Analysis of Yelp Dataset

Group Number: Final Project 1

Group members:

Akshaya Balan | 001091597 | Balan.a@northeastern.edu

Shriyah Maringanti | 001563531 | maringanti.s@northeastern.edu

Preet Mehta | 001520214 | Mehta.pre@northeastern.edu

Maulik Shah | 002133057 | shah.maul@northeastern.edu

All members have contributed equally

Objective:

Customers' buying patterns and decisions are heavily influenced by social networking sites. According to a 2017 Forbes survey, consumer social analysis is the second most valuable use case for big data analytics. Yelp is a social networking site that allows customers to provide reviews and rate businesses. According to Harvard Business School, a one-star rise in a restaurant's rating results in a 5 percent to 9 percent boost in sales earnings.

Our team is challenged to undertake research or analysis on this data and submit the findings. This project's purpose is to use data engineering and warehousing concepts to build data pipelines that receive data from a source, transform it, and store it in the best possible format for data visualization and to derive actionable and scalable insights from the data.

Questions we aim to answer:

1. Which restaurant categories have more positive number of reviews?
2. Top few restaurant categories based on Cuisines and Food Type?
3. Does median income in an area affect the number of restaurants around?
4. What are the top States which have the greatest number of restaurants?
5. When were the highest and lowest number of check-ins during the day and time in a week?
6. List the restaurants with the greatest number of users and highest number of ratings?

Data Source:

There are 3 datasets that we have used.
1) Yelp dataset: The Yelp dataset is a subset of Yelp's businesses, reviews, and user data that has been made publicly available for use for personal, educational, and academic purposes.
2) Median income dataset: This is derived from Census Bureau that collects median income on zip level
3) County dataset: The county is derived from zip code

The business entity in Yelp dataset will then be joined with income dataset and county dataset based on zip code.

Note: All schema mentioned below are schema according to data warehouse.

DATASET 1 - YELP DATASET
It consists of 4 tables: Business, Review, User and Checkin
Link of dataset: https://www.yelp.com/dataset

Business entity:

Contains different businesses and their attributes. The unique key is business id.
Source file type: Json
No of columns: 17 and Initial number of rows: 1,50,345

Schema (As per data warehouse):

Sn	Columns Name	Data Type	Description
1	business_id	string	unique string business id
2	name	string	business's name
3	address	string	full address of the business
4	city	string	the city
5	state	string	state code
6	postal code	string	postal code
7	latitude	float	latitude
8	longitude	float	longitude
9	stars	float	star rating, rounded to half-stars
10	review_count	integer	number of reviews
11	is_open	integer	0 or 1 for closed or open
12	attributes	string	Features of the business like BYOB, Parking etc

13	categories	string	Categories like fast food, cafe, restaurant etc
14	day_of_week	string	Day when restaurant is open
15	created_dt	timestamp	Creation date-time of the table in BigQuery
16	last_updated_dt	timestamp	Table last modified
17	last_updated_by	string	Contains values: “Initial Load” or “Stored Proc” depending on if it is initial load or updated by a stored procedure

Checkin Entity:

The CheckIn entity displays how many customers have currently checkedIn that they are present in a restaurant in Yelp. It captures the timestamp and the businessID which maps to the business entity mentioned above.
There are 8 columns and initially 121929 rows in the Checkin table.
Source file type: Json

Schema (As per data warehouse):

Sn	Columns Name	Data Type	Description
1	business_id	string	business id
2	date	timestamp	Date when a checkin has ocurred
3	month_name	string	Month of the check-in
4	Week_name	string	Week name of the check-in
5	Part_of_day	string	Afternoon/Evening/Night of the checkin
6	created_dt	timestamp	Creation date-time of the table in BigQuery
7	last_updated_dt	timestamp	Table last modified
8	last_updated_by	string	Contains values: “Initial Load” or “Stored Proc” depending on if it is initial load or updated by a stored procedure

Review Entity:

Each review in this dataset has a unique review_id along with the user_id of the user who has given a review, a business_id of the restaurant, the date and time, context, and the ratings/reactions it has received by other users who have seen the review.
There are 12 columns and initially 6990279 rows in the Review table.
Downloaded file type: Json

Schema (As per data warehouse):

Sn	Columns Name	Data Type	Description
1	review_id	string	22 character unique review id
2	user_id	string	22 character user id, maps to the user in user.json
3	business_id	string	22 character business id, maps to business in business.json
4	stars	float	star rating given by user
5	useful	integer	number of useful votes received
6	funny	integer	number of funny votes received
7	cool	integer	number of cool votes received
9	review_date	datetime	Date when review was made
10	created_dt	timestamp	Creation date-time of the table in BigQuery
11	last_updated_dt	timestamp	Table last modified
12	last_updated_by	String	Contains values: “Initial Load” or “Stored Proc” depending on if it is initial load or updated by a stored procedure

User Entity:

There are 23 columns and initially 1987897 rows in the User table. The User entity schema gives us information about the account holder information. Their unique user_id, name, number of reviews given and since when are they active on the site. Along with the basic details, even the reactions/ replies received to the user’s reviews are also collected in this dataset.
Downloaded file type: json

Schema (As per data warehouse):

Sn	Columns Name	Data Type	Description
1	user_id	string	unique user id
2	name	string	user's first name
3	review_count	integer	number of reviews
4	yelping_since	timestamp	when the user joined Yelp
5	useful	integer	number of useful votes sent by the user
6	funny	integer	number of funny votes sent by the user
7	cool	integer	number of cool votes sent by the user
8	fans	integer	number of fans the user has
9	average_stars	float	average rating of all reviews
10	compliment_hot	integer	number of hot compliments received by the user
11	compliment_more	integer	number of more compliments received by the user
12	compliment_profile	integer	number of profile compliments received by the user
13	compliment_cute	integer	number of cute compliments received by the user
14	compliment_list	integer	number of list compliments received by the user
15	compliment_note	integer	number of note compliments received by the user
16	compliment_plain	integer	number of plain compliments received by the user
17	compliment_cool	integer	number of cool compliments received by the user
18	compliment_funny	integer	number of funny compliments received by the user
19	compliment_writer	integer	number of writer compliments received by the user
20	compliment_photos	integer	number of photo compliments received by the user
21	created_dt	timestamp	Creation date-time of the table in BigQuery
22	last_updated_dt	timestamp	Table last modified
23	last_updated_by	String	Contains values: “Initial Load” or “Stored Proc” depending on if it is initial load or updated by a stored procedure

DATASET 2

The second dataset is from the American Community Survey (5 year estimate). We have taken the subject table S1903 which shows the median income. (https://data.census.gov/cedsci/table?q=median%20income&tid=ACSST5Y2020.S1903)

Steps to extract the dataset and apply basic transformations:

a) Extract dataset as csv from available API using the link - https://api.census.gov/data/2020/acs/acs5/subject?get=NAME,group(S1903)&for=zip%20code%20tabulation%20area:*
b) Match the variables using the file in https://www.census.gov/data/developers/data-sets/acs-5year.html
c) Take only the median income columns by deleting rest of the columns in excel
d) Remove characters like [ ] "

Final dataset details:
Initial No of rows: 33120
No of columns: 4
Downloaded file type: csv

Income entity

Schema (As per data warehouse):

Sn	Columns Name	Data Type	Description
1	name	string	The ZCTA code of an area
2	geo_id	string	geographical identifier by census bureau
3	Median_income_old	integer	Stores previous median income in a zip code when a change occurs
4	median_income	float	Current Median Income
5	zip_code_tabulation_area	string	5 digit zip code
6	created_dt	timestamp	Creation date-time of the table in BigQuery
7	last_updated_dt	timestamp	Table last modified
8	last_updated_by	string	Contains values: "Initial Load" or "Stored Proc" depending on if it is initial load or updated by a stored procedure

DATASET 3

The third dataset is a static file, which has County details for each zip codes across USA. This data was not available directly, but we extracted the data from a Tableau file which had all the geographical data of USA.
We have taken this file to have a county level analysis while creating visualizations in Tableau. We will join this table with the Business Table based on Zip code.
Downloaded file type: csv

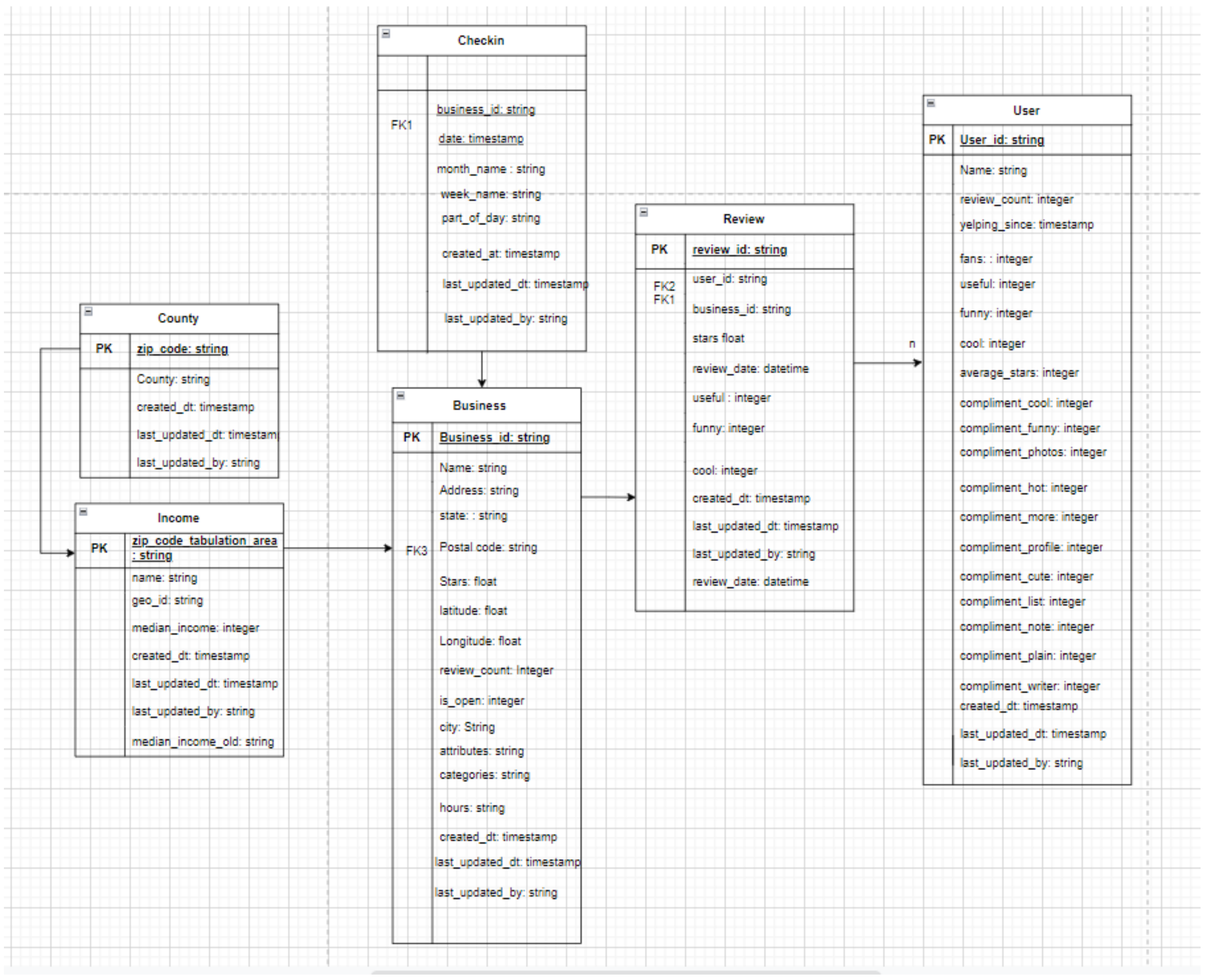
County Entity

Schema (As per data warehouse):

