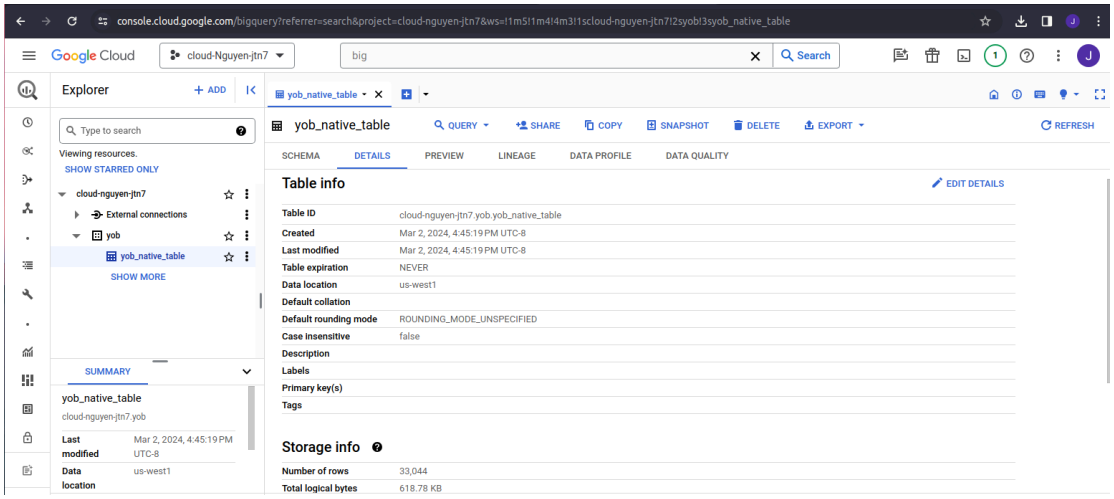
9.1g

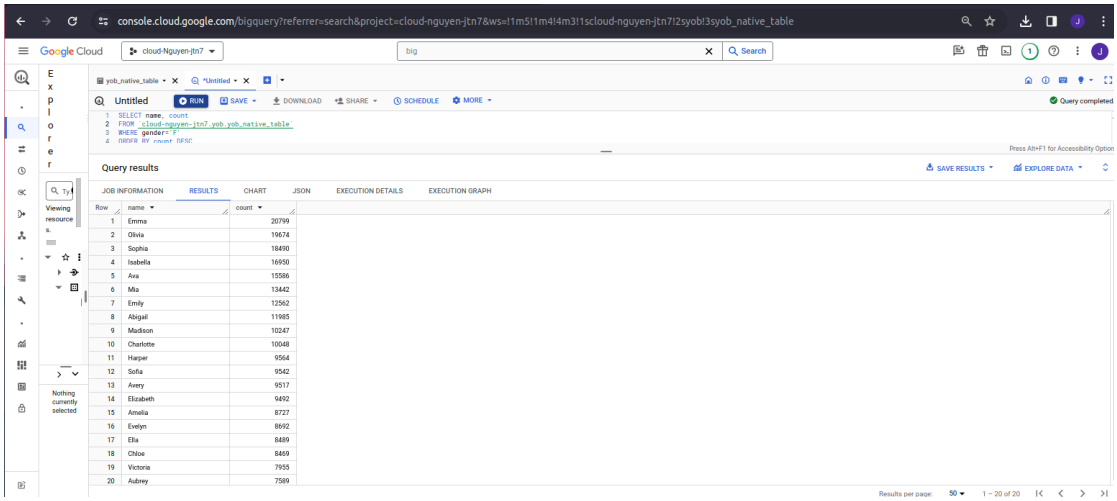
Then, click on the "Details" tab.

- Take a screenshot of the table's details that includes the number of rows in the table.



Run the query.

- Screenshot the query results and include it in your lab notebook



Via command-line bq command

- Screenshot your results and include it in your lab notebook

```
jtn7@cloudshell:~/cloud-nguyen-jtn7 (cloud-nguyen-jtn7)$ bq query "SELECT name, count
FROM [cloud-nguyen-jtn7.yob.yob_native_table]
WHERE gender='F'
ORDER BY count DESC
LIMIT 20;"
+-----------+-------+
|   name    | count |
+-----------+-------+
| Emma      | 20799 |
| Olivia    | 19674 |
| Sophia    | 18490 |
| Isabella  | 16950 |
| Ava       | 15586 |
| Mia       | 13442 |
| Emily     | 12562 |
| Abigail   | 11985 |
| Madison   | 10247 |
| Charlotte | 10048 |
| Harper    |  9564 |
| Sofia     |  9542 |
| Avery     |  9517 |
| Elizabeth |  9492 |
| Amelia    |  8727 |
| Evelyn    |  8692 |
| Ella      |  8489 |
| Chloe     |  8469 |
| Victoria  |  7955 |
| Aubrey    |  7589 |
+-----------+-------+
```

