Tony Nguyen

Dr. Shawn Bowers

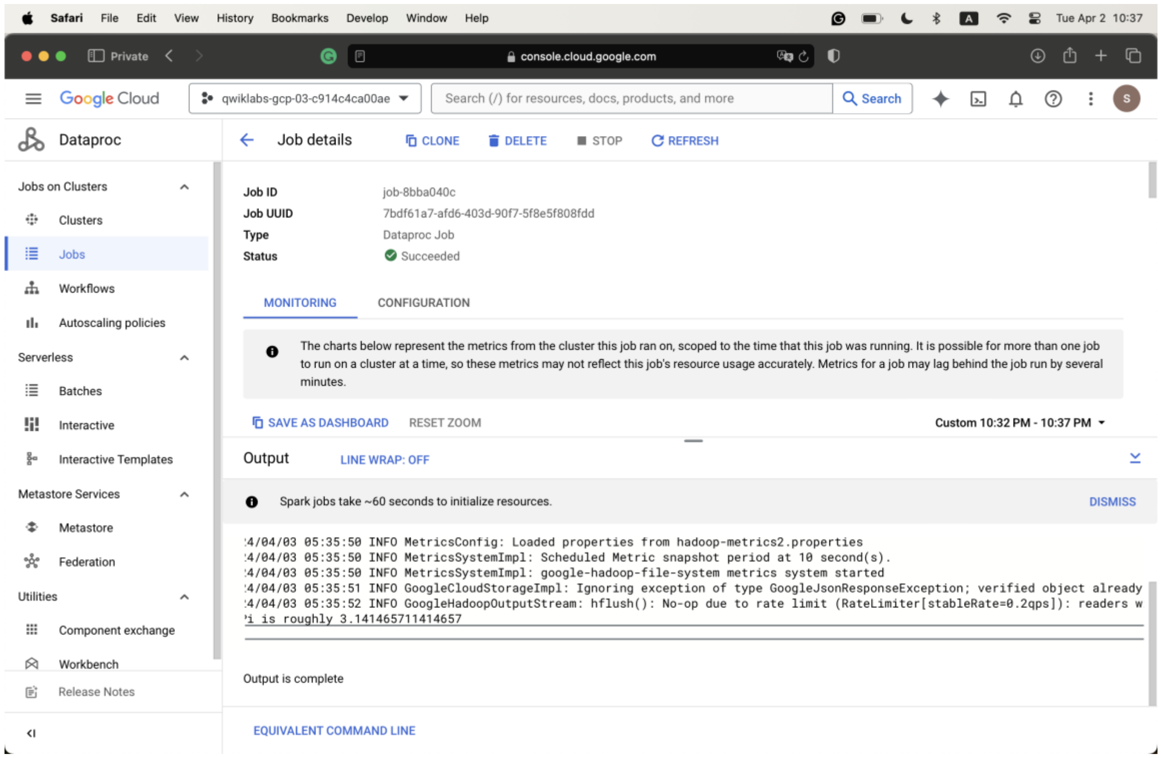
CPSC 324 01

28 March 2024

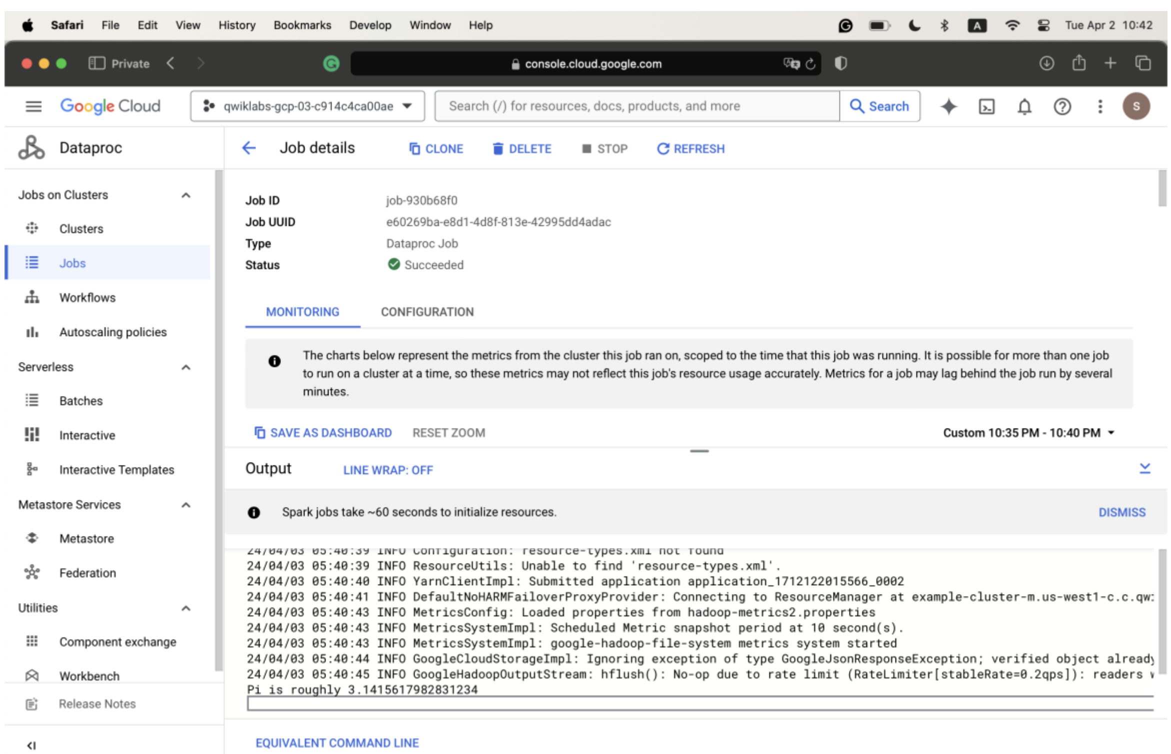
Homework 4

1.

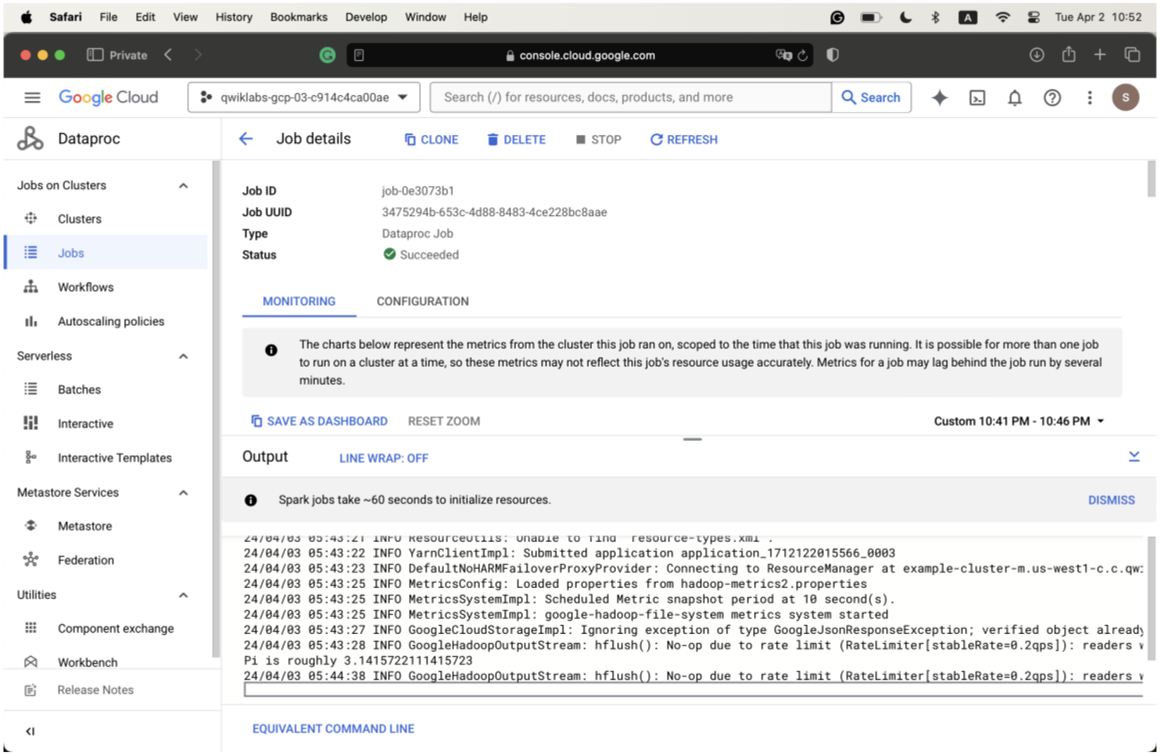
- Steps for creating a cluster and submitting the job
    - Confirm prerequisite
        - Navigation Menu -> API services -> Library -> Cloud Dataproc
    - Assign permission
        - Navigation Menu -> IAM & Admin -> IAM
        - Click on the `compute@developer.gserviceaccount.com` account
        - Select edit, and add "Storage Admin" role
    - Create a cluster
        - Navigation Menu -> Dataproc -> Cluster -> Cluster on Compute Engine
        - Remember to deselect "Configure all instances to have only internal IP addresses" in the Customer Cluster
    - Submitting a job
        - Under the "Job" tab, hit "Submit Job"
        - Choose the appropriate values
- Screenshots for 1,000