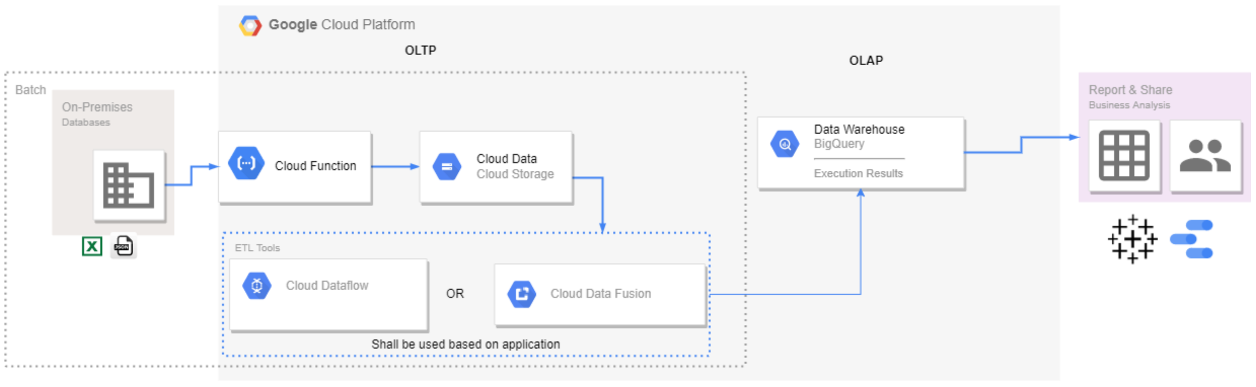
Sn	Columns Name	Data Type	Description
1	Zip	string	5 digit Zip Code
2	County	string	Couny Name for it's equivalent Zip
3	created_dt	timestamp	Creation date-time of the table in BigQuery
4	last_updated_dt	timestamp	Table last modified
5	last_updated_by	String	Contains values: “Initial Load” or “Stored Proc” depending on if it is initial load or updated by a stored procedure

Data Model:

The data is stored in key-value pair i.e in JSON format and CSV on premises. The below UML diagram defines the two data sources: Yelp Dataset and Census Data which we will be joining based on the zip_code attribute. The primary and foreign keys are defined and cardinality has been mentioned. Review acts like a factless fact table and other tables are dimension tables. The table names have intentionally not been prefixed/suffixed with fact/dimension.



Data Flow Design:



Overview of Data pipeline:

1. The raw data (OLTP) is stored in #6 tables, #4 are in Json format and #2 in csv format
2. The end user's details are captured in the user's table and review table while the business related are updated in business table and check-in table
3. A cloud function has been designed in Jupyter notebook which fetches the data from all the 5 tables and LOADED it into a single Google Cloud Storage bucket. This will be the Staging area, where we put all the data from different sources into a single location.
4. Using GCP Dataflow/Data fusion (ETL tool of GCP), we will extract data from all these stagging tables, provide necessary TRANSFORMATIONS to the data into Google BigQuery Data Warehouse (OLAP).
5. Once the data gets transformed into Data Warehouse, we shall use it to get further insights using BI Tools (Tableau and Data studio)

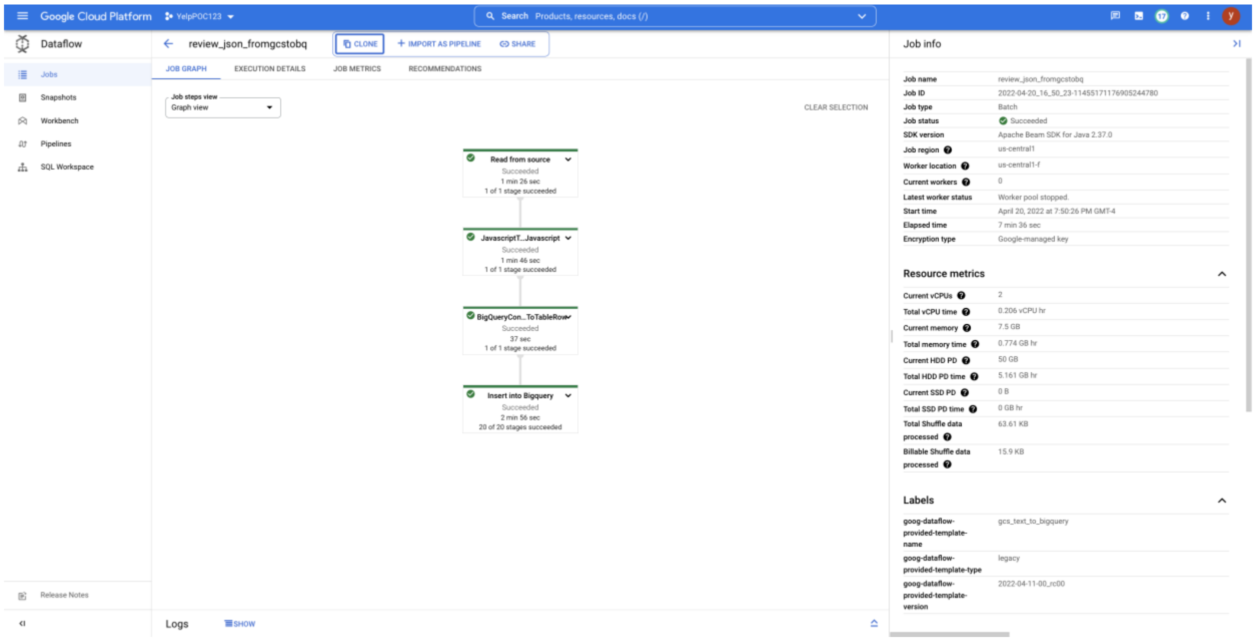
Migration of data from On-premise to Big Query:
Our aim to perform data visualization on a BI tool requires connection with the datasets which should be available on a data warehouse. We can also upload the main datasets directly over Big Query but that can not be considered as a data pipeline which gets updates on its own. So we used Python to open the Json file, and Google cloud platform (GCP) APIs to transfer the data from On-premise to Big Query. There are multiple services of GCP which can be used to migrate the data. We came up with 3 best approaches that we decided to use in order to get our datasets on Big Query.

Approach # 1 | Data flow (Templates)

This approach requires directly using the inbuilt teamplates available in the data flow. It requires following deliverables:

- Table Schema
- JSON UDF file
- Table in CSV format
- Directory to store Temp files

We created separate buckets for each of the #5 tables and performed the batch type Data flow job using templates and were successful in transferring all tables from Google cloud storage to Big Query



Read from source
Succeeded
1 min 26 sec
1 of 1 stage succeeded

JavascriptT...Javascript
Succeeded
1 min 46 sec
1 of 1 stage succeeded

BigQueryCon...ToTableRow
Succeeded
37 sec
1 of 1 stage succeeded

Insert into Bigquery
Succeeded
2 min 56 sec
20 of 20 stages succeeded

Job info

Job name	review_json_fromgcstobq
Job ID	2022-04-20_16_50_23-11455171176905244780
Job type	Batch
Job status	Succeeded
SDK version	Apache Beam SDK for Java 2.37.0
Job region	us-central1
Worker location	us-central1-f
Current workers	0
Latest worker status	Worker pool stopped.
Start time	April 20, 2022 at 7:50:26 PM GMT-4
Elapsed time	7 min 36 sec
Encryption type	Google-managed key

Resource metrics

Current vCPUs	2
Total vCPU time	0.206 vCPU hr
Current memory	7.5 GB
Total memory time	0.774 GB hr
Current HDD PD	50 GB
Total HDD PD time	5.161 GB hr
Current SSD PD	0 B
Total SSD PD time	0 GB hr
Total Shuffle data processed	63.61 KB
Billable Shuffle data processed	15.9 KB

Labels

goog-dataflow-provided-template-name	gcs_text_to_bigquery
goog-dataflow-provided-template-type	legacy
goog-dataflow-provided-template-version	2022-04-11-00_rc00

A snap shot of Review table is shown table.

Drawbacks in template:

It doesn’t allow to perform any kind of transformations over the data and simply transfers as it is in the staging table to the data warehouse. That’s the reason we tried using the another approach explained below.

Approach # 2 | Data flow (Apache beam – Python Code)

About Data flow:

Dataflow is the movement of data through a system comprised of software, hardware or a combination of both. Dataflow is often defined using a model or diagram in which the entire process of data movement is mapped as it passes from one component to the next within a program or a system, taking into consideration how it changes form during the process.

Note: The process shown below has been followed for all the tables. Here, we shall show the steps for User table. As a *Proof of Concept*, we followed the steps just for 10 rows.

Step 1: Importing the JSON dataset and converting it to dataframe

```
user=pd.read_json('/content/drive/MyDrive/Dataset/yelp_academic_dataset_us
er.json',lines=True)
```

Step 2: Selecting only top 10 rows

```
user_temp=user_temp.iloc[: , 1:]
```

Step 3: Saving the dataframe in CSV format

```
user_temp.to_csv('/content/drive/MyDrive/10 rows
dataset/user_temp.csv',index=False)
```

Step 4: Creating a function for migrating on-premise data to in Google cloud storage bucket

```
import sys

# [START storage_upload_file]
from google.cloud import storage


def upload_blob(bucket_name, source_file_name, destination_blob_name):

    storage_client = storage.Client()
    bucket = storage_client.bucket(bucket_name)
    blob = bucket.blob(destination_blob_name)

    blob.upload_from_filename(source_file_name)

    print(
        "File {} uploaded to {}.".format(
            source_file_name, destination_blob_name
        )
    )
```

Step 5: Defining the destination and source

```
    """Uploads a file to the bucket."""
    # The ID of your GCS bucket
    bucket_name = "yelp-poc"
    # The path to your file to upload
    source_file_name = "/content/drive/MyDrive/10 rows
dataset/user_temp.csv"
    # The ID of your GCS object
    destination_blob_name = "User_table"
```

Step 6: Running the function to upload the file in the selected bucket

```
upload_blob(bucket_name, source_file_name, destination_blob_name)
```

Result:
```
File /content/drive/MyDrive/10 rows dataset/user_temp.csv uploaded to
User_table.
```

Step 7: Creating functions to perform transformations using Apache beam

Discard incomplete rows:
```
def discard_incomplete(user_temp):
    """Filters out records that don't have an information."""
    return len(user_temp['user_id']) > 0 and len(user_temp['name']) > 0
and len(user_temp['yelping_since']) > 0
```

Discarding unwanted columns:

```
def del_unwanted_cols(user_temp):
    """Deleting unwanted columns"""
    del user_temp['elite']
    del user_temp['friends']
    return user_temp
```

Ensuring all columns have correct datatype:

```
def convert_types(data):
  data['user_id'] =  str(data['user_id']) if 'user_id' in data else None;
  data['name'] = str(data['name']) if 'name' in data else None;
  data['review_count'] = int(data['review_count']) if 'review_count' in
data else None;
  data['yelping_since'] = pd.to_datetime(data['yelping_since'],
format='%Y-%m-%d %H:%M:%S') if 'yelping_since' in data else None;
  data['useful'] = int(data['useful']) if 'useful' in data else None;
  data['funny'] = int(data['funny']) if 'funny' in data else None;
  data['cool'] = int(data['cool']) if 'cool' in data else None;
  data['fans'] = int(float(data['fans'])) if 'fans' in data else None;
  data['fans'] = data['fans'].astype(int) if 'fans' in data else None;
  data['average_stars'] = float(data['average_stars']) if 'average_stars'
in data else None;
  data['compliment_hot'] = int(data['compliment_hot']) if 'compliment_hot'
in data else None;
```

