

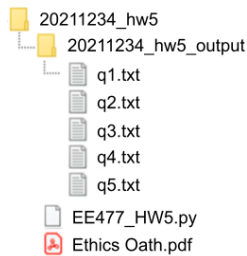
# EE477 Database and Big Data System, Spring 2021

## HW5\*

Due date: 06/20/2021 (11:59pm)

**Submission instructions:** Use [KAIST KLMS](#) to submit your homeworks. Your submission should be one gzipped tar file whose name is `YourStudentID_hw5.tar.gz`. For example, if your student ID is 20211234, and it is for homework #5, please name the file as `20211234_hw5.tar.gz`. You can also use these extensions: tar, gz, zip, tar.zip. Do not use other options not mentioned here.

Your zip file should contain three things; one python file (`EE477_HW5.py`), one folder named `YourStudentID_hw5_output`(put all output files `q1.txt`, `q2.txt`, ..., `q5.txt`), and the Ethics Oath pdf file. Do not include Korean letters in any file name or directory name when you submit.



If you do not follow this format, we will **deduct** 1 point of total score per mistake.

*Submitting code:* Each problem is accompanied by a programming part. Put all the code for each question into a single file. Good coding style (including comments) will be one criterion for grading. Please make sure your code is well structured and has descriptive comments.

*Ethics Oath:* For every homework submission, please fill out and submit the **PDF** version of [this document](#) that pledges your honor that you did not violate any ethics rules required by [this course](#) and KAIST. You can either scan a printed version into a PDF file or make the Word document into a PDF file after filling it out. Please sign on the document and submit it along with your other files.

Discussions with other people are permitted and encouraged. However, when the time comes to write your solution, such discussions (except with course staff members) are

---

\*Material adapted from Google Cloud Manual and University of Washington CSE344.

no longer appropriate: you must write down your own solutions independently. If you received any help, you must specify on the top of your written homework any individuals from whom you received help, and the nature of the help that you received. *Do not, under any circumstances, copy another person's solution.* We check all submissions for plagiarism and take any violations seriously.

## 1 Dataset Details

In this homework, you will be writing Spark and Spark SQL code using Google Cloud Platform.

We will be using flights dataset (`flights_full`) dump from the [US Bureau of Transportation Statistics](#), which consists of information about all domestic US flights from 1987 to 2011 or so.

The data is stored in a columnar format called [Parquet](#) and is available publicly on Cloud Storage. The data is around 4GB compressed (79GB uncompressed), and contains 162,212,419 rows. Hence you probably don't want to be crunching on this dataset using your laptop! Each row contains 109 different attributes (see [5](#) Appendix of the pdf for details)

To help you debug, we provide a subset of the data (`flights_small`). This dataset is a dump of all the flights that happened on November 4, 2010. We strongly encourage you to run your code using this small dataset before trying it out on full dataset.

- **Dataset Directory (Publicly Accessable Bucket)**

Full dataset : `gs://flight_data_kaistee/flights_full`

Subset of the data for debugging : `gs://flight_data_kaistee/flights_small`

## 2 Download Starter Code and Install Google Cloud SDK

- Download `EE477_HW5.py` from [KLMS](#).
- Install **Google Cloud SDK** (if you did not install)

In this homework, you can use Cloud Shell, but it is quite difficult and inconvenient to use GCP Dataproc. Therefore, we recommend you to use Google Cloud SDK.

You can install Google Cloud SDK on your local machine and connect to your google account by following instructions in <https://cloud.google.com/sdk/docs/>.

And then, you can command on your local machine, not Cloud Shell.

### 3 Run Code on Dataproc

**Cloud Dataproc** is a managed **Spark** and **Hadoop** service that lets you take advantage of open source data tools for batch processing, querying, streaming, and machine learning. Cloud Dataproc automation helps you create clusters quickly, manage them easily. With less time and money spent on administration, you can focus on your jobs and your data. Also, you don't need to install python or spark.

If you want to know more detail : [Cloud Dataproc](#)

We will use **Cloud Dataproc** to deploy our code. Follow these steps to do so.