- 

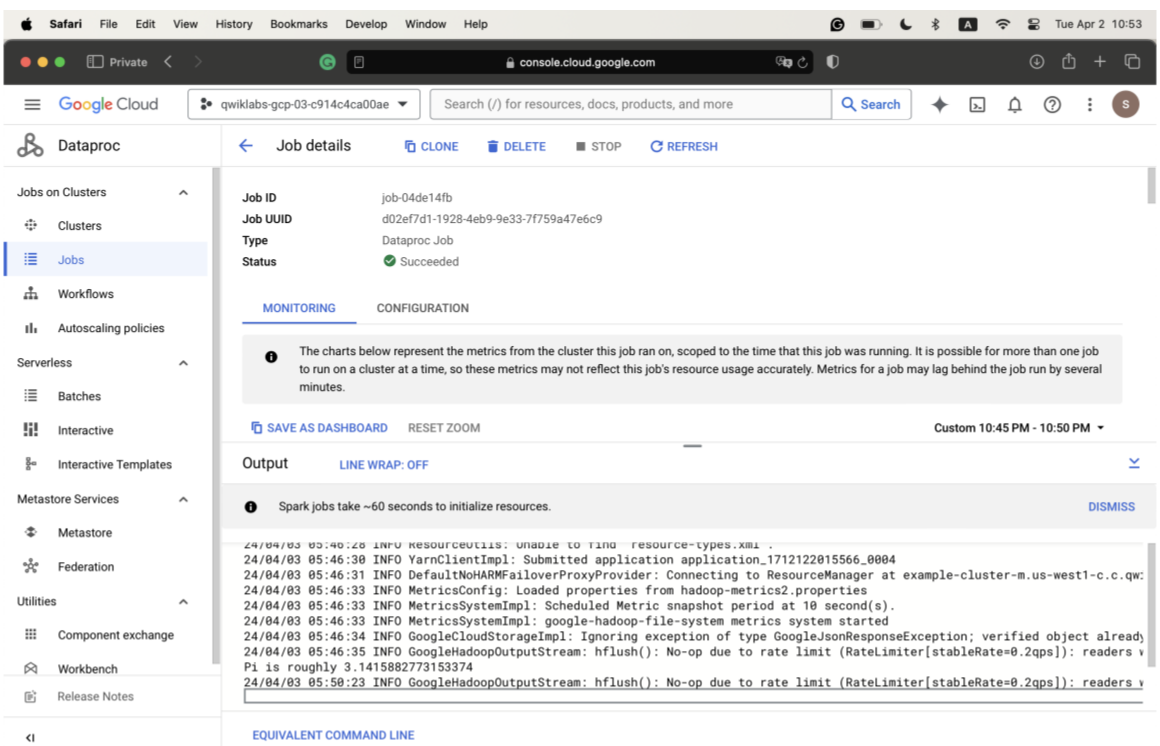
- Screenshot for 5,000

- 

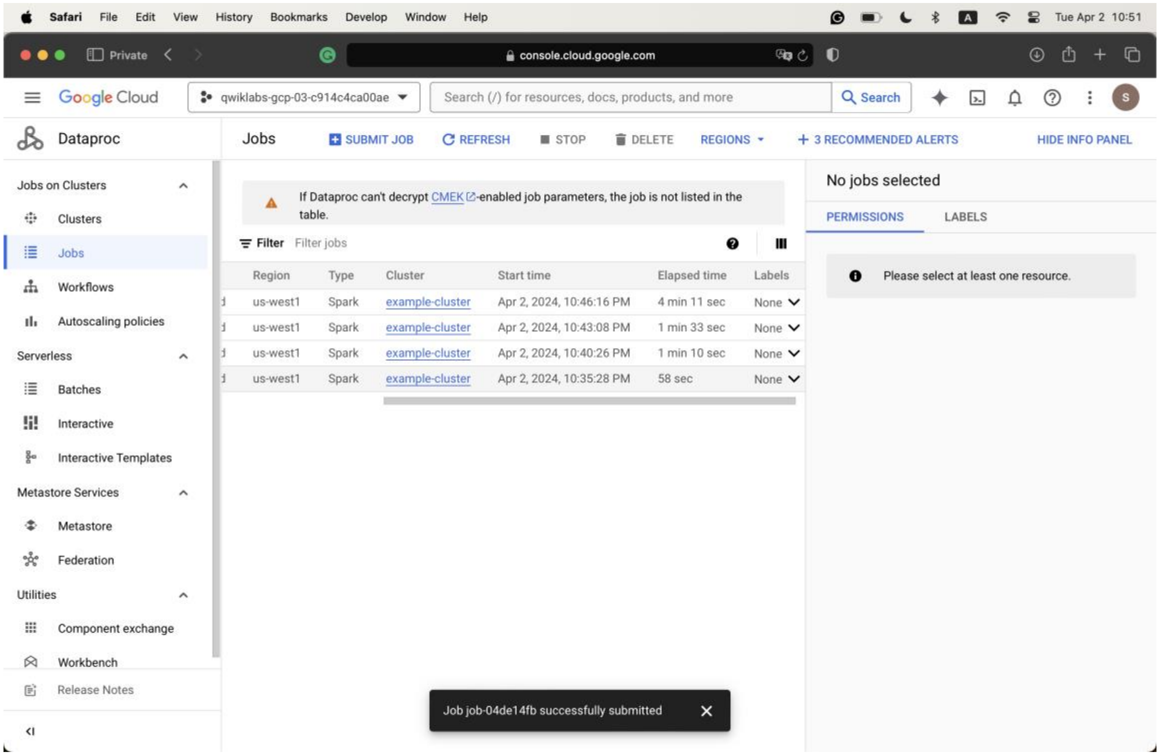
- Screenshot for 10,000

- 

- Screenshot for 100,000

- 

- Screenshot for amount of time

  - 

  - The more argument values it takes, the longer it is for the jobs to be accomplished.

2.