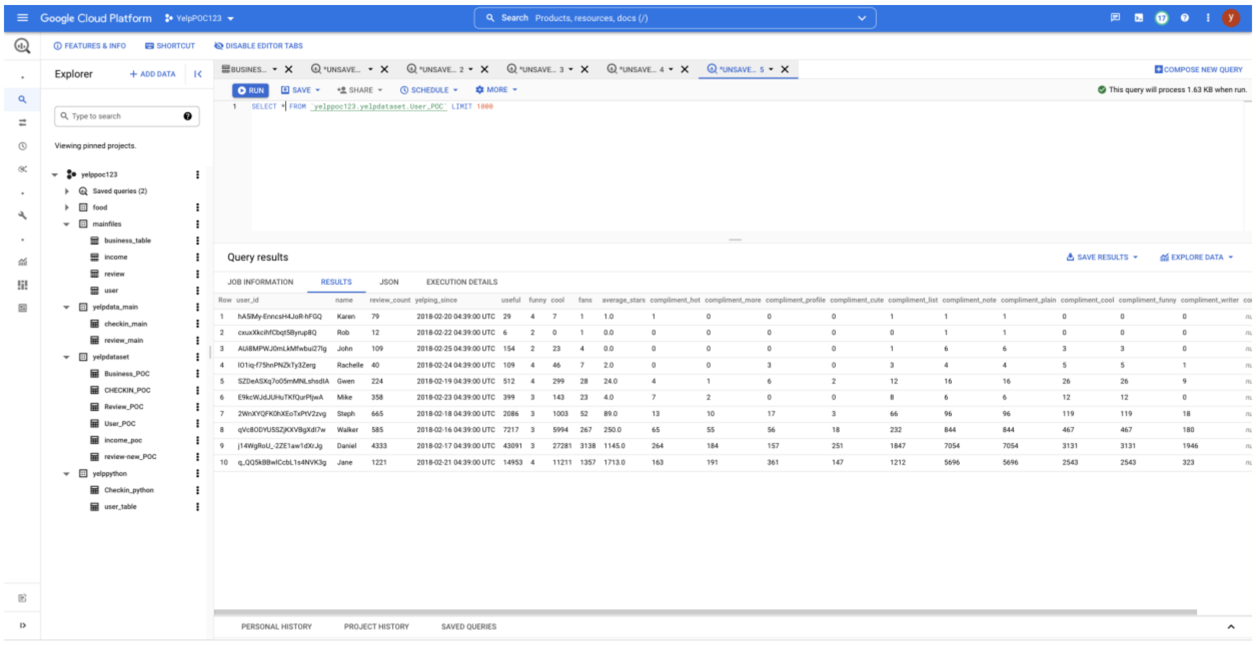
```
  data['compliment_more'] = int(data['compliment_more']) if
'compliment_more' in data else None;
  data['compliment_profile'] = int(data['compliment_profile']) if
'compliment_profile' in data else None;
  data['compliment_cute'] = int(data['compliment_cute']) if
'compliment_cute' in data else None;
  data['compliment_list'] = int(data['compliment_list']) if
'compliment_list' in data else None;
  data['compliment_note'] = int(data['compliment_note']) if
'compliment_note' in data else None;
  data['compliment_plain'] = int(data['compliment_plain']) if
'compliment_plain' in data else None;
  data['compliment_cool'] = int(data['compliment_cool']) if
'compliment_cool' in data else None;
  data['compliment_funny'] = int(data['compliment_funny']) if
'compliment_funny' in data else None;
  data['compliment_writer'] = int(data['compliment_writer']) if
'compliment_writer' in data else None;
  data['compliment_photos'] = int(data['compliment_photos']) if
'compliment_photos' in data else None;
  return data
```

```
SCHEMA='user_id:STRING,name:STRING,review_count:INTEGER,yelping_since:TIME
STAMP,useful:INTEGER,funny:INTEGER,cool:INTEGER,fans:INTEGER,average_stars
:FLOAT,compliment_hot:INTEGER,compliment_more:INTEGER,compliment_profile:I
NTEGER,compliment_cute:INTEGER,compliment_list:INTEGER,compliment_note:INT
EGER,compliment_plain:INTEGER,compliment_cool:INTEGER,compliment_funny:INT
EGER,compliment_writer:INTEGER,compliment_photos:INTEGER'


PROJECT_ID='yelppoc123'
```

Step 8: Performing Apache beam python code to perform transformations and upload table in Big Query:

```
p = beam.Pipeline(options=PipelineOptions())


if __name__ == '__main__':

    parser = argparse.ArgumentParser()
    known_args = parser.parse_known_args(argv)

    p = beam.Pipeline(options=PipelineOptions(
    temp_location="gs://user_python/temp/"))

    (p | 'ReadData' >> beam.io.ReadFromText('gs://user_python/User_table',
skip_header_lines =1)
       | 'Split' >> beam.Map(lambda x: x.split(','))
       | 'format to dict' >> beam.Map(lambda x: {"user_id" : x[0], "name"
: x[1],    "review_count" : x[2],       "yelping_since" : x[3], "useful" :
x[4], "funny" : x[5],    "cool" : x[6],       "fans" : x[7], "funny" :
x[8], "average_stars" : x[9],    "compliment_hot" : x[10],
"compliment_more" : x[11], "compliment_profile" : x[12], "compliment_cute"
: x[13],    "compliment_list" : x[14],       "compliment_note" : x[15],
"compliment_plain" : x[16], "compliment_cool" : x[17], "compliment_funny"
: x[18], "compliment_writer" : x[19], "compliment_photos" : x[20]})
       | 'DelIncompleteData' >> beam.Filter(discard_incomplete)
       | 'Convertypes' >> beam.Map(convert_types)
       | 'DelUnwantedData' >> beam.Map(del_unwanted_cols)
       | 'WriteToBigQuery' >>
beam.io.WriteToBigQuery('yelppoc123:yelppython.user_python'.format(PROJECT
_ID),
            schema=SCHEMA,
            write_disposition=beam.io.BigQueryDisposition.WRITE_APPEND))
    result = p.run()
    result.wait_until_finish()
```

Screen shot of User table uploaded in Big Query successfully



```
SELECT * FROM `yelppoc123.yelpdataset.User_POC` LIMIT 1000
```

Query results

Row	user_id	name	review_count	yelping_since	useful	funny	cool	fans	average_stars	compliment_hot	compliment_more	compliment_profile	compliment_cute	compliment_list	compliment_note	compliment_plain	compliment_cool	compliment_funny	compliment_writer
1	hA5lMy-EnncsH4JoR-hFGQ	Karen	79	2018-02-20 04:39:00 UTC	29	4	7	1	1.0	1	0	0	0	1	1	1	0	0	0
2	cxuxXkcihfCbqt5Byrup8Q	Rob	12	2018-02-22 04:39:00 UTC	6	2	0	1	0.0	0	0	0	0	0	1	1	0	0	0
3	AUi8MPWJ0mLkMfwbui27lg	John	109	2018-02-25 04:39:00 UTC	154	2	23	4	0.0	0	0	0	0	1	6	6	3	3	0
4	lO1iq-f75hnPNZkTy3Zerg	Rachelle	40	2018-02-24 04:39:00 UTC	109	4	46	7	2.0	0	0	3	0	3	4	4	5	5	1
5	SZDeASXq7o05mMNLshsdIA	Gwen	224	2018-02-19 04:39:00 UTC	512	4	299	28	24.0	4	1	6	2	12	16	16	26	26	9
6	E9kcWJdJUHuTKfQurPljwA	Mike	358	2018-02-23 04:39:00 UTC	399	3	143	23	4.0	7	2	0	0	8	6	6	12	12	0
7	2WnXYQFK0hXEoTxPtV2zvg	Steph	665	2018-02-18 04:39:00 UTC	2086	3	1003	52	89.0	13	10	17	3	66	96	96	119	119	18
8	qVc8ODYU5SZjKXVBgXdI7w	Walker	585	2018-02-16 04:39:00 UTC	7217	3	5994	267	250.0	65	55	56	18	232	844	844	467	467	180
9	j14WgRoU_-2ZE1aw1dXrJg	Daniel	4333	2018-02-17 04:39:00 UTC	43091	3	27281	3138	1145.0	264	184	157	251	1847	7054	7054	3131	3131	1946
10	q_QQ5kBBwlCcbL1s4NVK3g	Jane	1221	2018-02-21 04:39:00 UTC	14953	4	11211	1357	1713.0	163	191	361	147	1212	5696	5696	2543	2543	323

The above steps were done for each of the tables and we were successful in uploading all tables in Big Query.

Drawback in the Apache beam Python approach:

It requires a lot of coding and manual intervention. For example, in Business Table we had 150 columns. Writing schema for each of these columns manually in Python requires a lot of efforts in error handling. That’s the reason we tried the approach shown below.

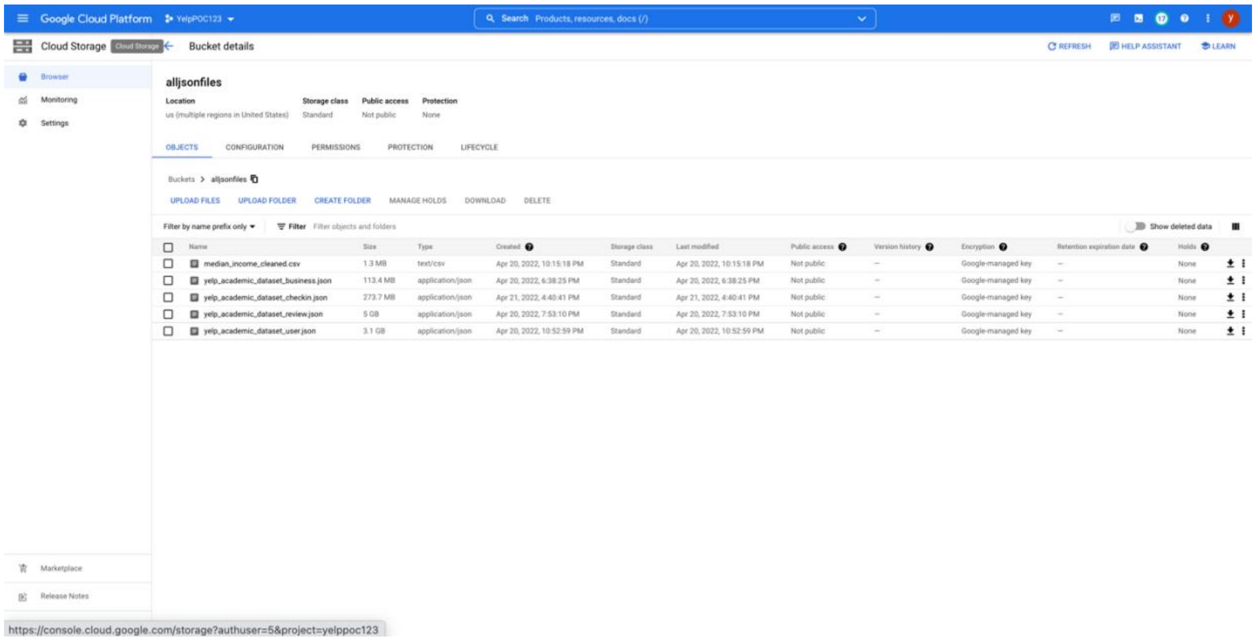
Approach # 3 | Data Fusion:

About Data Fusion:

Data Fusion is built using open source project CDAP, and this open core ensures data pipeline portability for users. CDAP's broad integration with on-premises and public cloud platforms gives Cloud Data Fusion users the ability to break down silos and deliver insights that were previously inaccessible.

In this approach, we used data fusion technology, which allows us to perform all transformations over the columns without writing any code.

