNO	NYSE	\$35308
21	CNOB	NASDAQ	\$744,138
22	CO	NYSE	\$966,528

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.

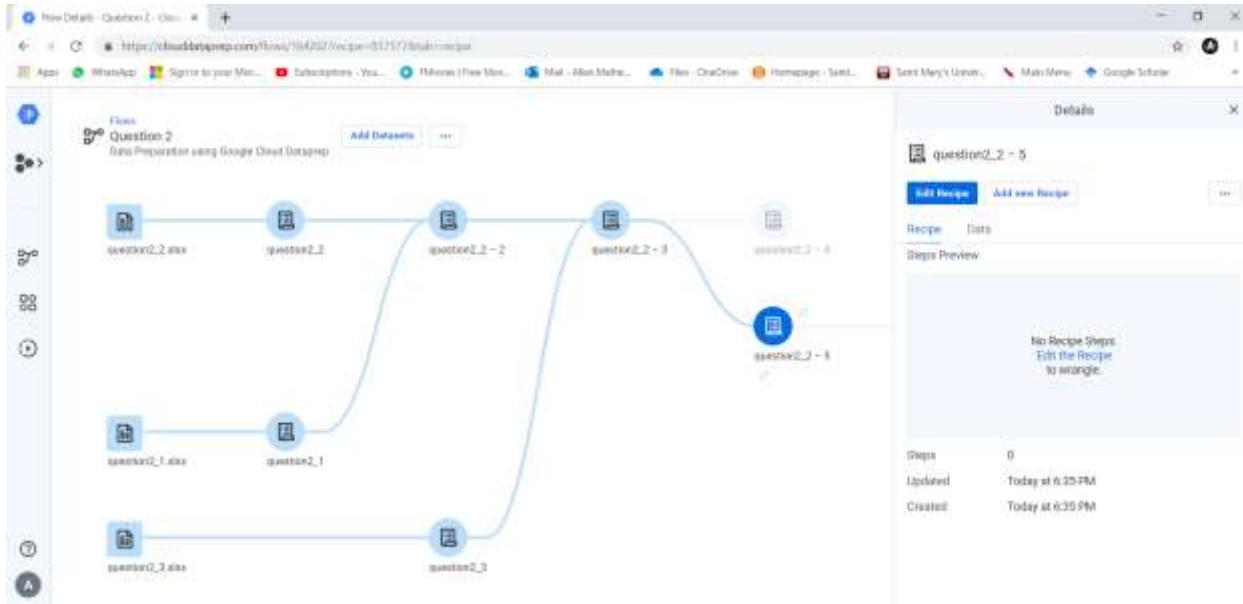


Fig 113: Adding a new Recipe.

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’

stock_symbol	stock_sector	stock_market	stock_market_cap_bill
AAC	Energy	NYSE	\$11.9B
ABE	Consumer Goods	NYSE	\$11.5B
ABIL	Finance	NASDAQ	\$11.4B
ABIF	Consumer Services	NYSE	\$11.3B
ADHD	Health Care	NASDAQ	\$11.3B
ADK	Technology	NASDAQ	\$10.6B
ADSK	Public Utilities	NYSE	\$10.6B
AEI	Finance	NYSE	\$10.5B
AET	Health Care	NYSE	\$9.9B
AFF	Finance	NASDAQ	\$9.8B
AGDQ	Health Care	NASDAQ	\$9.6B
AGLE	Health Care	NASDAQ	\$9.5B
AGNC	Consumer Services	NASDAQ	\$9.2B
AGR	Public Utilities	NYSE	\$9.1B
AGRD	Consumer Non-Durables	NYSE	\$9.1B
AHFI	Consumer Services	NYSE	\$9.1B
AI	Finance	NYSE	\$9.1B
ALBK	Health Care	NASDAQ	\$8.8B
ALF	Public Utilities	NYSE	\$8.7B
ALL	Finance	NYSE	\$8.7B
ALUT	Technology	NASDAQ	\$8.7B
AM	Public Utilities	NYSE	\$8.2B

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”.

	stock_symbol	stock_sector	stock_market	stock_market_cap_bill
00	ABMD	Health Care	NASDAQ	\$38.07B
01	ACT	Health Care	NYSE	\$449.99B
02	AGDZ	Health Care	NASDAQ	\$23.09B
03	AGLE	Health Care	NASDAQ	\$47.00B
04	ALDX	Health Care	NASDAQ	\$5.07B
05	AMED	Health Care	NASDAQ	\$21.20B
06	AMBI	Health Care	NASDAQ	\$229.63B
07	AMDO	Health Care	NASDAQ	\$486.70B
08	AMAY	Health Care	NASDAQ	\$357.52B
09	AMLE	Health Care	NASDAQ	\$361.09B
10	APBS	Health Care	NASDAQ	\$79.49B
11	ATRY	Health Care	NASDAQ	\$119.09B
12	ATRZ	Health Care	NASDAQ	\$133.07B
13	AZM	Health Care	NYSE	\$274.00B
14	BCLL	Health Care	NASDAQ	\$22.31B
15	BKRX	Health Care	NASDAQ	\$446.23B
16	BYNS	Health Care	NASDAQ	\$383.40B
17	CARA	Health Care	NASDAQ	\$356.53B
18	CGRX	Health Care	NASDAQ	\$391.57B
19	CHMB	Health Care	NASDAQ	\$797.63B
20	CRIS	Health Care	NASDAQ	\$234.19B
21	SRIS	Health Care	NASDAQ	\$254.40B

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.

	stock_symbol	stock_market	stock_market_cap_bill
00	ABMD	NASDAQ	\$38.07B
01	ACT	NYSE	\$449.99B
02	AGDZ	NASDAQ	\$23.09B
03	AGLE	NASDAQ	\$47.00B
04	ALDX	NASDAQ	\$5.07B
05	AMED	NASDAQ	\$21.20B
06	AMBI	NASDAQ	\$229.63B
07	AMDO	NASDAQ	\$486.70B
08	AMAY	NASDAQ	\$357.52B
09	AMLE	NASDAQ	\$361.09B
10	APBS	NASDAQ	\$79.49B
11	ATRY	NASDAQ	\$119.09B
12	ATRZ	NASDAQ	\$133.07B
13	AZM	NYSE	\$274.00B
14	BCLL	NASDAQ	\$22.31B
15	BKRX	NASDAQ	\$446.23B
16	BYNS	NASDAQ	\$383.40B
17	CARA	NASDAQ	\$356.53B
18	CGRX	NASDAQ	\$391.57B
19	CHMB	NASDAQ	\$797.63B
20	CRIS	NASDAQ	\$234.19B
21	SRIS	NASDAQ	\$254.40B

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.

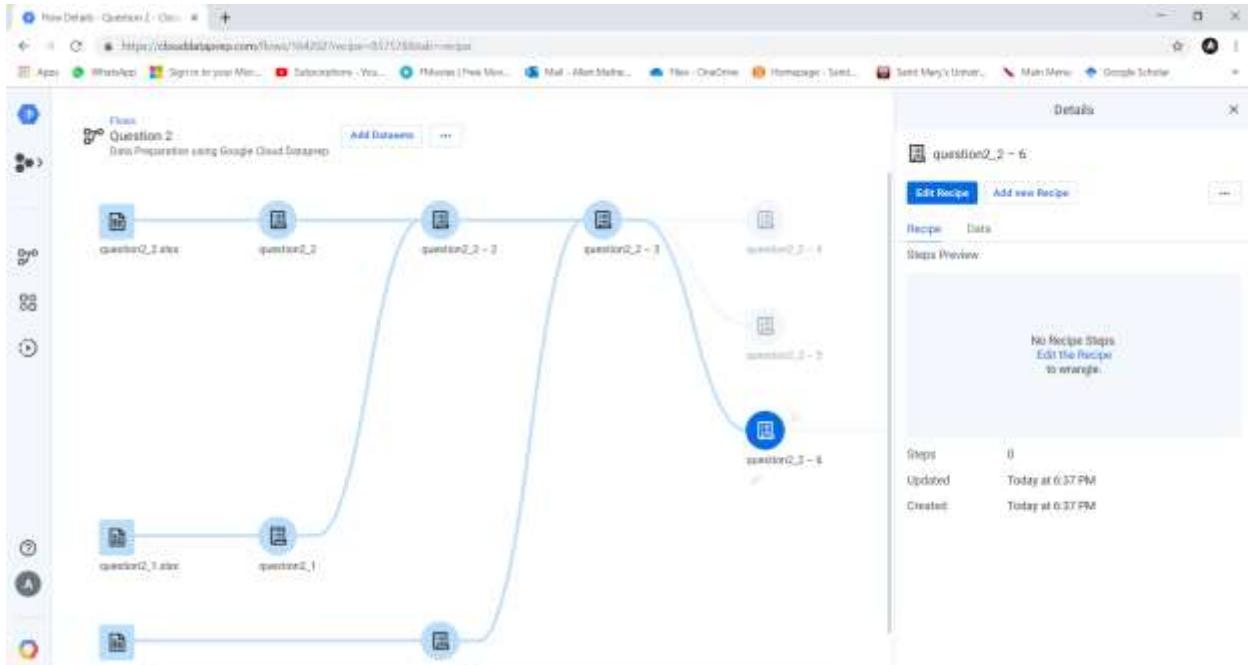


Fig 117: Adding a new Recipe.

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’

stock_symbol	stock_sector	stock_market	stock_market_cap_bill
AAC	Automobiles	NYSE	\$11.00B
ABE	Energy	NYSE	\$11.90B
ABIL	Consumer Staples	NASDAQ	\$21.94B
ABIF	Finance	NYSE	\$31.36B
ABIN	Consumer Services	NASDAQ	\$38.37B
ABND	Health Care	NASDAQ	\$23.00B
ABX	Technology	NASDAQ	\$23.00B
ABNB	Public Utilities	NYSE	\$21.00B
ABR	Finance	NYSE	\$18.00B
ACT	Consumer Staples	NASDAQ	\$49.00B
AFH	Health Care	NASDAQ	\$18.22B
AGDZ	Health Care	NASDAQ	\$25.00B
AGLE	Health Care	NASDAQ	\$47.00B
AGNCB	Consumer Services	NASDAQ	\$31.20B
AGR	Public Utilities	NYSE	\$141.60B
AGRD	Consumer Non-Durables	NYSE	\$12.00B
AHF	Consumer Services	NYSE	\$213.01B
AT	Finance	NYSE	\$332.64B
ALDX	Health Care	NASDAQ	\$63.07B

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.

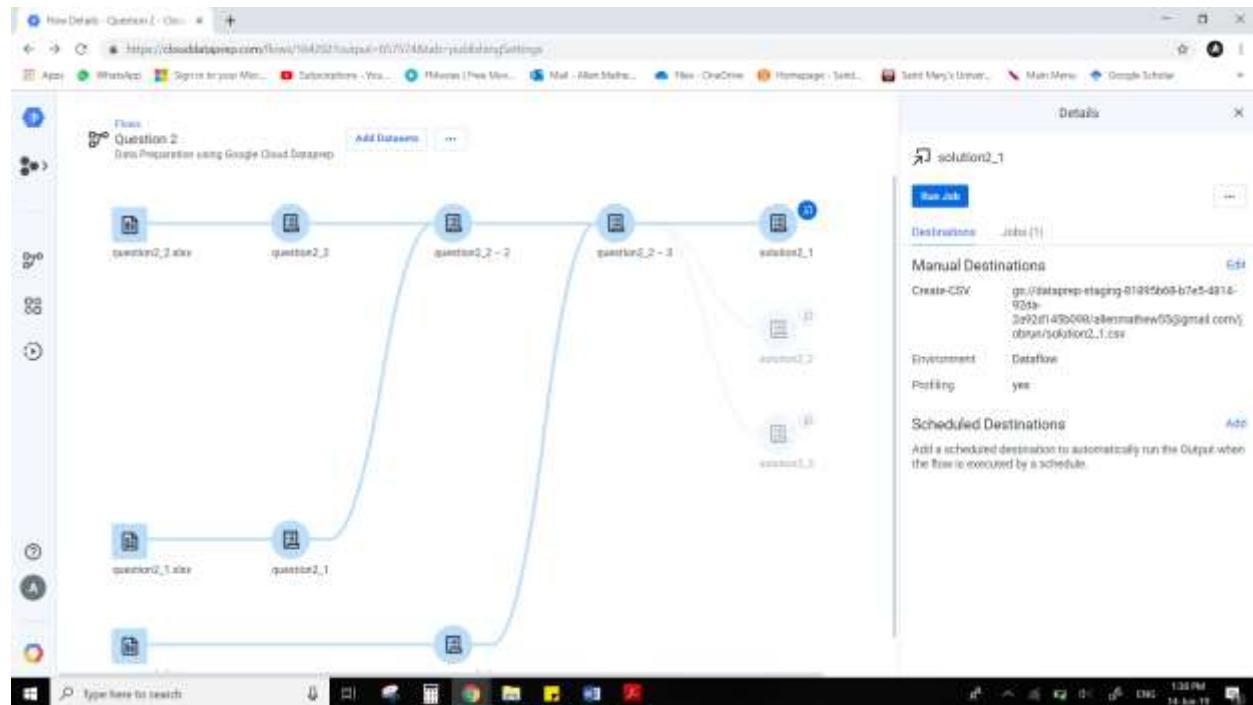


Fig 121: Create the Output to Run

Job	User	Output	Status	Started
2445648	alemmathew05@gmail.com (you)	solution2_3 Question 2	Completed	Last Sunday at 9:07 PM Ran for 7 minutes
2445647	alemmathew05@gmail.com (you)	solution2_2 Question 2	Completed	Last Sunday at 9:06 PM Ran for 7 minutes
2445650	alemmathew05@gmail.com (you)	solution2_1 Question 2	Completed	Last Sunday at 9:01 PM Ran for 6 minutes

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.

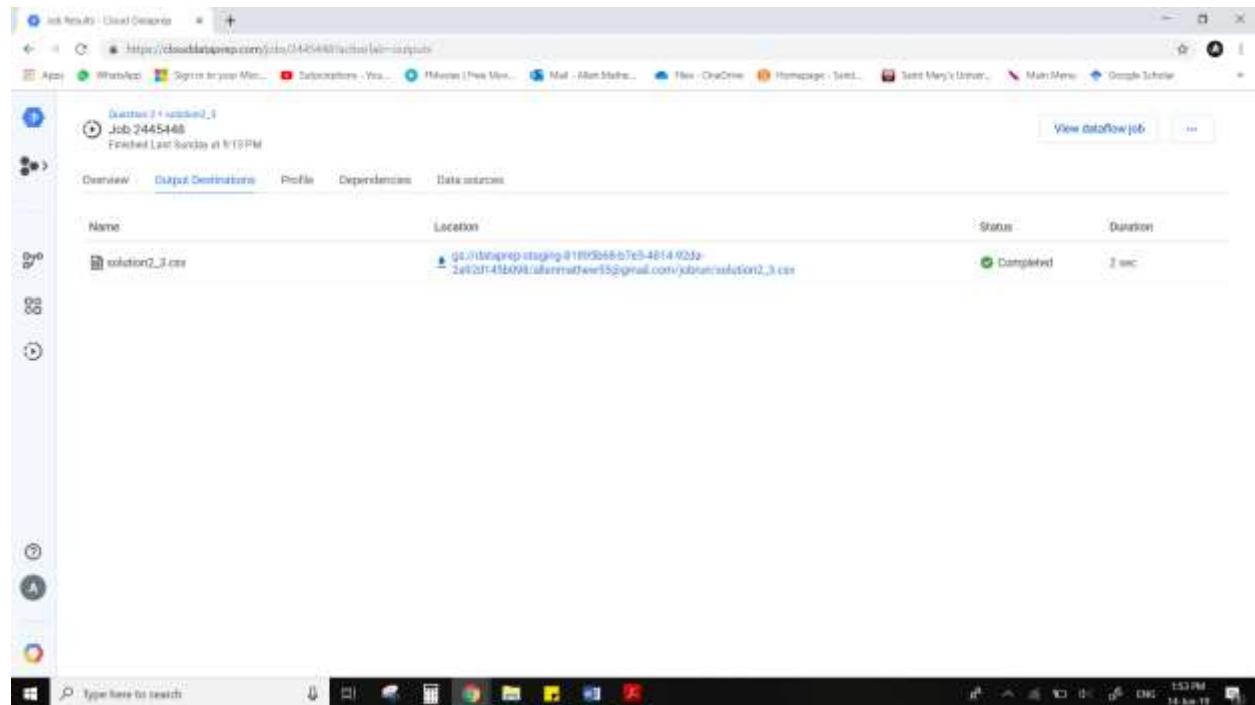


Fig 123: Download the document

All the CSV files were exported successfully.