- Set region
  - gcloud config set dataproc/region us-west1
- Get PROJECT_ID and PROJECT_NUMBER variables
  - echo $PROJECT_ID
  - echo $PROJECT_NUMBER
  - PROJECT_ID=$(gcloud config get-value project) && \
  - gcloud config set project $PROJECT_ID
  - PROJECT_NUMBER=$(gcloud projects describe $PROJECT_ID --format='value(projectNumber)')
- Set Storage Admin privilage

- gcloud projects add-iam-policy-binding $PROJECT_ID \

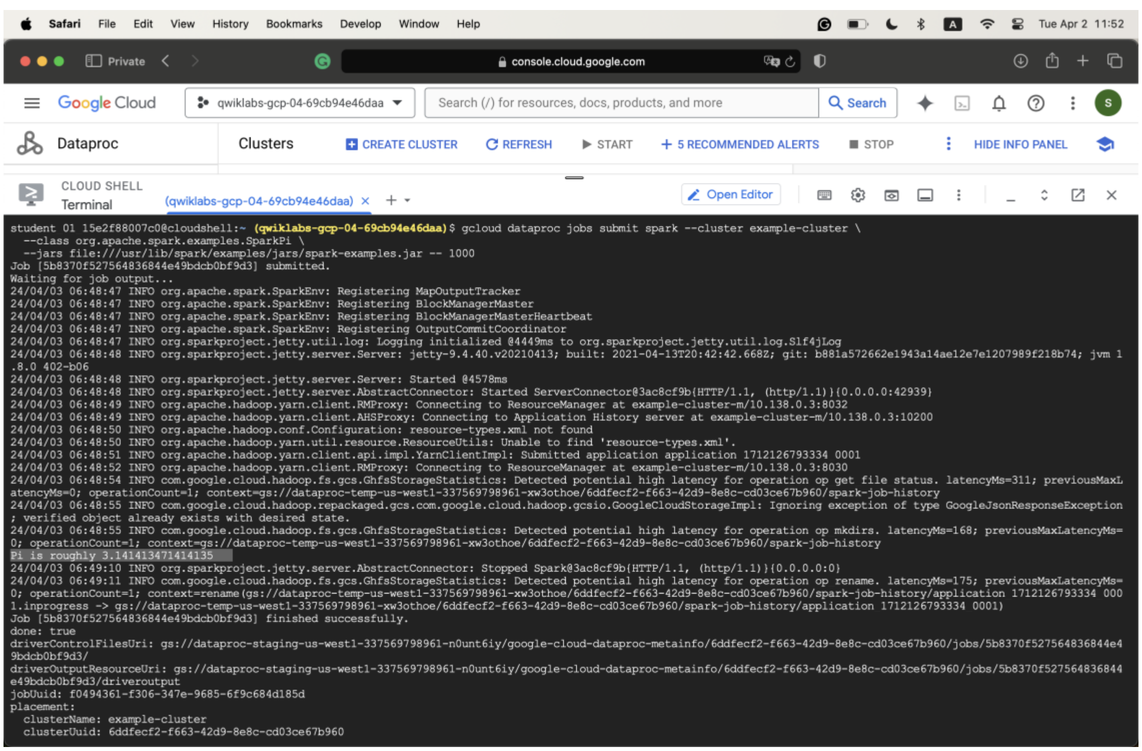
  --member=serviceAccount:$PROJECT_NUMBER-

  compute@developer.gserviceaccount.com \

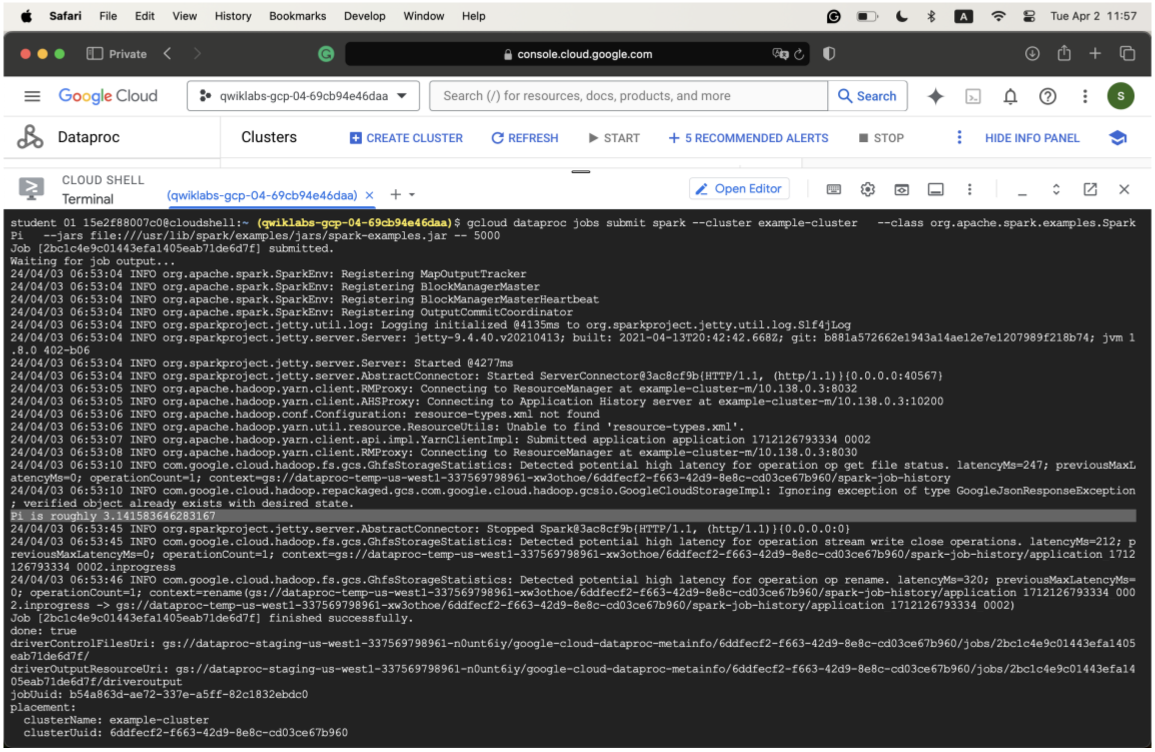
  --role=roles/storage.admin

- Create a cluster
  - gcloud dataproc clusters create example-cluster --worker-boot-disk-size 500 --worker-machine-type=e2-standard-4 --master-machine-type=e2-standard-4
- Run job
  - gcloud dataproc jobs submit spark --cluster example-cluster \

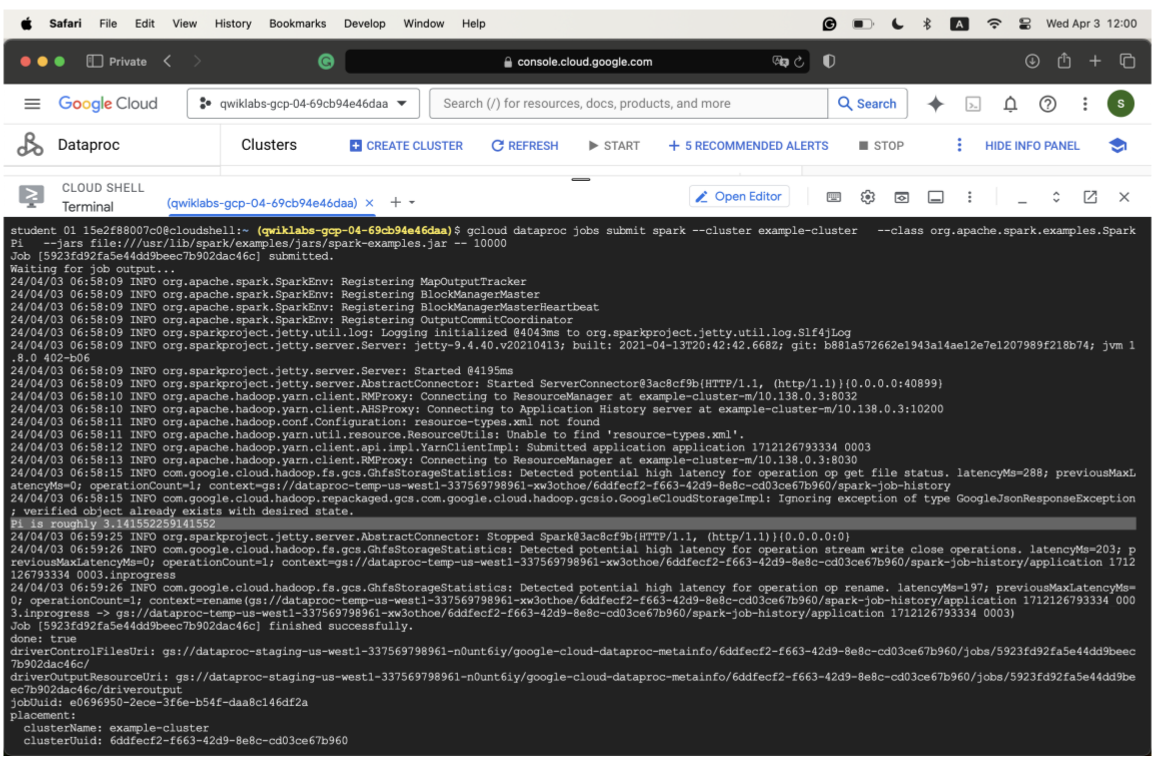
    --class org.apache.spark.examples.SparkPi \