Step 1: Uploading all the #5 tables in JSON format itself in the bucket



alljsonfiles

Location	Storage class	Public access	Protection
us (multiple regions in United States)	Standard	Not public	None

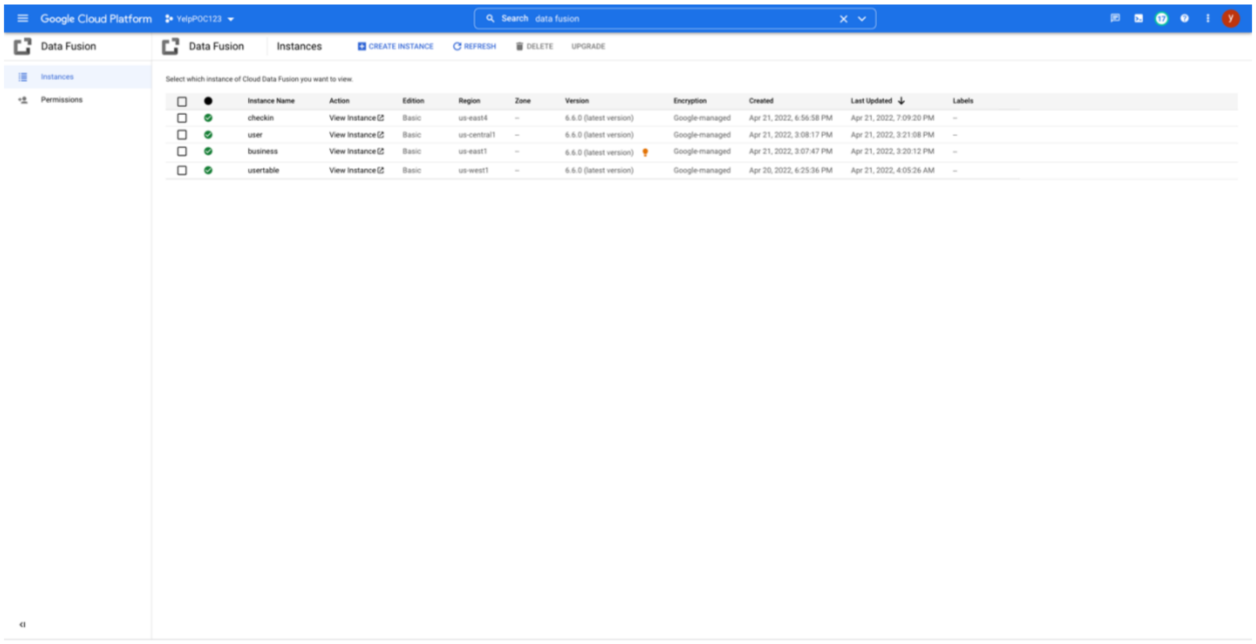
Buckets > alljsonfiles

Name	Size	Type	Created	Storage class	Last modified	Public access	Version history	Encryption	Retention expiration date	Holds
median_income_cleaned.csv	1.3 MB	text/csv	Apr 20, 2022, 10:15:18 PM	Standard	Apr 20, 2022, 10:15:18 PM	Not public	–	Google-managed key	–	None
yelp_academic_dataset_business.json	113.4 MB	application/json	Apr 20, 2022, 6:38:25 PM	Standard	Apr 20, 2022, 6:38:25 PM	Not public	–	Google-managed key	–	None
yelp_academic_dataset_checkin.json	273.7 MB	application/json	Apr 21, 2022, 4:40:41 PM	Standard	Apr 21, 2022, 4:40:41 PM	Not public	–	Google-managed key	–	None
yelp_academic_dataset_review.json	5 GB	application/json	Apr 20, 2022, 7:53:10 PM	Standard	Apr 20, 2022, 7:53:10 PM	Not public	–	Google-managed key	–	None
yelp_academic_dataset_user.json	3.1 GB	application/json	Apr 20, 2022, 10:52:59 PM	Standard	Apr 20, 2022, 10:52:59 PM	Not public	–	Google-managed key	–	None

https://console.cloud.google.com/storage?authuser=5&project=yelppoc123

As we can see in the above Snap Shot, we have uploaded all the tables as it is Google cloud storage in a single bucket in their original JSON format.

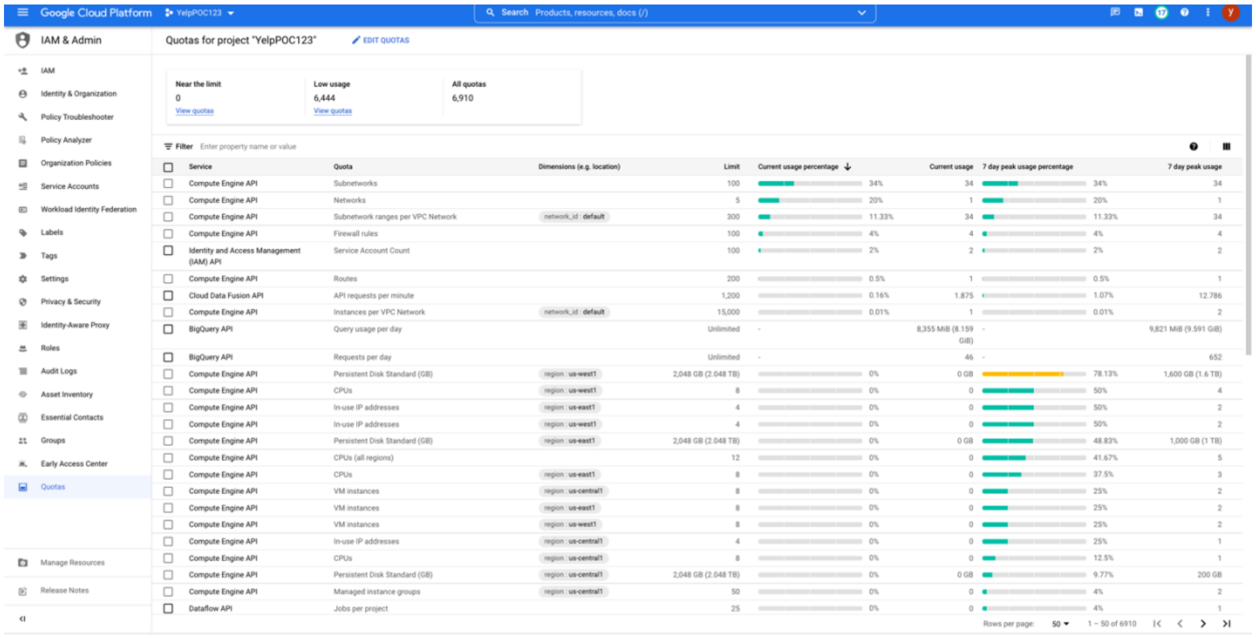
Step 2: Creating instances in Data Fusion



Select which instance of Cloud Data Fusion you want to view.

	●	Instance Name	Action	Edition	Region	Zone	Version	Encryption	Created	Last Updated	Labels
☐	✓	checkin	View Instance	Basic	us-east4	–	6.6.0 (latest version)	Google-managed	Apr 21, 2022, 6:56:58 PM	Apr 21, 2022, 7:09:20 PM	–
☐	✓	user	View Instance	Basic	us-central1	–	6.6.0 (latest version)	Google-managed	Apr 21, 2022, 3:08:17 PM	Apr 21, 2022, 3:21:08 PM	–
☐	✓	business	View Instance	Basic	us-east1	–	6.6.0 (latest version)	Google-managed	Apr 21, 2022, 3:07:47 PM	Apr 21, 2022, 3:20:12 PM	–
☐	✓	usertable	View Instance	Basic	us-west1	–	6.6.0 (latest version)	Google-managed	Apr 20, 2022, 6:25:36 PM	Apr 21, 2022, 4:05:26 AM	–

Each instance as a limit of 2.048 TB Persistent Disk Standard (GB) for each instance made in one region (as shown in below snapshot). However, while transformation we realized each data pipe line requires 3 TB of Persistent Disk Standard (GB). That’s the reason we could not make a single data pipeline in one instance (shown in the above snap shot).



Quotas for project "YelpPOC123"

Near the limit	Low usage	All quotas
0	6,444	6,910
View quotas	View quotas	

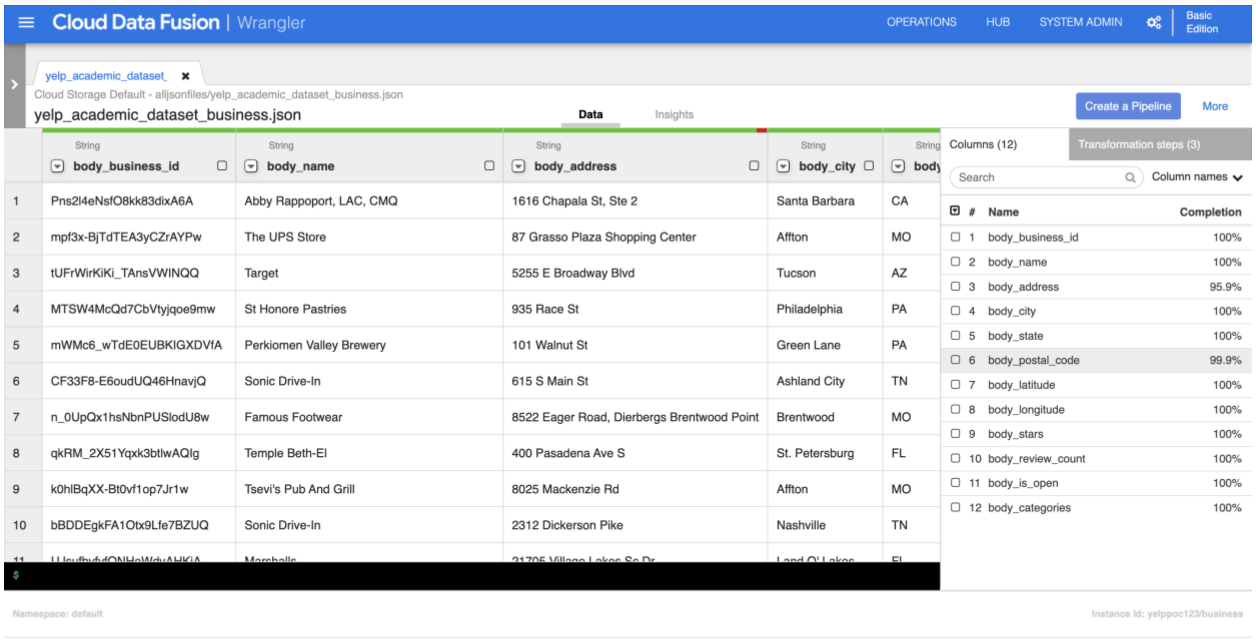
Service	Quota	Dimensions (e.g. location)	Limit	Current usage percentage	Current usage	7 day peak usage percentage	7 day peak usage
Compute Engine API	Subnetworks		100	34%	34	34%	34
Compute Engine API	Networks		5	20%	1	20%	1
Compute Engine API	Subnetwork ranges per VPC Network	network_id : default	300	11.33%	34	11.33%	34
Compute Engine API	Firewall rules		100	4%	4	4%	4
Identity and Access Management (IAM) API	Service Account Count		100	2%	2	2%	2
Compute Engine API	Routes		200	0.5%	1	0.5%	1
Cloud Data Fusion API	API requests per minute		1,200	0.16%	1.875	1.07%	12.786
Compute Engine API	Instances per VPC Network	network_id : default	15,000	0.01%	1	0.01%	2
BigQuery API	Query usage per day		Unlimited	-	8,355 MiB (8.159 GiB)	-	9,821 MiB (9.591 GiB)
BigQuery API	Requests per day		Unlimited	-	46	-	652
Compute Engine API	Persistent Disk Standard (GB)	region : us-west1	2,048 GB (2.048 TB)	0%	0 GB	78.13%	1,600 GB (1.6 TB)
Compute Engine API	CPUs	region : us-west1	8	0%	0	50%	4
Compute Engine API	In-use IP addresses	region : us-east1	4	0%	0	50%	2
Compute Engine API	In-use IP addresses	region : us-west1	4	0%	0	50%	2
Compute Engine API	Persistent Disk Standard (GB)	region : us-east1	2,048 GB (2.048 TB)	0%	0 GB	48.83%	1,000 GB (1 TB)
Compute Engine API	CPUs (all regions)		12	0%	0	41.67%	5
Compute Engine API	CPUs	region : us-east1	8	0%	0	37.5%	3
Compute Engine API	VM instances	region : us-central1	8	0%	0	25%	2
Compute Engine API	VM instances	region : us-east1	8	0%	0	25%	2
Compute Engine API	VM instances	region : us-west1	8	0%	0	25%	2
Compute Engine API	In-use IP addresses	region : us-central1	4	0%	0	25%	1
Compute Engine API	CPUs	region : us-central1	8	0%	0	12.5%	1
Compute Engine API	Persistent Disk Standard (GB)	region : us-central1	2,048 GB (2.048 TB)	0%	0 GB	9.77%	200 GB
Compute Engine API	Managed instance groups	region : us-central1	50	0%	0	4%	2
Dataflow API	Jobs per project		25	0%	0	4%	1

Rows per page: 50 1 – 50 of 6910

Step 3: Performing transformation in each the table using WRANGLER:

In Wrangler, we need to select the table from google cloud storage and using Parse option and mentioning the depth of JSON, we can unnest/normalize each of the columns to get the table in the final desired format/schema.

Note: Here we have shown the steps for Business table, and all the steps were processed for other #4 table as well.