Via interactive bq session

- Screenshot your results and include it in your lab notebook

```
jtn7@cloudshell:~/cloud-nguyen-jtn7 (cloud-nguyen-jtn7)$ bq shell
Welcome to BigQuery! (Type help for more information.)
Cannot read termcap database;
using dumb terminal settings.
cloud-nguyen-jtn7> SELECT name, count FROM [cloud-nguyen-jtn7.yob.yob_native_table] WHERE gender='F' ORDER BY count DESC LIMIT 20;
+-----------+-------+
|   name    | count |
+-----------+-------+
| Emma      | 20799 |
| Olivia    | 19674 |
| Sophia    | 18490 |
| Isabella  | 16950 |
| Ava       | 15586 |
| Mia       | 13442 |
| Emily     | 12562 |
| Abigail   | 11985 |
| Madison   | 10247 |
| Charlotte | 10048 |
| Harper    |  9564 |
| Sofia     |  9542 |
| Avery     |  9517 |
| Elizabeth |  9492 |
| Amelia    |  8727 |
| Evelyn    |  8692 |
| Ella      |  8489 |
| Chloe     |  8469 |
| Victoria  |  7955 |
| Aubrey    |  7589 |
+-----------+-------+
```

Finally, run a query on your name. How popular was it?

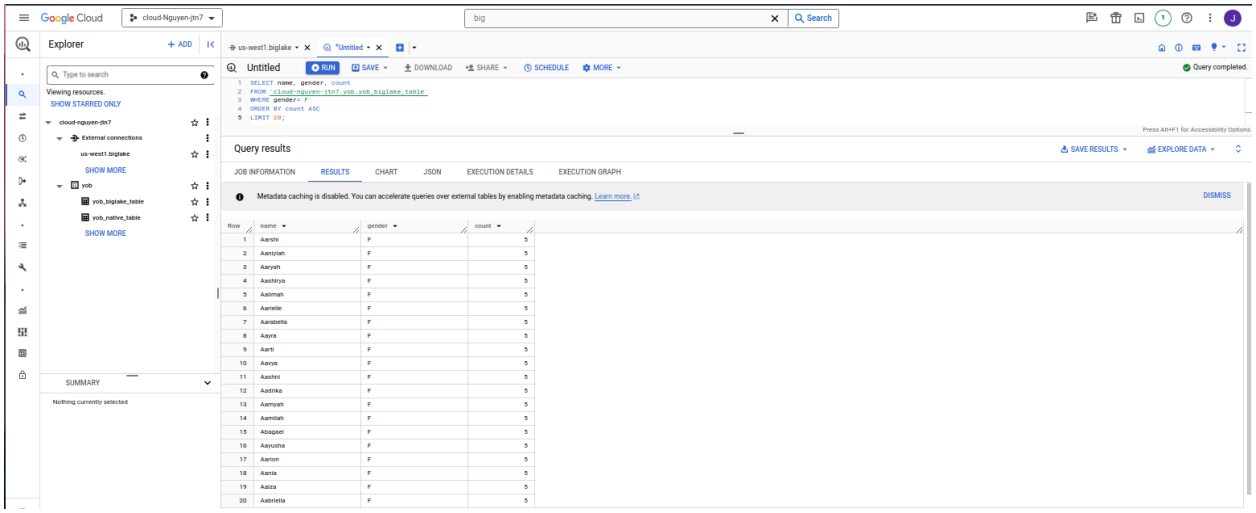
- Screenshot your results and include it in your lab notebook

```
cloud-nguyen-jtn7> SELECT name, gender, count FROM [cloud-nguyen-jtn7.yob.yob_native_table] WHERE name='Joseph';
+--------+--------+-------+
|  name  | gender | count |
+--------+--------+-------+
| Joseph | F      |     6 |
| Joseph | M      | 11995 |
+--------+--------+-------+
```

There were 6 women named Joseph and 11,995 men named Joseph in the dataset.

Run the query.

- Screenshot the query results and include it in your lab notebook



9.2g

Before running the query, answer the following question for your lab notebook:

- How much less data does this query process compared to the size of the table?

Selecting the data takes 21.94 GB while querying with the given query statement takes 3.05 GB. Thus the query process is 21.94 - 3.05 = 18.89 GB less than the size of the table.

Run the query and answer the following for your lab notebook:

- How many twins were born during this time range? **375362 twins were born**
- How much lighter on average are they compared to single babies? **~2.17116 lbs lighter**

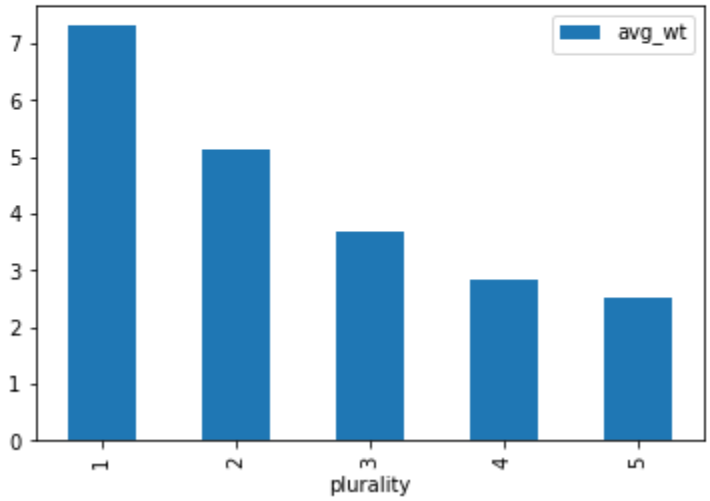
In examining the plots, which two features are the strongest predictors for a newborn baby's weight?

