Name : Manikandan Mohan

Case Study 1

Amazon Books Catalog Search, Data Cleansing, Analytics, Pipeline &Reporting using Apache Beam, Python pySpark on a Google Data Proc Cluster [ubuntu 20.04 VMs].

PROBLEM STATEMENT:

1. Find the average user rating for Fiction books, Non-fiction books and for all books together and write the result in separate files. – python (analytics), PySpark

2. Sort the books in alphabetical order and the last 5 books and print the output – Pyspark 3. Kaggle Dataset - https://www.kaggle.com/sootersaalu/amazon-top-50-bestselling-books-2009-2019

Notes - This is a dataset on Amazon's Top 50 bestselling books from 2009 to 2019., it contains 550+ books, and has been categorized into fiction and non-fiction using Goodreads.

[Note, one can add/multiple the data set to about 5000+] books to utilize power of GCP Cluster, Analytics, Spark Job, BigQuery, BigTable capabilities.

Implementation Steps -

· Pre-requisite – One should download this csv and do data cleaning (replace the commas in the Name column with semicolons). [Upload this csv in the GCP bucket] – Pyspark, Hive [10% credits]

One has to build a DataProc Cluster [at least 2 nodes] to run Python pySpark batch jobs [Kaggle dataset, from GCP bucket] on Google Cloud. – DataProc [15% credits]

· Advisable to connect to the nodes using CLI and provision the VMs with required configurations using [15% credits] o GCP Cloud SDK o gsutil tool

· Create/Configure a pipeline [10% credits] using Command Line

Pipeline = Reading input + Transforming Data + Writing Output

Output results should be stored as raw data in Hive[BigQuery] before stored finally into HBase[BigTable] for further processing & reporting to Analytics Engines. – Hbase, Hive [15% Credits]

· Run the pipeline – Apache Beam [10% credit]

· Print the results to console and store for further analytics – Hbase [10% credits]

Extract of the pipeline execution and output is here – (good to use HBase) – [10% credits]

Engineering Best Practices to be followed [30% credits]

1. Logging has to be implemented in all the Python Components

2. Exception handling framework need to be implemented

3. Unit testing of each Python component/service using Pytest/UnitTest/Mock Objects

4. Unit Testing of Spark Cluster, Data Pipeline [is desirable]

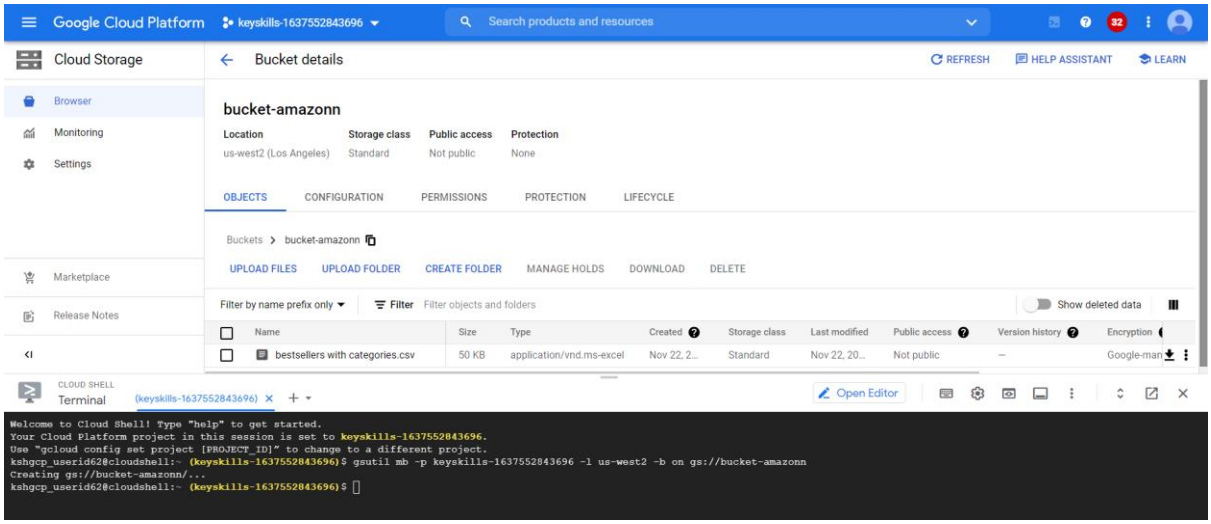
5. Usage of 4 to 5 Big Data Design Patterns using Python [is desirable]