Cloud Data Fusion | Wrangler

yelp_academic_dataset_

Cloud Storage Default - alljsonfiles/yelp_academic_dataset_business.json

yelp_academic_dataset_business.json

	String body_business_id	String body_name	String body_address	String body_city	String body
1	Pns2l4eNsfO8kk83dixA6A	Abby Rappoport, LAC, CMQ	1616 Chapala St, Ste 2	Santa Barbara	CA
2	mpf3x-BjTdTEA3yCZrAYPw	The UPS Store	87 Grasso Plaza Shopping Center	Affton	MO
3	tUFrWirKiKi_TAnsVWINQQ	Target	5255 E Broadway Blvd	Tucson	AZ
4	MTSW4McQd7CbVtyjqoe9mw	St Honore Pastries	935 Race St	Philadelphia	PA
5	mWMc6_wTdE0EUBKIGXDVfA	Perkiomen Valley Brewery	101 Walnut St	Green Lane	PA
6	CF33F8-E6oudUQ46HnavjQ	Sonic Drive-In	615 S Main St	Ashland City	TN
7	n_0UpQx1hsNbnPUSlodU8w	Famous Footwear	8522 Eager Road, Dierbergs Brentwood Point	Brentwood	MO
8	qkRM_2X51Yqxk3btlwAQlg	Temple Beth-El	400 Pasadena Ave S	St. Petersburg	FL
9	k0hlBqXX-Bt0vf1op7Jr1w	Tsevi's Pub And Grill	8025 Mackenzie Rd	Affton	MO
10	bBDDEgkFA1Otx9Lfe7BZUQ	Sonic Drive-In	2312 Dickerson Pike	Nashville	TN
11	[illegible]	Marshalls	[illegible]	[illegible]	FL

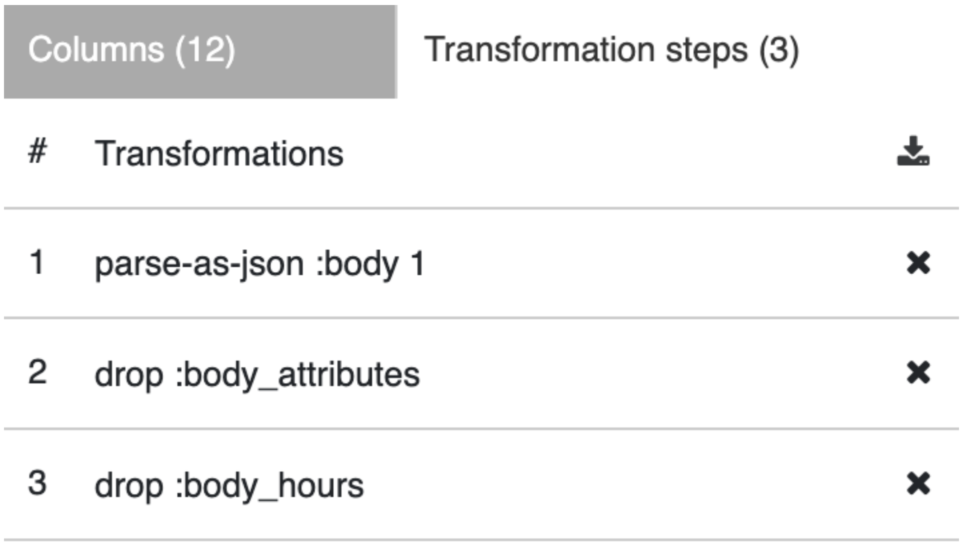
Columns (12) | Transformation steps (3)

#	Name	Completion
1	body_business_id	100%
2	body_name	100%
3	body_address	95.9%
4	body_city	100%
5	body_state	100%
6	body_postal_code	99.9%
7	body_latitude	100%
8	body_longitude	100%
9	body_stars	100%
10	body_review_count	100%
11	body_is_open	100%
12	body_categories	100%

Namespace: default

Instance Id: yelppoc123/business

As show in the above snap shot, we can see all the columns of the JSON file are normalized, renamed and changed the required data type for each columns.



Columns (12) | Transformation steps (3)

#	Transformations
1	parse-as-json :body 1
2	drop :body_attributes
3	drop :body_hours

As shown in the above snap shot, all the transformations gets captured and can easily be edited.

Error handling:

Missing Values:

The first step would be to check for missing/null values and certain important columns like name in user entity, median_income in income entity etc with missing values are removed.

Garbage Values:

Garbage values like -66666 were found in income table and they were removed.

Datatype conversion:

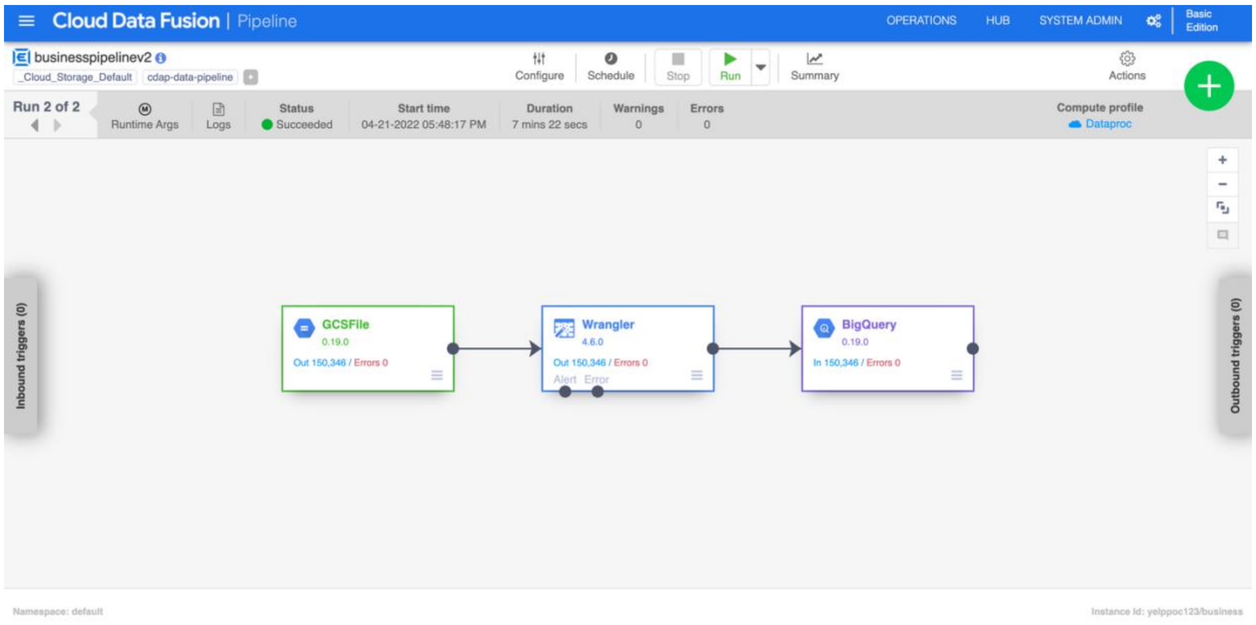
Data types for each attribute are converted to the one given in UML Eg: format the zip code in income dataset and yelp dataset similarly to join the two tables

Derived attribute:

Day_of_month, time and other date fields are derived from the “date” field of checkin entity which was of type timestamp. This enabled easier querying for analysis.
Once the transformations are done, next step is to set up the pipeline

Step 4: Creating the pipeline

Once the wrangling is done, we have plenty of other transformation options which data fusion provides. As we have done all the transformations in the wrangler itself, we can now connect it with Big Query.

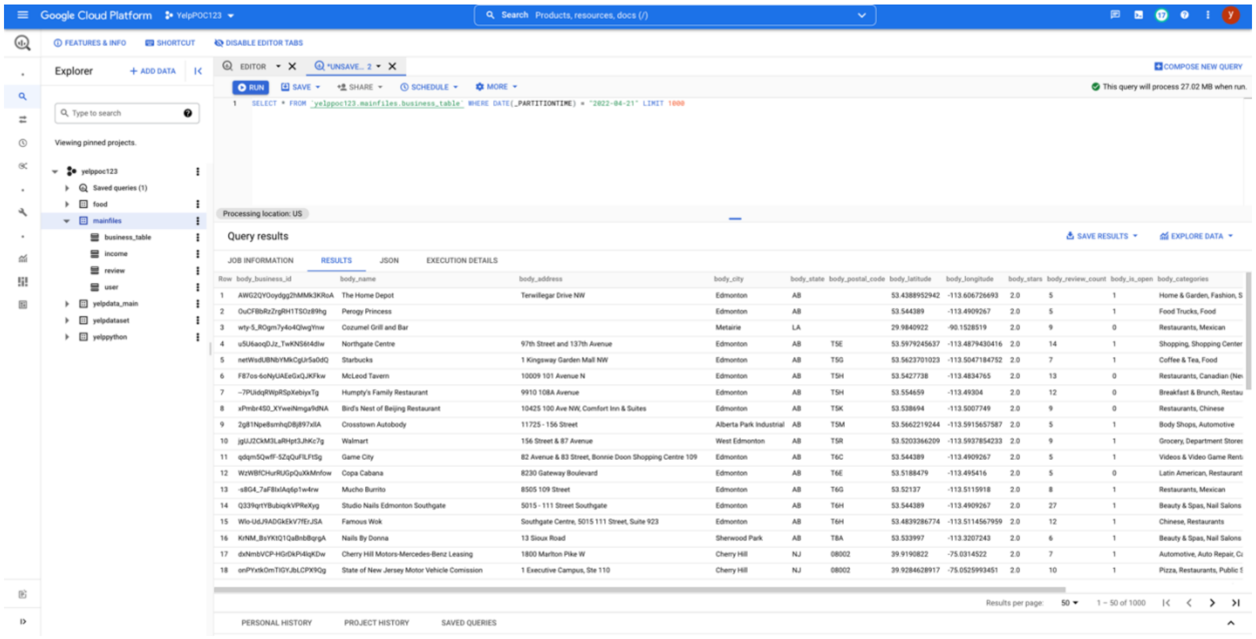


Step 5: Setting up the node size

As we are using free tier, we cannot change the quota size of disk storage. So for that reason we need to limit the working node disk size to 500 GB from 1000 GB.

Step 6: Running the pipeline

We can see in the above snap shot the pipe line was success and took 7 mins 22 seconds to execute



```
SELECT * FROM `yelppoc123.mainfiles.business_table` WHERE DATE(_PARTITIONTIME) = "2022-04-21" LIMIT 1000
```

This query will process 27.02 MB when run.