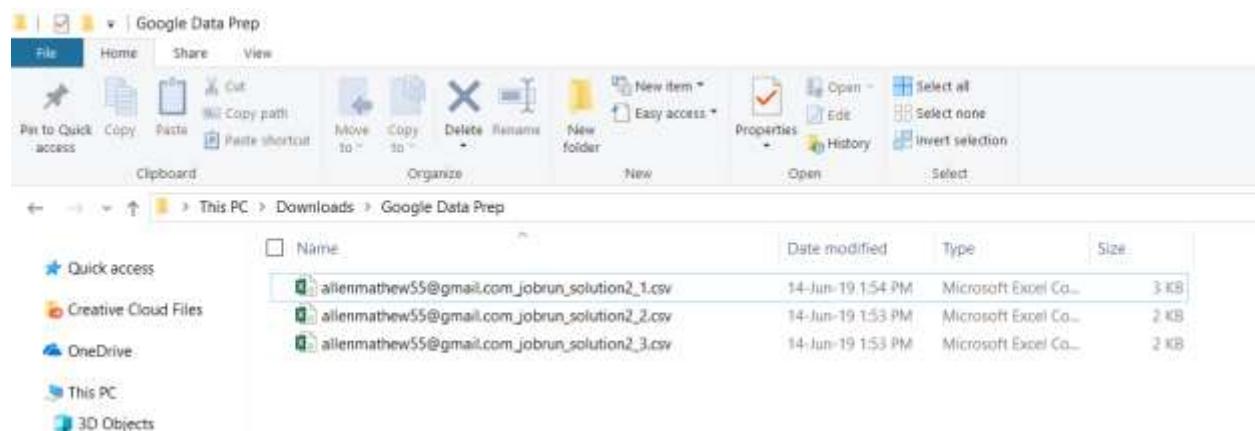


Fig 124: The Exported CSV files

## 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.

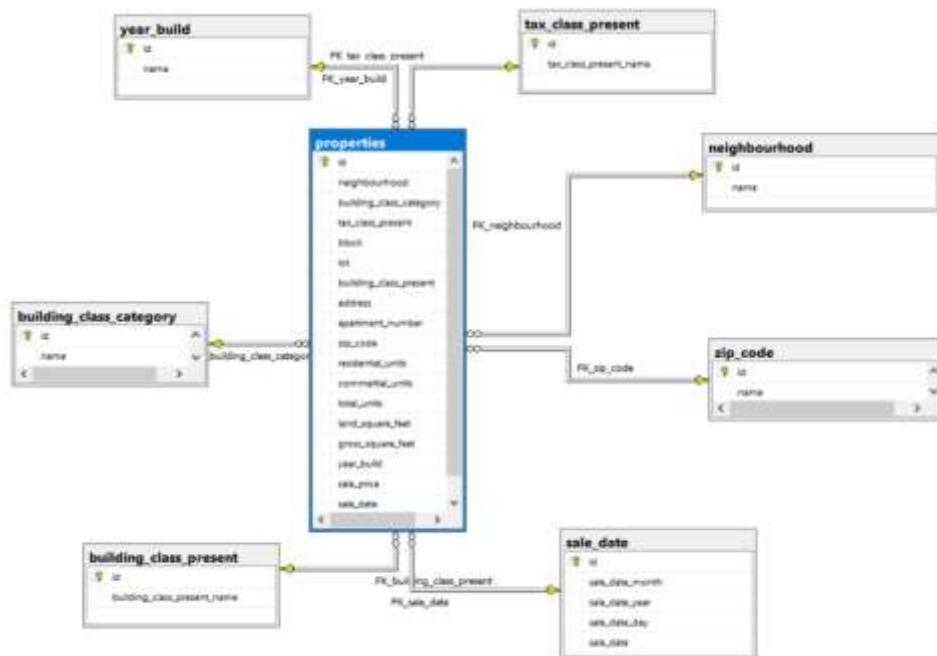


Fig 125:Star schema implementation of our NYCPROPERTYSALES Database

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

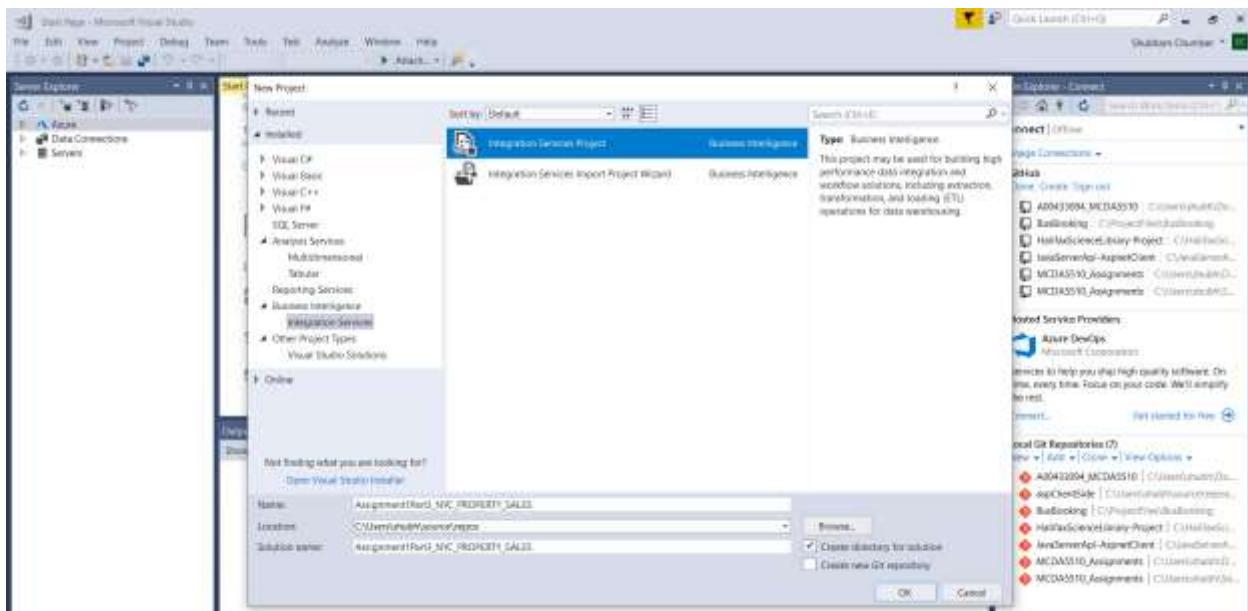


Fig 127:Creating a new project

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

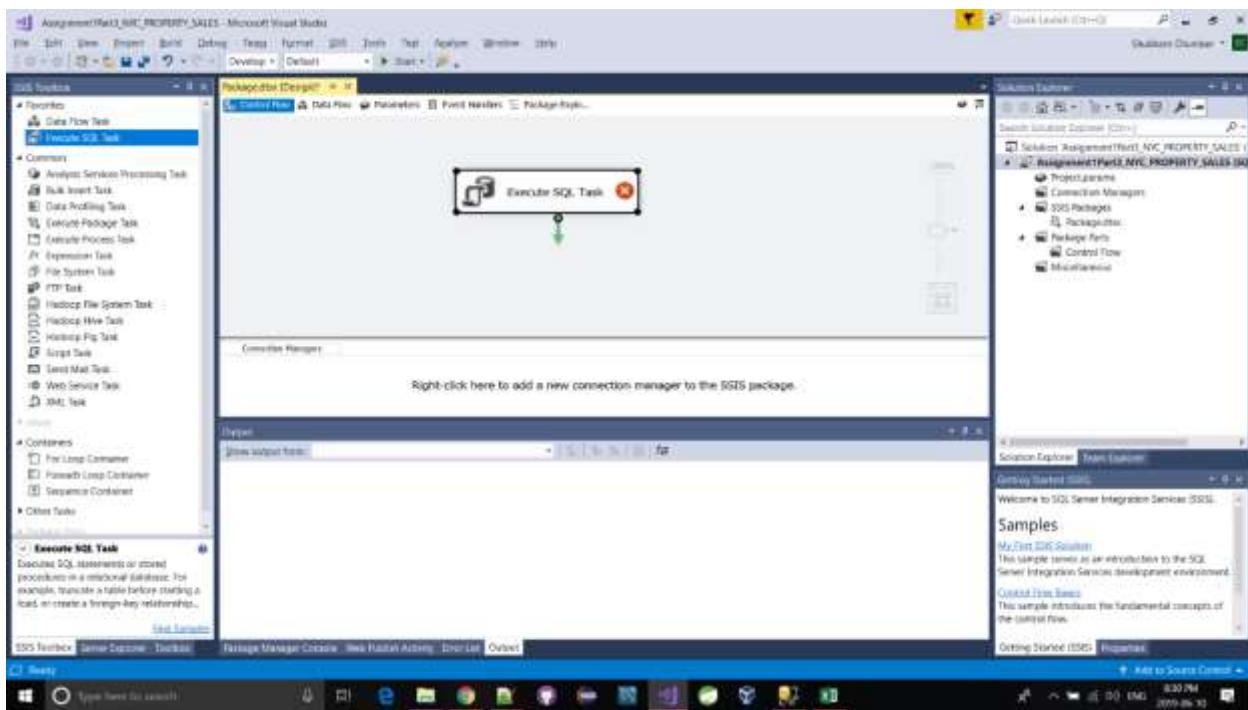


Fig 128:First task of control flow

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.

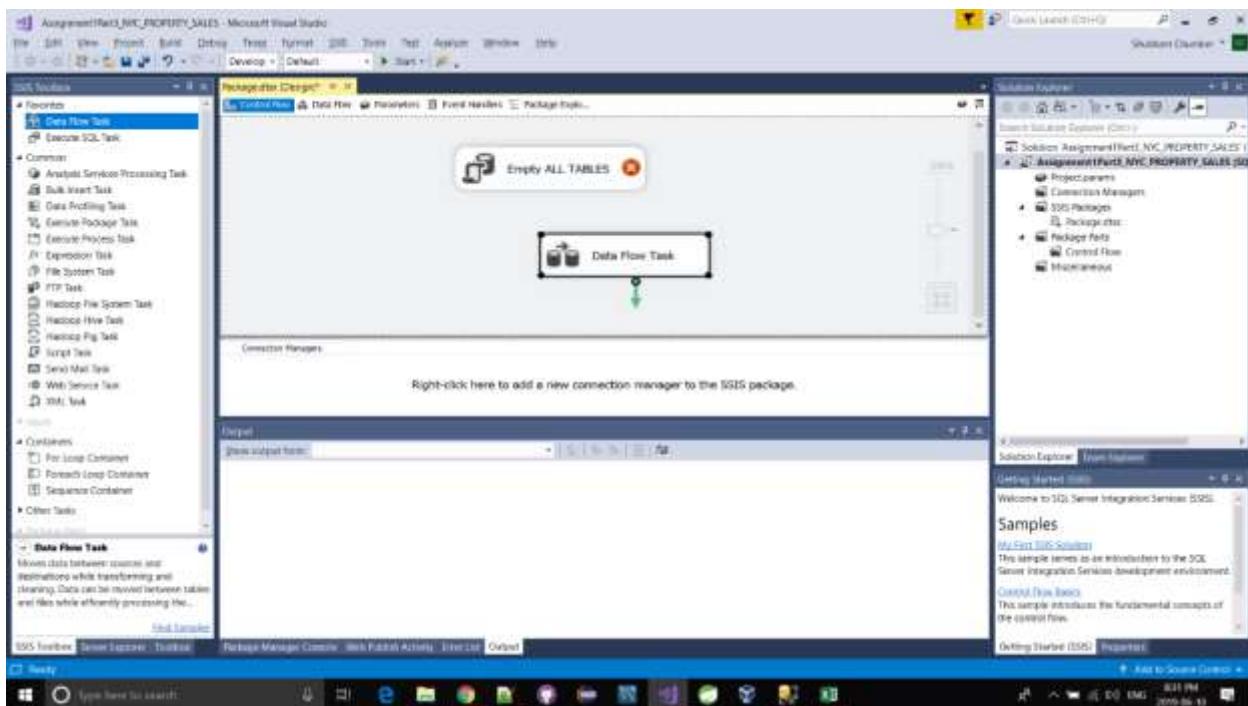


Fig 129:Drag a data flow task

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.

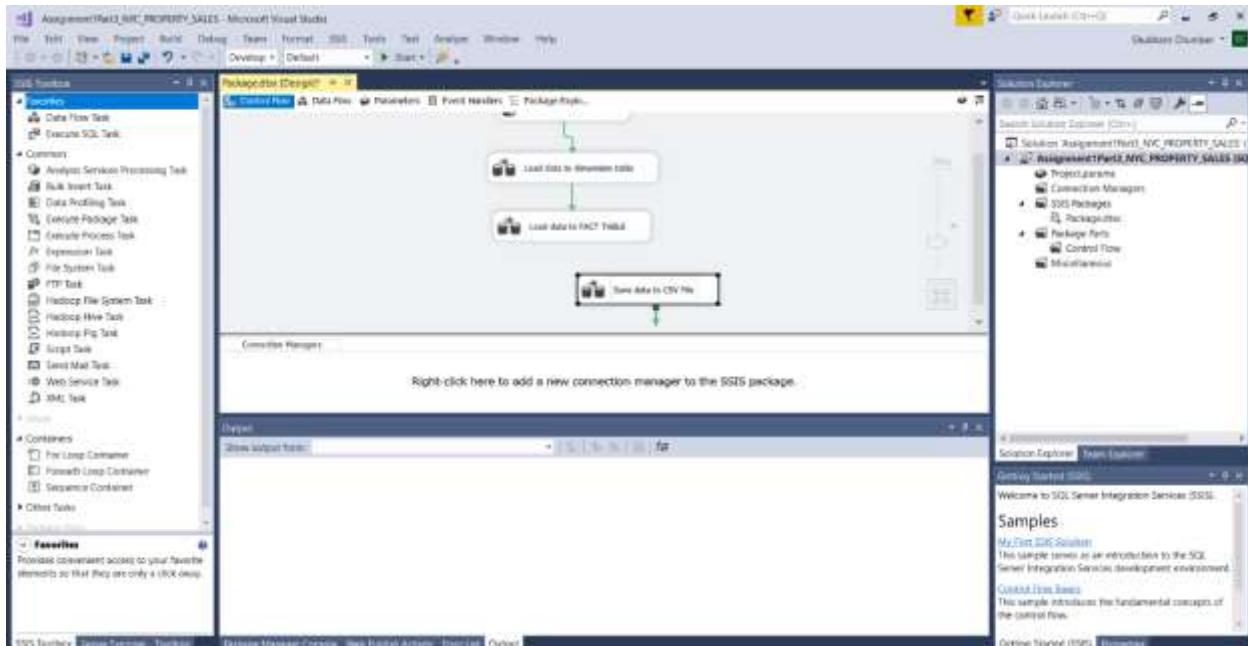


Fig 130:Creating task for saving CSV file

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.