#### 1) Create a Cluster

The image shows the Google Cloud Dataproc console. The top section is a 'Cluster' overview card with a 'CREATE CLUSTER' button. Below this is the 'Create Cluster' configuration page. The 'Name' field is 'cluster-0fbc'. The 'Location' is set to 'Region: asia-northeast3' and 'Zone: asia-northeast3-c'. The 'Cluster type' is 'Standard (1 master, N workers)'. The 'Autoscaling' policy is 'None'. The 'Versioning' section is empty. The 'Master node' section shows 'Machine family: GENERAL-PURPOSE', 'Series: N1', and 'Machine type: Custom'. The 'Worker nodes' section shows 'Machine family: GENERAL-PURPOSE', 'Series: N1', and 'Machine type: n1-standard-2 (2 vCPU, 7.5 GB memory)'. The 'Number of worker nodes' is 4. The 'Primary disk size' is 500 GB and the 'Primary disk type' is 'Standard Persistent Disk'. The 'Number of local SSDs' is 0. Below the configuration page is a table of clusters. The first cluster is 'cluster-0fbc' with status 'Running' and region 'asia-northeast3'. To the right is a 'Browser' section showing a list of buckets. The first bucket is 'dataproc-staging-asia-northeast3-921563987094-yc0yqgyk' with a size of 'M'. The second bucket is 'dataproc-temp-asia-northeast3-921563987094-hniuinzg' with a size of 'M'.

**Cluster Configuration:**

- Name: cluster-0fbc
- Location: Region: asia-northeast3, Zone: asia-northeast3-c
- Cluster type: Standard (1 master, N workers)
- Autoscaling: Policy: None
- Versioning: (empty)
- Master node: Machine family: GENERAL-PURPOSE, Series: N1, Machine type: Custom
- Worker nodes: Machine family: GENERAL-PURPOSE, Series: N1, Machine type: n1-standard-2 (2 vCPU, 7.5 GB memory), Number of worker nodes: 4
- Primary disk size: 500 GB, Primary disk type: Standard Persistent Disk
- Number of local SSDs: 0

**Clusters List:**

Name	Status	Region	Zone
cluster-0fbc	Running	asia-northeast3	asia-northeast3-c

**Buckets List:**

Name	Size
dataproc-staging-asia-northeast3-921563987094-yc0yqgyk	M
dataproc-temp-asia-northeast3-921563987094-hniuinzg	M

## 2) Run python code on Dataproc using Google Cloud SDK

Command on your **LOCAL** machine (NOT Cloud Shell) :

```
gcloud dataproc jobs submit pyspark --cluster YOUR_CLUSTER_NAME
--region=asia-northeast3 PYTHON_FILE_DIRECTORY
-- DATASET_DIRECTORY OUTPUT_DIRECTORY
```

### Example

(a) Subset of dataset

```
gcloud dataproc jobs submit pyspark --cluster=cluster-0fbe
--region=asia-northeast3 ./EE477_HW5.py
-- gs://flight_data_kaistee/flights_full
gs://dataproc-staging-asia-northeast3-921563987094-yc0yqgyk/q1
```

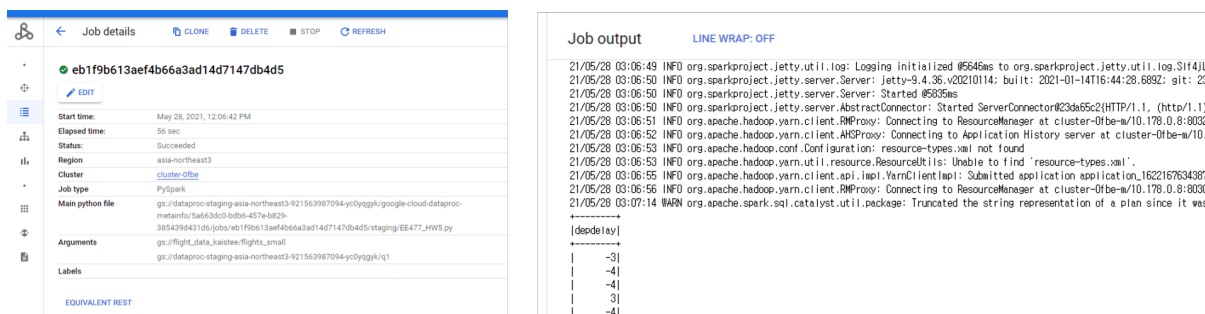
(b) Whole dataset

```
gcloud dataproc jobs submit pyspark --cluster=cluster-0fbe
--region=asia-northeast3 ./EE477_HW5.py
-- gs://flight_data_kaistee/flights_full
gs://dataproc-staging-asia-northeast3-921563987094-yc0yqgyk/q1
```

CAUTION : In Spark, the output text file will be saved in OUTPUT\_DIRECTORY, and there should **NOT** be the directory whose name is same with OUTPUT\_DIRECTORY.

You can not use local directory as OUTPUT\_DIRECTORY, so we recommend you to use the storage for cluster.