Answer

Project Name : keyskills-1637220444729

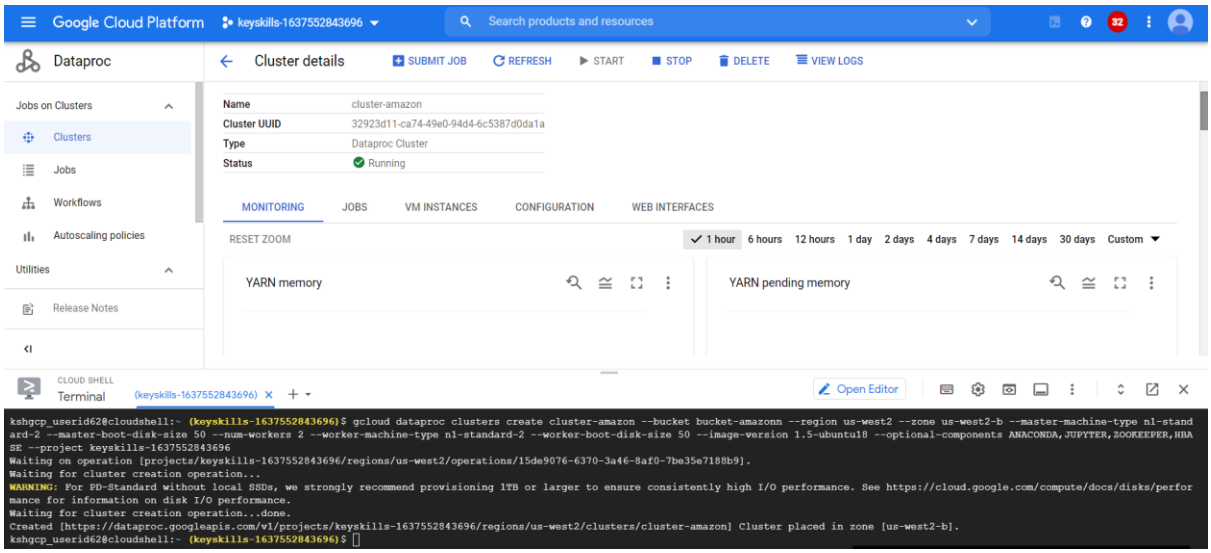
Creating a bucket storage named "bucket-amazonn" using CLI and uploading the kaggle dataset.

gsutil mb -p keyskills-1637552843696 -l us-west2 -b on gs://bucket-amazonn



Creating a cluster using Command Line Interface (CLI) :

gcloud dataproc clusters create cluster-amazon --bucket bucket-amazonn --region us-west2 --zone us-west2-b --master-machine-type n1-standard-2 --master-boot-disk-size 50 --num-workers 2 --worker-machine-type n1-standard-2 --worker-boot-disk-size 50 --image-version 1.5-ubuntu18 --optional-components ANACONDA,JUPYTER,ZOOKEEPER,HBASE --project keyskills-1637552843696



Creating a python (spark) file with the required queries and uploading it into the bucket.

Pyspark code :

amazon_book_analysis.py

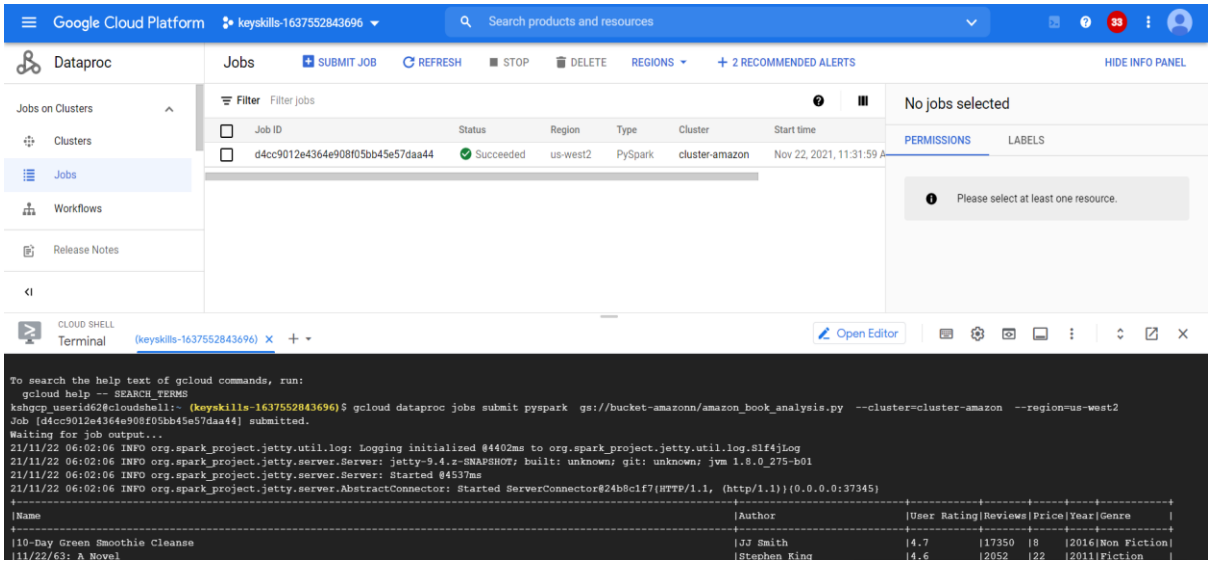
```
import pyspark as sc
from pyspark import SparkConf, SparkContext
from pyspark.sql import SparkSession
```

```
from pyspark.sql.functions import translate

#Reading Data using SparkSession
spark = SparkSession.builder.master("local[*]").getOrCreate()
data =
spark.read.option("header",True).options(inferSchema='True',delimiter=',').csv("gs
://bucket-amazonn/bestsellers with categories.csv" )

data.show(truncate=False)
data.printSchema()

#Replacing ‘,’ with ‘;’ in the name column
data = data.withColumn('Name', translate('Name', ',', ';'))
data.show(truncate=False)

#Find averages of fiction , non- fiction and overall

data.filter(data['Genre']=='Fiction').agg({'User Rating' : 'avg'}).show()
data.filter(data['Genre']=='Non Fiction').agg({'User Rating' : 'avg'}).show()
data.agg({'User Rating': 'avg'}).show()
avg_fict = data.filter(data['Genre']=='Fiction').agg({'User Rating' : 'avg'})
avg_non_fict = data.filter(data['Genre']=='Non Fiction').agg({'User Rating' : 'avg'})
avg_all = data.agg({'User Rating' : 'avg'})

avg_fict.write.csv('gs://bucket-amazonn/Avg_of_Fiction')
avg_non_fict.write.csv('gs://bucket-amazonn/Avg_of_Non_Fiction')
avg_all.write.csv('gs://bucket-amazonn/Avg_of_Allbooks')

data.orderBy(data.Name.desc()).limit(5).show(truncate=False)
```