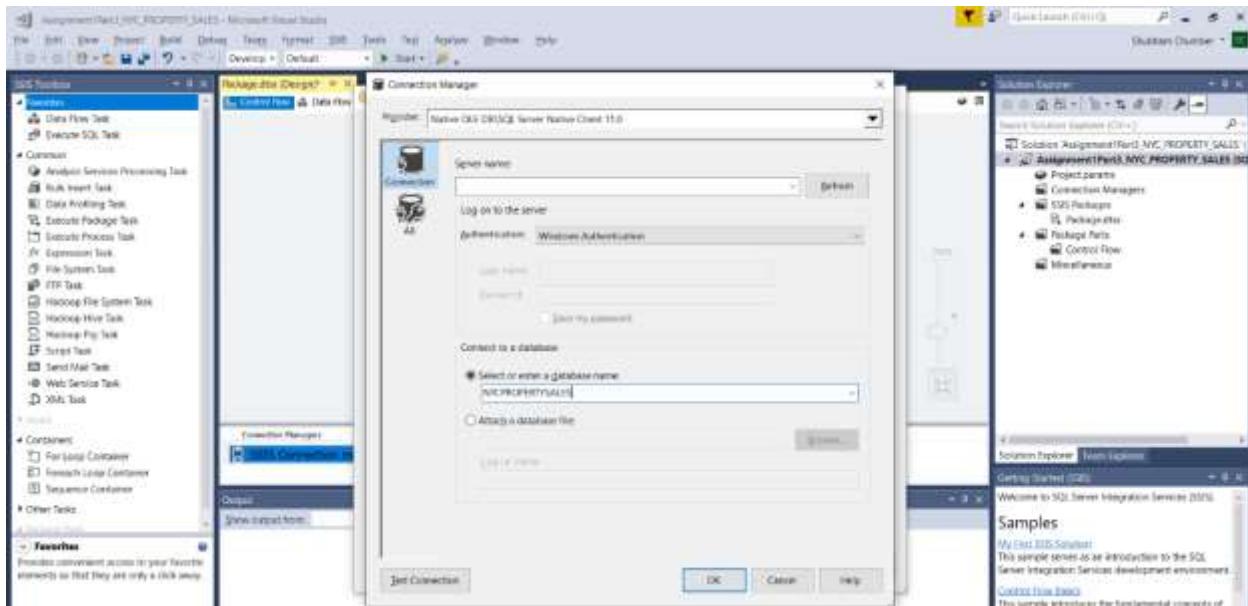


Fig 131: Making connection with the server

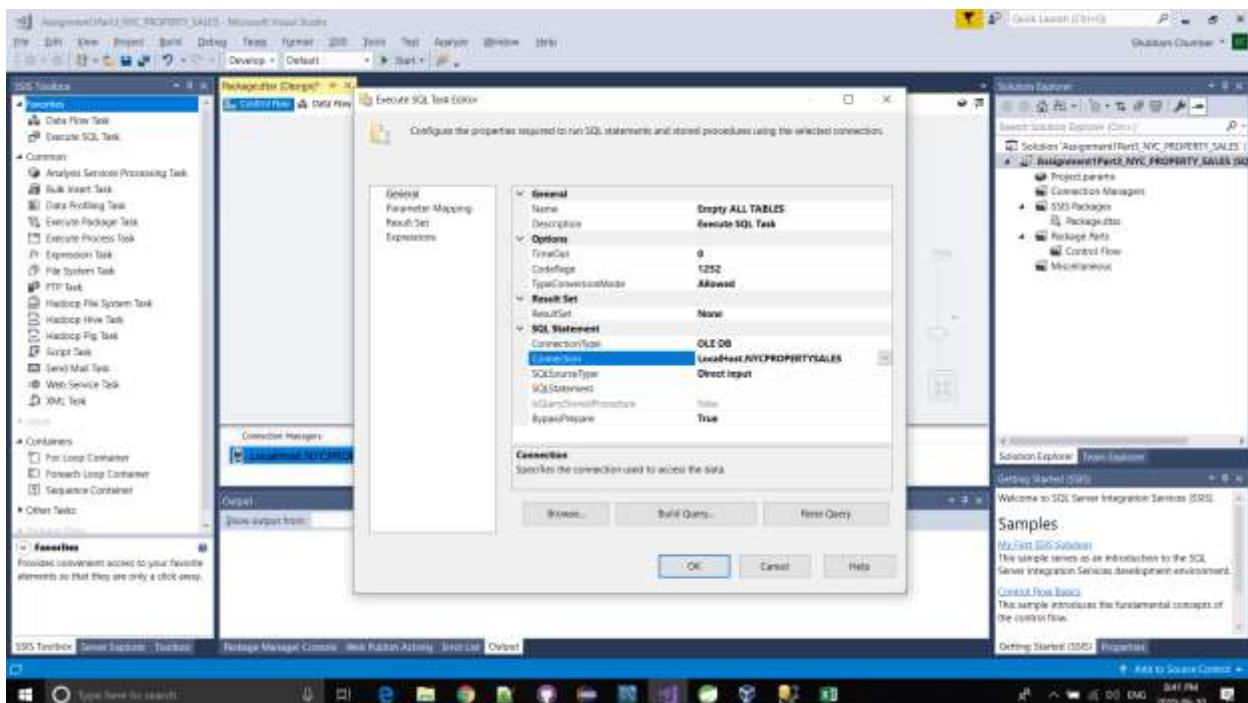


Fig 132: Configuring the properties required to run SQL statements

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

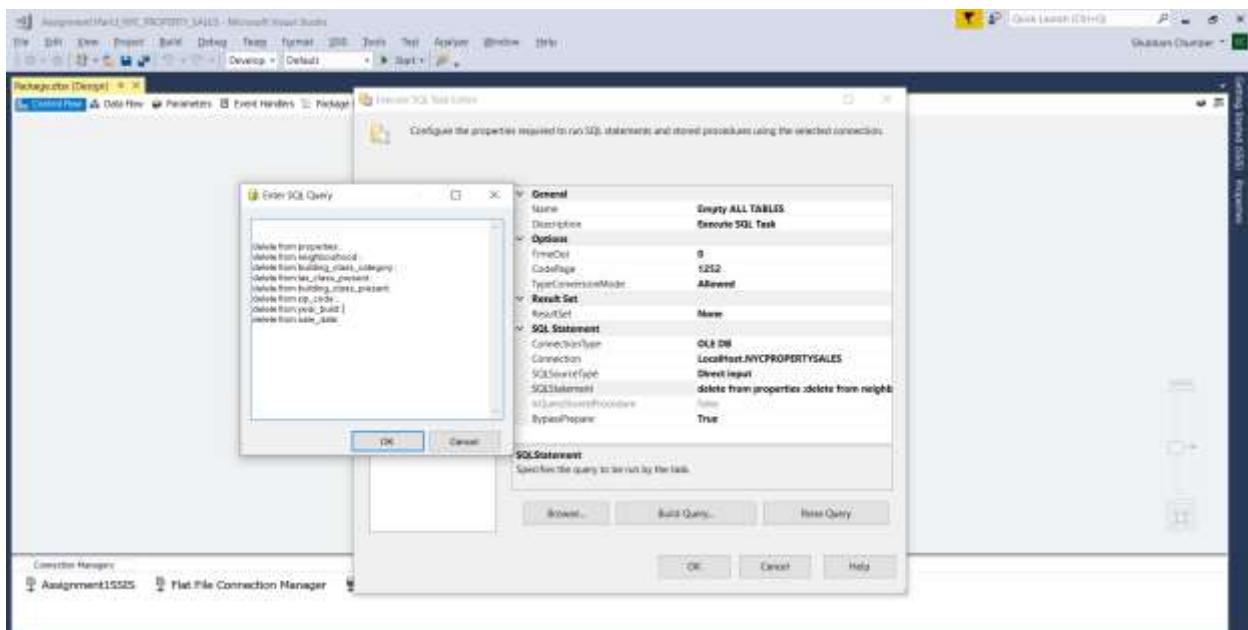


Fig 133: Executing SQL statements to empty the tables

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

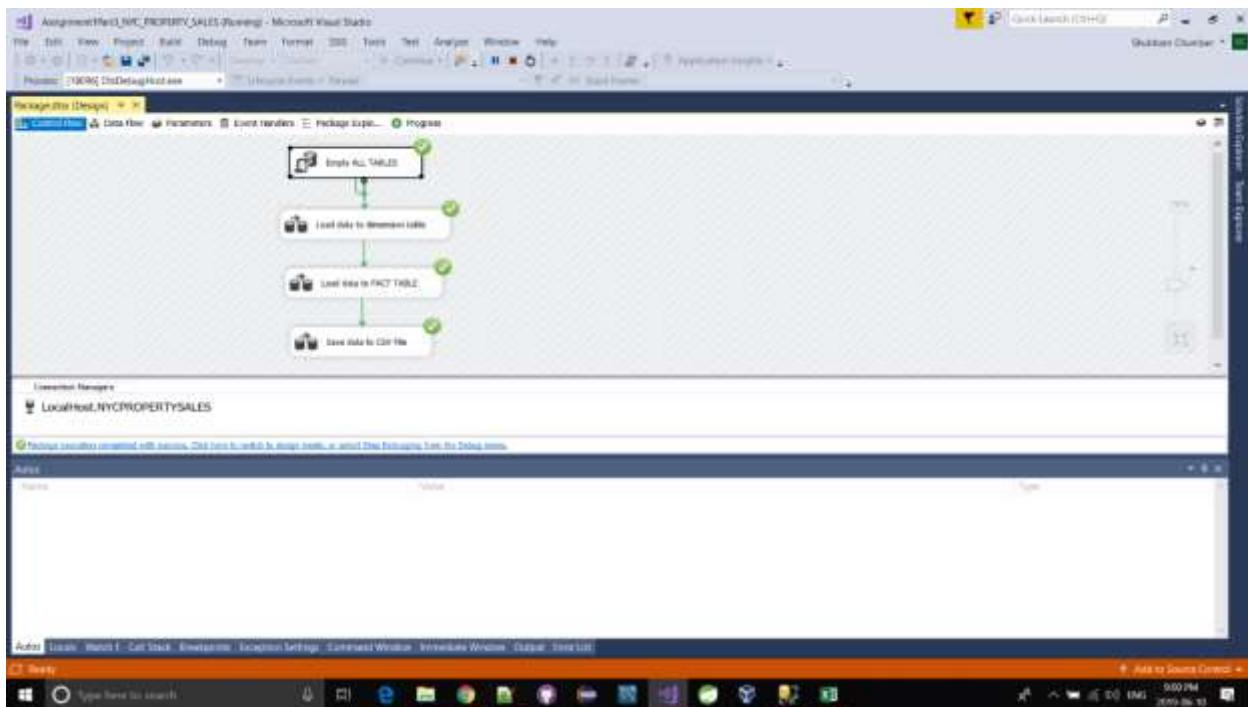


Fig 134: Screen after successful in emptying all tables

## 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.

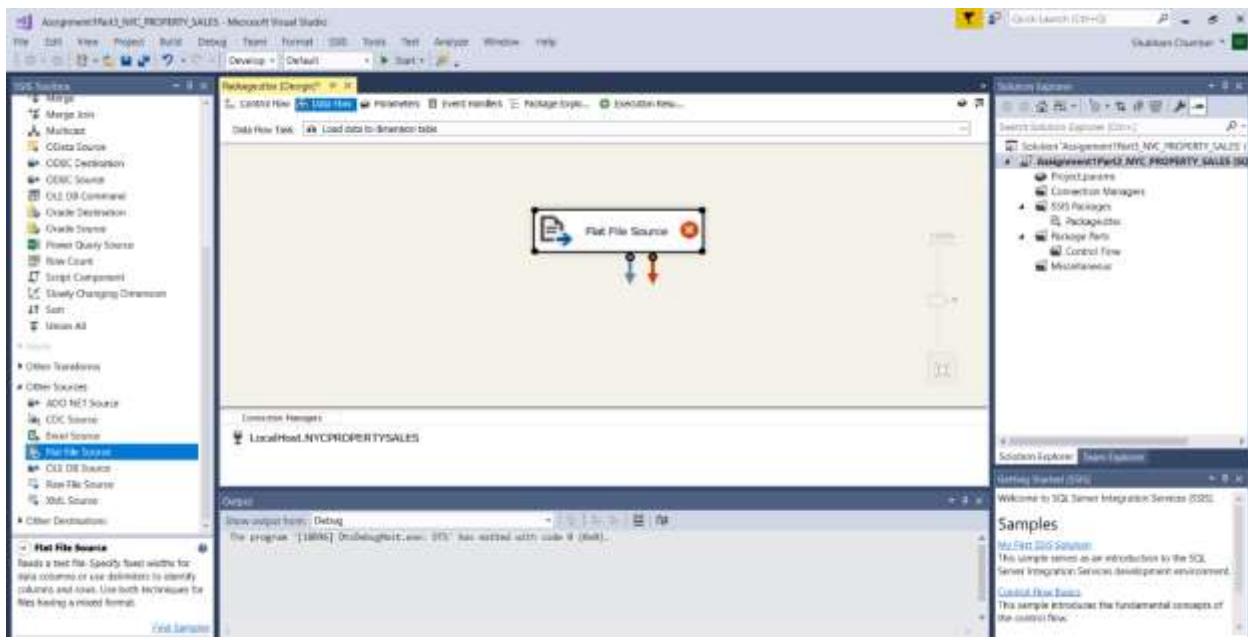


Fig 135:Created a flat file source to fetch data from dataset

Selecting the file from local storage.

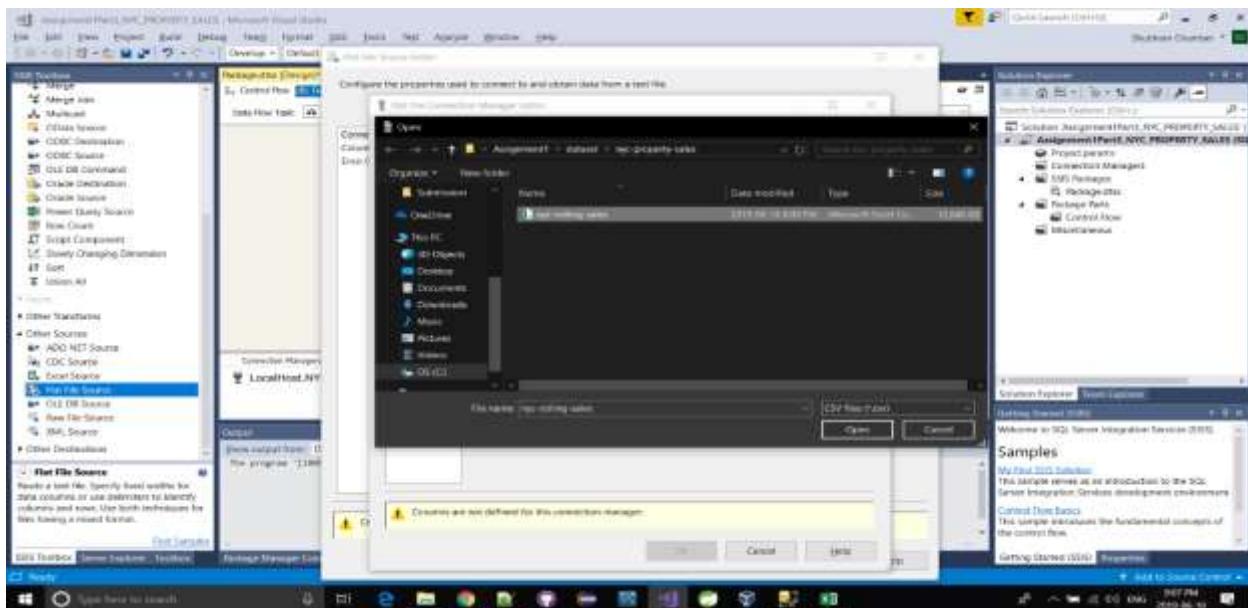


Fig 136:Uploading Dataset “NYC rolling sales”

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

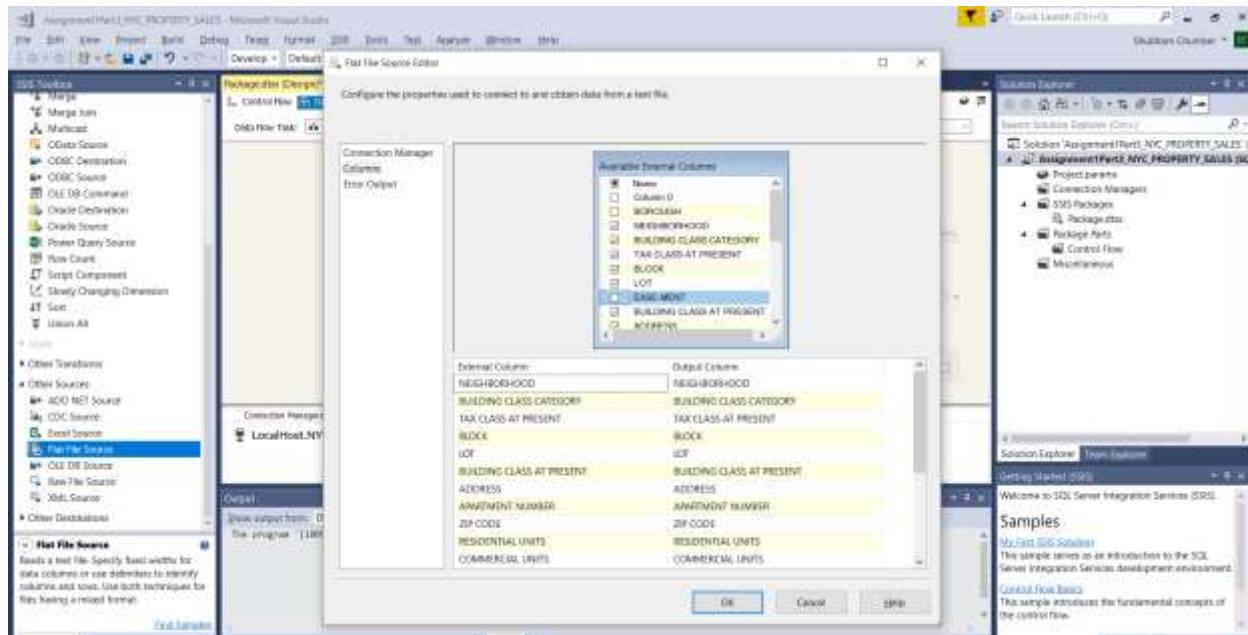


Fig 137:Filtering uploaded dataset

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.