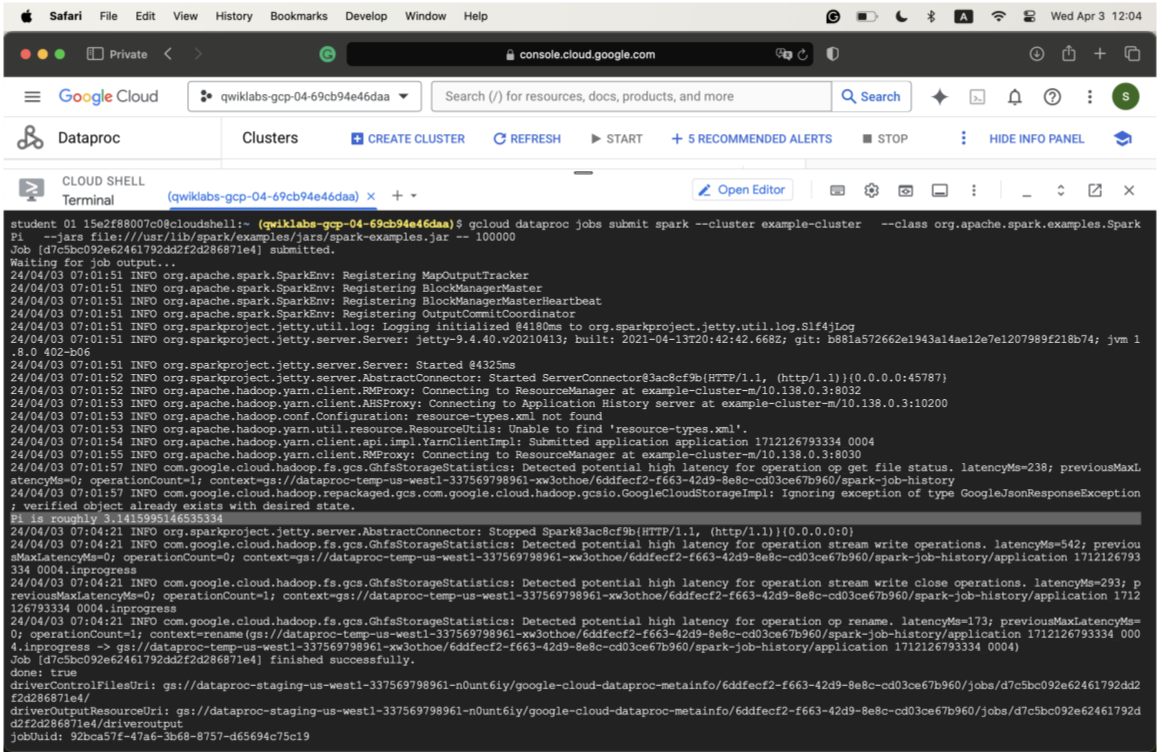
    --jars file:///usr/lib/spark/examples/jars/spark-examples.jar -- 1000
  - Note that the 1000 value depicts the Argument
- Screenshot of 1,000
  - 

- Screenshot of 5,000

  - 

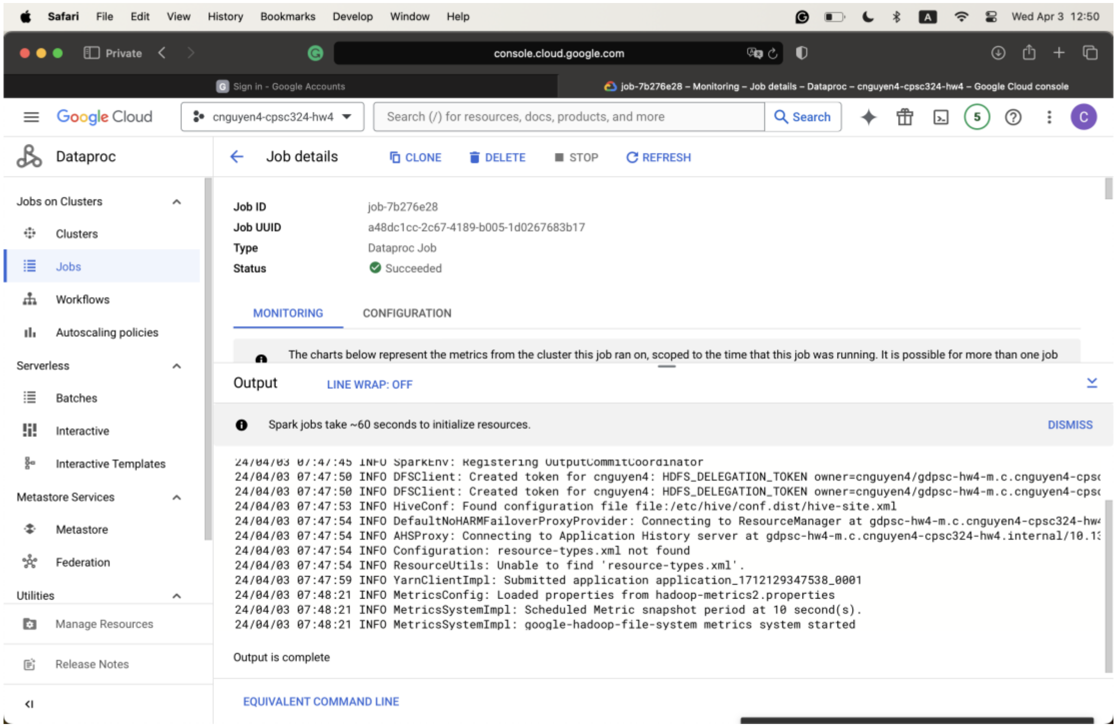
- Screenshot of 10,000

  - 

- Screenshot of 100,000
    - 

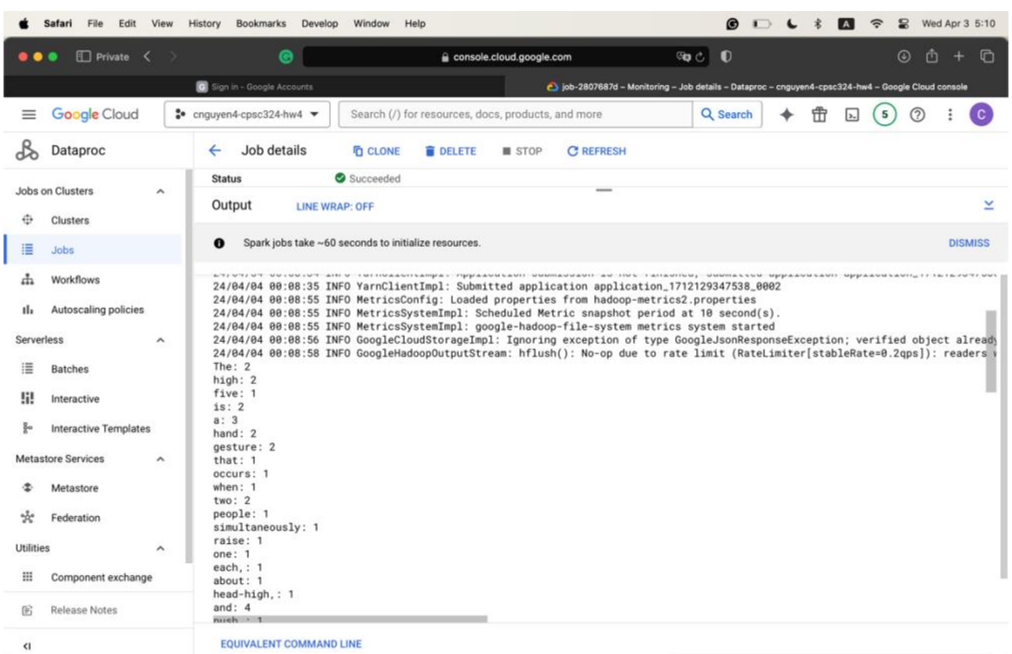
3.