## 3) Check current status of the job (Dataproc)

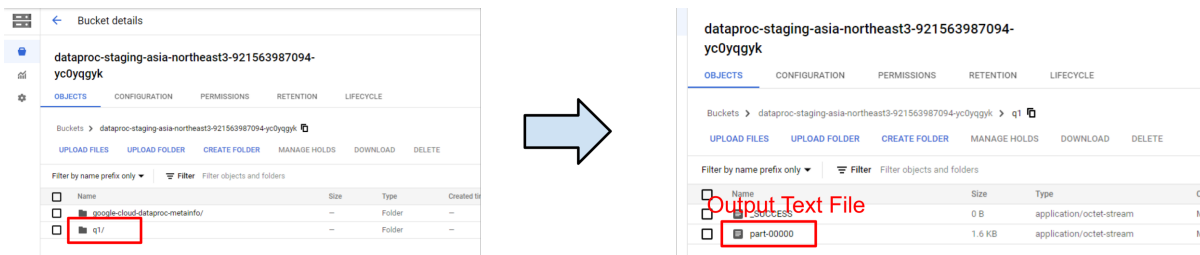


The screenshot shows the Google Cloud Dataproc interface. On the left, the 'Job details' panel for job 'eb1f9b613aef4b6a3ad14d7147db4d5' is displayed. It shows the job is 'Succeeded' and provides details like start time, elapsed time, region, cluster, and the main python file. On the right, the 'Job output' panel shows the Spark logs, including information about the Spark version, JVM options, and the start of the application. The logs indicate that the application is running successfully and has submitted to the ResourceManager.

```
gcloud dataproc jobs submit pyspark --cluster=hw5cluster --
region=asia-northeast3 /Users/yangheewon/hw5/HW5.py -- gs://
flight_data_kaistee/flights_small gs://dataproc-staging-asia-
northeast3-701898616707-blhsrpbo/q1
```

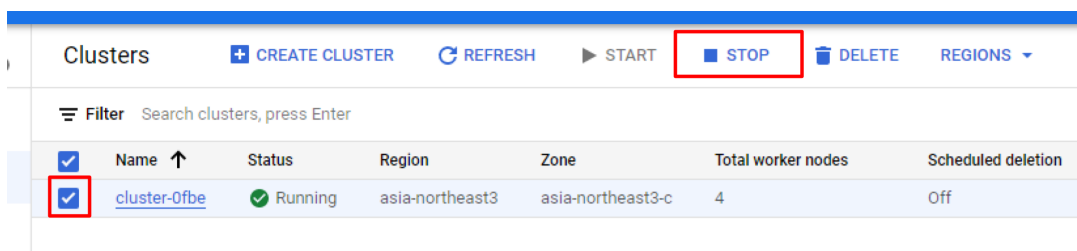
```
gcloud dataproc jobs submit pyspark --cluster=hw5cluster --
region=asia-northeast3 /Users/yangheewon/hw5/HW5.py -- gs://
flight_data_kaistee/flights_full gs://dataproc-staging-asia-
northeast3-701898616707-blhsrpbo/q1
```

#### 4) Download output text file (CloudStorage)



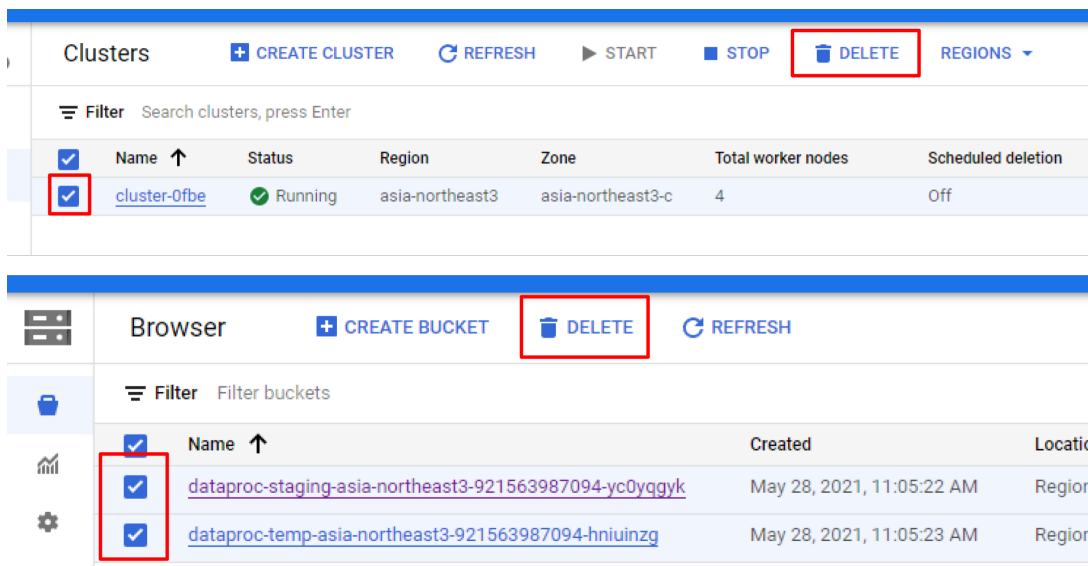
You should change the name of output text file (For example, `part-00000` -> `q1.txt`) and download it.