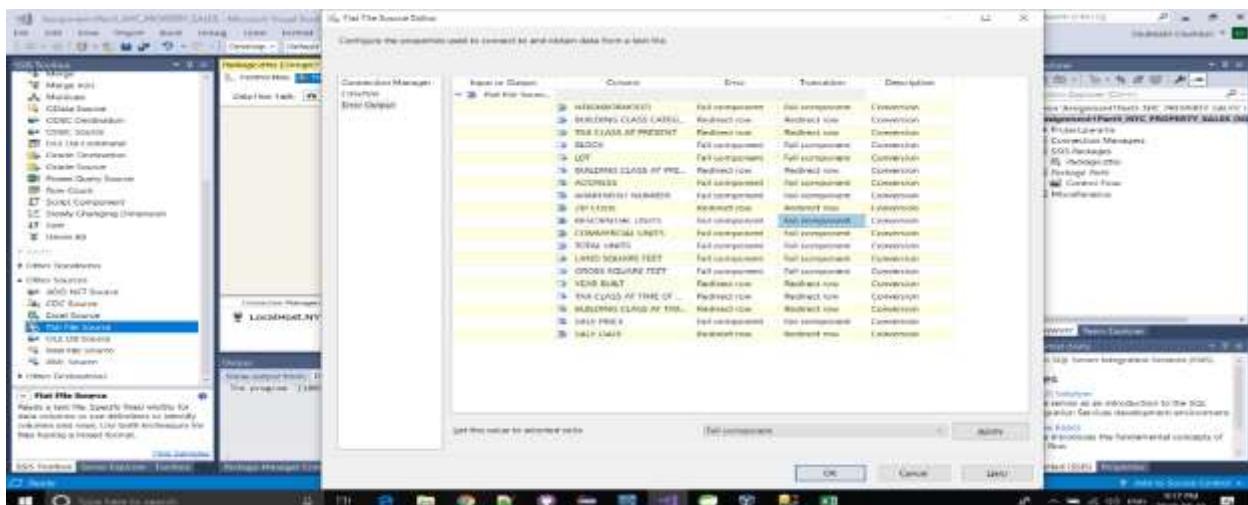


Fig 138:Changing the truncation of attributes

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

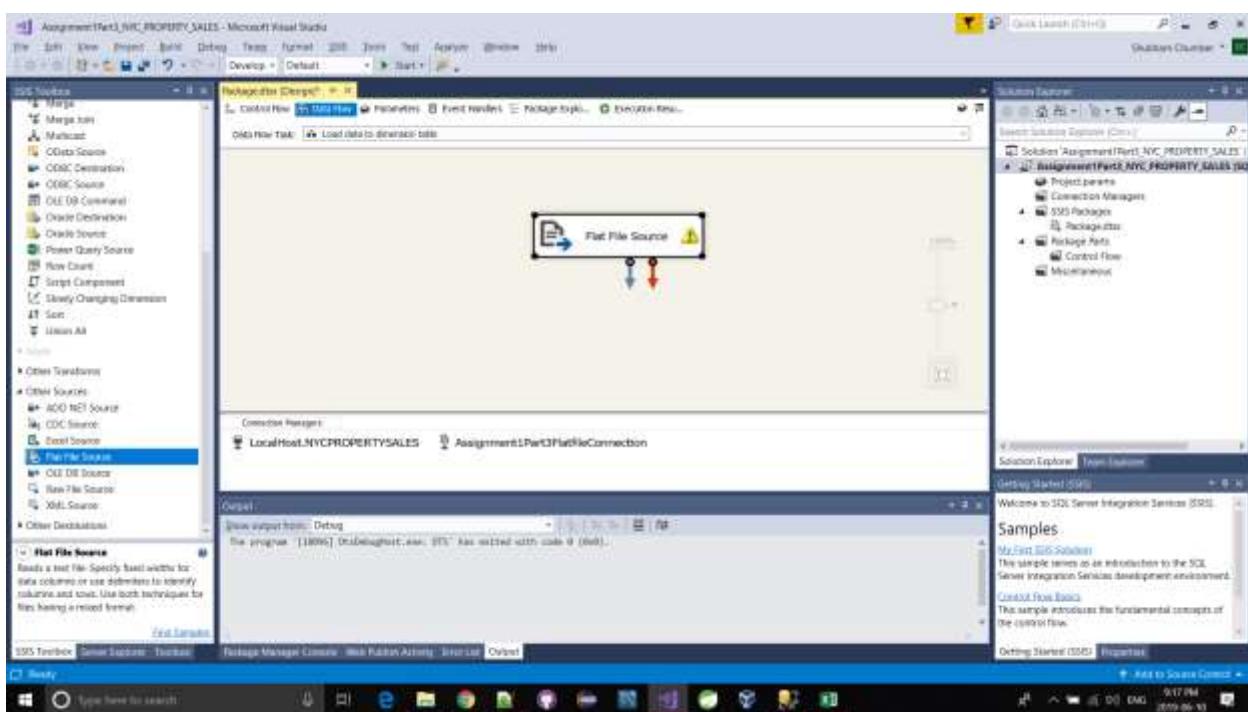


Fig 139:Successfully load the data from dataset

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

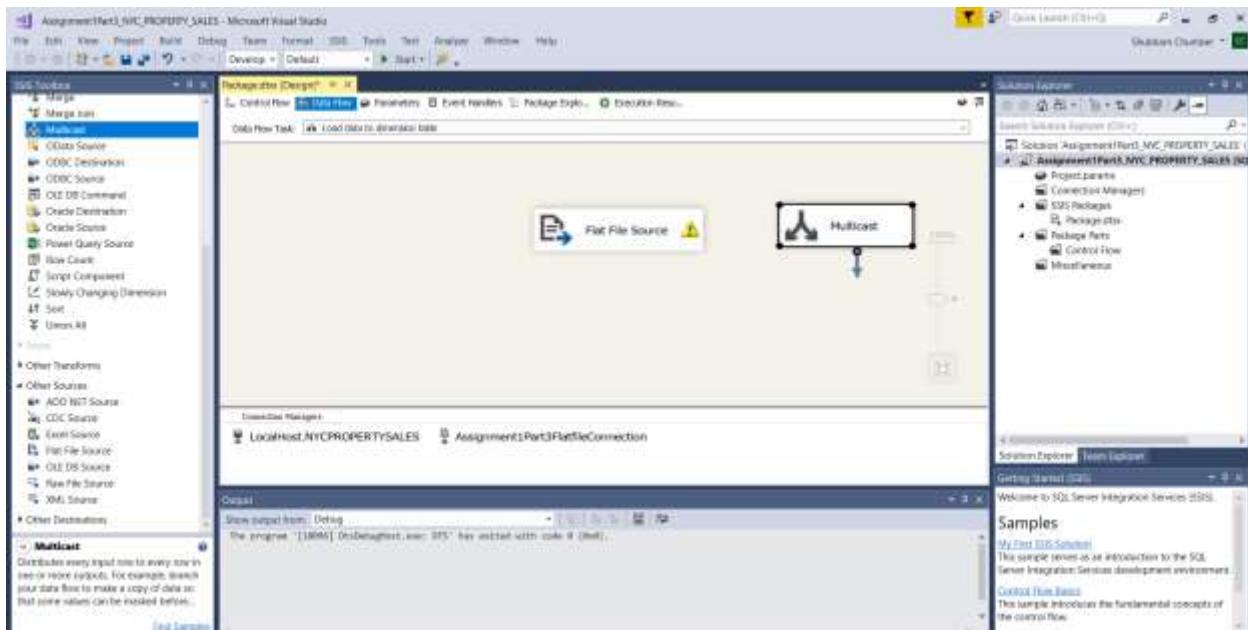


Fig 140:Adding new Multicast block

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

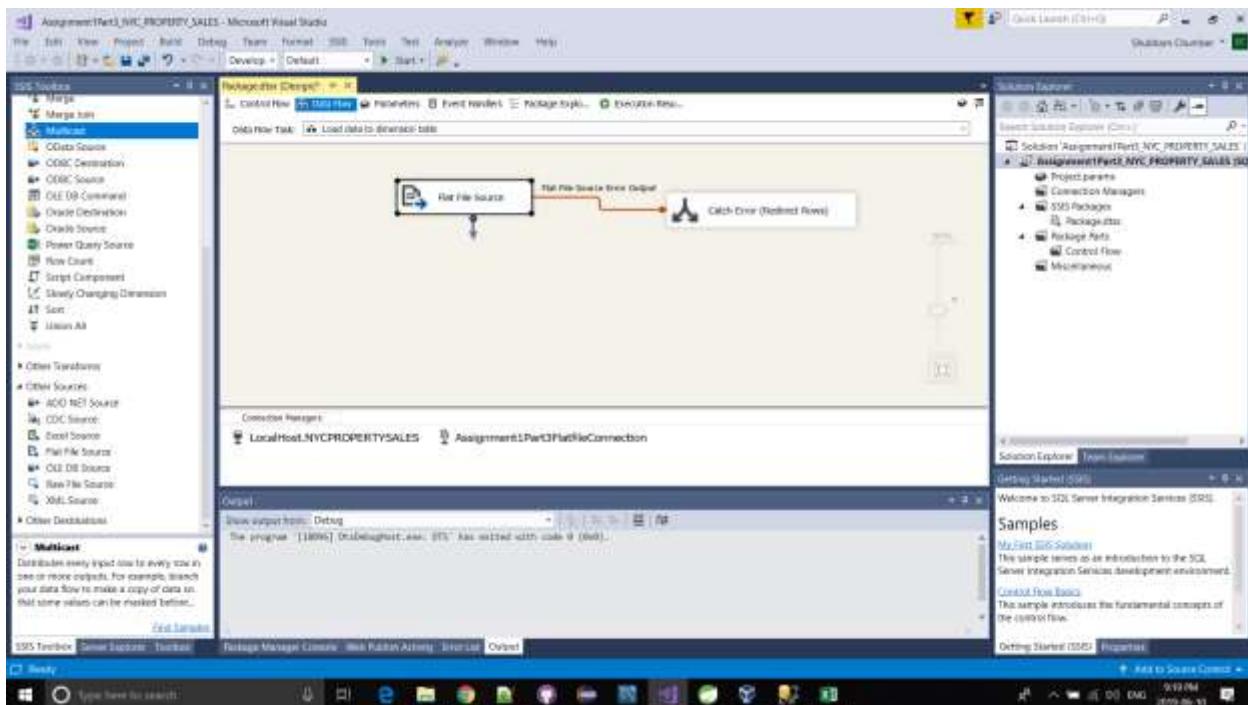


Fig 141:Assigning a task to multicast of catching error

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

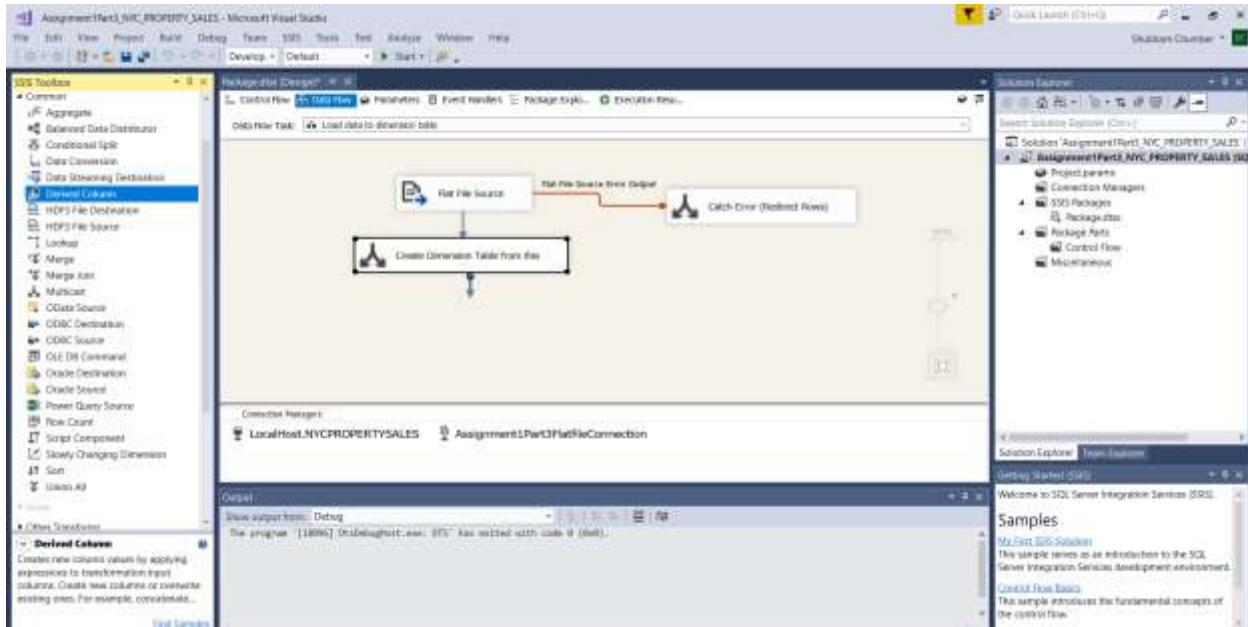


Fig 142:Adding a new multitask block to load dimension tables

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

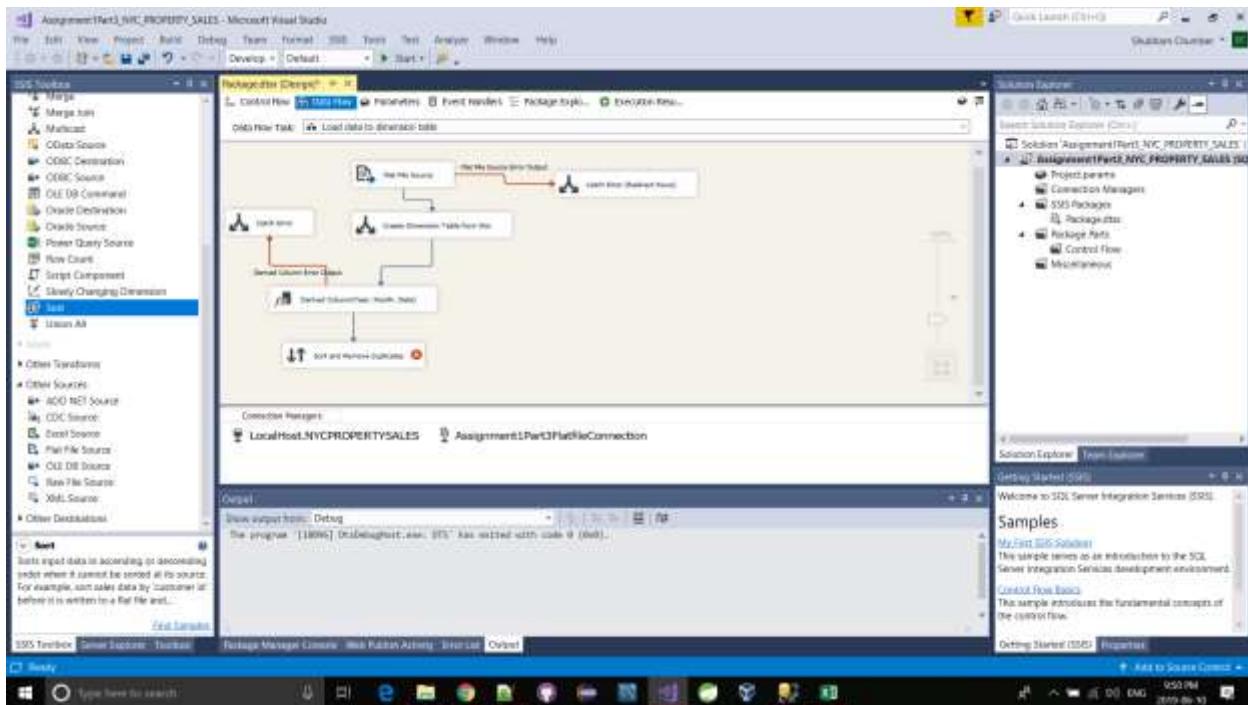


Fig 143:Deriving attributes from the parent attribute

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.