Processing location: US

Query results

Row	body_business_id	body_name	body_address	body_city	body_state	body_postal_code	body_latitude	body_longitude	body_stars	body_review_count	body_is_open	body_categories
1	AWG2QYOoydgg2hMMk3KRoA	The Home Depot	Terwillegar Drive NW	Edmonton	AB		53.4388952942	-113.606726693	2.0	5	1	Home & Garden, Fashion, S
2	OuCFBbRzZrgRH1TSOz89hg	Perogy Princess		Edmonton	AB		53.544389	-113.4909267	2.0	5	1	Food Trucks, Food
3	wty-5_ROgm7y4o4QlwgYnw	Cozumel Grill and Bar		Metairie	LA		29.9840922	-90.1528519	2.0	9	0	Restaurants, Mexican
4	u5U6aoqDJz_TwKNS6t4dlw	Northgate Centre	97th Street and 137th Avenue	Edmonton	AB	T5E	53.5979245637	-113.4879430416	2.0	14	1	Shopping, Shopping Center
5	netWsdUBNbYMkCgUr5a0dQ	Starbucks	1 Kingsway Garden Mall NW	Edmonton	AB	T5G	53.5623701023	-113.5047184752	2.0	7	1	Coffee & Tea, Food
6	F87os-6oNyUAEeGxQJKFkw	McLeod Tavern	10009 101 Avenue N	Edmonton	AB	T5H	53.5427738	-113.4834765	2.0	13	0	Restaurants, Canadian (Ne
7	--7PUidqRWpRSpXebiyxTg	Humpty's Family Restaurant	9910 108A Avenue	Edmonton	AB	T5H	53.554659	-113.49304	2.0	12	0	Breakfast & Brunch, Restau
8	xPmbr4S0_XYweiNmga9dNA	Bird's Nest of Beijing Restaurant	10425 100 Ave NW, Comfort Inn & Suites	Edmonton	AB	T5K	53.538694	-113.5007749	2.0	9	0	Restaurants, Chinese
9	2g81Npe8smhqDBj897xllA	Crosstown Autobody	11725 - 156 Street	Alberta Park Industrial	AB	T5M	53.5662219244	-113.5915657587	2.0	5	1	Body Shops, Automotive
10	jgUJ2CkM3LaRHpt3JhKc7g	Walmart	156 Street & 87 Avenue	West Edmonton	AB	T5R	53.5203366209	-113.5937854233	2.0	9	1	Grocery, Department Stores
11	qdqm5QwfF-5ZqQuFlLFtSg	Game City	82 Avenue & 83 Street, Bonnie Doon Shopping Centre 109	Edmonton	AB	T6C	53.544389	-113.4909267	2.0	5	1	Videos & Video Game Rent
12	WzWBfCHurRUGpQuXkMnfow	Copa Cabana	8230 Gateway Boulevard	Edmonton	AB	T6E	53.5188479	-113.495416	2.0	5	0	Latin American, Restaurant
13	-s8G4_7aF8IxlAq6p1w4rw	Mucho Burrito	8505 109 Street	Edmonton	AB	T6G	53.52137	-113.5115918	2.0	8	1	Restaurants, Mexican
14	Q339qrtYBubiqrkVPReXyg	Studio Nails Edmonton Southgate	5015 - 111 Street Southgate	Edmonton	AB	T6H	53.544389	-113.4909267	2.0	27	1	Beauty & Spas, Nail Salons
15	Wlo-UdJ9ADGkEkV7fErJSA	Famous Wok	Southgate Centre, 5015 111 Street, Suite 923	Edmonton	AB	T6H	53.4839286774	-113.5114567959	2.0	12	1	Chinese, Restaurants
16	KrNM_BsYKtQ1QaBnbBqrgA	Nails By Donna	13 Sioux Road	Sherwood Park	AB	T8A	53.533997	-113.3207243	2.0	6	1	Beauty & Spas, Nail Salons
17	dxNmbVCP-HGrDkPi4lqKDw	Cherry Hill Motors-Mercedes-Benz Leasing	1800 Marlton Pike W	Cherry Hill	NJ	08002	39.9190822	-75.0314522	2.0	7	1	Automotive, Auto Repair, C
18	onPYxtkOmTIGYJbLCPX9Qg	State of New Jersey Motor Vehicle Comission	1 Executive Campus, Ste 110	Cherry Hill	NJ	08002	39.9284628917	-75.0525993451	2.0	10	1	Pizza, Restaurants, Public S

Results per page: 50 1 – 50 of 1000

We can see in the above snap shot that business_table is created in the Big Query. The same process was followed for each of the tables.

Incremental loading:

Incremental loading was performed in Big Query using scheduled query and stored procedure.

A stored procedure was created to run incremental loading task to fetch only the newly added rows in Big Query. This query was scheduled to run daily by enabling **Big Query Data Transfer API.**

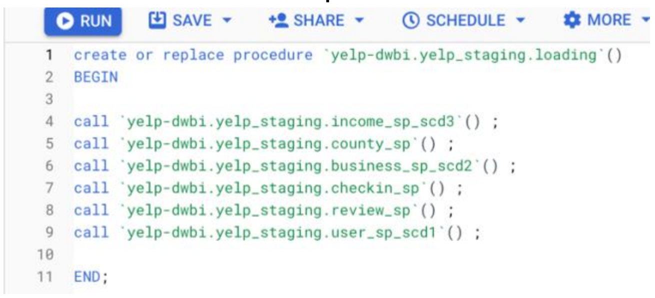
Logic: Any new row added to source table will be identified by primary key if it exists or check entire row using **“except distinct”**. Accordingly, a new record will be inserted in the table in datawarehouse.

To track changes in tables in datawarehouse, a **run_control_table** was created which contains the following columns to track changes in a table:

Sn	Columns Name	Data Type	Description
1	table_nm	string	Table name in data warehouse that was modified
2	src_table_nm	string	The source table – usually from OLTP
3	rec_inserted	integer	Number of records inserted
4	rec_deleted	integer	Number of records deleted
5	rec_updated	integer	Number of records updated
6	run_dt	timestamp	Run time of the update

Procedure:

a) Enable Big Query Data Transfer API
b) Create a stored procedure for each table that checks for new records in the source
c) Create another stored procedure to call all the above stored procedures



```
create or replace procedure `yelp-dwbi.yelp_staging.loading`()
BEGIN

call `yelp-dwbi.yelp_staging.income_sp_scd3`() ;
call `yelp-dwbi.yelp_staging.county_sp`() ;
call `yelp-dwbi.yelp_staging.business_sp_scd2`() ;
call `yelp-dwbi.yelp_staging.checkin_sp`() ;
call `yelp-dwbi.yelp_staging.review_sp`() ;
call `yelp-dwbi.yelp_staging.user_sp_scd1`() ;

END;
```

d) Create a scheduled query to call this stored block that runs everyday

Slowly changing dimension:

About Slowly changing dimension:

Slowly Changing Dimensions (SCD) - dimensions that change slowly over time, rather than changing on regular schedule, time-base. In Data Warehouse there is a need to track changes in dimension attributes in order to report historical data. In other words, implementing one of the SCD types should enable users assigning proper dimension's attribute value for given date. Example of such dimensions could be: customer, geography, employee.
There are many approaches how to deal with SCD. The most popular are:

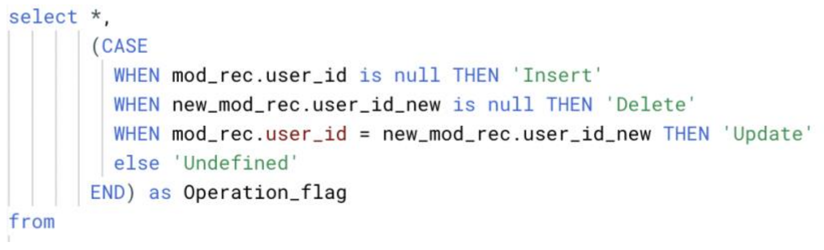
- Type 0 - The passive method
- Type 1 - Overwriting the old value
- Type 2 - Creating a new additional record
- Type 3 - Adding a new column
- Type 4 - Using historical table
- Type 6 - Combine approaches of types 1,2,3 (1+2+3=6)

We applied SCD type 1,2,3 in this project.

Implementation of Slowly Change Dimension in this project:

Type 1: name field in user_entity
Type 2: category field in business enitty
Type 3: median_income field in income entity
General logic:

a) The stored procedure in incremental loading explained in the incremental loading accommodates the SCD as well.
b) For a given primary key, if any of the other fields changes then the operation flag is update. *mod_rec = modified records in source; new_mod_rec = new/modified records in src*

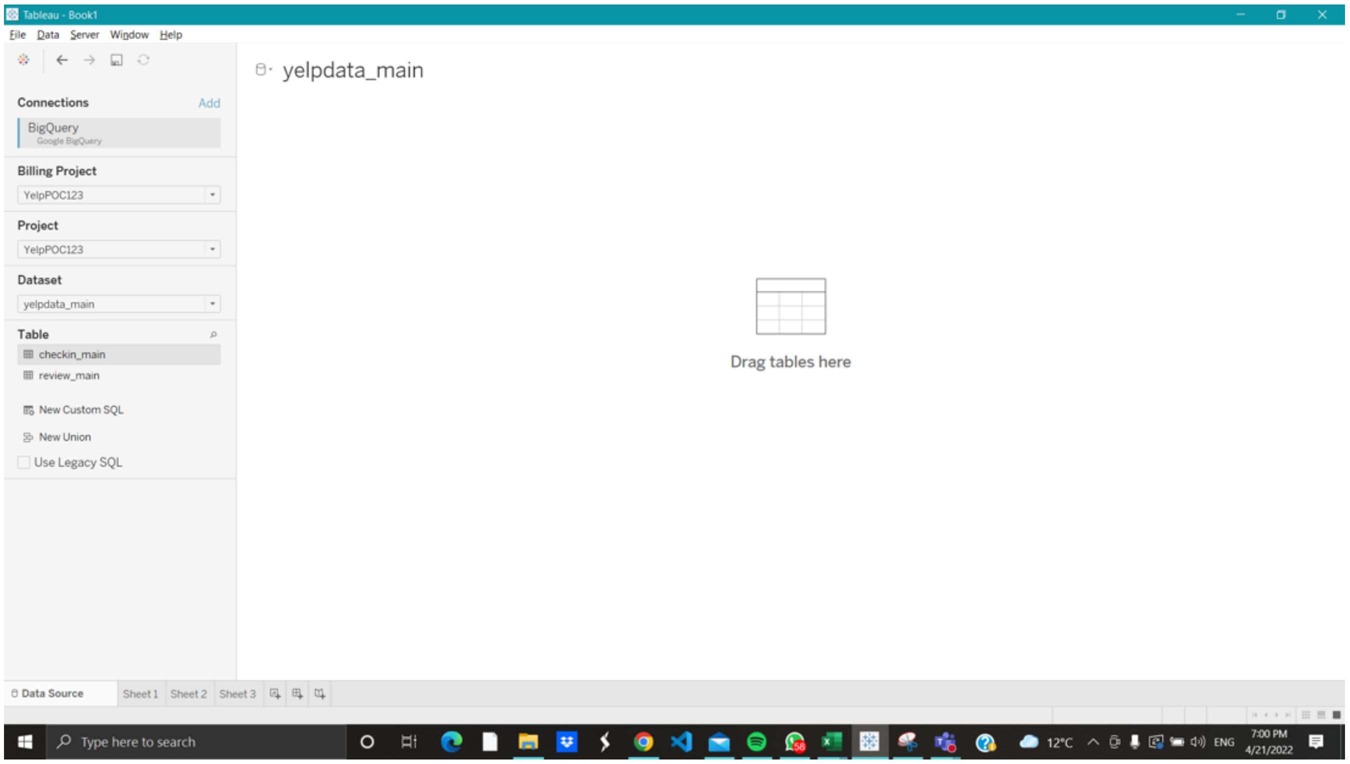


```
select *,
        (CASE
          WHEN mod_rec.user_id is null THEN 'Insert'
          WHEN new_mod_rec.user_id_new is null THEN 'Delete'
          WHEN mod_rec.user_id = new_mod_rec.user_id_new THEN 'Update'
          else 'Undefined'
        END) as Operation_flag
from
```