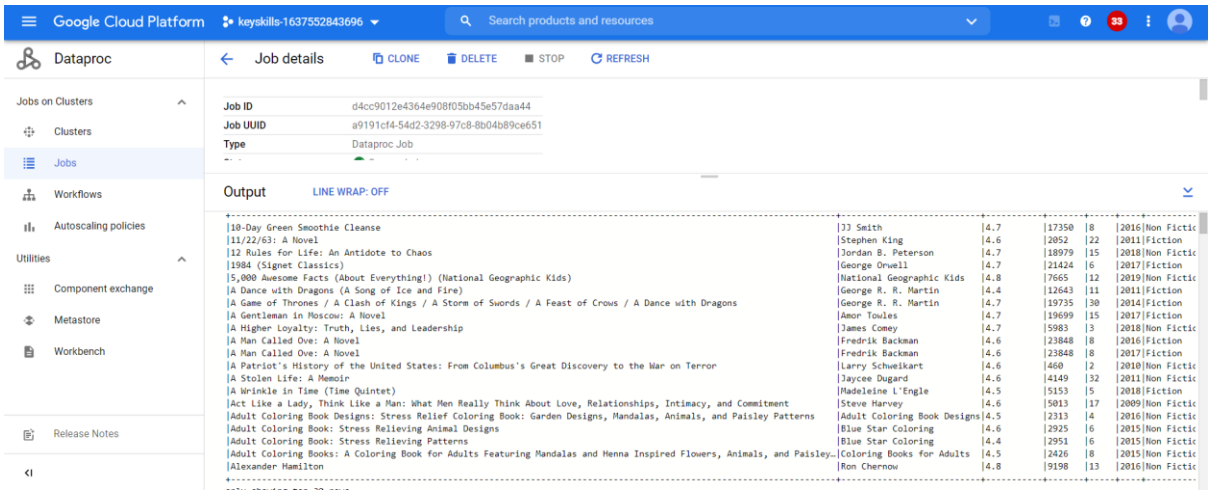
data.write.csv('gs://bucket-amazonn/clean_books_amazon.csv')

Executing the python file by submitting a job in dataproc using CLI.

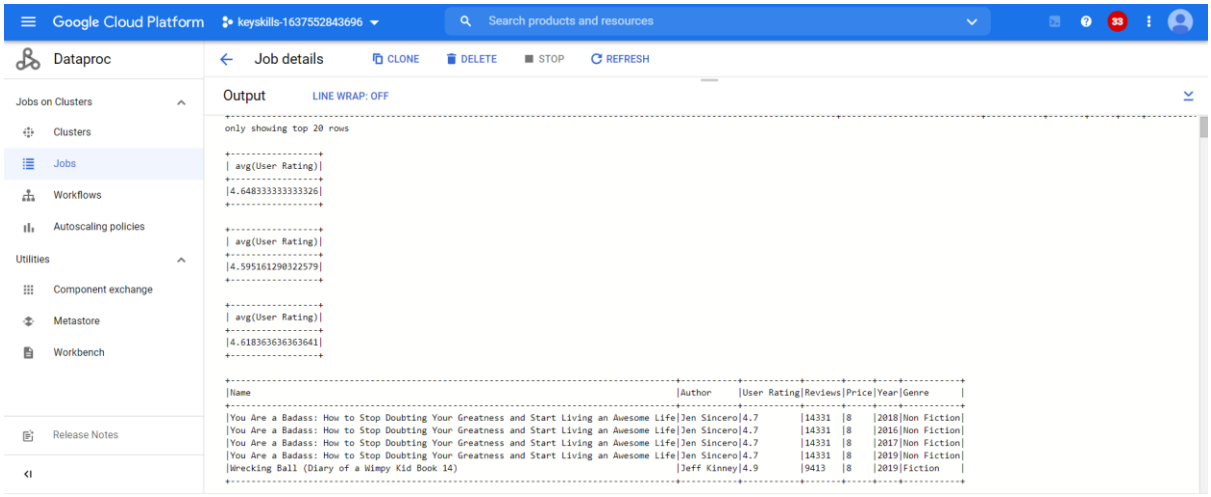
gcloud dataproc jobs submit pyspark gs://bucket-amazonn /amazon_book_analysis.py --cluster=cluster-amazon --region=us-west2