- Steps
    - Enable APIs
    - Upload the script file to a bucket
    - Create a cluster
    - Submit job
- Screenshot

○ 

4.

- I don’t encounter any issues while running this question
- For this script, I noticed that it runs much faster
- Screenshot

○ 

5.

- Compared to other steps
    - When creating a cluster
        - Change the Versioning section to 2.1
        - Under “Component,” enable “Jupyter Notebook”
- Commands
    - Clone the git repo
        - git -C ~ clone https://github.com/GoogleCloudPlatform/training-data-analyst
    - Export the default Cloud Storage Bucket
        - export DP_STORAGE="gs://$(gcloud dataproc clusters describe sparktodp --region=europe-west1 --format=json | jq -r '.config.configBucket')"
    - Copy the sample notebooks into the Jupyter working folder
        - gcloud storage cp ~/training-data-analyst/quests/sparktobq/*.ipynb $DP_STORAGE/notebooks/jupyter
    - Copy the Python script to run as a Cloud Dataproc Job
        - gcloud storage cp gs://$PROJECT_ID/sparktodp/spark_analysis.py spark_analysis.py
    - Create a launch script
        - nano submit_onejob.sh
        - #!/bin/bash

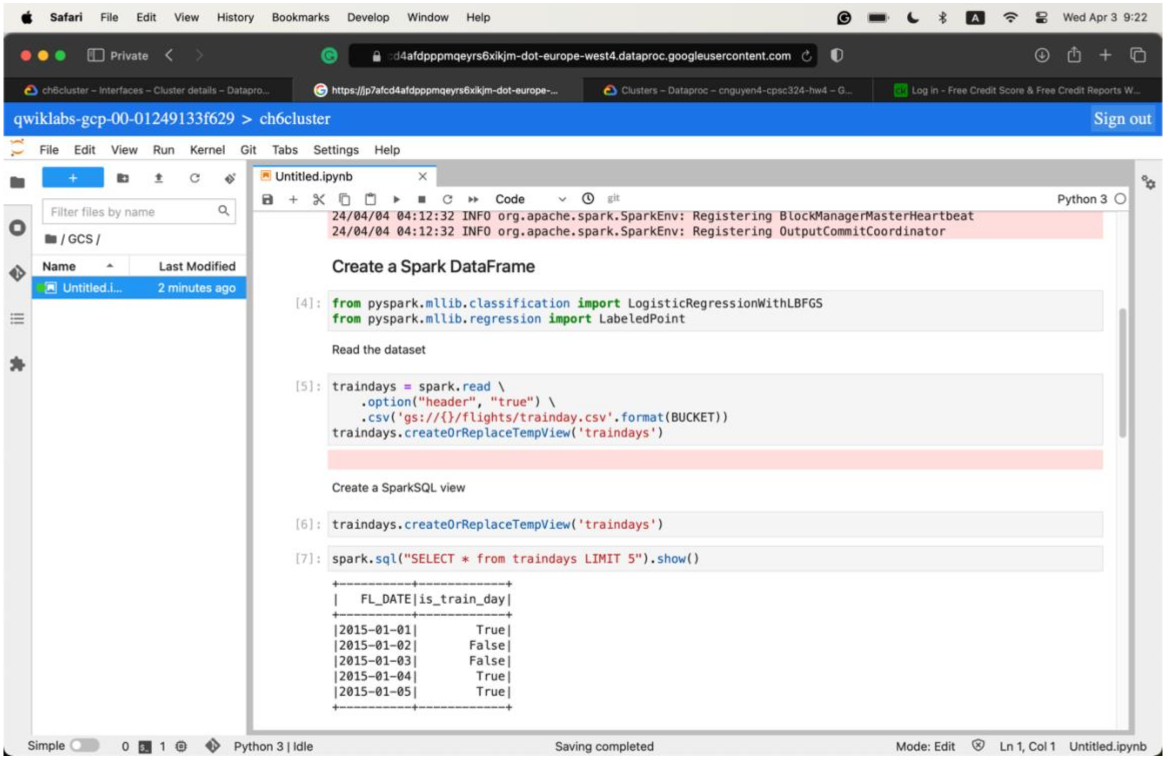
          gcloud dataproc jobs submit pyspark \

--cluster sparktodp \
--region europe-west1 \
spark_analysis.py \
-- --bucket=$1

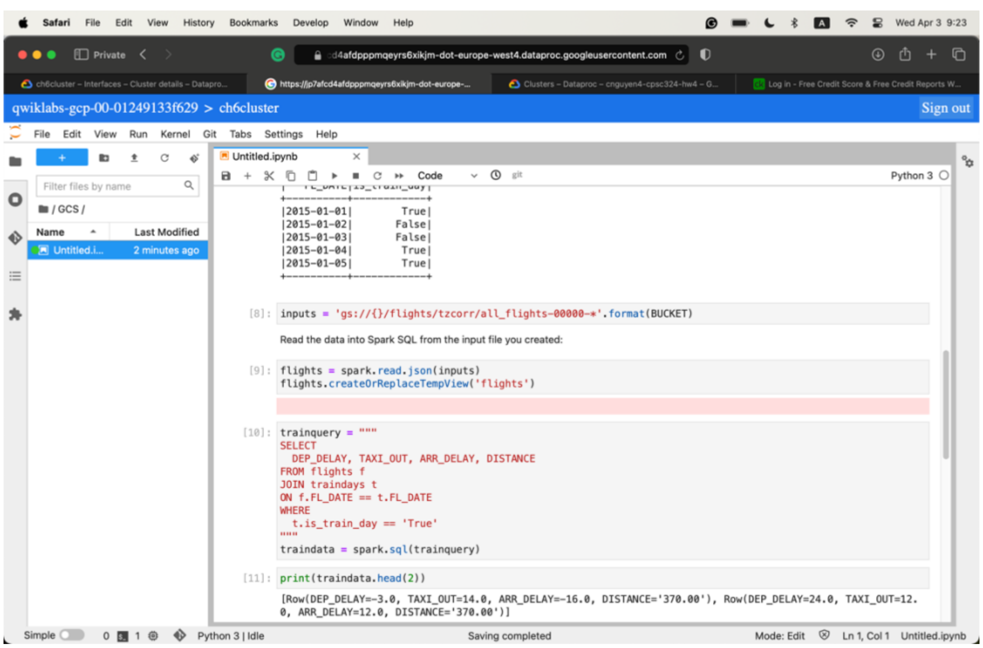
        - Press CTRL+X then Y and Enter key to exit and save.
    - Make the script executable
        - chmod +x submit_onejob.sh
    - Launch the PySpark Analysis job
        - ./submit_onejob.sh $PROJECT_ID
- Steps to open the Jupyter Notebook
    - Get into the cluster main menu, click on “Web Interfaces”
    - Under “Component Gateway,” choose “Jupyter”

6.