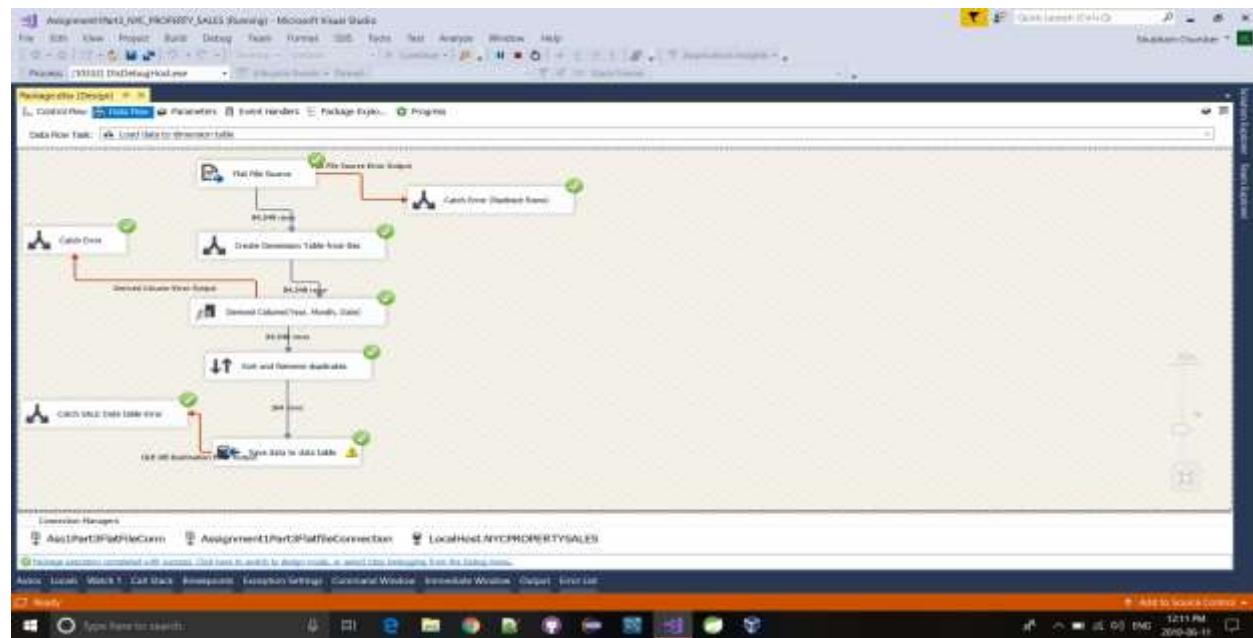


Fig 144:Sorting, removing duplicates and catching records having error

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

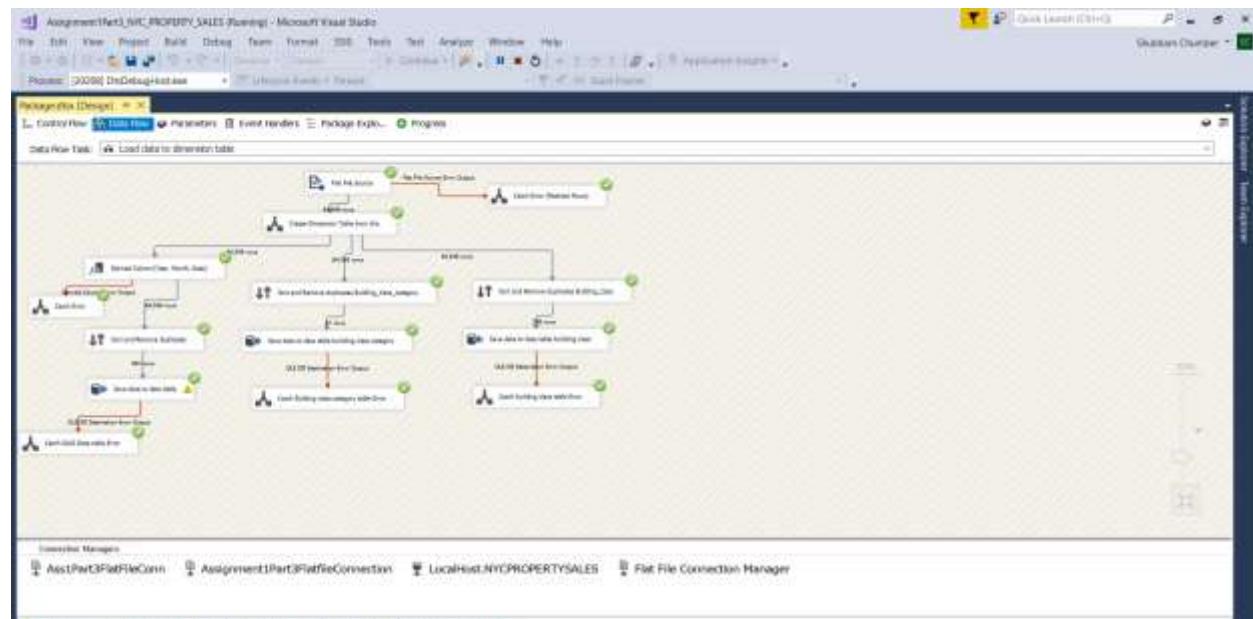


Fig 145:Successful added three dimension tables

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

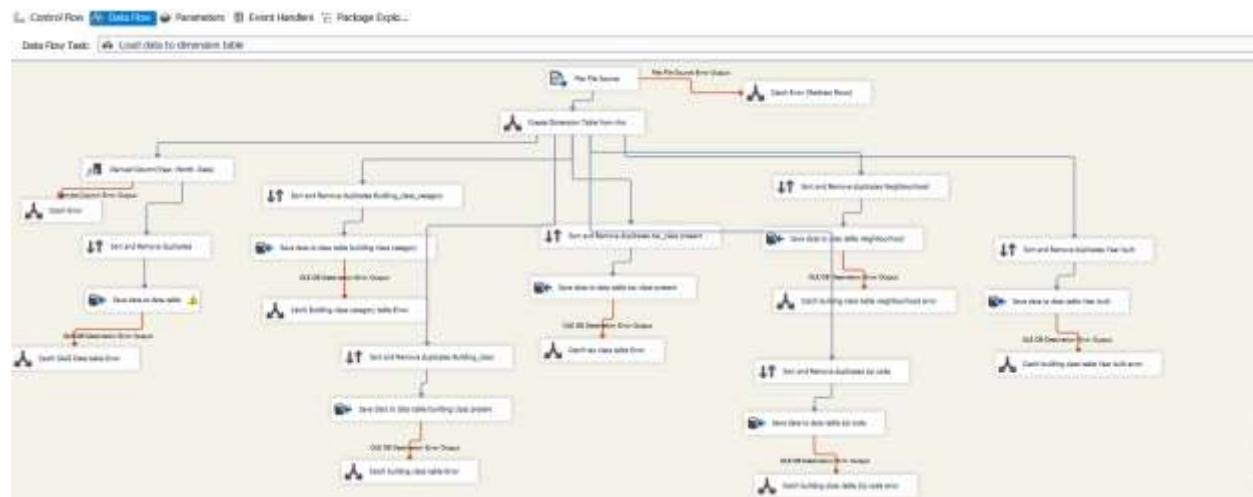


Fig 146:Successfully loaded data into all Dimension tables

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

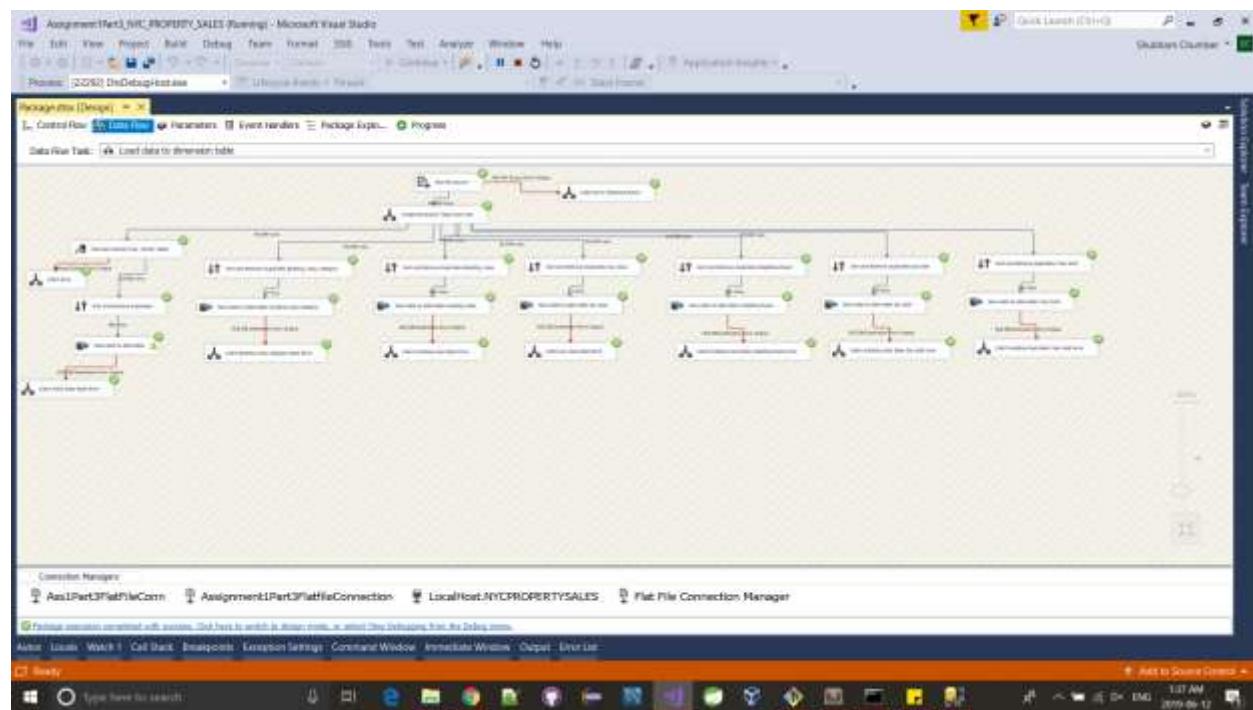


Fig 147:Successfully loaded data into all Dimension tables

### 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.

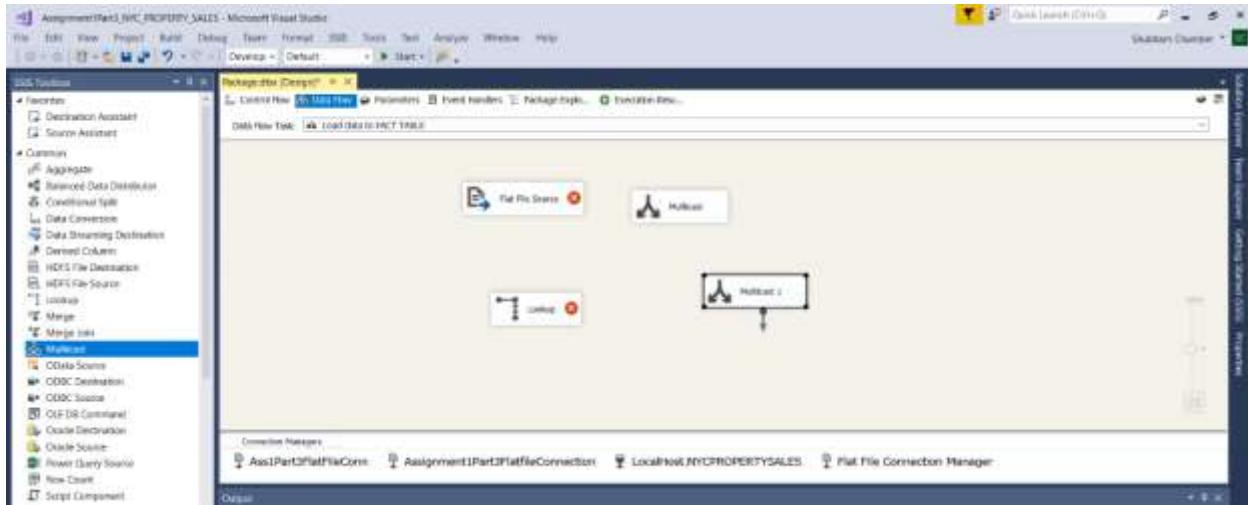


Fig 148:Adding source and multitask block to load data into Fact table

Setting up the Flat file Source and its mappings

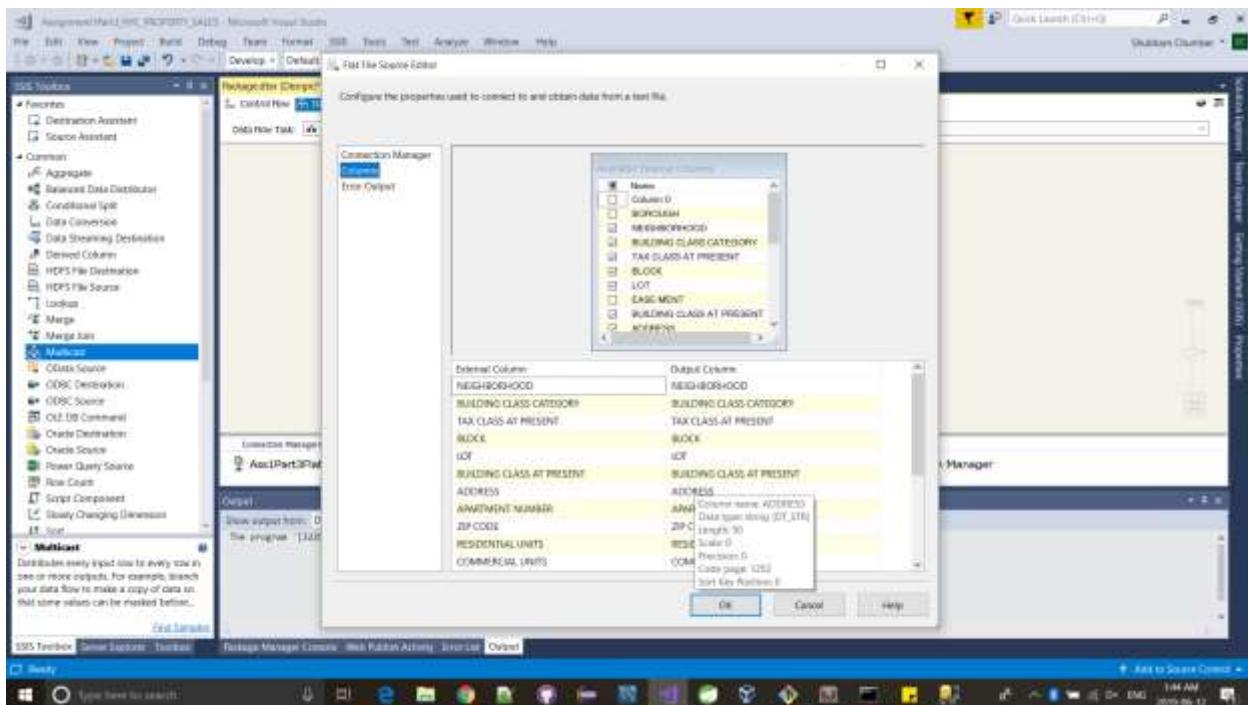


Fig 149: Selecting the required columns

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

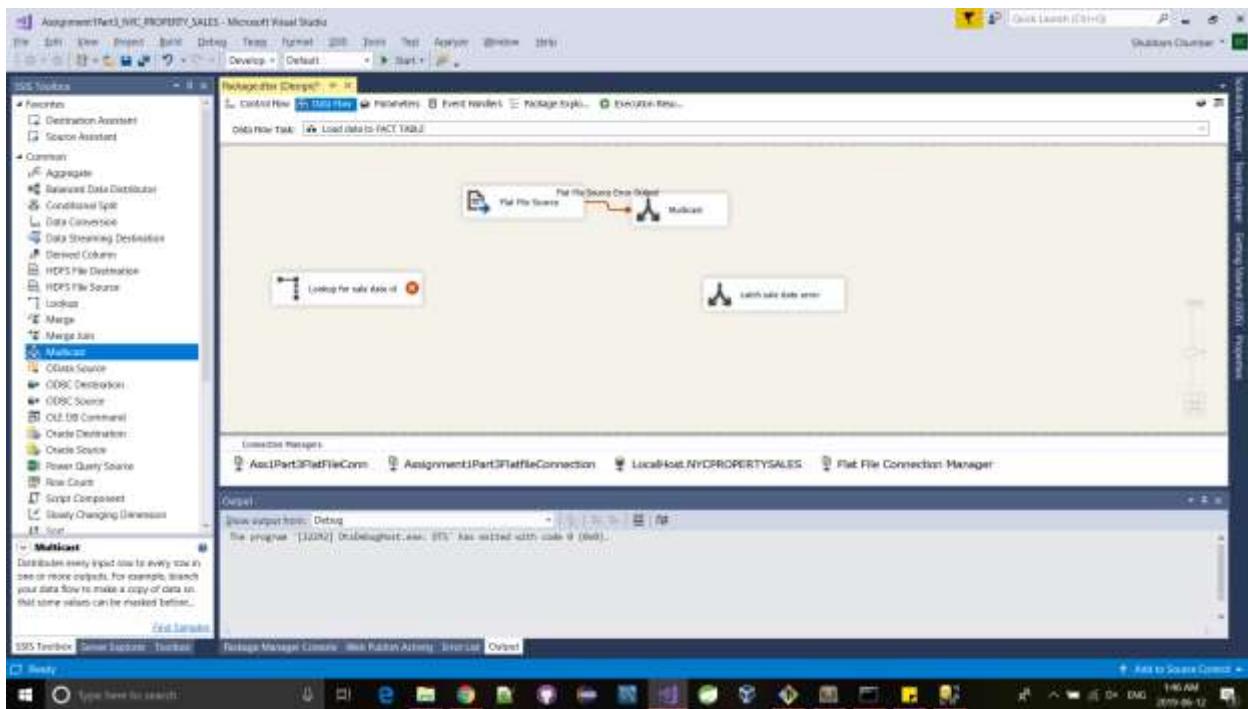


Fig 150: Redirecting the errors into catch error block

Setting up the Lookup for sale\_date\_id by mapping the component to sale\_date table

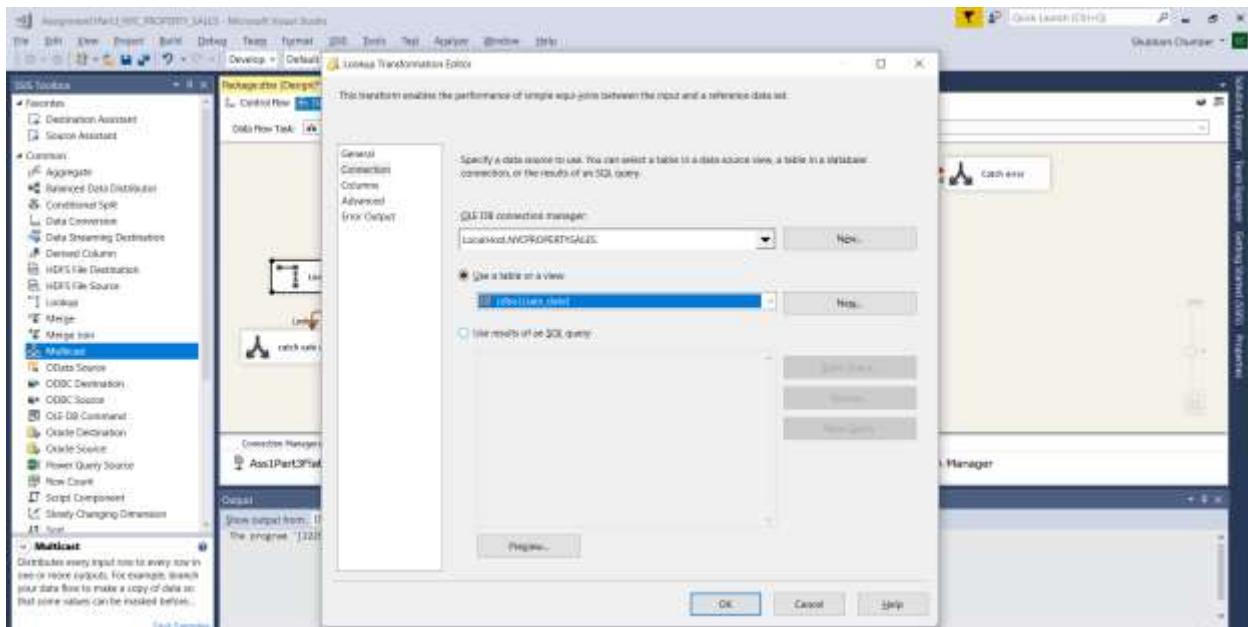


Fig 151:Lookup transformation editor

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

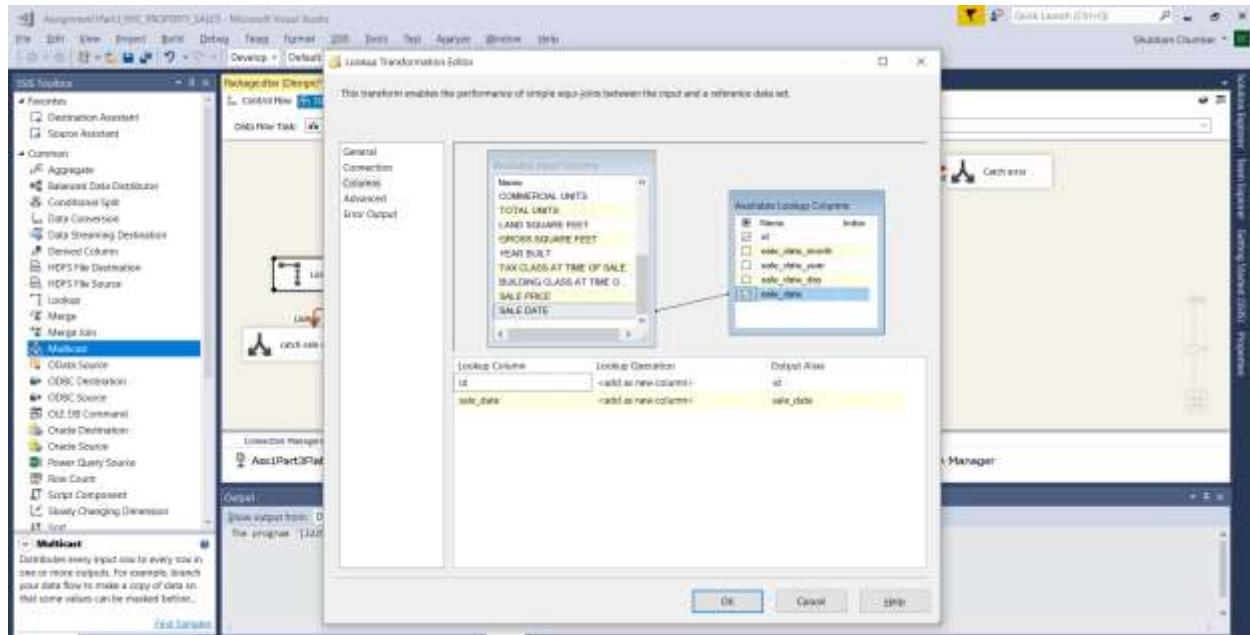


Fig 152: Mapping Dimension table attribute

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

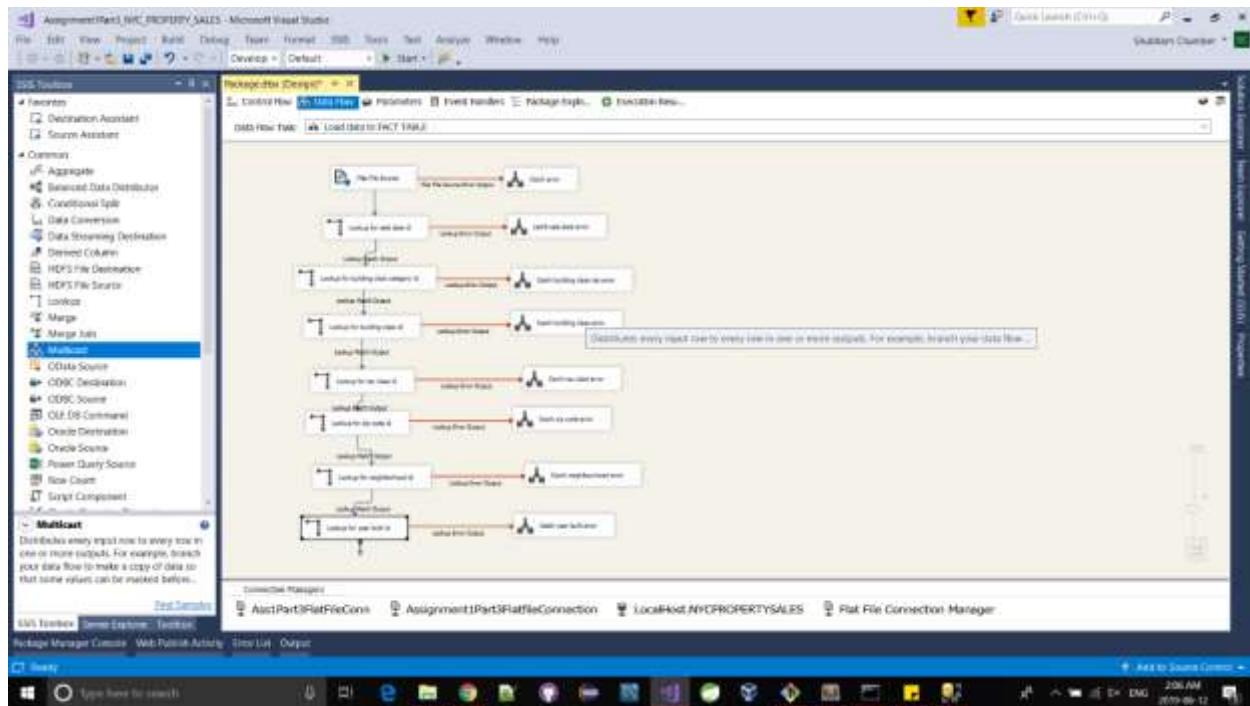


Fig 153: Lookups for Dimension tables

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

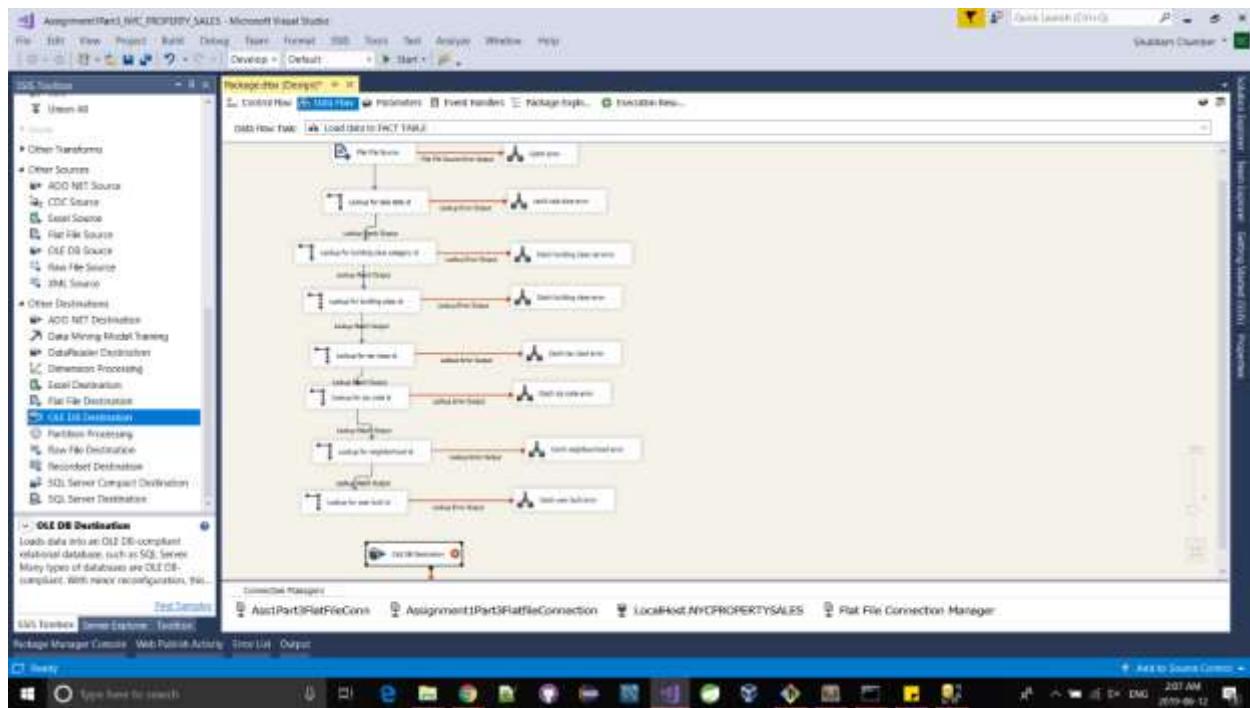


Fig 154:Successfully added OLE DB Destination component

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

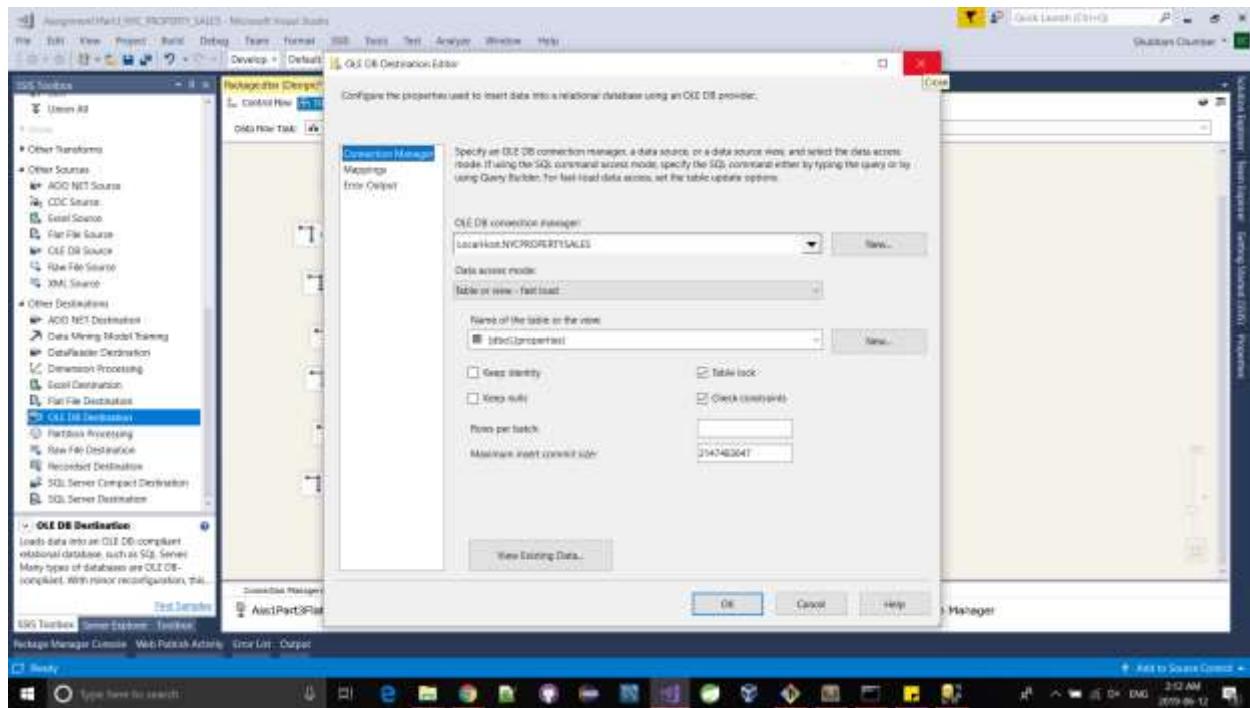


Fig 155:OLE DB connection

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

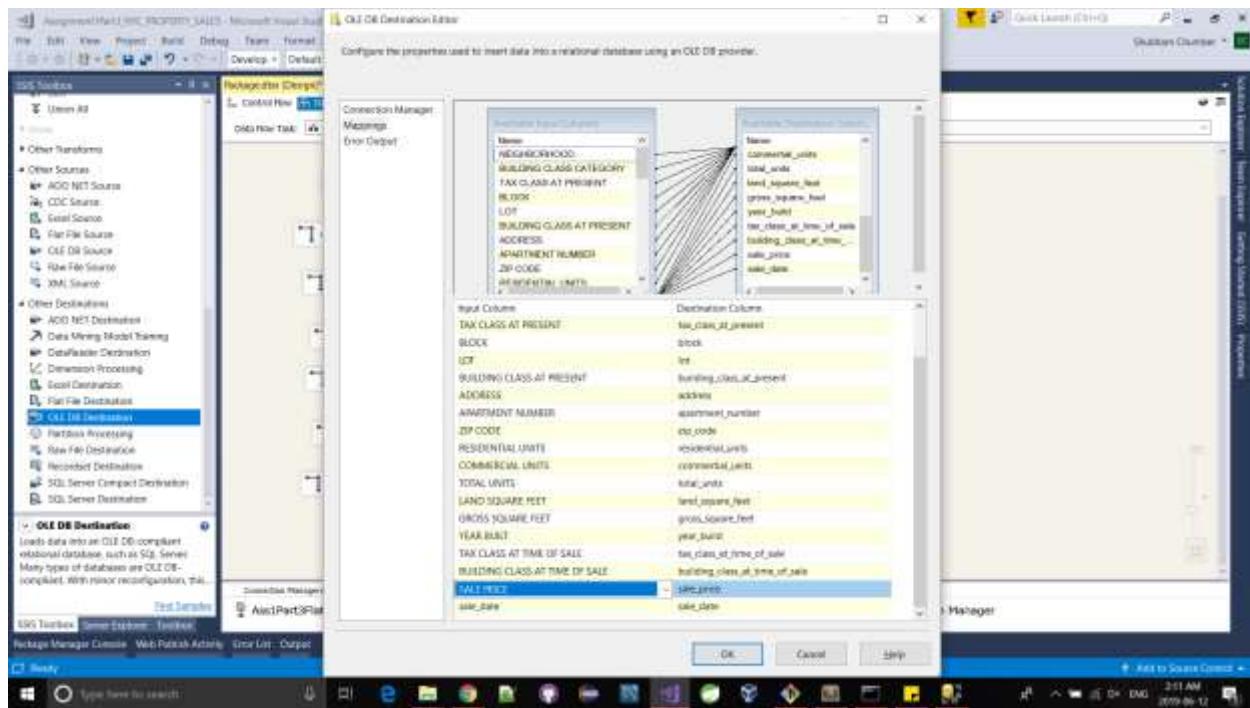


Fig 156: Mapping the OLE DB component with required fields

Complete data flow task of ‘LOAD DATA INTO FACT TABLES’