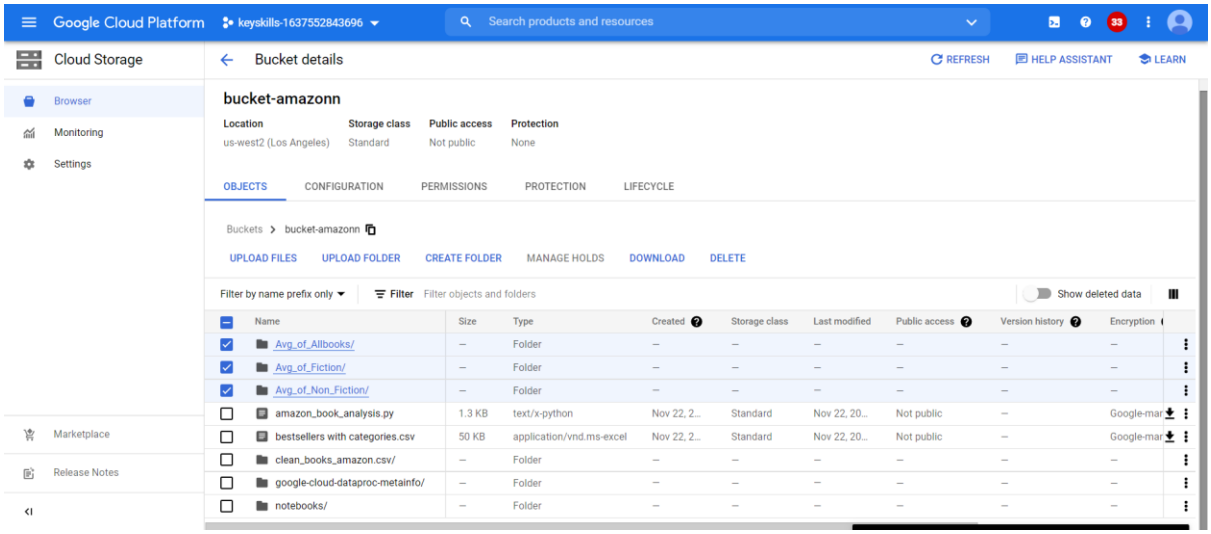
Job output : Displays the top 20 rows.



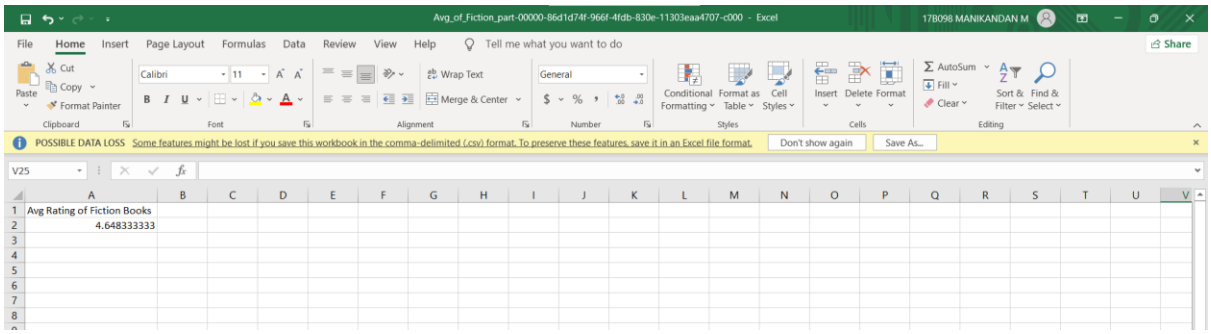
Displays the average rating of books and the last 5 rows of the dataset.



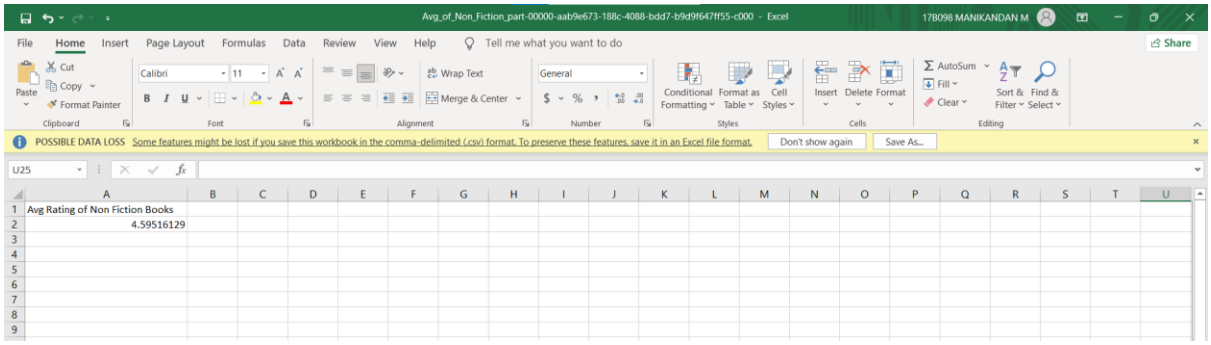
Output CSV files written to the bucket.