#### 5) After running code, stop the cluster



If you don't use the cluster, you should **STOP** the cluster. If you do not stop it, it will be charged consistently.

#### 6) After submission, remove the cluster and storage



If you finish the project, you should **DELETE** the cluster and storage of that.

## 4 Task : Using Spark SQL and RDD API

We have created empty method bodies for each of the questions below (Q1, Q2, etc). Please don't change any of the method signatures. You are free to define extra methods if you need to. Run all of the following problems on the full dataset. Except for problem 1, you should use the pyspark RDD API (and optionally reduceByKey from RDD) to implement your solution. Save the output from Dataproc to q1.txt, q2.txt, ...

1. Select all flights that leave from "Seattle, WA", and return the destination city names. Only return each city name once. Implement this using the Dataset API and writing a SQL query. This should be trivial and is intended for you to learn about the Dataset API. You can either use the functional form (i.e., `join(...)`, `select(...)`) or write a SQL query using `SparkSession.sql`. Check the corresponding Spark Javadoc for the parameters and return values. Save the Dataproc output as q1.txt.

**Result Size: 79 rows (50 rows on the small dataset), less than 5 min on Dataproc**

**Hint:** If you decide to write a SQL query, note that you can use single quotes inside your query for string literals (e.g., 'Seattle'). Also, it does not matter what you name the output column as, since that information is not dumped to the output.

2. Implement the same query as above, but use the RDD API. You can convert a Dataset to a RDD by calling `RDD`, which we did for you in the skeleton code. Save the Dataproc output as q2.txt.

**Result Size: 79 rows (50 rows on the small dataset), less than 21 mins on Dataproc**

3. Find the number of non-cancelled flights per month that departs from each city, return the results in a RDD where the key is a pair (i.e., a Tuple2 object), consisting of a String for the departing city name, and an Integer for the month. The value should be the number of non-cancelled flights. Save the Dataproc output as q3.txt.

**Result Size: 4383 rows (281 rows on the small dataset), less than 25 mins on Dataproc**

4. Find the name of the city that is connected to the most number of other cities within a single hop flight. Return the result as a pair that consists of a String for the city name, and an Integer for the total number of cities connected within a single hop. Save the Dataproc output as q4.txt.

**Result Size: 1 row, less than 24 mins on Dataproc**

**Hint:** You can assume the (origin city A, dest city B) pair on same row is connected within single hop and means  $A \rightarrow B$ , not  $A \leftrightarrow B$ .

5. Compute the average delay from all departing flights for each city. Flights with null delay values (due to cancellation or otherwise) should not be counted. Return the results in a RDD where the key is a String for the city name, and the value is a Double for the average delay in minutes. Save the Dataproc output as q5.txt.  
**Result Size: 383 rows (281 rows on the small dataset), less than 25 mins on Dataproc**