- Bucket name after creating the cluster
    - qwiklabs-gcp-00-01249133f629-dsongcp
- Screenshot showing the result of the first spark command

- 

- Hi

  - 

  - I think these attributes represent
    - Departure delay
    - Taxi out (time)

        - Arrival delay
        - Distance
- The result of running the statement *traindata.describe().show()*
    -

summary	DEP_DELAY	TAXI_OUT	ARR_DELAY	DISTANCE
count	46439	46422	46355	46936
mean	8.561769202609876	15.427685149282668	3.2853413871211306	916.0707133117437
stddev	30.752752455053308	8.427384168645757	32.98848343691196	591.9164453757172
min	-22.0	2.0	-77.0	1009.00
max	711.0	178.0	719.0	980.00

    - This table shows the summary statistics over the values of those four attributes.
- Data cleaning using SQL
    - The first query
        - This query joins the master flight list with the traindays set to find the flights that are used to train. It is used to calculate the summary statistics.
        - The query filters out dep_delay and arr_delay that are empty and is_train_day that is false.
        -

summary	DEP_DELAY	TAXI_OUT	ARR_DELAY	DISTANCE
count	46355	46355	46355	46355
mean	8.539531873584295	15.421507927947363	3.2853413871211306	917.660230827311
stddev	30.700034730525516	8.41130660980497	32.98848343691196	592.0960248192869
min	-22.0	2.0	-77.0	1009.00
max	711.0	178.0	719.0	980.00

    - The second query
        - This query joins the master flight list with the traindays set to find the flights that are used to train. It is used to calculate the summary statistics.

        - This query filters out those that are canceled and diverted as well as those who belong to is_train_day equal to true.

summary	DEP_DELAY	TAXI_OUT	ARR_DELAY	DISTANCE
count	46355	46355	46355	46355
mean	8.539531873584295	15.421507927947363	3.2853413871211306	917.660230827311
stddev	30.700034730525516	8.41130660980497	32.98848343691196	592.0960248192869
min	-22.0	2.0	-77.0	1009.00
max	711.0	178.0	719.0	980.00

    - I don’t personally like cleaning the dataset using SQL. I think one of the biggest problems is that the cleaning operations are not saved. Hence, every time we need to run the report again, we have to perform the cleaning operation another time.

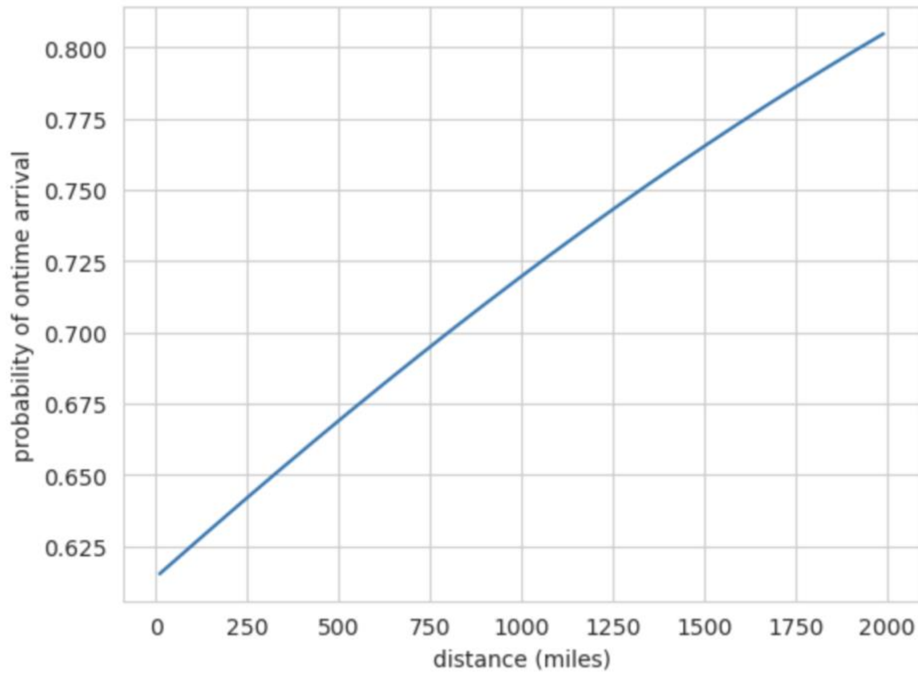
- The map function acts as a “proxy” to apply the to_example function on the traindata set. In this case, the to_example acts as a filter that picks out the flights that match the condition within the function. The result is saved to the example variable.

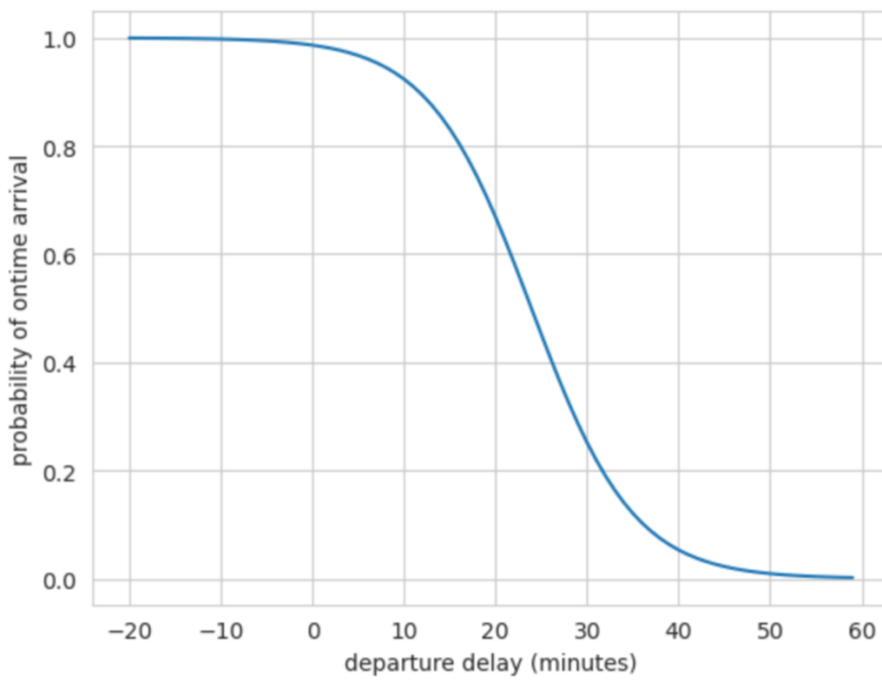
- The result of two predictions after clearing the threshold

    - ```
      lrmodel.clearThreshold()
      print(lrmodel.predict([6.0,12.0,594.0]))
      print(lrmodel.predict([36.0,12.0,594.0]))
      ```