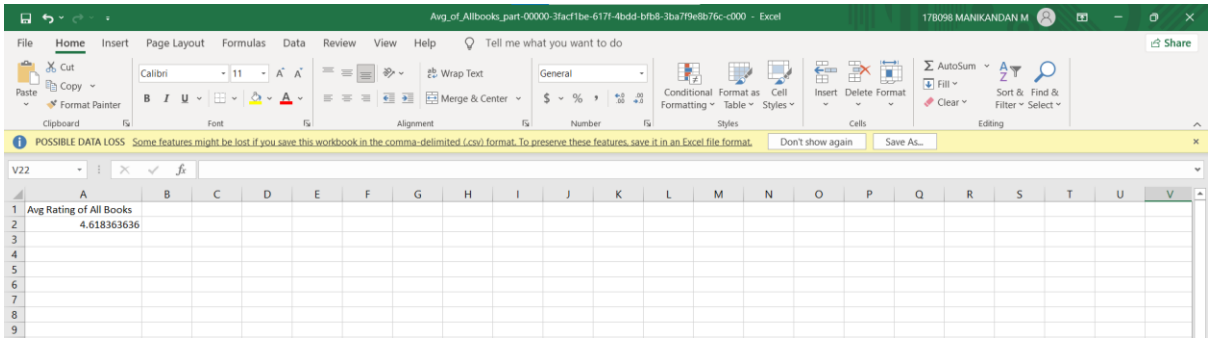
Average Rating of Fiction Books



Average Rating of Non Fiction Books



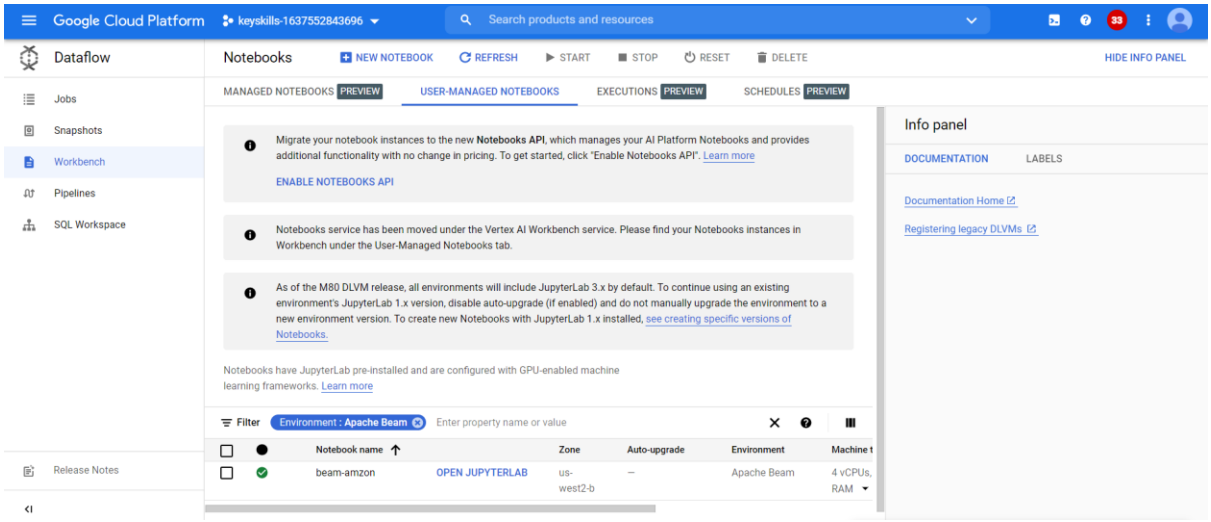
Average Rating of all the Books.



Last 5 rows while the dataset in alphabetical order

Creating a dataflow pipeline job using apache beam (python) for executing all the queries and to write the table in the bigquery.

Creating a juputer lab notebook with apache beam on dataflow workbench.



Pipeline code :