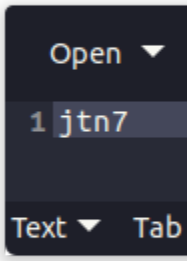
- Show the plots generated for the two most important features for your lab notebook

Plurality and Gestation time are the two most important features for predicting a newborn baby's weight.

```
[6]: df = get_distinct_values('plurality')
     df.plot(x='plurality', y='avg_wt', kind='bar')
```

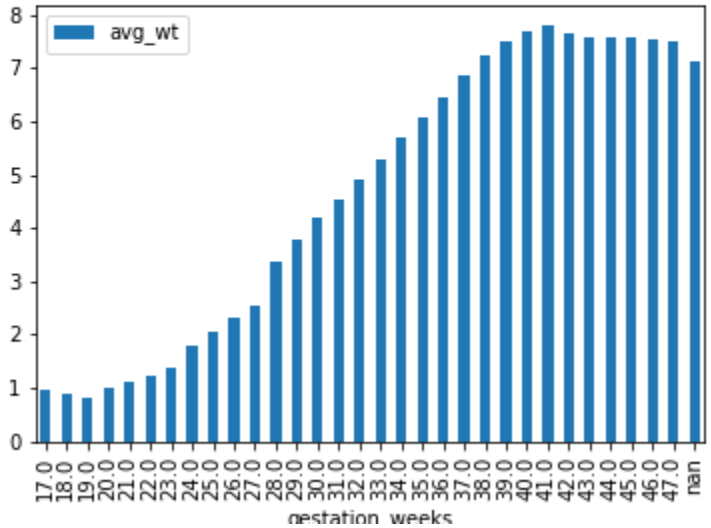
```
[6]: <matplotlib.axes._subplots.AxesSubplot at 0x7f8d2d6c6f10>
```

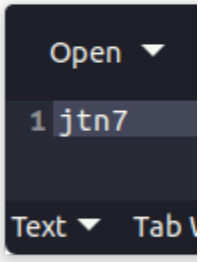





```
[8]: df = get_distinct_values('gestation_weeks')
     df.plot(x='gestation_weeks', y='avg_wt', kind='bar')
```

```
[8]: <matplotlib.axes._subplots.AxesSubplot at 0x7f8d2c997b50>
```





- What day saw the largest spike in trips to grocery and pharmacy stores?

On March 13, 2020, there was a 17% increase from baseline to grocery and pharmacy stores.

- On the day the stay-at-home order took effect (3/23/2020), what was the total impact on workplace trips?

There was a 49% decrease on workplace trips from the normal baseline.

- Which three airports were impacted the most in April 2020 (the month when lockdowns became widespread)?

Detroit Metropolitan Wayne County, McCarran International, and San Francisco International

- Run the query again using the month of August 2020. Which three airports were impacted the most?

McCarran International, Detroit Metropolitan Wayne County, and San Francisco International

- What table and columns identify the place name, the starting date, and the number of excess deaths from COVID-19?

Table: excess_death
Columns: placename, start_date, excess_deaths

- What table and columns identify the date, county, and deaths from COVID-19?

Table: us_counties
Columns: date, county, deaths

- What table and columns identify the date, state, and confirmed cases of COVID-19?

Table: us_states
Columns: date, state_name, confirmed_cases

- What table and columns identify a county code and the percentage of its residents that report they always wear masks?

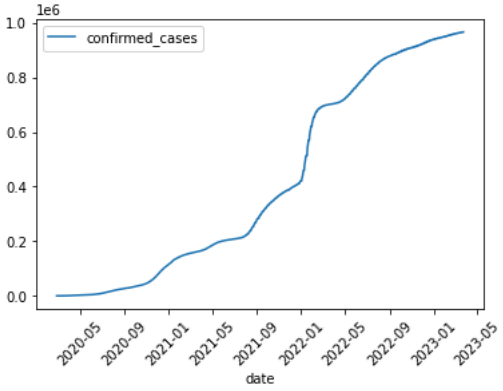
Table: mask_use_by_county
Column: county_fips_code, always

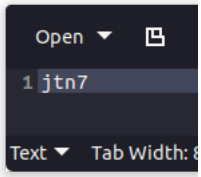
Within the notebook, perform the following query and plot the results using a line graph. The query pulls out the confirmed cases of COVID-19 across the state of Oregon by date.

- Show a screenshot of the plot and the code used to generate it for your lab notebook

[11]:
```
query_string = """
SELECT date, confirmed_cases
FROM bigquery-public-data.covid19_nyt.us_states
WHERE state_name = 'Oregon'
ORDER BY date ASC
"""
from google.cloud import bigquery
df = bigquery.Client().query(query_string).to_dataframe()
df.plot(x='date', y='confirmed_cases', kind='line', rot=45)
```