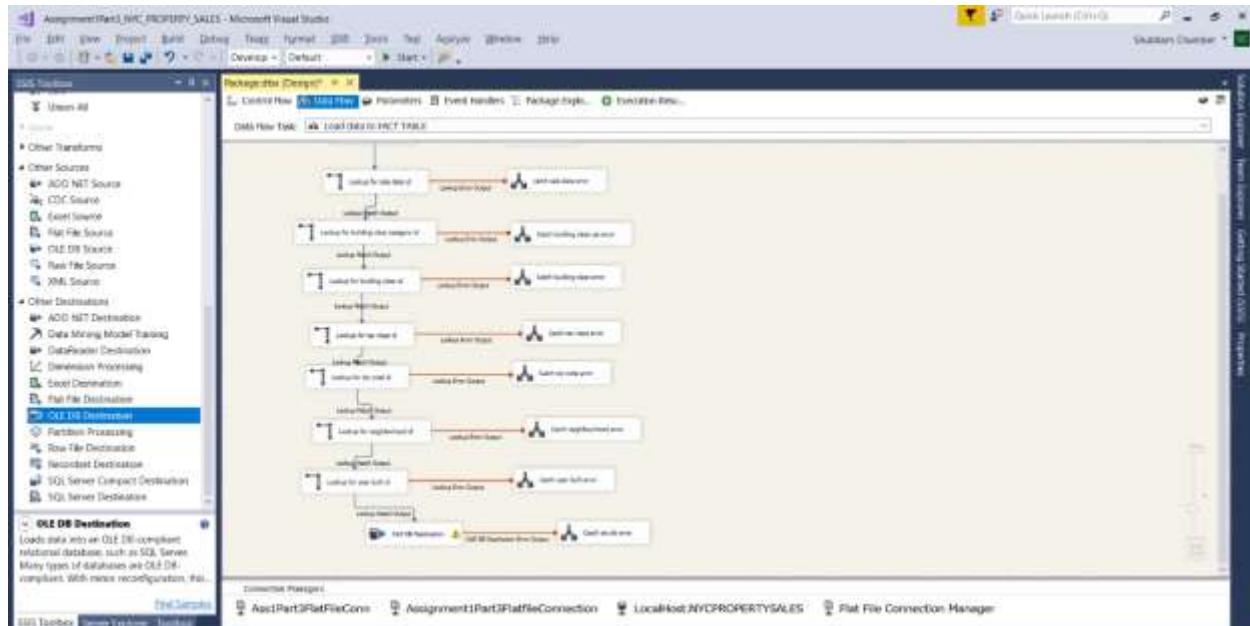


Fig 157: Successfully loaded data into all Fact table

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.