Dataflowpipeline.ipynb [Using Jupyter Lab on Google Cloud]

```
import apache_beam as beam
from apache_beam.runners.interactive.interactive_runner import InteractiveRunner
from apache_beam.options.pipeline_options import PipelineOptions
import google.auth
from apache_beam.options import pipeline_options
from apache_beam.options.pipeline_options import GoogleCloudOptions
from apache_beam.runners import DataflowRunner
from google.cloud import bigquery
from apache_beam.runners.runner import PipelineState
from IPython.core.display import display, HTML


def Split(element):
    return element.split(",")


# Finding the average of given list of values
class avgRatingFn(beam.CombineFn):
```

```
    def create_accumulator(self):
        return (0.0, 0)  # initialize (sum, count)

    def add_input(self, sum_count, input1):
        (sum1, count) = sum_count
        (sum2, c2) = input1
        return sum1 + float(sum2) * c2, count + c2

    def merge_accumulators(self, accumulators):
        ind_sums, ind_counts = zip(*accumulators)
        return sum(ind_sums), sum(ind_counts)

    def extract_output(self, sum_count):
        (sum1, count) = sum_count
        return sum1 / count if count else float('NaN')

pipe = beam.Pipeline(InteractiveRunner())

# Setting up the Apache Beam pipeline options.
options = pipeline_options.PipelineOptions(flags=[])

# Sets the project to the default project in the current Google Cloud environment.
_, options.view_as(GoogleCloudOptions).project = google.auth.default()
options.view_as(GoogleCloudOptions).region = 'us-west2'
dataflow_gcs_location = 'gs://bucket-amazonn/dataflowAmz'
options.view_as(GoogleCloudOptions).staging_location = '%s/staging' %
dataflow_gcs_location
```

```
# The Dataflow Temp Location location is used to store intermediate results before outputting the final result.
options.view_as(GoogleCloudOptions).temp_location = '%s/temp' % dataflow_gcs_location


clientBQ = bigquery.Client()


dataset_id = "keyskills-1637552843696.dataflowpipelineAmz"
dataset = bigquery.Dataset(dataset_id)
dataset.location = "us-west2"
dataset.description = "Amazon dataset books"
clientBQ.create_dataset(dataset, timeout = 30)


#Converts csv to json
def to_json(csv_file):
    fields = csv_file.split(',')
    json_file = {"Name": fields[0],
                "Author": fields[1],
                "User_Rating": fields[2],
                "Reviews": fields[3],
                "Price": fields[4],
                "Year": fields[5],
                "Genre": fields[6]
                }
    return json_file
table_schema = 'Name:STRING,Author:STRING,User_Rating:FLOAT,Reviews:INTEGER,Price:INTEGER,Year: INTEGER ,Genre:STRING'
```

```
bookAmz = (pipe | beam.io.ReadFromText("gs://bucket-amazonn/clean_books_amazon.csv/part-00000-44653796-5293-4a2a-b34e-225c74788cd8-c000.csv"))

(bookAmz | 'cleaned_data to json' >> beam.Map(to_json)

 | 'write to bigquery' >> beam.io.WriteToBigQuery(

        "keyskills-1637552843696:dataflowpipelineAmz.tableAmz",

        schema=table_schema,

        create_disposition=beam.io.BigQueryDisposition.CREATE_IF_NEEDED,

        write_disposition=beam.io.BigQueryDisposition.WRITE_APPEND,

        custom_gcs_temp_location="gs://bucket-amazonn/dataflowAmz/temp"

    )

 )


pipelineAmz = pipe.run()

if pipelineAmz.state == PipelineState.DONE:

   print('The pipeline is running successfully \n The table is sent to big query successfully !!!')

else:

   print('Error running the pipeline')


# Reads data and split based on ‘,’

pipe2 = beam.Pipeline(InteractiveRunner())

AmzBooks = (pipe2 | beam.io.ReadFromText("gs://bucket-amazonn/clean_books_amazon.csv/part-00000-44653796-5293-4a2a-b34e-225c74788cd8-c000.csv") | beam.Map(Split))


# Filter records having fiction. Map each rating as a set (rating,1) .Using combineperkey to count the number of each rating. Running the average function and write the result to Fiction_result1
```

```
avg_fict = (
    AmzBooks
    | beam.Filter(lambda rec: rec[6] == "Fiction")
    | beam.Map(lambda rec: (rec[2], 1))
    | "Fict Combine keys1" >> beam.CombinePerKey(sum)
    | "Fict Combine Global keys1" >> beam.CombineGlobally(avgRatingFn())
    | "Fict Write to  bucket1" >> beam.io.WriteToText("gs://bucket-amazonn/dataflowAmz/FictionAvg_Result")
)

#Same for "non fiction"
avg_non_fict = (
    AmzBooks
    | beam.Filter(lambda rec: rec[6] == "Non Fiction")
    | beam.Map(lambda rec: (rec[2], 1))
    | "N_Fict Combine keys" >> beam.CombinePerKey(sum)
    | "N_Fict Combine Global keys" >> beam.CombineGlobally(avgRatingFn())
    | "N_Fict Write to bucket" >> beam.io.WriteToText("gs://bucket-amazonn/dataflowAmz/NonFictionAvg_Result")
)

#Same for "all genre"
avg_all = (
    AmzBooks
    | beam.Map(lambda rec: (rec[2], 1))
    | "All Combine keys" >> beam.CombinePerKey(sum)
    | "All Combine Global keys" >> beam.CombineGlobally(avgRatingFn())
    | "All Write to bucket" >> beam.io.WriteToText("gs://bucket-amazonn/dataflowAmz/AllBookAvg_Result")
```

)

Map each record’s 0th column that is name with value 1.

Running top.of(5) to sort and get the last5 books alphabetically and store it.