[11]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f8d2c728f10>
```



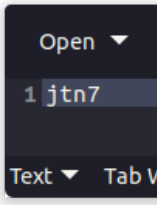


- From within your Jupyter notebook, run the query and write code that shows the first 10 states that reached 1000 deaths from COVID-19. Take a screenshot for your lab notebook.

[13]:
```
query_string = """
SELECT state_name, MIN(date) as date_of_1000
FROM bigquery-public-data.covid19_nyt.us_states
WHERE deaths > 1000
GROUP BY state_name
ORDER BY date_of_1000 ASC
"""
from google.cloud import bigquery
df = bigquery.Client().query(query_string).to_dataframe()
df.head(10)
```

[13]:

	state_name	date_of_1000
0	New York	2020-03-29
1	New Jersey	2020-04-06
2	Michigan	2020-04-09
3	Louisiana	2020-04-14
4	Massachusetts	2020-04-15
5	Illinois	2020-04-16
6	California	2020-04-17
7	Connecticut	2020-04-17
8	Pennsylvania	2020-04-17
9	Florida	2020-04-24

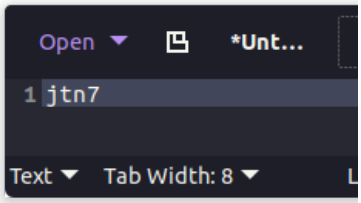


- Take a screenshot for your lab notebook of the Top 5 counties and the states they are located in.

[15]:
```
query_string = """
SELECT DISTINCT mu.county_fips_code, mu.always, ct.county, ct.state_name
FROM bigquery-public-data.covid19_nyt.mask_use_by_county as mu
LEFT JOIN bigquery-public-data.covid19_nyt.us_counties as ct
ON mu.county_fips_code = ct.county_fips_code
ORDER BY mu.always DESC
"""
from google.cloud import bigquery
df = bigquery.Client().query(query_string).to_dataframe()
df.head(5)
```

[15]:

	county_fips_code	always	county	state_name
0	06027	0.889	Inyo	California
1	36123	0.884	Yates	New York
2	48229	0.880	Hudspeth	Texas
3	06051	0.880	Mono	California
4	48141	0.877	El Paso	Texas



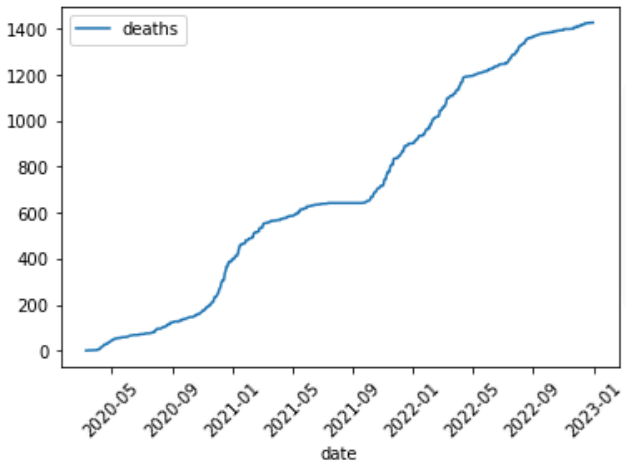
Deaths in Multnomah county

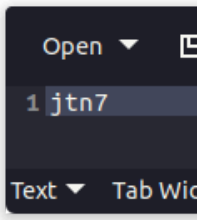
Construct a query string that obtains the number of deaths from COVID-19 that have occurred in Multnomah county for each day in the dataset, ensuring the data is returned in ascending order of date. Run the query and obtain the results.

- Plot the results and take a screenshot for your lab notebook.

[20]:
```
query_string = """
SELECT date, deaths
FROM bigquery-public-data.covid19_nyt.us_counties as ct
WHERE ct.county = 'Multnomah'
ORDER BY date
"""
from google.cloud import bigquery
df = bigquery.Client().query(query_string).to_dataframe()
df.plot(x='date', y='deaths', kind='line', rot=45)
```

[20]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f8d2c568a50>
```



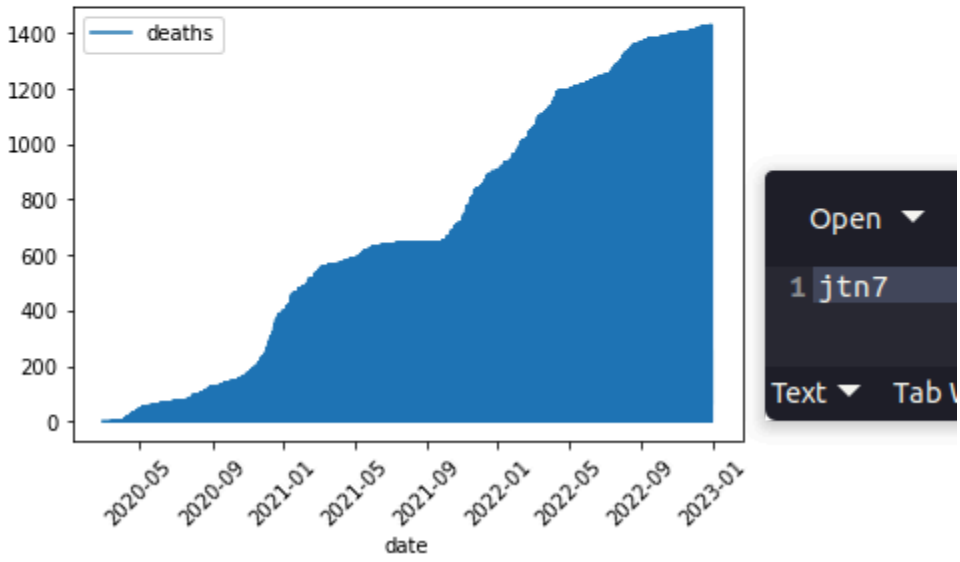


Deaths in Oregon
Construct a query string that obtains the number of deaths from COVID-19 that have occurred in Oregon for each day in the dataset, ensuring the data is returned in ascending order of date. Run the query and obtain the results.