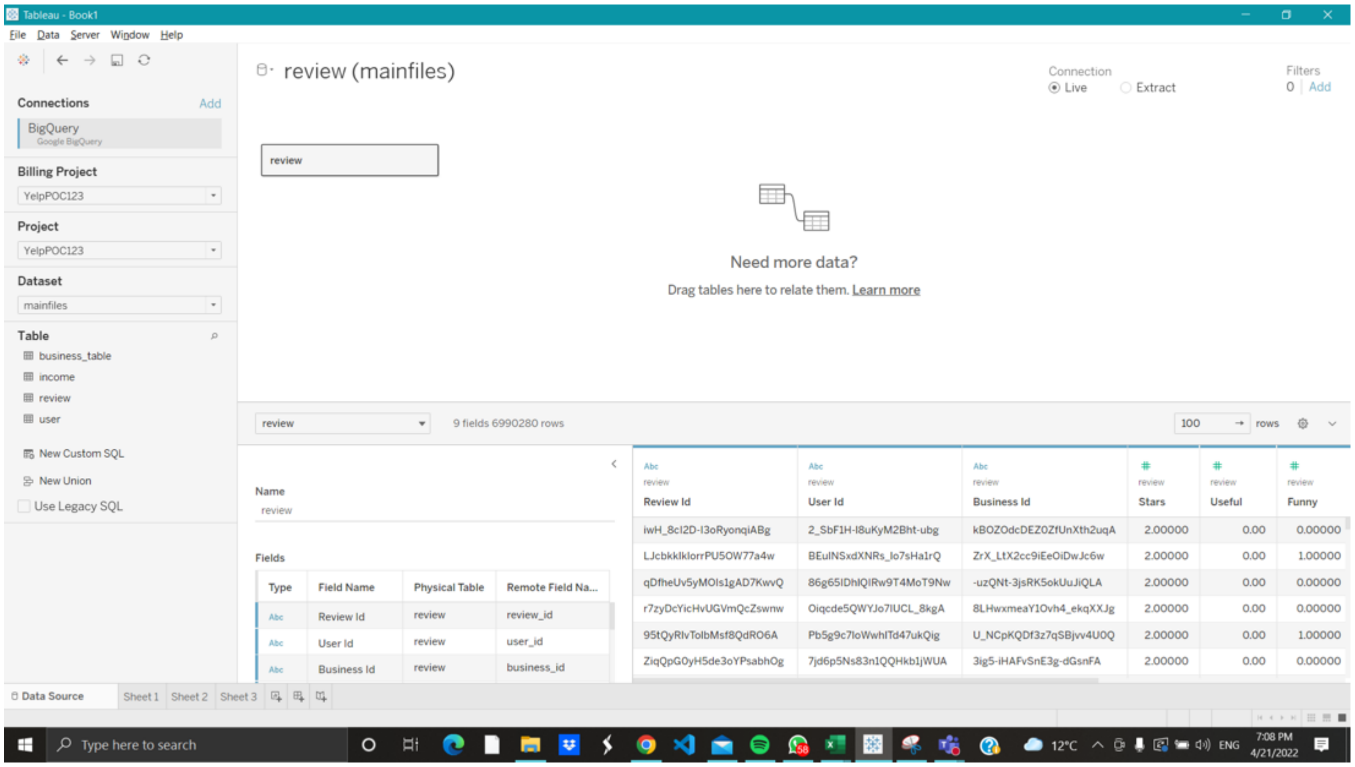
c) According to the scd type, the update occurs reflecting the timestamp in *last_updated* field in data warehouse table
d) The number of records updated/deleted will also be reflected in run_control_tbl
e) The scheduled query runs daily so data warehouse is updated daily

Data Visualization:

Connecting Tableau with Google Big Query fetch all #5 tables from Data warehouse:



As shown above, we have connected with Google big Query using OAUTH method.



review (mainfiles)

Billing Project: YelpPOC123

Project: YelpPOC123

Dataset: mainfiles

Table:
- business_table
- income
- review
- user

review — 9 fields 6990280 rows — 100 rows

Name: review

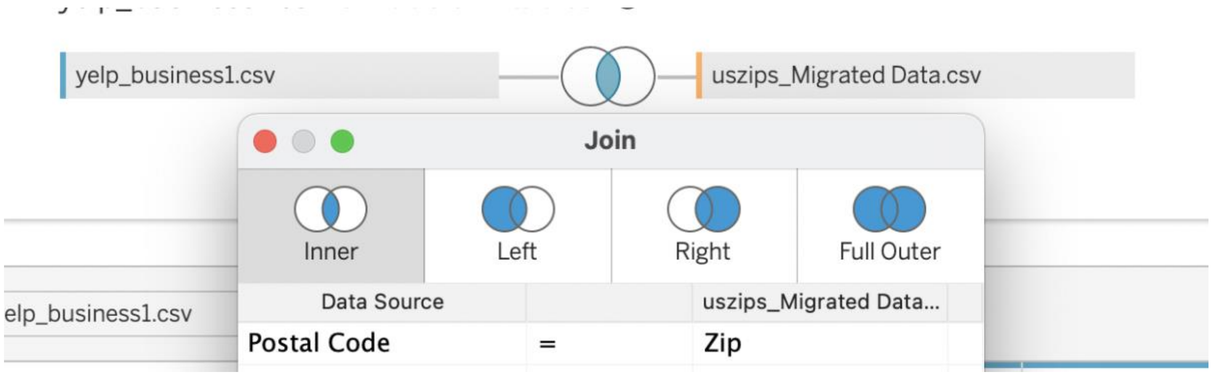
Fields

Type	Field Name	Physical Table	Remote Field Na...
Abc	Review Id	review	review_id
Abc	User Id	review	user_id
Abc	Business Id	review	business_id

Review Id	User Id	Business Id	Stars	Useful	Funny
iwH_8cI2D-I3oRyonqiABg	2_SbF1H-I8uKyM2Bht-ubg	kBOZOdcDEZ0ZfUnXth2uqA	2.00000	0.00	0.00000
LJcbkkIklorrPU5OW77a4w	BEuINSxdXNRs_Io7sHa1rQ	ZrX_LtX2cc9iEeOiDwJc6w	2.00000	0.00	1.00000
qDfheUv5yMOIs1gAD7KwvQ	86g65IDhIQIRw9T4MoT9Nw	-uzQNt-3jsRK5okUuJiQLA	2.00000	0.00	0.00000
r7zyDcYicHvUGVmQcZswnw	Oiqcde5QWYJo7IUCL_8kgA	8LHwxmeaY1Ovh4_ekqXXJg	2.00000	0.00	0.00000
95tQyRlvTolbMsf8QdRO6A	Pb5g9c7loWwhITd47ukQig	U_NCpKQDf3z7qSBjvv4U0Q	2.00000	0.00	1.00000
ZiqQpG0yH5de3oYPsabhOg	7jd6p5Ns83n1QQHkb1jWUA	3ig5-iHAFvSnE3g-dGsnFA	2.00000	0.00	0.00000

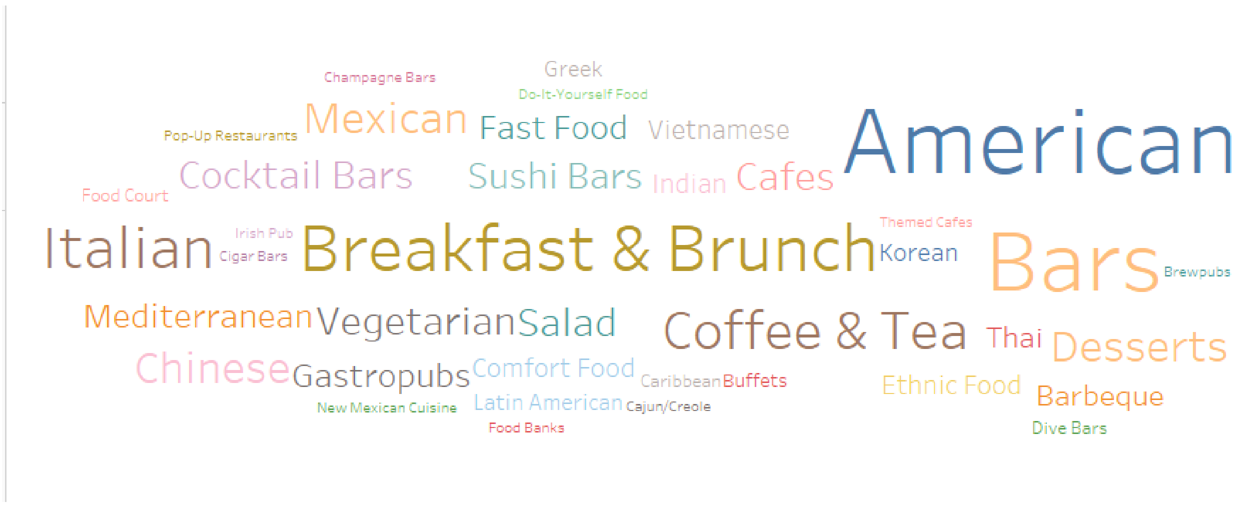
12°C ENG 7:08 PM 4/21/2022

We have taken data for 15 states of USA and created various visual graphs for various Food Businesses such as Restaurants, cafe, bars, coffeeshops, desserts, etc. Our Source data have Geographical fields such as State, City, Zip Code but County was MISSING, that's why we took another csv from the web which had all the mapping between zip codes and county across US. So, we joined both the data based on zip codes to get the name of the County and have a County-level analysis as well. Check below snapshot for the same.



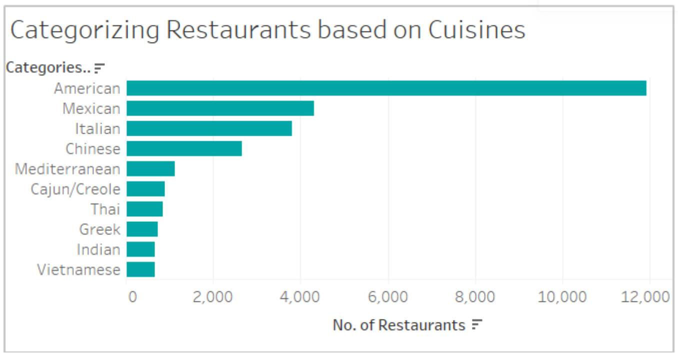
Data Source		uszips_Migrated Data...
Postal Code	=	Zip

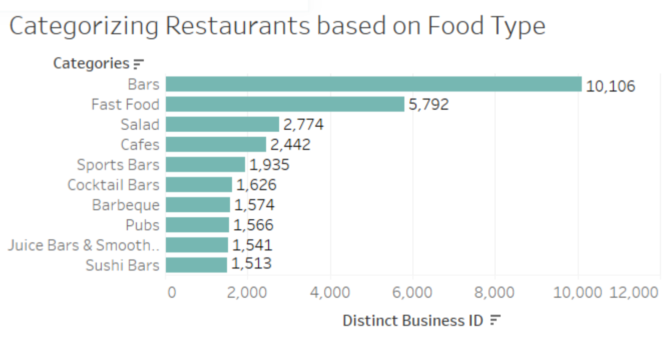
1. Which restaurant categories have more/less positive reviews?



Based on the number of positive reviews received on Yelp, American Food Category stands at the top with 3Million reviews followed by Bars at 2M and Breakfast and Brunch at 1M. American is the most preferred cuisine based on reviews.

2. How many restaurants are there in US based on Cuisines and Food Type grouped by category?



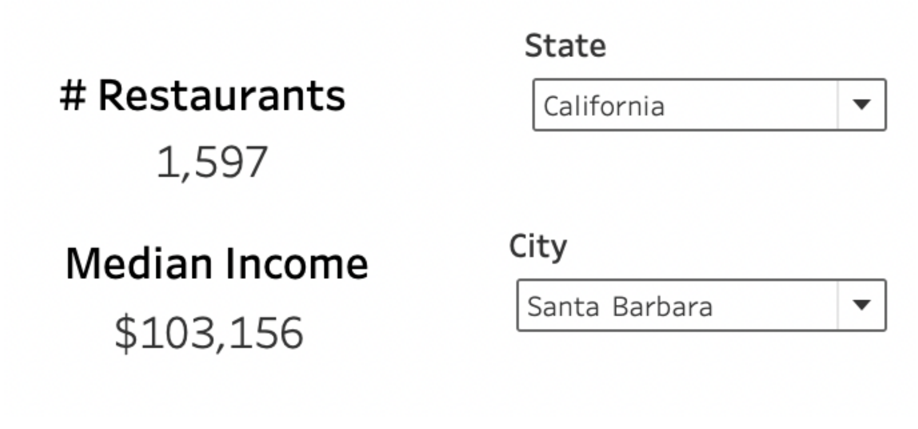


Most of the USA restaurants offers American Cuisine – 12K restaurants which justifies the number of positive reviews discussed earlier. Followed by Mexican Cuisine adopted by 4,5k restaurants and Italian at 3.9k.

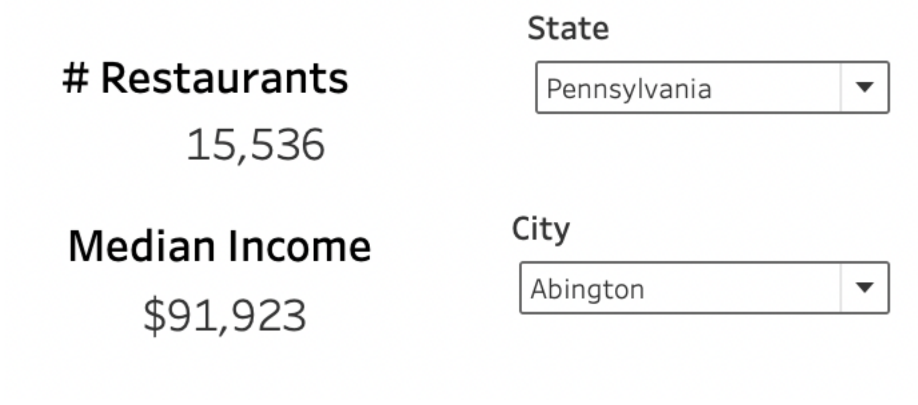
Bars with a count of 10,000 are the most offered food type by restaurants.