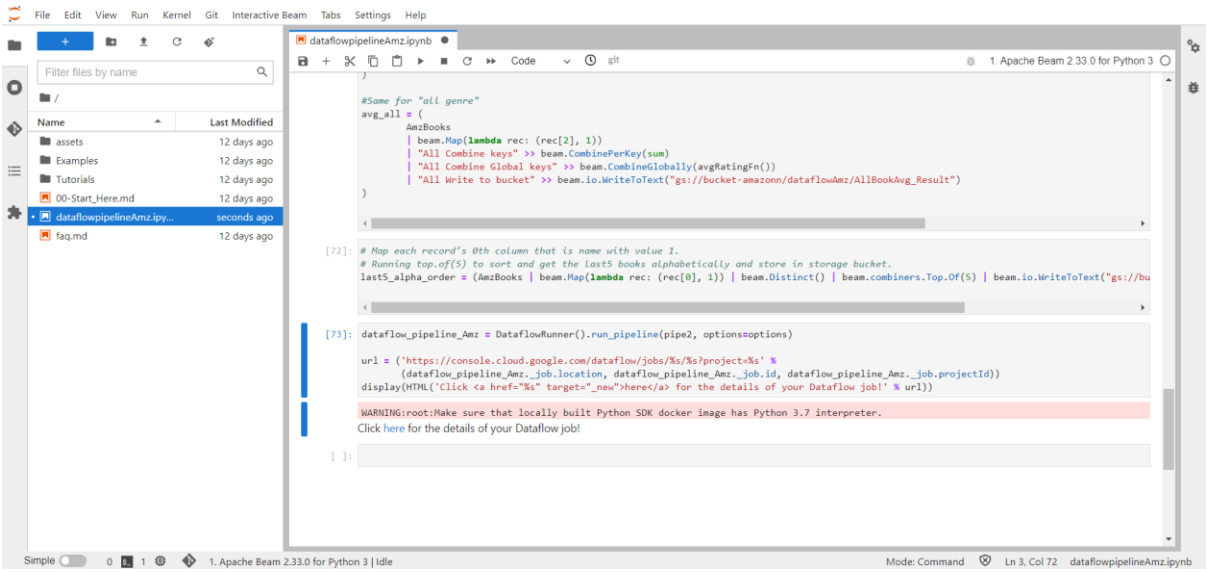
last5_alpha_order = (AmzBooks | beam.Map(lambda rec: (rec[0], 1)) | beam.Distinct() | beam.combiners.Top.Of(5) | beam.io.WriteToText("gs://bucket-amazonn/dataflowAmz/Last5_Result"))

dataflow_pipeline_Amz = DataflowRunner().run_pipeline(pipe2, options=options)

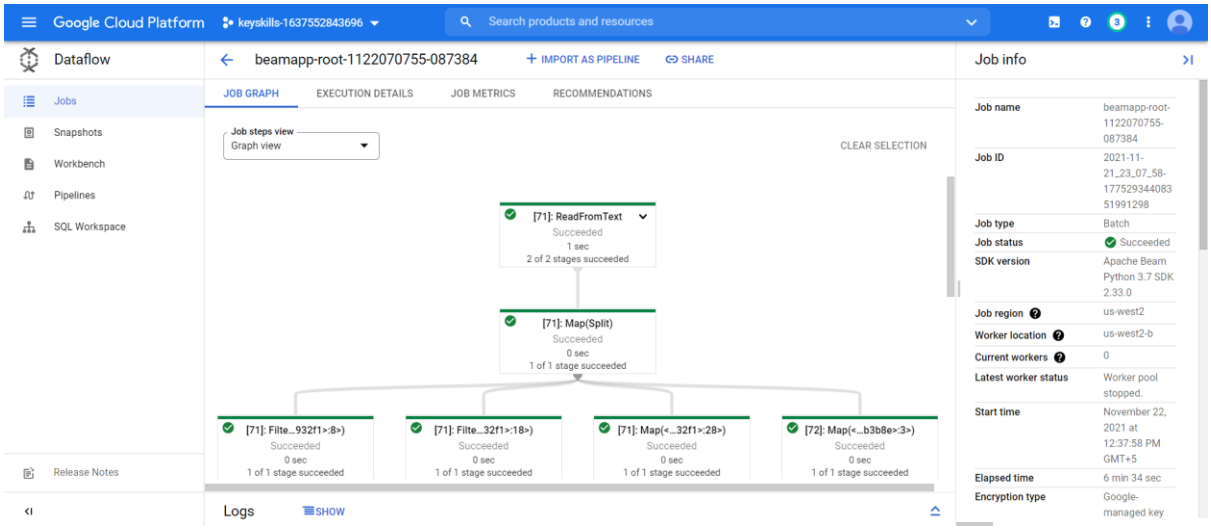
url = ('https://console.cloud.google.com/dataflow/jobs/%s/%s?project=%s' %

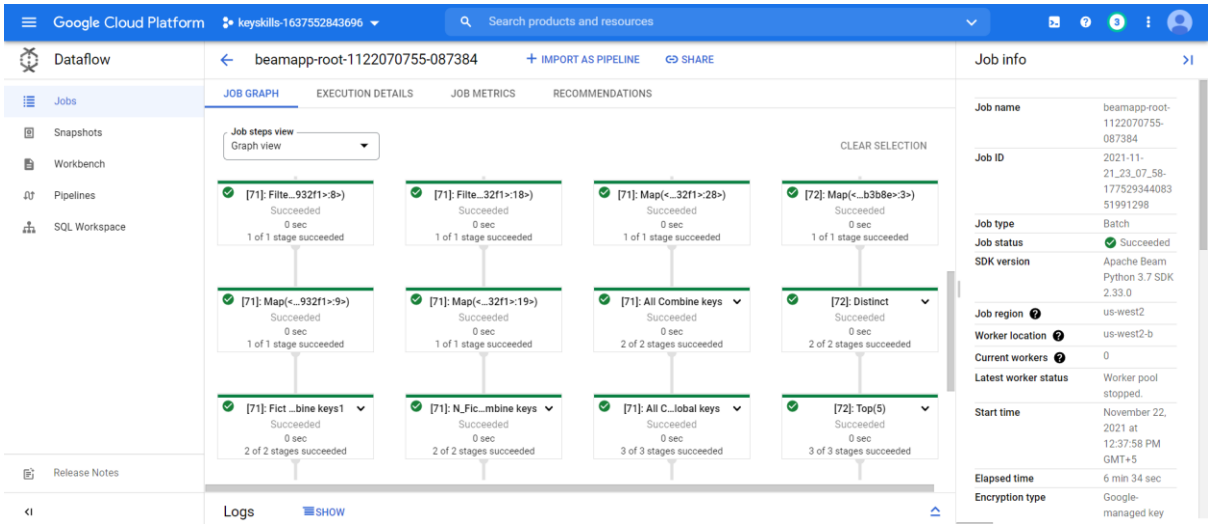
(dataflow_pipeline_Amz._job.location, dataflow_pipeline_Amz._job.id, dataflow_pipeline_Amz._job.projectId))