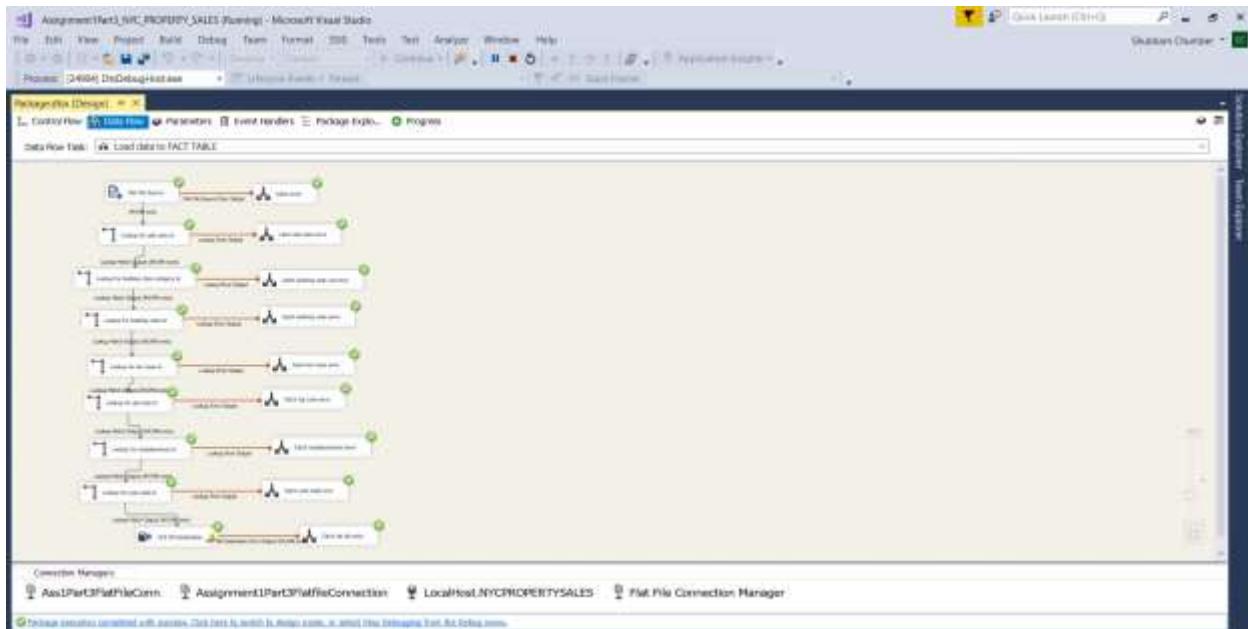


Fig 158:observing the error records transferred to the Catch error block

#### 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.

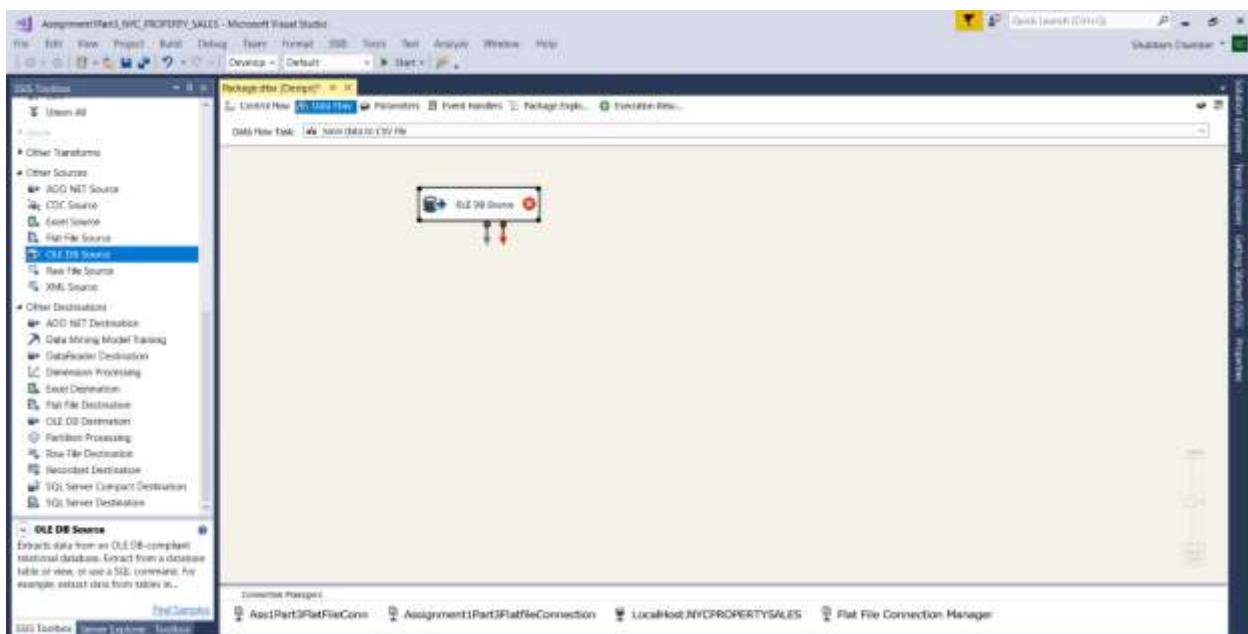


Fig 159:Successfully added OLE DB source block

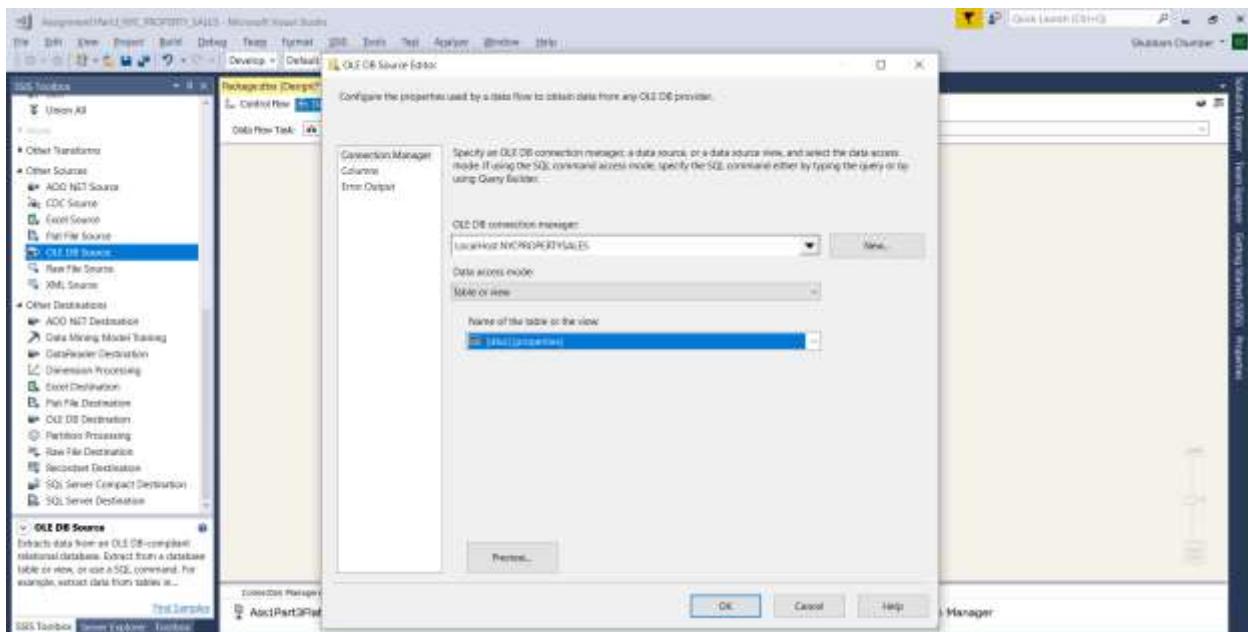


Fig 160:OLE DB connection manager

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

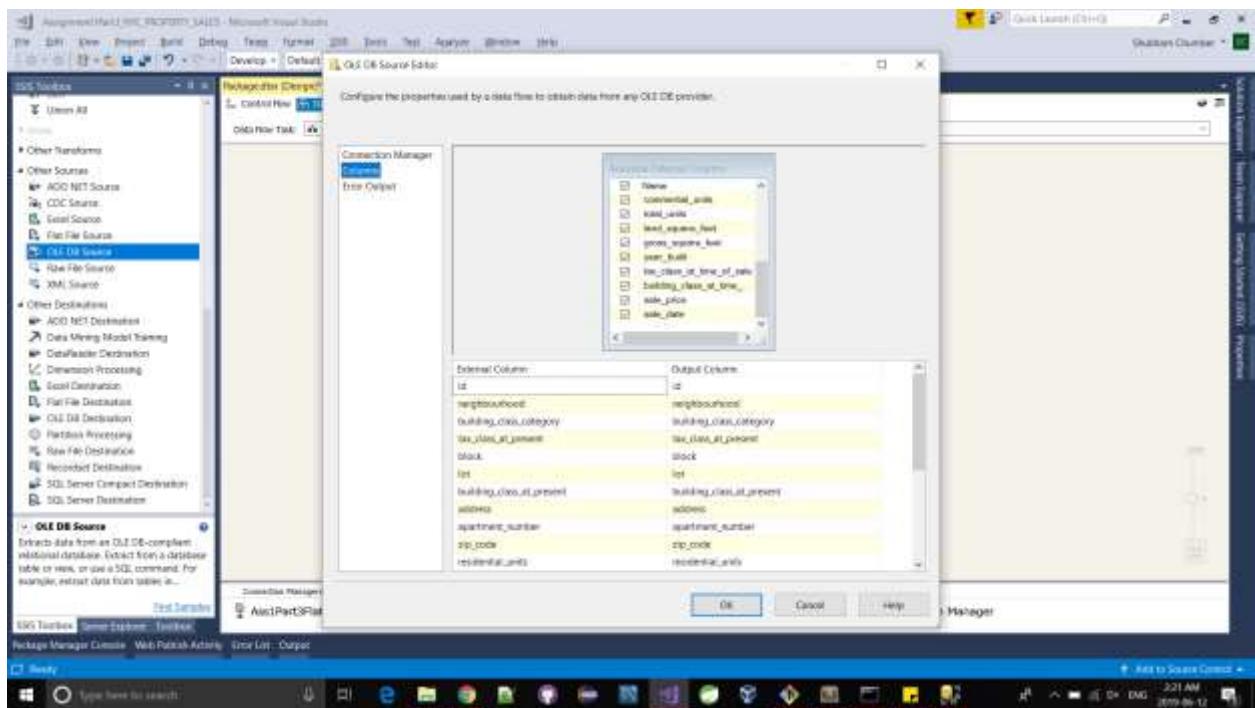


Fig 161:Setting DB source component

## Setting up the DB source component. Redirecting the errors.

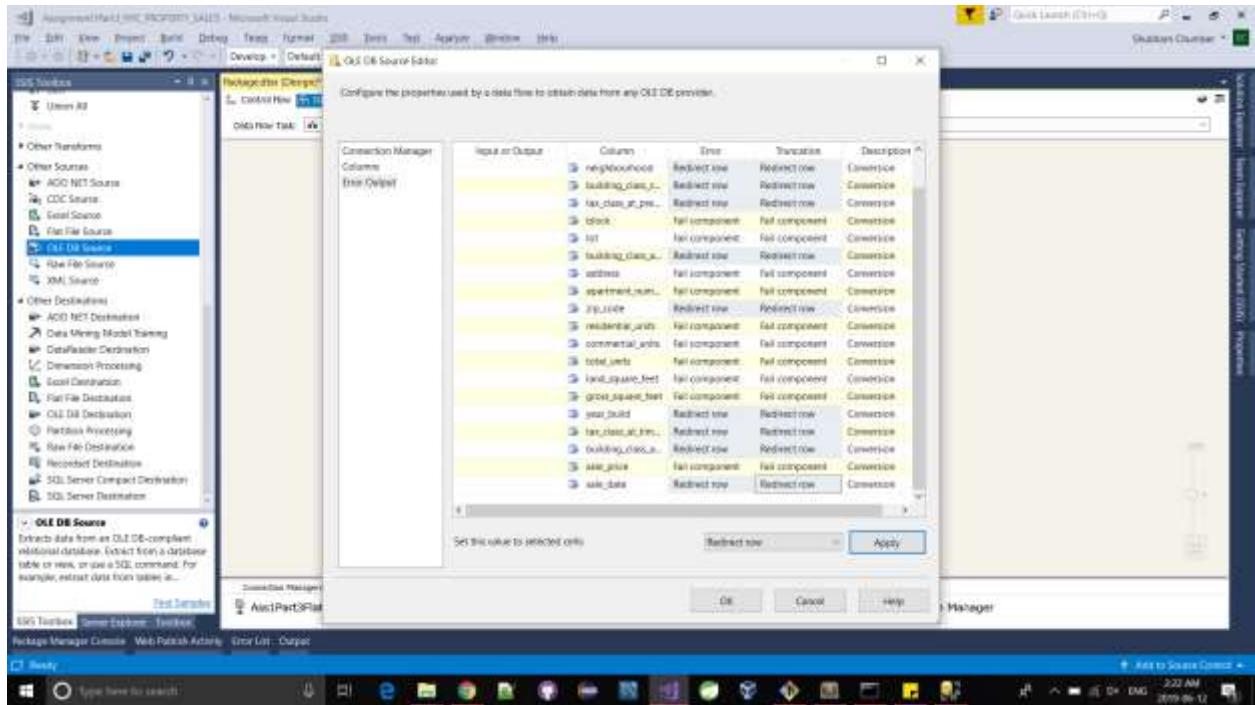


Fig 162:Changing the status of the truncation

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

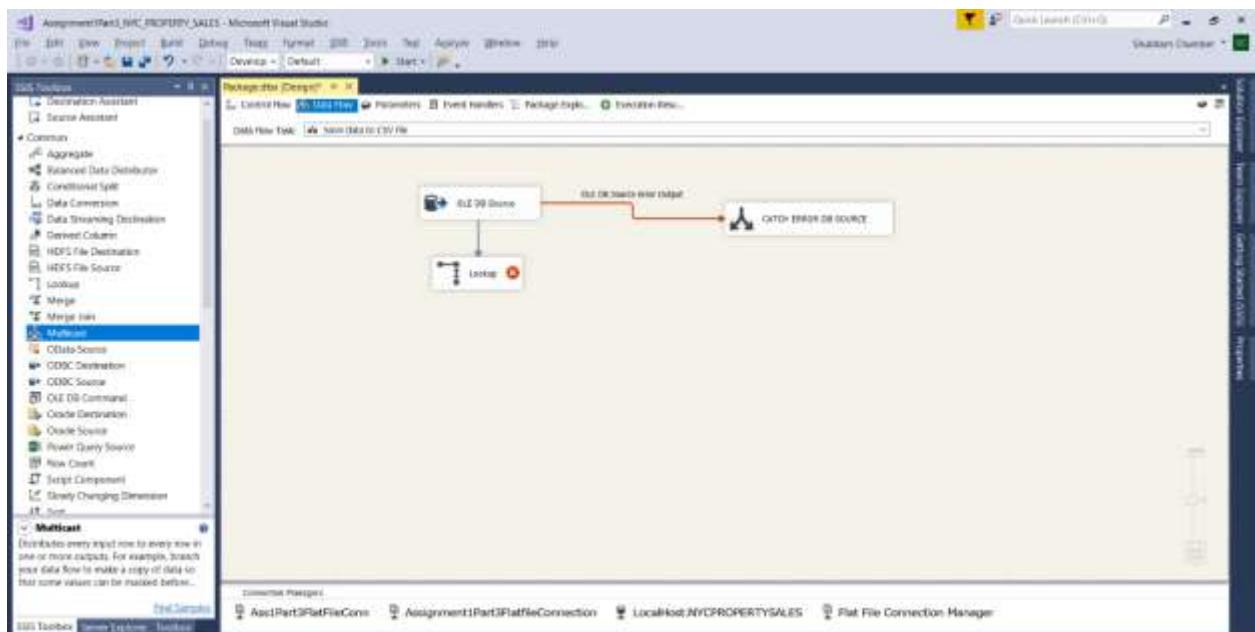


Fig 163:Adding Lookup and Catch error block

## Setting up the Sale date lookup

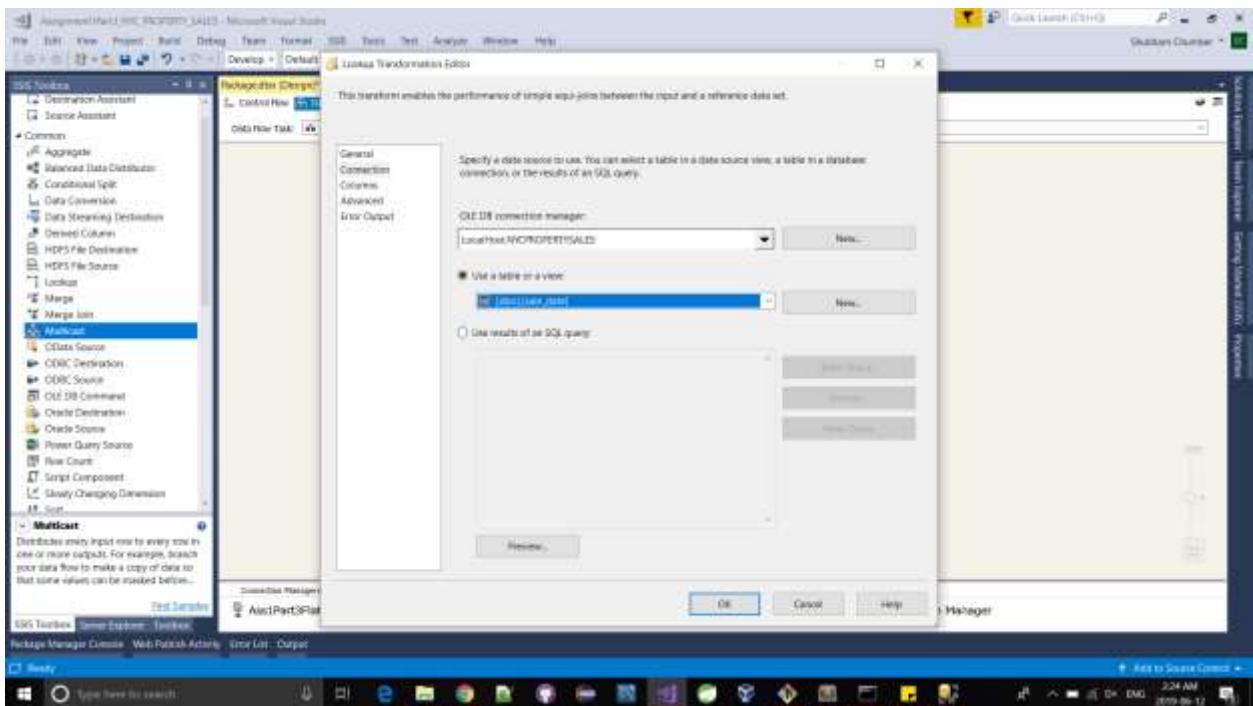


Fig 164:Sales Date Lookup

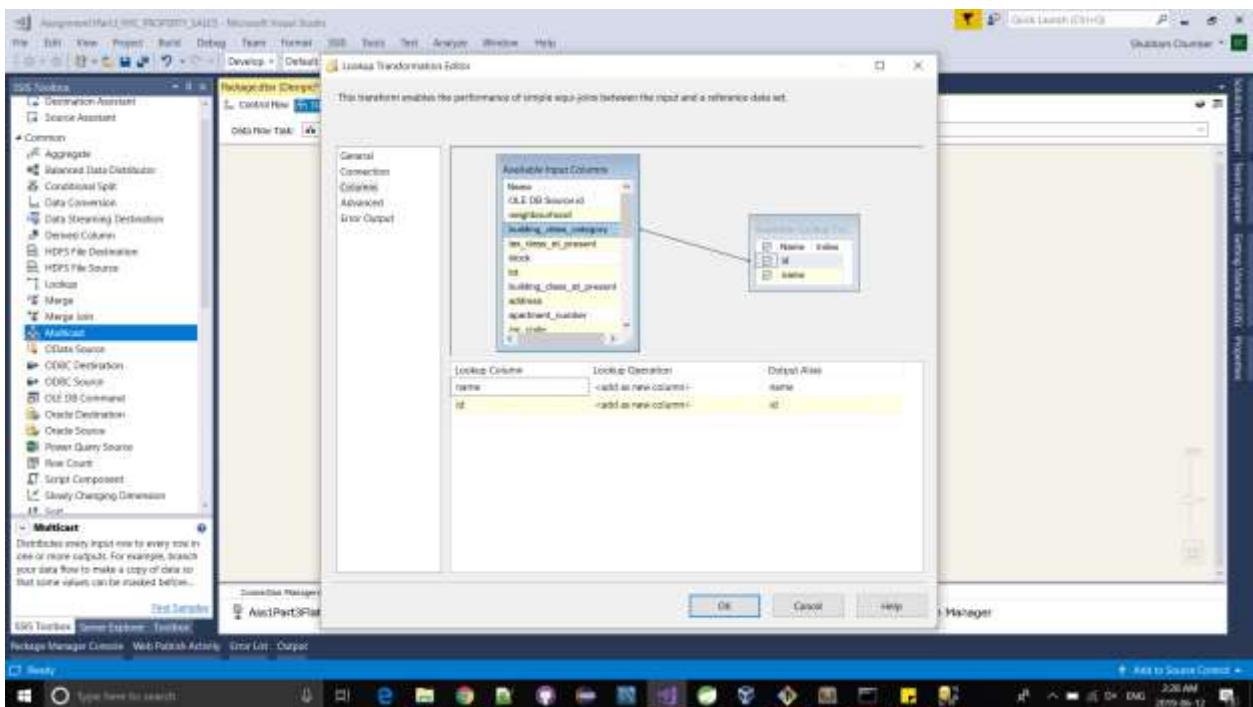


Fig 165:Mapping the Building class category similarly

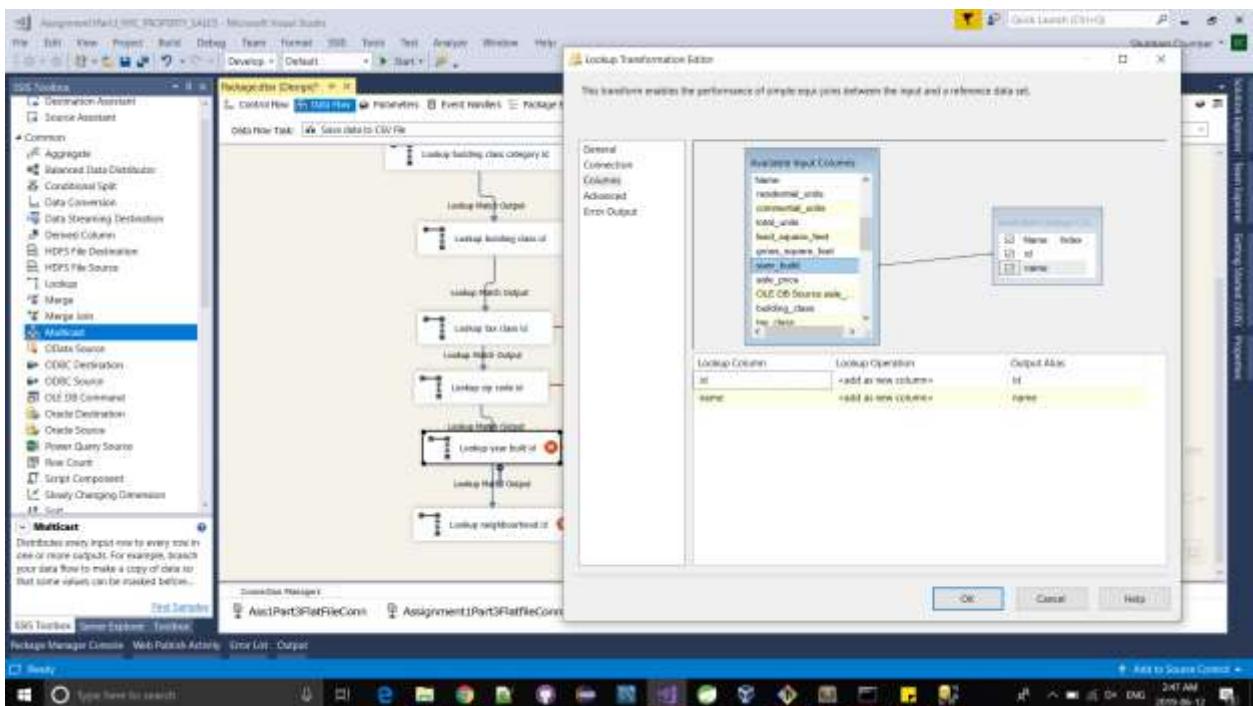


Fig 166:Mapping the Year built similarly

Added all the dimension table columns following same approach

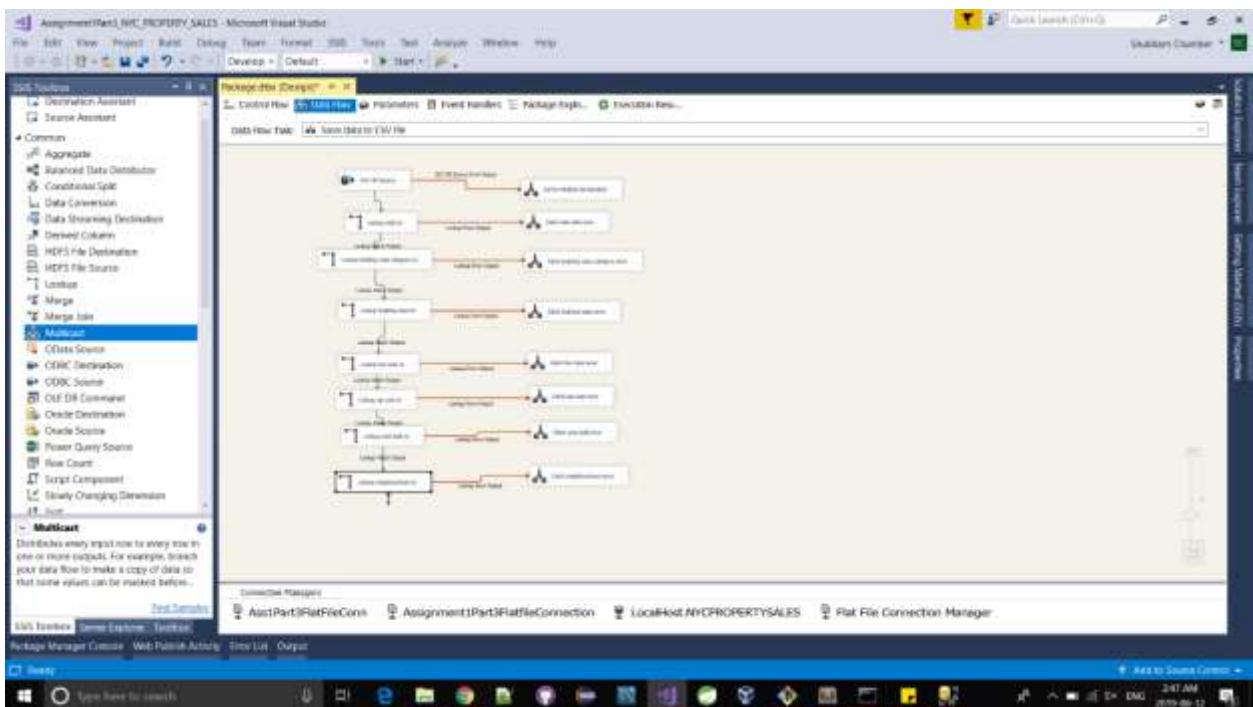


Fig 167:Adding Dimension tables

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

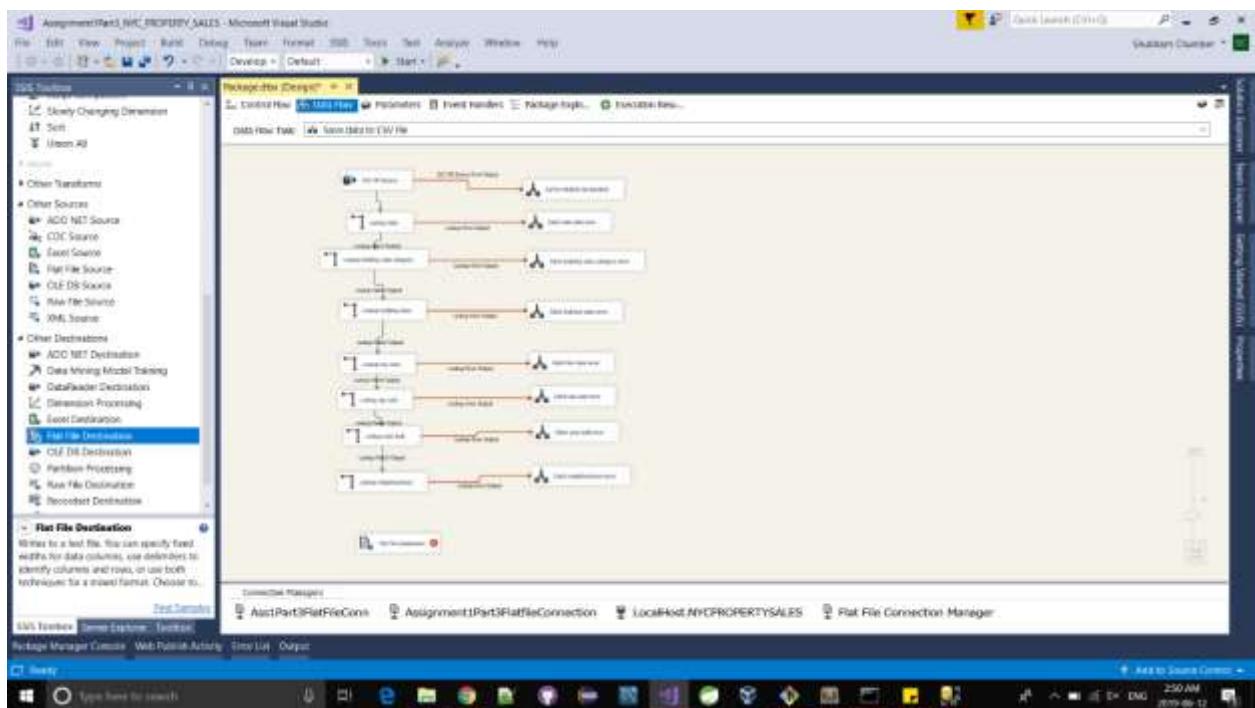


Fig 168:Destination component to csv file

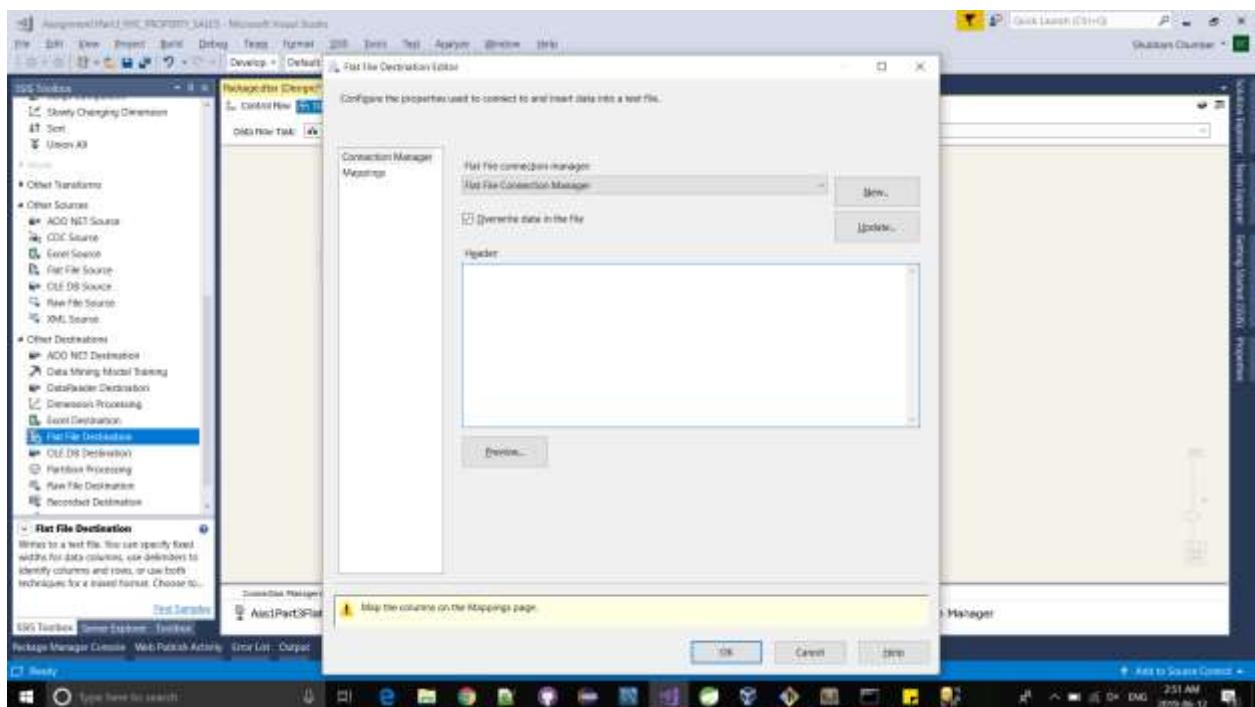


Fig 169;Setting up the Flat file component.