3. What is the Average Median Income and number of restaurants based on US states and cities?

State- California

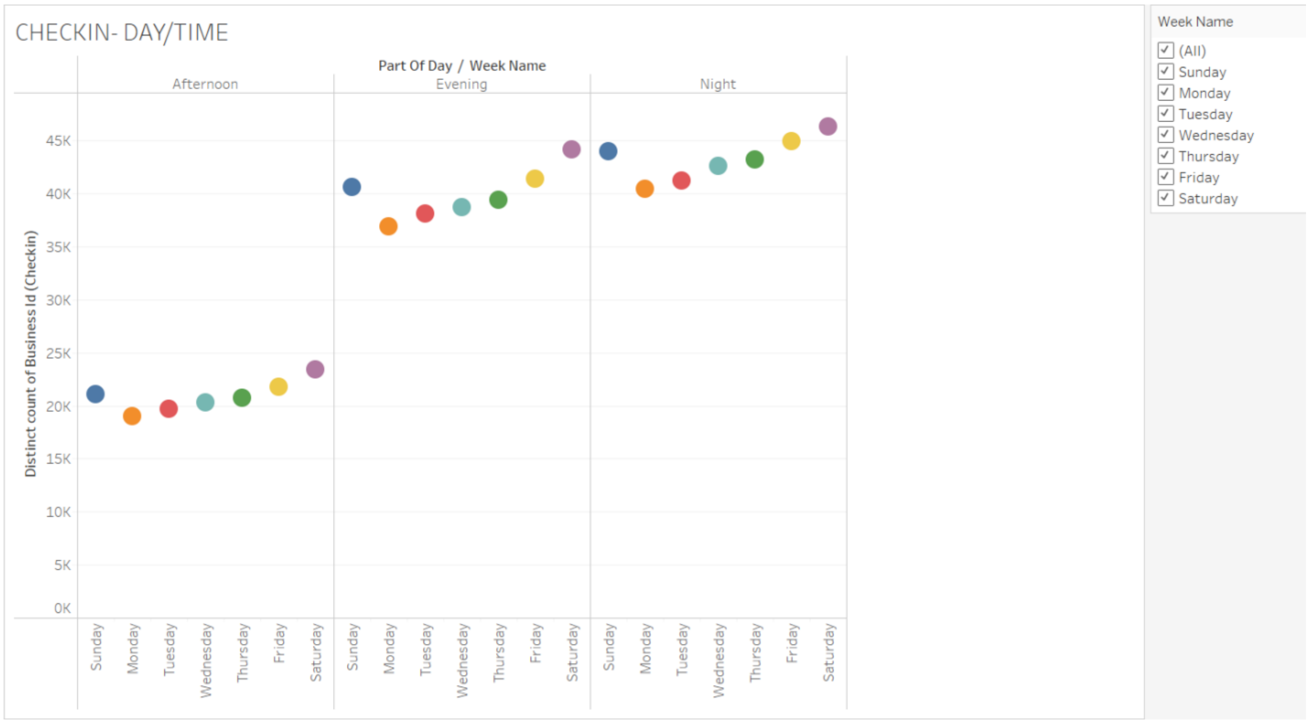

Restaurants
1,597
Median Income
$103,156
State
California
City
Santa Barbara

State- Pennsylvania


Restaurants
15,536
Median Income
$91,923
State
Pennsylvania
City
Abington

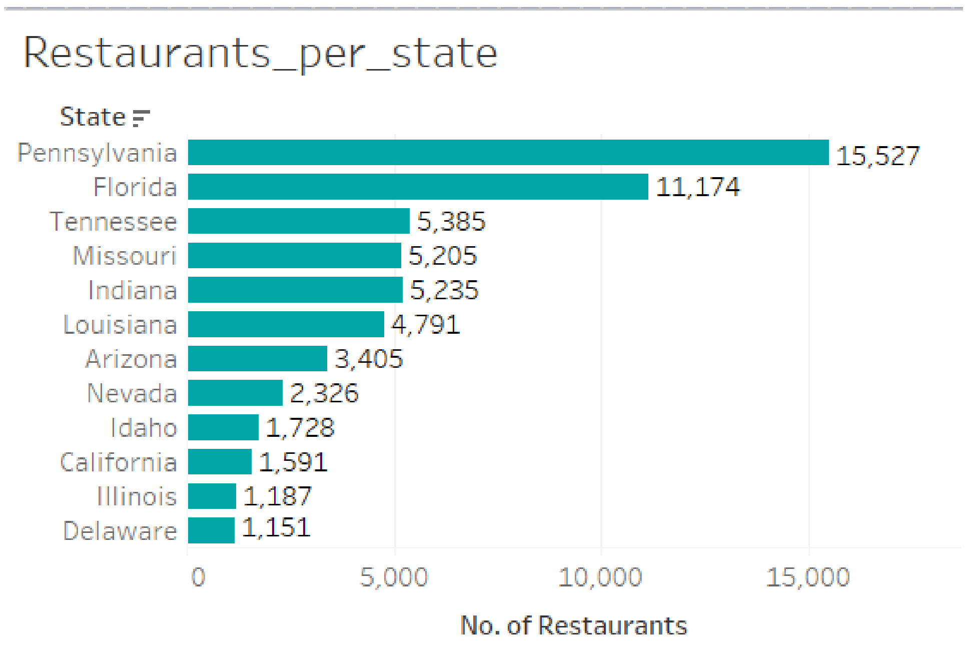
The dashboard contains filters where you can select the state and city and compare the number of restaurants and median income. Although median income for California is higher than Pennsylvania the number of restaurants are higher in Pennsylvania. However, 1597 restaurants in California doesn't seem true. It could be because fewer businesses are registered in Cali, or the data in the original dataset itself is not true. There are many hypothesis that we can come up with but these are left for future scope of this project .

4. When were the highest and lowest number of check-ins during the day and time in a week?



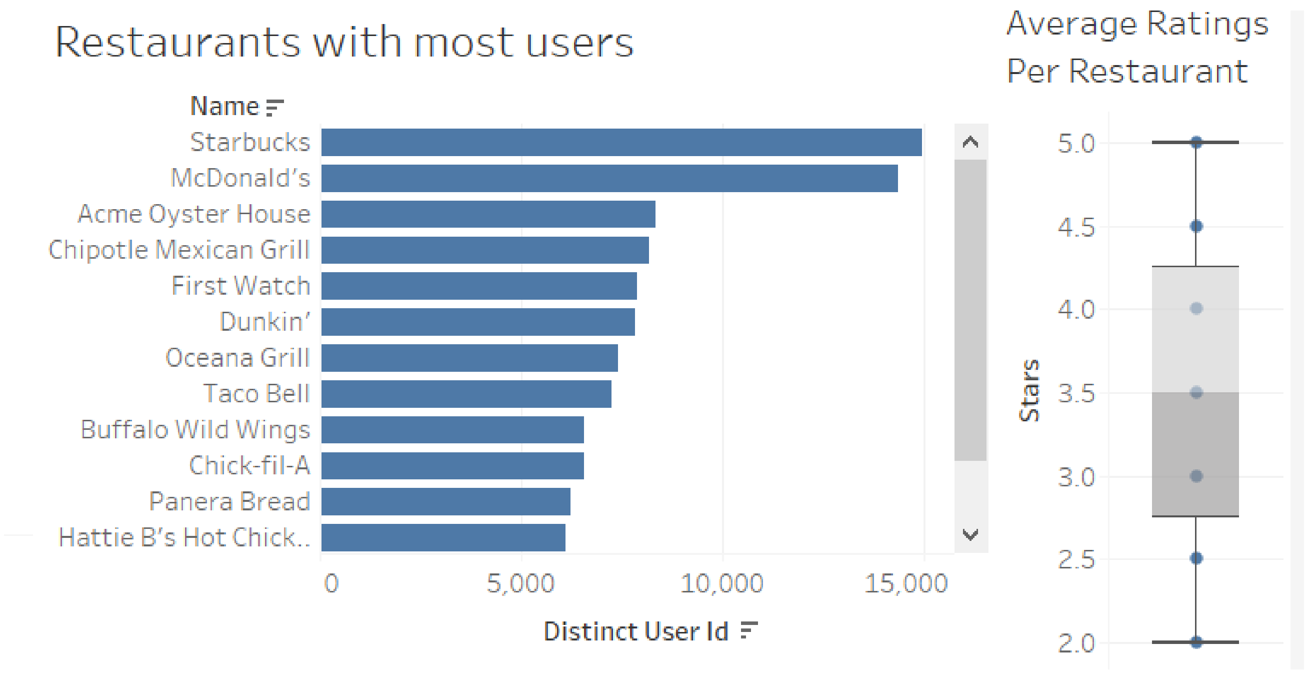
In the above visualization, we have taken “Time of the day” and “Day of the week” parameters into consideration. With the Week Day filter, we can see that on the weekends there are more number of checkins during the evening and night times and especially during Saturdays where the restaurant checkins are reaching almost 46000 on yelp.

5. **What are the top States which have most number of restaurants?**



Pennsylvania has around 15,000 restaurants alone whose median income is $88,000 followed by Florida having around 11,000 restaurants.

6. List the restaurants with most number of users and highest number of ratings?



We can clearly see that Starbucks is the most preferred restaurants by the users where it has to be the highest rating of 5 and average of 3.5 stars. McDonalds stands second in the number of restaurants with most users at 14,000, with a average rating of 2.3.
While Chipotle has lesser number significantly less number of users, it receives high rating with the high of 4.

TAKE AWAYS!

- Median income does not necessarily play a huge factor when restaurants open. We confirmed that places with a lesser median income had more restaurants than places with higher median income.
- The big chain of restaurants like Star Bucks, McD, Chipotle are very popular on Yelp
- People prefer visiting restaurants during evenings and nights especially on Saturday which has most number of checkins
- Pennsylvania has the most number of restaurants registered in Yelp which is funny because you usually tend to think California would have the highest. It could be because fewer users are on Yelp these days which is again a hypothesis that is left for future work.
- American cuisine is the most popular in the US which is obvious and highest number of restaurant category being Bars.

Appendix:

Modification History

Version	Date	Author	Changes
1	03/11/22	Team	Initial proposal
2	04/07/22	Team	1. Added another dataset 2. What questions we aim to answer overview added 3. Included additional dataset description 4. Added data model 5. Added data flow design 6. Added pencil sketch visualizations
3	04/21/22	Team	1. Migration of On-premise data to Big Query Approach #1 Data flow Approach #2 Data Fusion 2. Performed transformations using Wrangler in Data Fusion and built end to end pipeline 3. Performed SCD and Incremental Loading using stored procedures in big query 4. Added a static dataset which has County names for each zip code, which can be used for County level analytical purposes in Tableau. 5. Connected Datasets from Big Query to Tableau to create interactive visualizations based on earlier drafts and feedback • What are the restaurant categories based on number of reviews? • What are the restaurant categories based on Cuisines and Food Type? • What is the Average Median Income and number if restaurants based on US states and cities? • What are the top States which have most number of restaurants? • When were the highest and lowest number of check-ins during the day and time in a week? • List the restaurants with most number of users and highest number of ratings?
4	5/4/2022	Team	1. Explained normalization process in Data Fusion 2. Updated the schema of tables in data warehouse instead of data source 3. Data model made as per the RDBMS in big query 4. Added questions in Objective section 5. Mention metadata for all tables

Instructor Notes

Version	Comments
Initial Proposal	1. Add another data source 2. What questions do you aim to answer
First Draft	1. Why are we limiting analysis to MA? 2. What states are you including? 3. Try to add zip code to county table for analysis
Final Draft	1. What does final normalized data model look like? 2. Add the attribute descriptions 3. More comments in code 4. Little more visualization and watch labels in visualizations too