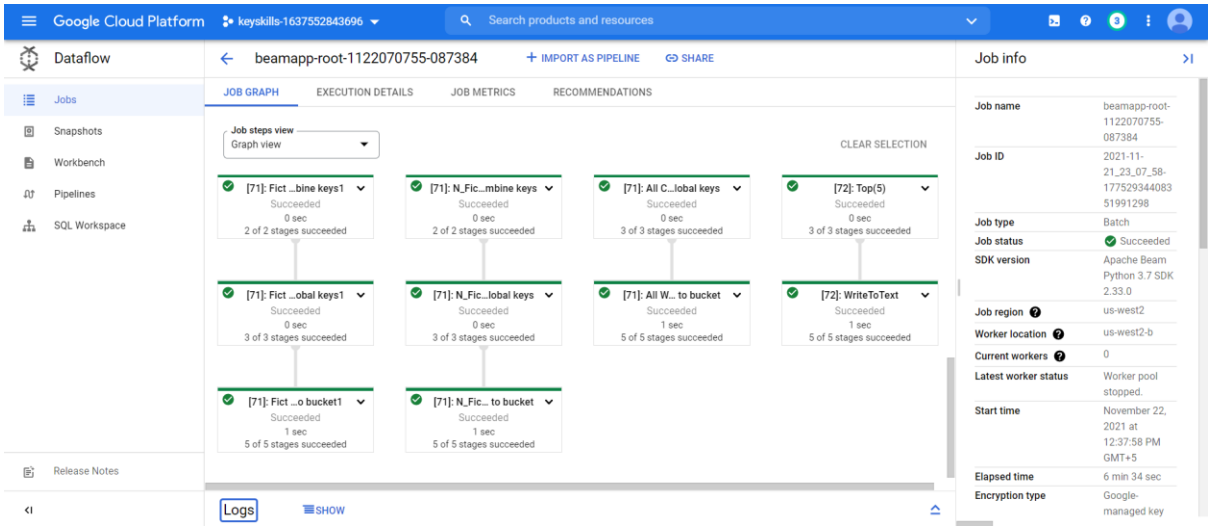
display(HTML('Click here for the details of your Dataflow job!' % url))



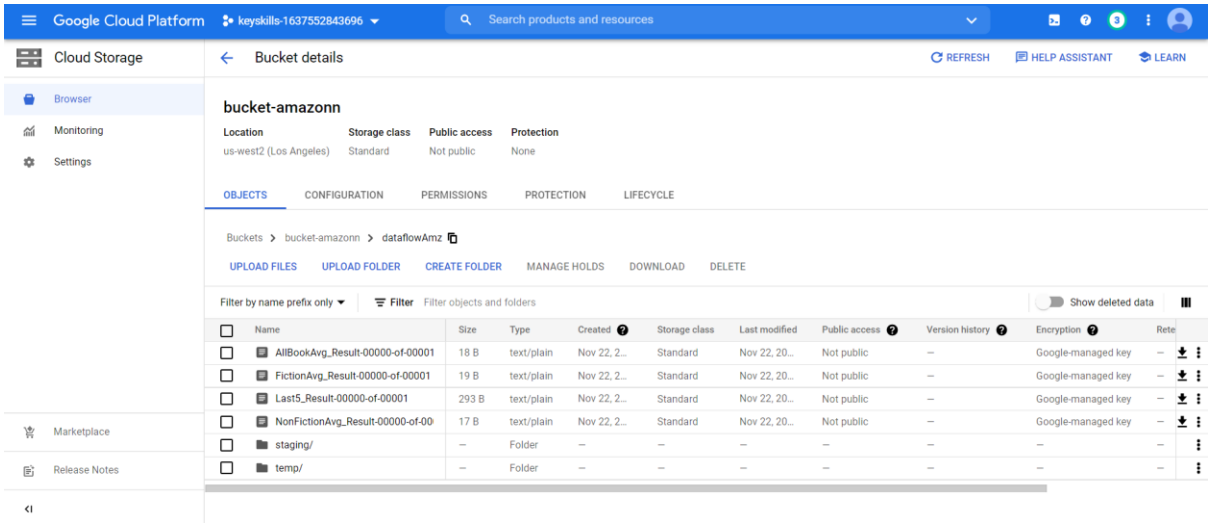
Dataflow Pipeline Job







All the necessary dataflow files written to the bucket.



The dataflow pipeline read the csv file from the bucket and wrote it as a table to the dataset in BigQuery.