      ```
      0.9520080900763146
      0.08390675828170738
      ```

- Two arrival probability graphs
      ```

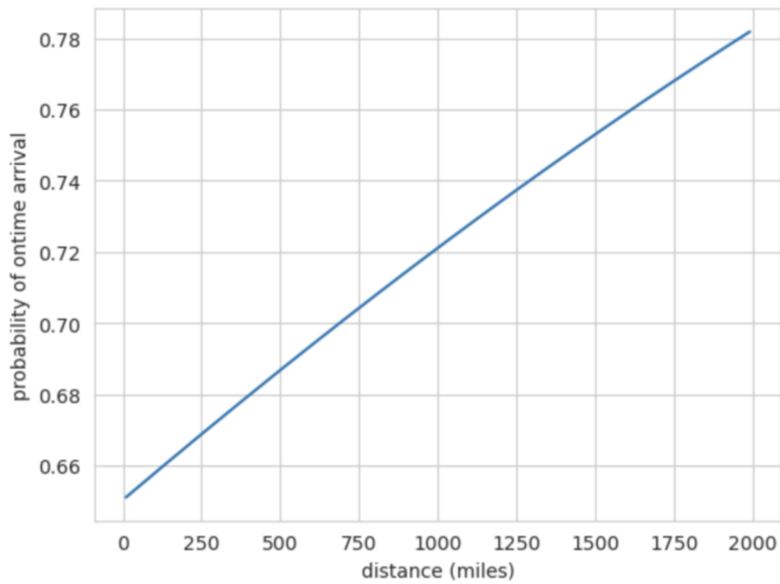
- 

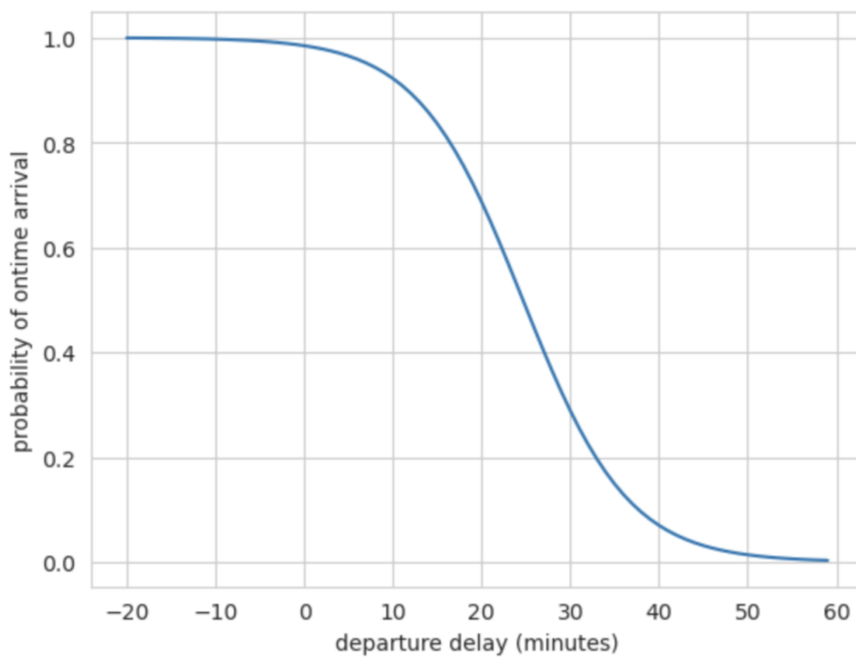
- 

- The map and filter functions
  - The map function maps the ground truth and the predicted label of the testing set.

- The filter function filters out results that are within the 65-75% range.

- Running on the full-flight dataset





7.

- RDD operations
  - Transformations: create a new dataset from an existing one

        - flatMap: process each element into multiple output elements
        - reduceByKey: aggregate values of a key using a specified function
    - Actions: return a value to the driver program after running a computation on the dataset
        - collect: returns all elements of the RDD to the driver
        - countByKey: counts the occurrences of each distinct key
- SQL functions in Spark
    - Offer a seamless and efficient way to perform complex data transformations and analyses. Somewhat similar to SQL
    - Interesting examples
        - Use SQL directly on DataFrame to filter and aggregate result
        - Use the window function in SparkSQL to perform computations over a range of rows related to the current rows.

8.

- I find the ML support in Spark impressively extensive. It offers a wide range of algorithms and utilities for machine-learning tasks involving scalability and integration with Spark's ecosystem. The system also simplifies the development process of machine learning systems.
- Basic statistics
    - This article gives a summary of basic statistic computation, including correlation, hypothesis testing with ChiSquareTest, and data summarization through Summarizer.
- Extracting, transforming, and selecting features

    - Spark supports many feature-selecting tools, such as TF-DIF and Word2Vec, for text analysis, which is good for developing generative AI models.
- Classification and regression
    - Spark ML provides frameworks for complex algorithms better suited to handle real-world data. It promises to be more efficient than the current Python library as it can solve performance issues in many datasets.
- Clustering
    - This is mainly supported by the k-means algorithm to group instances that are in common. This can be helpful for handling large data structures and relationships.
- ML Tuning
    - This one emphasizes the importance of feature selection used in machine learning algorithms and pipelines. This can be used to find the attributes that are impactful in order to improve performance and accuracy.

9.

- Steps
    - Check Project ID
        - echo ${GOOGLE_CLOUD_PROJECT}
    - If it is not set
        - export \
          GOOGLE_CLOUD_PROJECT=PROJECT-ID
    - Configure it to the current Project ID
        - gcloud config set project \

cnguyen4-cpsc324-hw4

- Set region
  - export REGION=us-west1
- Enable private access
  - gcloud compute networks subnets \

    update default \

    --region=${REGION} \

    --enable-private-ip-google-access
  - gcloud compute networks subnets \

    describe default \

    --region=${REGION} \

    --format="get(privateIpGoogleAccess)"
- Choose a bucket
  - export BUCKET=gs://cnguyen4-cpsc324-hw4-bucket1
  - export BUCKET=BUCKET

    gsutil mb -l ${REGION} \

    gs://${BUCKET}
    - To create a new bucket
- Choose a dataset
  - export DATASET=DATASET

    bq --location=${REGION} mk -d \

    ${DATASET}
- Set a name for the persistent history

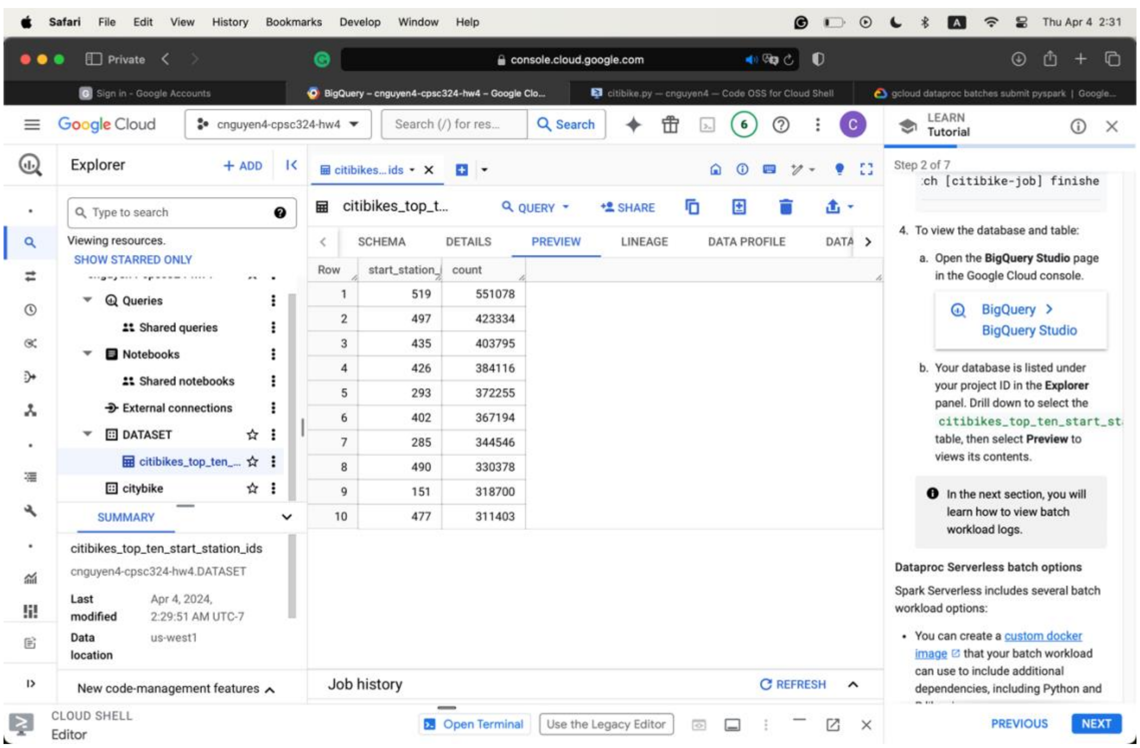
- PHS_CLUSTER_NAME=cnguyen4-cpsc324-hw4-cluster1
- Create a new cluster
  - gcloud dataproc clusters create \
    ${PHS_CLUSTER_NAME} \
    --region=${REGION} \
    --single-node \
    --enable-component-gateway \
    --
    properties=spark:spark.history.fs.logDirectory=gs://${BUCKET}/phs/
    */spark-job-history
- Set a batch workload
  - BATCH_NAME=batch1
- Submit job
  - gcloud dataproc batches submit \
    pyspark citibike.py \
    --batch=${BATCH_NAME} \
    --region=${REGION} \
    --deps-bucket=gs://${BUCKET} \
    --version=1.1 \
    --history-server-
    cluster=projects/${GOOGLE_CLOUD_PROJECT}/regions/${REGIO
    N}/clusters/${PHS_CLUSTER_NAME} \
    -- ${DATASET}