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

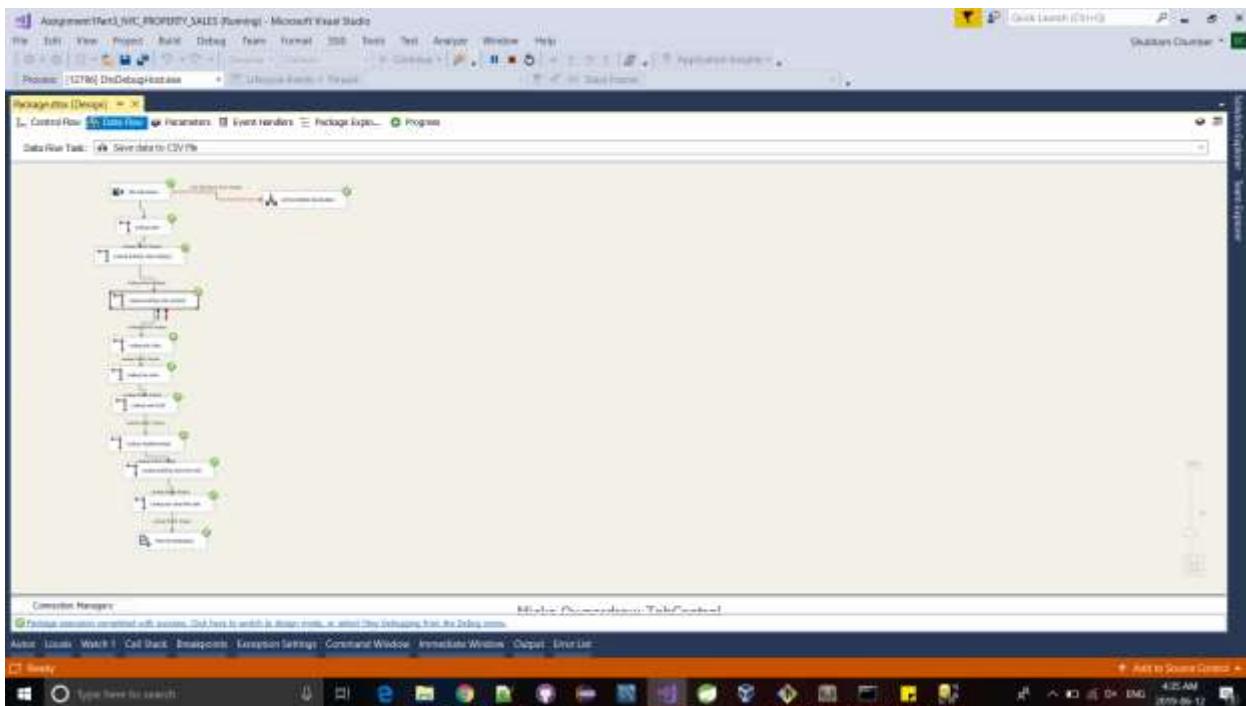


Fig 170: Working Data flow

## Complete Working ETL pipeline of NYCPROPERTYSALES dataset

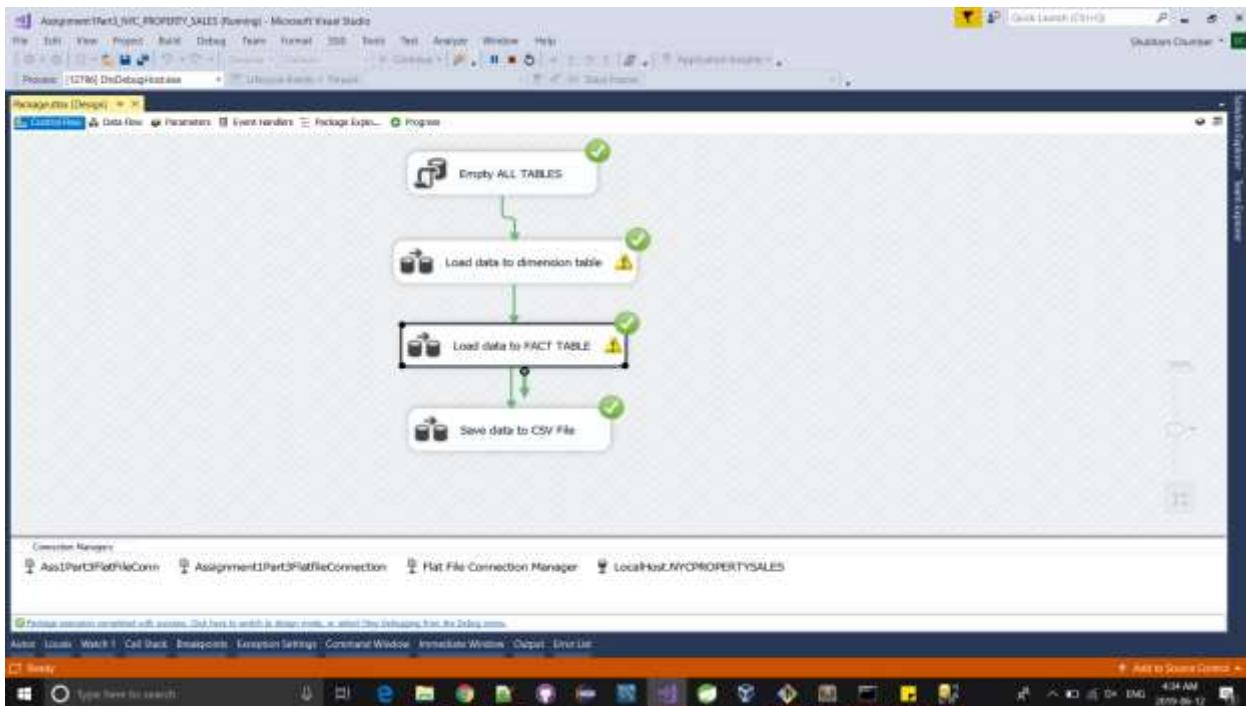


Fig 171: Status of complete ETL Pipeline process

## 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.

The screenshot shows the Microsoft SQL Server Management Studio interface. The Object Explorer on the left shows the database structure, including tables like Neighborhood, Building\_Class\_Categories, Tax\_Class\_Present, Building\_Class\_Present, Sale\_Date, Zip\_Code, and Year\_Build. The central pane displays a T-SQL query result set titled 'Results' for the top 1000 rows of the 'Properties' table. The columns listed in the query are Neighborhood, building\_class\_category, tax\_class\_present, stock, lot, building\_class\_present, address, and apartment\_number. The results show various addresses across New York City, such as 101 Avenue B, 234 EAST 4TH STREET, 187 EAST 3RD STREET, 154 EAST 7TH STREET, 301 EAST 10TH STREET, 506 EAST 12TH STREET, 216 AVENUE B, 520 EAST 14TH STREET, 141 AVENUE D, 629 EAST 5TH STREET, 191 AVENUE C, and 530-52 EAST 5TH STREET. The bottom right pane shows connection details for the current session.

	Neighborhood	building_class_category	tax_class_present	stock	lot	building_class_present	address	apartment_number
1	3006	673	163	302	6	2063	101 AVENUE B	
2	3006	673	162	309	26	2068	234 EAST 4TH STREET	
3	3006	673	168	309	38	2068	187 EAST 3RD STREET	
4	3006	673	170	402	21	2066	154 EAST 7TH STREET	
5	3006	673	169	404	55	2063	301 EAST 10TH STREET	
6	3006	673	168	405	16	2068	506 EAST 12TH STREET	
7	3006	673	170	408	32	2066	216 AVENUE B	
8	3006	673	168	407	18	2064	520 EAST 14TH STREET	
9	3006	674	168	379	34	2077	141 AVENUE D	
10	3006	674	168	387	183	2087	629 EAST 5TH STREET	
11	3006	673	168	384	44	2078	191 AVENUE C	
12	2006	673	170	400	21	2073	530-52 EAST 5TH STREET	
13	3006	678	168	378	14	2067	344 EAST 7TH STREET	

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

The screenshot shows the Microsoft SQL Server Management Studio interface. The left pane displays the Object Explorer with the database 'NYCOPROPERTIESALES' selected. The right pane shows a query results grid and a properties window.

**Query Results:**

```
SELECT TOP (1000) [id]
FROM [NYCOPROPERTIESALES].[dbo].[neighborhood]
```

ID	Name
1	AIRPORT LA GUARDIA
2	ALBANY CITY
3	AMHADE
4	ARON HEIGHTS
5	ARRONCHAR
6	ARROCHA SHORE ACRES
7	ARVERNE
8	ASTORIA
9	BATH BEACH
10	BATHSTATE
11	BAY RIDGE
12	BAYCHESTER
13	BAYSIDE
14	MARVELL HALL-ASTORIA

**Properties Window:**

- Current connection parameters:
  - Aggregate Status: Connection is active. Elapsed time: 00:00:00.137. Result time: 2019-06-11 10:00:00 AM. Rows returned: 254. Start time: 2019-06-11 10:00:00 AM. State: Open.
  - Connections: Connection name: DESKTOP-NCOBRA0. Connection state: Active-Active-Standby. Connection step: 00:00:00.137. Connection warning: Not encrypted. Connection failed: 2019-06-11 10:00:00 AM. Connection max: 254. Connection count: 1. Connection state: Open. Display name: DESKTOP-NCOBRA0. Logon name: DESKTOP-NCOBRA0\user. Server name: DESKTOP-NCOBRA0. Server version: 14.0.3000. Session Tracing ID: 0. SPID: 57.
- Name: The name of the connection.

Fig 173: Neighborhood

### → Building\_class\_present

The screenshot shows the Microsoft SQL Server Management Studio interface. The left pane displays the Object Explorer with the database 'NYCOPROPERTIESALES' selected. The right pane shows a query results grid and a properties window.

**Query Results:**

```
SELECT TOP (1000) [id]
FROM [NYCOPROPERTIESALES].[dbo].[building_class_present]
```

ID	Building Class Present Name
1	CM7
2	AB
3	CM2
4	AB2
5	CM3
6	AB3
7	CM4
8	AB4
9	CM5
10	AB5
11	CM6
12	AB6
13	CM7
14	AB7

**Properties Window:**

- Current connection parameters:
  - Aggregate Status: Connection is active. Elapsed time: 00:00:02.186. Result time: 2019-06-11 10:00:00 AM. Rows returned: 167. Start time: 2019-06-11 10:00:00 AM. State: Open.
  - Connections: Connection name: DESKTOP-NCOBRA0. Connection state: Active-Active-Standby. Connection step: 00:00:02.186. Connection warning: Not encrypted. Connection failed: 2019-06-11 10:00:00 AM. Connection max: 167. Connection count: 1. Connection state: Open. Display name: DESKTOP-NCOBRA0. Logon name: DESKTOP-NCOBRA0\user. Server name: DESKTOP-NCOBRA0. Server version: 14.0.3000. Session Tracing ID: 0. SPID: 57.
- Name: The name of the connection.

Fig 174: Build class present

## → Building\_class\_category

The screenshot shows the Microsoft SQL Server Management Studio interface. The Object Explorer on the left shows the database structure, including tables like `dbo.building_class_category`, `dbo.building_class_present`, and `dbo.sale_date`. The central pane displays the results of a query:

```

SELECT TOP (1000) [id]
,[name]
FROM [NYCPropetySales].[dbo].[building_class_category];

```

The results table has two columns: `id` and `name`. The data is as follows:

<code>id</code>	<code>name</code>
1	01 ONE FAMILY DWELLINGS
2	02 TWO FAMILY DWELLINGS
3	03 THREE FAMILY DWELLINGS
4	04 TAX CLASS 1 CONDO
5	05 TAX CLASS 1 VACANT LAND
6	06 TAX CLASS 1 - OTHER
7	07 RENTALS - WALKUP APARTMENTS
8	08 RENTALS - ELEVATOR APARTMENTS
9	09 COOPS - WALKUP APARTMENTS
10	10 COOPS - ELEVATOR APARTMENTS
11	11 SPECIAL CONDO BILLING LOTS
12	12A CONDO RENTALS
13	12B CONDOS - WALKUP APARTMENTS
14	13 CONDOS - ELEVATOR APARTMENTS

The status bar at the bottom indicates "Query executed successfully".

Fig 175: Building class category

## → Sale\_date

The screenshot shows the Microsoft SQL Server Management Studio interface. The Object Explorer on the left shows the database structure, including tables like `dbo.sale_date`. The central pane displays the results of a query:

```

SELECT TOP (1000) [id]
,[sale_date_month]
,[sale_date_year]
,[sale_date_day]
,[sale_date]
FROM [NYCPropetySales].[dbo].[sale_date];

```

The results table has five columns: `id`, `sale_date_month`, `sale_date_year`, `sale_date_day`, and `sale_date`. The data is as follows:

<code>id</code>	<code>sale_date_month</code>	<code>sale_date_year</code>	<code>sale_date_day</code>	<code>sale_date</code>
1	5097	9	2016	1
2	5098	9	2016	2
3	5099	9	2016	3
4	5100	9	2016	4
5	5101	9	2016	5
6	5102	9	2016	6
7	5103	9	2016	7
8	5104	9	2016	8
9	5105	9	2016	9
10	5106	9	2016	10
11	5107	9	2016	11
12	5108	9	2016	12
13	5109	9	2016	13
14	5110	9	2016	14

The status bar at the bottom indicates "Query executed successfully".

Fig 176: Sale date

→ Tax\_class\_present

The screenshot shows the Microsoft SQL Server Management Studio interface. The Object Explorer on the left shows the database structure, including tables like 'tax\_class\_present'. The central pane displays the results of a query:

```

SELECT TOP (1000) [id]
      ,[tax_class_present_name]
  FROM [NYCOPROPERTIES].[dbo].[tax_class_present]
  
```

The results grid shows 11 rows of data:

	name
1	103
2	104
3	105
4	106
5	107
6	108
7	109
8	110
9	111
10	112
11	113
12	114
13	115
14	116
15	117
16	118
17	119
18	120
19	121
20	122
21	123
22	124

Properties pane on the right shows the connection status and details.

Fig 177: Tax class present

→ Year build

The screenshot shows the Microsoft SQL Server Management Studio interface. The Object Explorer on the left shows the database structure, including tables like 'year\_build'. The central pane displays the results of a query:

```

SELECT TOP (1000) [id]
      ,[name]
  FROM [NYCOPROPERTIES].[dbo].[year_build]
  
```

The results grid shows 27 rows of data:

	name
1	2001
2	2002
3	2003
4	2004
5	2005
6	2006
7	2007
8	2008
9	2009
10	2010
11	2011
12	2012
13	2013
14	2014
15	2015
16	2016
17	2017
18	2018
19	2019
20	2020
21	2021
22	2022

Properties pane on the right shows the connection status and details.

Fig 178: Year built

→ Zip\_code

The screenshot shows the Microsoft SQL Server Management Studio interface. The Object Explorer on the left lists the database structure, including tables like 'AssignmentID', 'BUSTCET', 'ClassDatabase', 'In-Class', 'NYCPROPERTYSALES', 'Tables', 'System Tables', 'FileTables', 'External Tables', 'Graph Tables', 'dbo.building\_class\_category', 'dbo.building\_class\_present', 'dbo.building\_class\_use', 'dbo.property', 'dbo.sale\_date', 'dbo.sale\_year', 'dbo.zip\_code', 'VIEWS', 'External Resources', 'Synonyms', 'Programmability', 'Service Broker', 'Storage', 'Security', 'Security', 'Server Objects', 'Replication', 'Policies', 'Always On High Availability', 'Management', and 'Integration Services Catalog'. The central pane displays a query result set titled 'Results' with 14 rows of data. The right pane shows connection details for 'DESKTOP-NC0B4AO\shubha'.

```

SELECT TOP (1000) /*[name]
FROM [NYCPROPERTYSALES].[dbo].[zip_code]

```

ID	NAME
1	2611 10001
2	2612 10002
3	2613 10003
4	2614 10004
5	2615 10005
6	2616 10006
7	2617 10007
8	2618 10008
9	2621 10010
10	2622 10011
11	2623 10011
12	2624 10012
13	2625 10013
14	2626 10014

Fig 179: Zip code

E. CSV File generated:

The screenshot shows a Microsoft Excel spreadsheet titled 'OutputCSV\_NYCPROPERTYSALES - Excel'. The table contains 29 rows of data, each representing a property record with columns for ID, NEIGHBORHOOD, BUILDING TAX CLASS, BLOCK, LOT, BUILDING ADDRESS, ZIP CODE, RESIDENTIAL COMMERCIAL TOTAL, URN, LAND, SQU, GROSS SQ FT, YEAR BUILT, SALE PRICE, and DATE. The data spans from row 1 to row 30.

ID	NEIGHBORHOOD	BUILDING TAX CLASS	BLOCK	LOT	BUILDING ADDRESS	ZIP CODE	RESIDENTIAL	COMMERCIAL	TOTAL	URN	LAND	SQU	GROSS SQ FT	YEAR BUILT	SALE PRICE	DATE
1	ALPHABET 07 RENTAL A	391	# CZ	334 AVENUE B,	5205	3	D	5	1033	6440	4473	1625000	2017-07-19 0:00			
2	ALPHABET 07 RENTAL	2	109	264 EAST 4TH STREET,	5205	38	I	11	6616	18890	4473			1016-12-14 0:00		
3	ALPHABET 07 RENTAL	3	106	38 C7 187 EAST 180 STREET,	5225	16	I	17	2213	7803	4473			2018-13-09 0:00		
4	ALPHABET 07 RENTAL B	403	23 C6	334 EAST 7TH STREET,	5225	10	D	10	2272	6704	4488	3046377	2018-09-23 0:00			
5	ALPHABET 07 RENTAL 28	404	55 C2	303 EAST 10TH STREET,	5225	6	D	6	2369	4615	4473	8000000	2018-11-17 0:00			
6	ALPHABET 07 RENTAL 28	2	426	516 EAST 12TH STREET,	5225	20	D	20	2581	9730	4473			2017-07-20 0:00		
7	ALPHABET 07 RENTAL 28	406	32 C6	210 AVENUE B,	5225	8	D	8	1750	4214	4463	2102848	2018-09-29 0:00			
8	ALPHABET 07 RENTAL 28	2	407	34 C7	520 EAST 14TH STREET,	5225	44	I	48	5163	21007	4473			2017-07-20 0:00	
9	ALPHABET 08 RENTAL	2	379	34 D3 141 AVENUE D,	5225	15	D	15	1534	9388	4493			2017-09-20 0:00		
10	ALPHABET 08 RENTAL	2	383	153 D9 525 EAST 5TH STREET,	5225	24	D	24	4488	18523	4488	1625000	2019-11-07 0:00			
11	ALPHABET 08 RENTAL	2	384	153 D9 525 EAST 5TH STREET,	5225	30	A	34	6395	21328	4483			2017-07-20 0:00		
12	ALPHABET 08 RENTAL	2	384	153 D9 525 EAST 5TH STREET,	5225	30	A	34	6395	21328	4483			2017-07-20 0:00		
13	ALPHABET 08 RENTAL 29	406	31 D1	520-522 EAST 5TH STREET,	5225	10	D	10	2717	12150	4582	10280000	2018-10-17 0:00			
14	ALPHABET 09 COOPS	2	278	24 C6	244 EAST 7TH STREET,	5225	34	D	24	4123	16778	4501	11000000	2017-06-21 0:00		
15	ALPHABET 09 RENTAL 28	301	29 53	624 EAST 10TH STREET,	5225	3	I	4	1520	3366	4463	3330000	2018-11-15 0:00			
16	ALPHABET 14 RENTAL 28	303	4 54	169 AVENUE B,	5225	4	I	5	2103	5468	4473	7215000	2017-01-30 0:00			
17	ALPHABET 14 RENTAL 28	304	5 55	183 AVENUE B,	5225	5	I	6	1279	2713	4463	4750000	2017-04-09 0:00			
18	ALPHABET 14 RENTAL 28	304	5 55	183 AVENUE B,	5225	5	I	6	1279	2713	4463			2017-04-03 0:00		
19	ALPHABET 22 STORE	4	380	34 K4 125 AVENUE C,	5225	0	I	1	1218	3588	4472	3700000	2018-09-28 0:00			
20	ALPHABET 37 RETAIL	4	388	13 MY 258 EAST 4TH STREET,	5225	0	I	1	52146	39246	4473			2017-01-13 0:00		
21	ALPHABET 37 RETAIL	4	388	15 MY 173 EAST 10TH STREET,	5225	0	I	1	1608	16632	4473			2017-01-13 0:00		
22	CHELSEA 01 ONE FA	3	766	13 49 254 WEST 16TH STREET	5227	3	D	1	1260	2716	4483			2017-01-00 0:00		
23	CHELSEA 01 ONE FA	3	766	13 49 255 WEST 16TH STREET	5227	1	D	1	1260	2716	4483			2018-10-20 0:00		
24	CHELSEA 01 ONE FA	3	771	52 49 224 WEST 22ND STREET	5227	1	D	1	1975	5855	4474	1625000	2018-12-30 0:00			
25	CHELSEA 01 ONE FA	3	772	29 46 235 WEST 22ND STREET	5227	1	D	1	1588	5653	4473	1625000	2017-01-30 0:00			
26	CHELSEA 01 ONE FA	3	772	29 46 235 WEST 22ND STREET	5227	1	D	1	1588	5653	4473	1625000	2017-01-30 0:00			
27	CHELSEA 01 ONE FA	3	772	28 46 233 WEST 22ND STREET	5227	1	D	1	1588	5653	4473	1625000	2017-01-30 0:00			
28	CHELSEA 02 TWO FA	3	743	63 03 328 WEST 20TH STREET,	5227	2	D	2	1520	3168	4483			2017-02-24 0:00		
29	CHELSEA 02 TWO FA	3	768	18 81 245 WEST 16TH STREET	5227	2	D	2	2000	2520	4474			2017-03-28 0:00		

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 prediction/ 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 Triflacta 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]))