- Plot the results and take a screenshot for your lab notebook.

```
[22]: query_string = """
      SELECT date, deaths
      FROM bigquery-public-data.covid19_nyt.us_counties as ct
      WHERE ct.state_name = 'Oregon'
      ORDER BY date
      """
      from google.cloud import bigquery
      df = bigquery.Client().query(query_string).to_dataframe()
      df.plot(x='date', y='deaths', kind='line', rot=45)
```

```
[22]: <matplotlib.axes._subplots.AxesSubplot at 0x7f8d2c6a8e90>
```



9.3g

- How long did the job take to execute?

It took about 43 seconds.

- Examine output.txt and show the estimate of π calculated.

Pi is roughly 3.141501591415016

After rescaling

- How long did the job take to execute? How much faster did it take?

It took 31 seconds so It was over 20% faster than previously.

- Examine output2.txt and show the estimate of π calculated.

Pi is roughly 3.1413466714134666

9.4
Reverse-engineer the code and make note of where the input is taken from and where the output goes to by default.

Answer the following questions for your lab notebook.

- Where is the input taken from by default?

'../javahelp/src/main/java/com/google/cloud/training/dataanalyst/javahelp/'

- Where does the output go by default?

'/tmp/output'

- Examine both the getPackages() function and the splitPackageName() function. What operation does the 'PackageUse()' transform implement?

PackageUse() calls getPackages() as a generator function. getPackages() searches the keyword by line where ';' denotes the end of a line; it calls splitPackageName() which splits up the package names at each '.' by using python brackets for slicing the strings and returns an array of these substrings.

- Look up Beam's CombinePerKey. What operation does the TotalUse operation implement?

CombinePerKey is called with sum, meaning that it takes in an iterable of numbers and adds them together. In this case, we are summing up how many times the PackageUse() function is call, so TotalUse represents how many package uses there are.

The operations in the pipeline mimic a Map-Reduce pattern, demonstrating Beam's ability to support it.
Answer the following questions for your lab notebook.

- Which operations correspond to a "Map"?

GetImport and PackageUse call beam.FlatMap() which corresponds to Map.

- Which operation corresponds to a "Shuffle-Reduce"?

TotalUse calls beam.CombinePerKey() which shuffle and reduce the data.

- Which operation corresponds to a "Reduce"?

Top_5 calls beam.transforms.combiners.Top.Of() which corresponds to a reduce.

Go to where the output file is written out and cat the file.

- Take a screenshot of its contents
- Explain what the data in this output file corresponds to based on your understanding of the program

The data in the file represents the name of the Java package and how many times that package was called.

When the code is invoked as a module, it executes the run() function. Within the function, a pipeline p is incrementally constructed using the Beam syntax (lines, counts, output) before the entire pipeline is executed via p.run(). Examine the code that implements the function and answer the following questions for your lab notebook:

- What are the names of the stages in the pipeline?

Split, PairWithOne, GroupAndSum

- Describe what each stage does.

Split: splits a line of text into individual words
PairWithOne: Each word is paired with a number
GroupAndSum: Matching words are grouped together and their number summed together. Thus we will get a list of unique words and their count.

After running the script, perform the following and show a screenshot of the results in your lab notebook:

- Use wc with an appropriate flag to determine the number of different words in King Lear.

```
(env) jtn7@cloudshell:~/training-data-analyst/courses/data_analysis/lab2/python (cloud-nguyen-jtn7)$ wc -w outputs-00000-of-00001
9568 outputs-00000-of-00001
(env) jtn7@cloudshell:~/training-data-analyst/courses/data_analysis/lab2/python (cloud-nguyen-jtn7)$
```

- Use sort with appropriate flags to perform a numeric sort on the key field containing the count for each word in descending order. Pipe the output into head to show the top 3 words in King Lear and the number of times they appear

```
(env) jtn7@cloudshell:~/training-data-analyst/courses/data_analysis/lab2/python (cloud-nguyen-jtn7)$ sort -t: -k 2nr,2 outputs-00000-of-00001 > sorted_output
(env) jtn7@cloudshell:~/training-data-analyst/courses/data_analysis/lab2/python (cloud-nguyen-jtn7)$ ls
env       grep.py  install_packages_OLD.sh  is_popular.py                JavaProjectsThatNeedHelp_PY2_Version.py  outputs-00000-of-00001
grepc.py  head     install_packages.sh      JavaProjectsThatNeedHelp.py  OLD_grepc.py                             sorted_output
(env) jtn7@cloudshell:~/training-data-analyst/courses/data_analysis/lab2/python (cloud-nguyen-jtn7)$ head -n 3 sorted_output
the: 786
I: 622
and: 594
```

Perform the following and show a screenshot of the results in your lab notebook:

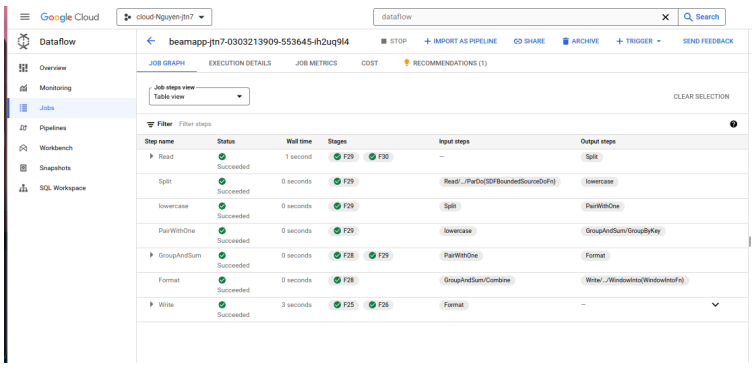
- Use the previous method to show the top 3 words in King Lear, case-insensitive, and the number of times they appear.

```
(env) jtn7@cloudshell:~/training-data-analyst/courses/data_analysis/lab2/python (cloud-nguyen-jtn7)$ sort -t: -k 2nr,2 outputs-00000-of-00001 > sorted_output
(env) jtn7@cloudshell:~/training-data-analyst/courses/data_analysis/lab2/python (cloud-nguyen-jtn7)$ head -n 3 sorted_output
the: 908
and: 738
i: 622
(env) jtn7@cloudshell:~/training-data-analyst/courses/data_analysis/lab2/python (cloud-nguyen-jtn7)$
```

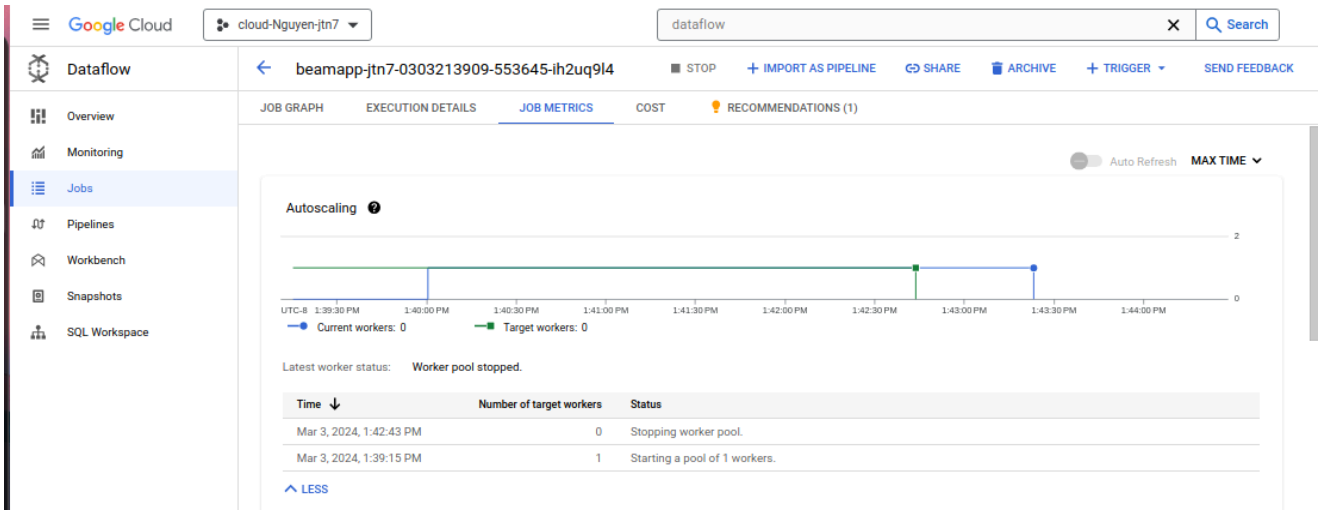
After executing the program, which takes around 5 minutes to complete, visit Dataflow in the web console and click on the Dataflow job that was executed. Examine both "Job Graph" and "Job Metrics". Include the following in your lab notebook:

- The part of the job graph that has taken the longest time to complete.

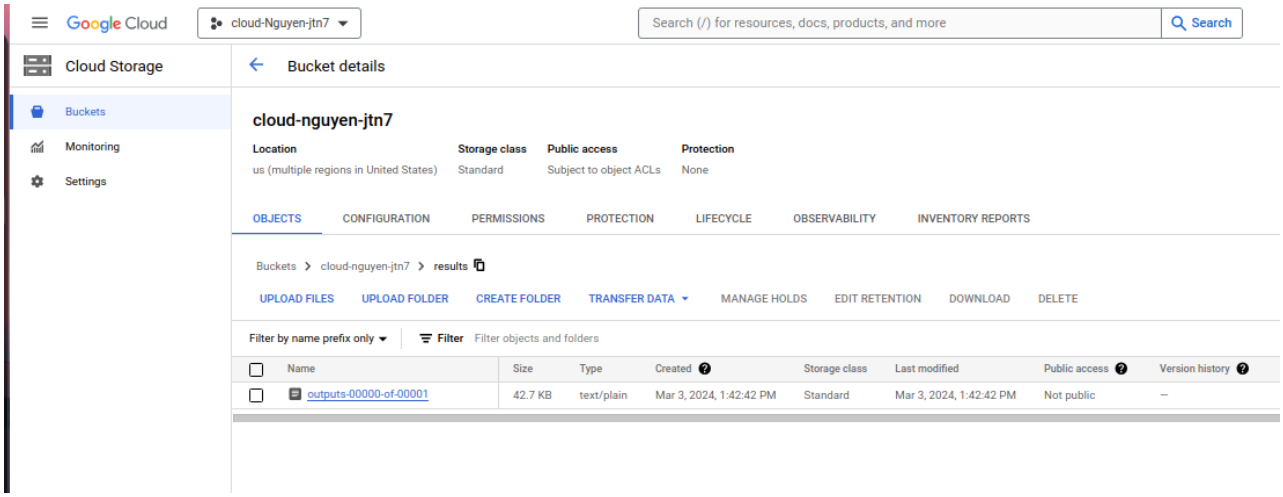
Writing the data took the longest time to complete.