- Statement to create the data frame
  - df = spark.read.format("bigquery").load(table)
- Top ten data frame SQL equivalent:
  - SELECT start_station_id, COUNT(*) AS count

    FROM df

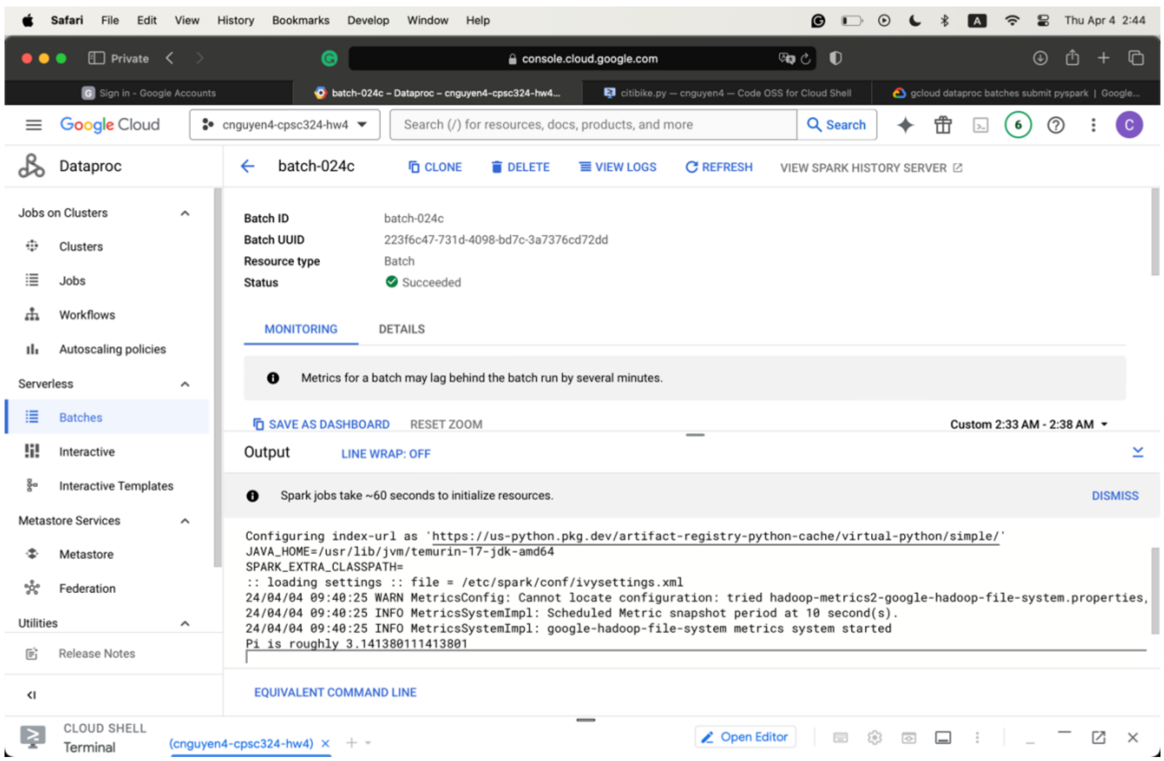
    GROUP BY start_station_id

    WHERE start_station_id IS NOT NULL

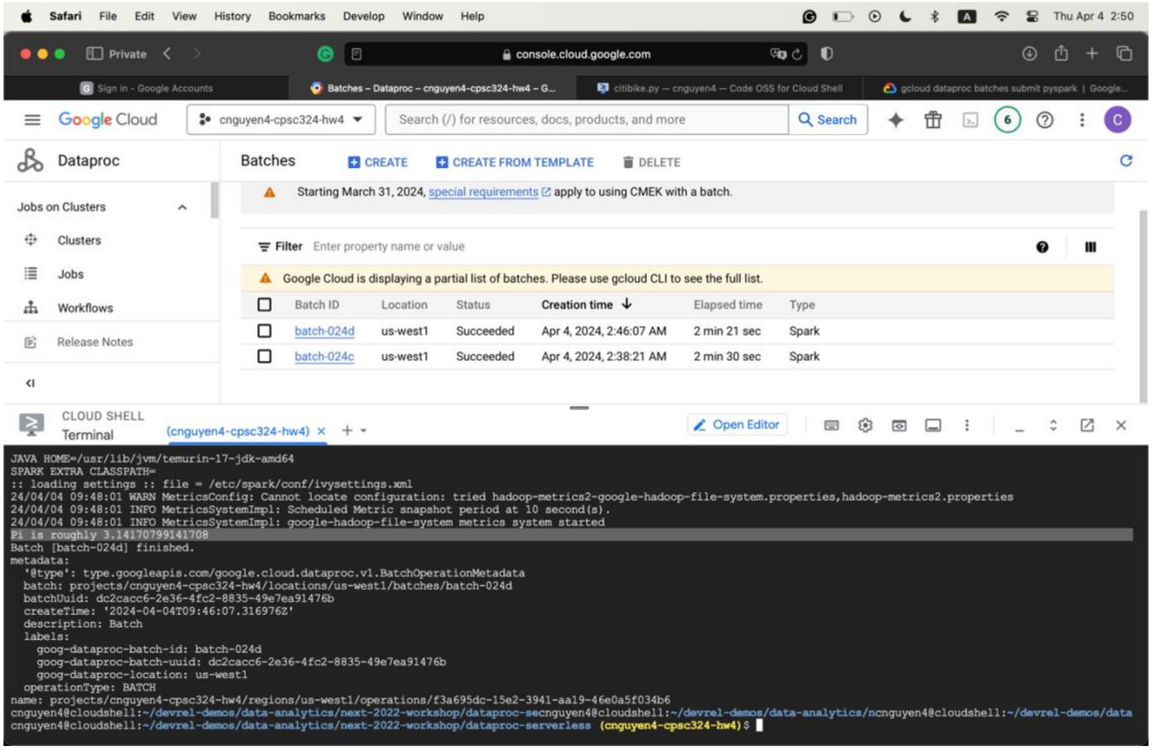
    ORDER BY count ASC LIMIT 10;
- Screenshot of BigQuery Studio
  - 

- I noticed that the command to submit the job was weird. It returned an HTTP 400 error without telling me exactly what happened. It turned out that the BATCH_NAME only allows (a-z) and numbers, and I did not know that. I also hard-coded the command.

10.

- Equivalent command line
    - gcloud dataproc batches submit --project cnguyen4-cpsc324-hw4 --region us-west1 spark --batch batch-024c --class org.apache.spark.examples.SparkPi --version 2.2 --jars file:///usr/lib/spark/examples/jars/spark-examples.jar --subnet default -- 1000
- Screenshot of resulting pi computation
    - 

- Running again but using Cloud Shell

- 

11.

- How is it different?
    - The wordcount.py of the other approaches uses the Spark functions liberally to match and count the word.
    - The wordcount.py, on the other hand, focuses on using SQL to find what the word count process is like.
- Paste a screenshot

-