**Hint:** if some attributes have null values in table, those values are represented by “None” in Python

## 5 Appendix : Attributes of Dataset

We list all the fields in the input data file for your reference root

```
|-- year: integer (nullable = true)    // index 0
|-- quarter: integer (nullable = true)
|-- month: integer (nullable = true)
|-- dayofmonth: integer (nullable = true)
|-- dayofweek: integer (nullable = true)
|-- flightdate: string (nullable = true)
|-- uniquecarrier: string (nullable = true)
|-- airlineid: integer (nullable = true)
|-- carrier: string (nullable = true)
|-- tailnum: string (nullable = true)
|-- flightnum: integer (nullable = true)
|-- originairportid: integer (nullable = true)
|-- originairportseqid: integer (nullable = true)
|-- origincitymarketid: integer (nullable = true)
|-- origin: string (nullable = true)    // airport short name
|-- origincityname: string (nullable = true) // e.g., Seattle, WA
|-- originstate: string (nullable = true)
|-- originstatefips: integer (nullable = true)
|-- originstatename: string (nullable = true)
|-- originwac: integer (nullable = true)
|-- destairportid: integer (nullable = true)
|-- destairportseqid: integer (nullable = true)
|-- destcitymarketid: integer (nullable = true)
|-- dest: string (nullable = true)
|-- destcityname: string (nullable = true)
|-- deststate: string (nullable = true)
|-- deststatefips: integer (nullable = true)
|-- deststatename: string (nullable = true)
|-- destwac: integer (nullable = true)
|-- crsdeptime: integer (nullable = true)
|-- deptime: integer (nullable = true)
|-- depdelay: integer (nullable = true)
```

```

|-- depdelayminutes: integer (nullable = true)
|-- depdel15: integer (nullable = true)
|-- departedelaygroups: integer (nullable = true)
|-- deptimeblk: integer (nullable = true)
|-- taxiout: integer (nullable = true)
|-- wheelsoff: integer (nullable = true)
|-- wheelson: integer (nullable = true)
|-- taxiin: integer (nullable = true)
|-- crsarrrtime: integer (nullable = true)
|-- arrtime: integer (nullable = true)
|-- arrdelay: integer (nullable = true)
|-- arrdelayminutes: integer (nullable = true)
|-- arrdel15: integer (nullable = true)
|-- arrivaldelaygroups: integer (nullable = true)
|-- arrtimeblk: string (nullable = true)
|-- cancelled: integer (nullable = true)
|-- cancellationcode: integer (nullable = true)
|-- diverted: integer (nullable = true)
|-- crselapsedtime: integer (nullable = true)
|-- actualelapsedtime: integer (nullable = true)
|-- airtime: integer (nullable = true)
|-- flights: integer (nullable = true)
|-- distance: integer (nullable = true)
|-- distancegroup: integer (nullable = true)
|-- carrierdelay: integer (nullable = true)
|-- weatherdelay: integer (nullable = true)
|-- nasdelay: integer (nullable = true)
|-- securitydelay: integer (nullable = true)
|-- lateaircraftdelay: integer (nullable = true)
|-- firstdeptime: integer (nullable = true)
|-- totaladdgtime: integer (nullable = true)
|-- longestaddgtime: integer (nullable = true)
|-- divairportlandings: integer (nullable = true)
|-- divreacheddest: integer (nullable = true)
|-- divactualelapsedtime: integer (nullable = true)
|-- divarrdelay: integer (nullable = true)
|-- divdistance: integer (nullable = true)
|-- div1airport: integer (nullable = true)
|-- div1airportid: integer (nullable = true)
|-- div1airportseqid: integer (nullable = true)
|-- div1wheelson: integer (nullable = true)
|-- div1totalgtime: integer (nullable = true)
|-- div1longestgtime: integer (nullable = true)
|-- div1wheelsoff: integer (nullable = true)
|-- div1tailnum: integer (nullable = true)
|-- div2airport: integer (nullable = true)
|-- div2airportid: integer (nullable = true)
|-- div2airportseqid: integer (nullable = true)

```



```
|-- div2wheelson: integer (nullable = true)
|-- div2totalgtime: integer (nullable = true)
|-- div2longestgtime: integer (nullable = true)
|-- div2wheelsoff: integer (nullable = true)
|-- div2tailnum: integer (nullable = true)
|-- div3airport: integer (nullable = true)
|-- div3airportid: integer (nullable = true)
|-- div3airportseqid: integer (nullable = true)
|-- div3wheelson: integer (nullable = true)
|-- div3totalgtime: integer (nullable = true)
|-- div3longestgtime: integer (nullable = true)
|-- div3wheelsoff: integer (nullable = true)
|-- div3tailnum: integer (nullable = true)
|-- div4airport: integer (nullable = true)
|-- div4airportid: integer (nullable = true)
|-- div4airportseqid: integer (nullable = true)
|-- div4wheelson: integer (nullable = true)
|-- div4totalgtime: integer (nullable = true)
|-- div4longestgtime: integer (nullable = true)
|-- div4wheelsoff: integer (nullable = true)
|-- div4tailnum: integer (nullable = true)
|-- div5airport: integer (nullable = true)
|-- div5airportid: integer (nullable = true)
|-- div5airportseqid: integer (nullable = true)
|-- div5wheelson: integer (nullable = true)
|-- div5totalgtime: integer (nullable = true)
|-- div5longestgtime: integer (nullable = true)
|-- div5wheelsoff: integer (nullable = true)
|-- div5tailnum: integer (nullable = true)
```