- The autoscaling graph showing when the worker was created and stopped.



- Examine the output directory in Cloud Storage. How many files has the final write stage in the pipeline created? **1 file, the output file which is stored in the results directory.**

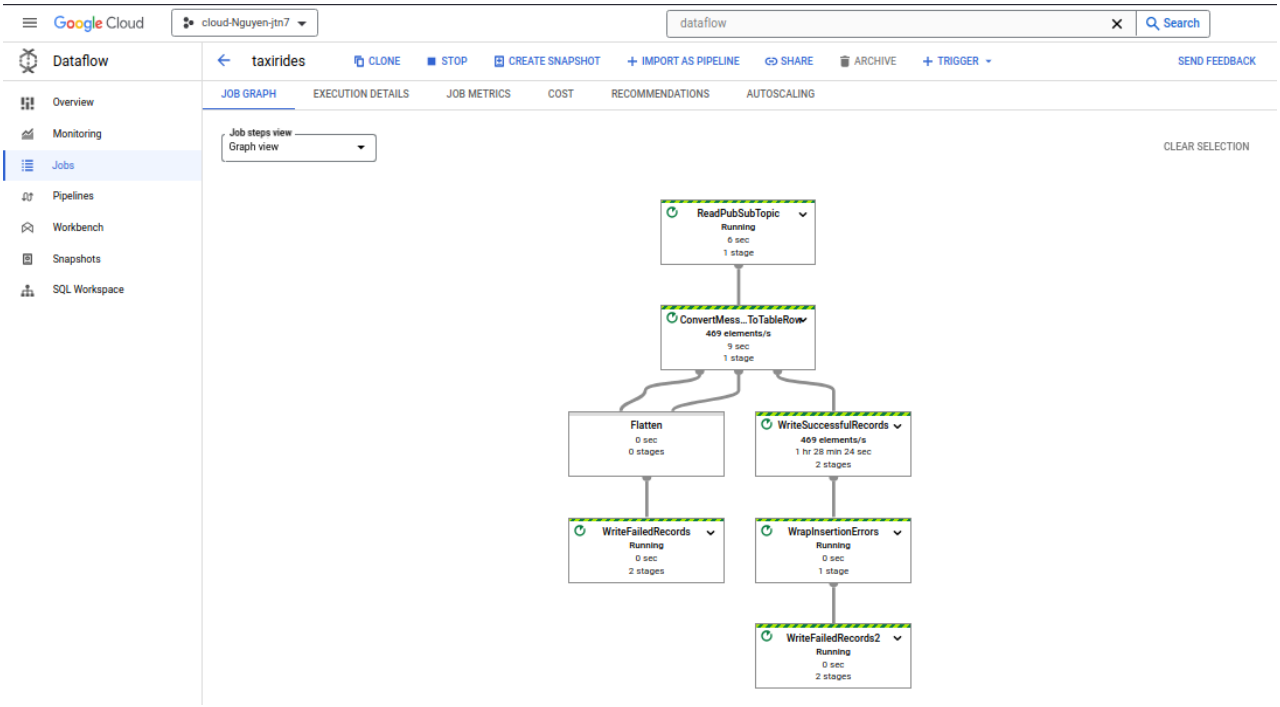


The data that is returned is a JSON object.

- Take a screenshot listing the different fields of this object.

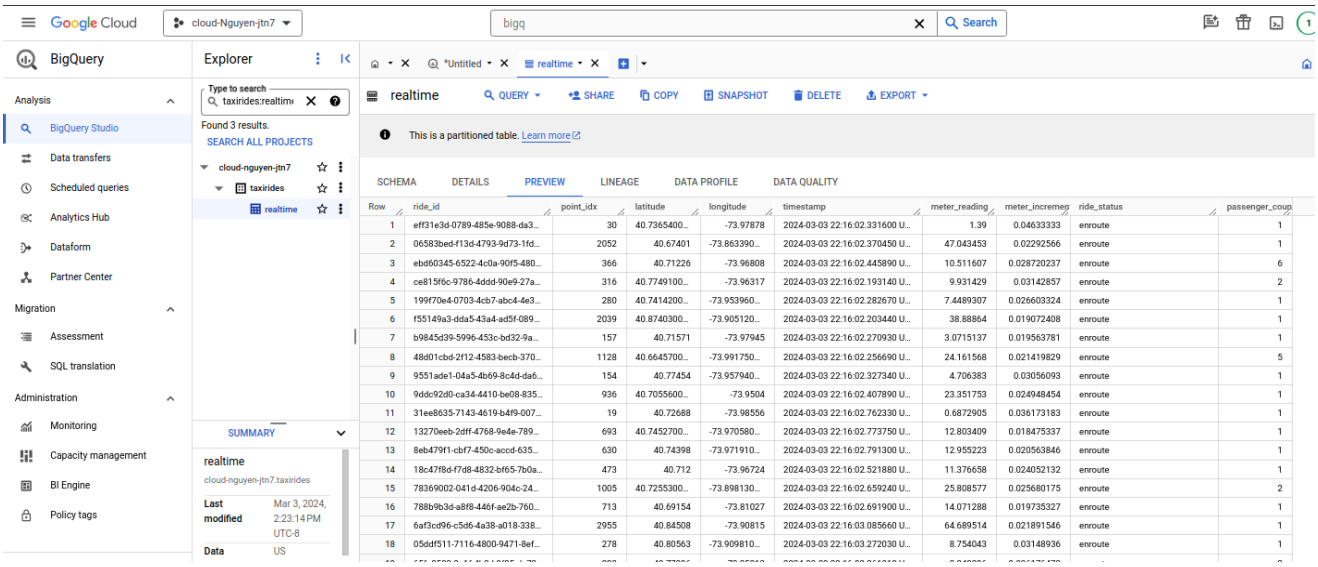
Then, visit the Cloud Dataflow console in the web UI and view the running job. Wait until the job is functioning and has processed 5-10 minutes worth of raw data from the stream..

- Take a screenshot of the pipeline that includes its stages and the number of elements per second being handled by individual stages.

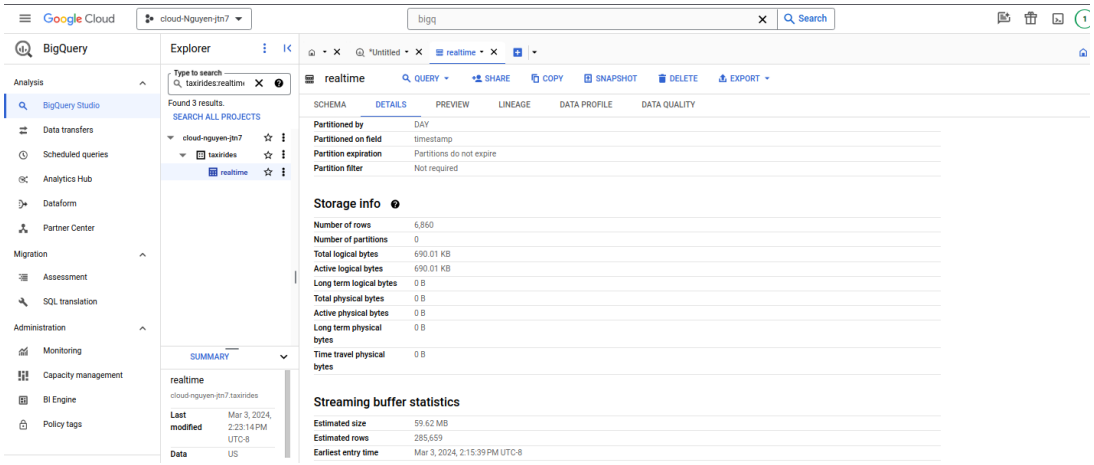


Visit the BigQuery web console and navigate to the table we've created within our project at taxirides:realtime. Examine the table.

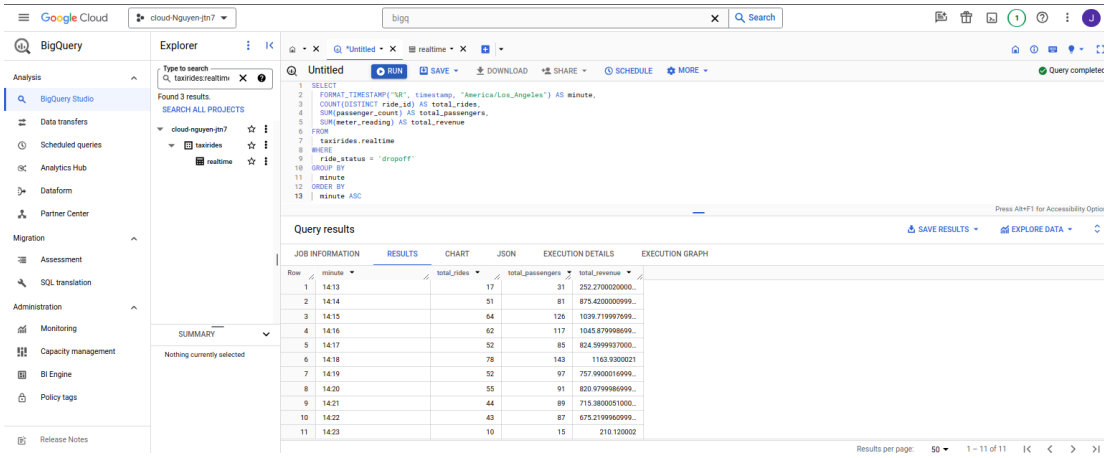
- Take a screenshot showing the number of passengers and the amount paid for the first ride



- Take a screenshot showing the estimated number of rows in the table.



- Take a screenshot showing the per-minute number of rides, passengers, and revenue for the data collected



- Take a screenshot showing the plot for your data for your lab notebook