CSC 643 – Big Data & Web Intelligence

Team1.cache(Ahmad, Sean)

*Project 4*

Submission Date: December 11, 2019

Table of Contents

[**Task Ranking** 3](#_Toc26825633)

[**Introduction** 4](#_Toc26825634)

[**Starter Kits** 5](#_Toc26825635)

[**1.** **Loading data into Databricks** 5](#_Toc26825636)

[**2.** **Creating cluster in Databricks** 7](#_Toc26825637)

[**3.** **Creating notebook in Databricks** 8](#_Toc26825638)

[**Answers:**  9](#_Toc26825639)

[**Part1: Use Apache Spark Core to answer the following queries** 9](#_Toc26825640)

[**Part2: Use Apache Spark Core to answer the following queries** 10](#_Toc26825642)

[**Part3: Use Apache Spark Core and Spark SQL to answer the following queries.** 16](#_Toc26825644)

[**Part4: use Data Aggregation functions and compare numeric values using Apache Spark Core** 18](#_Toc26825647)

[**References:** 21](#_Toc26825648)

# **Task Ranking**

* Member 1: **Ahmad Alshawaf**

Handled 50% of the project from start to complete

* + - * Task 1: analyzing data
      * Task 2: solving part 1
      * Task 3: solving part 2
      * Task 4: solving part 3
      * Task 5: solving part 4
      * Task 6: organizing report
* Member 2: **Sean Sothey**

Handled 50% of the project from start to complete

* + - * Task 1: analyzing data
      * Task 2: solving part 1
      * Task 3: solving part 2
      * Task 4: solving part 3
      * Task 5: solving part 4
      * Task 6: organizing report

# **Introduction**

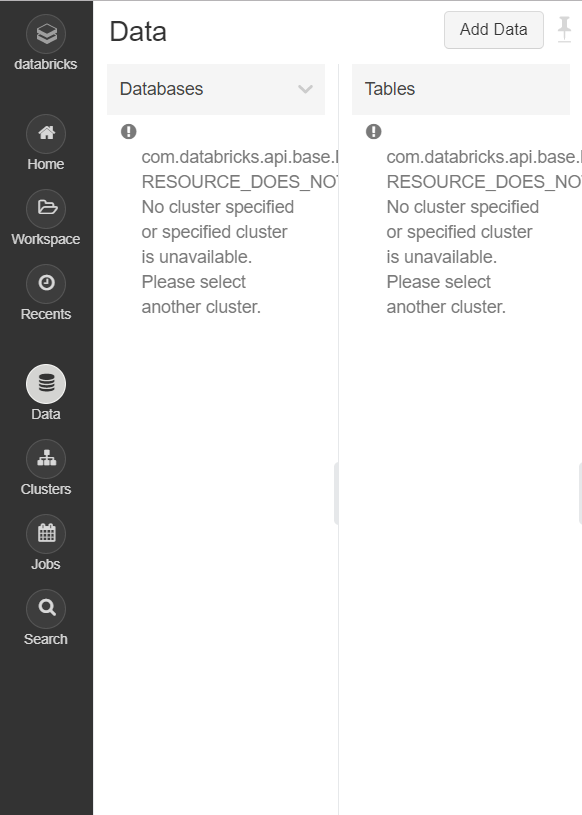
Data science has become an extremely rewarding career choice for people interested in extracting, manipulating, and generating insights out of large volumes of data. To fully leverage the power of data science, scientists often need to obtain skills in databases, statistical programming tools, and data visualizations. Companies surely need data scientists to help them empower their analytics processes, build a numbers-based strategy that will boost their bottom line, and ensure that enormous amounts of data are translated into actionable insights. In this project, we would like to introduce one of the most popular platform tools to analyze the data.

Databricks is one of the most powerful cloud base computations, managed platform for running Apache Spark - that means that you do not have to learn complex cluster management concepts nor perform tedious maintenance tasks to take advantage of Spark. Databricks also provides a host of features to help its users be more productive with Spark. It's a point and click platform for those that prefer a user interface like data scientists or data analysts. However, this UI is accompanied by a sophisticated API for those that want to automate aspects of their data workloads with automated jobs. To meet the needs of enterprises, Databricks also includes features such as role-based access control and other intelligent optimizations that not only improve usability for users but also reduce costs and complexity for administrators.

In this project, we will explore our process for discovering on numbers of petition visas base on random cases. It is worth mentioning that this notebook (in Databricks) is intended to be an academically usage, but simply a good example of the work that, a data scientist might be performing using Apache Spark and Databricks.

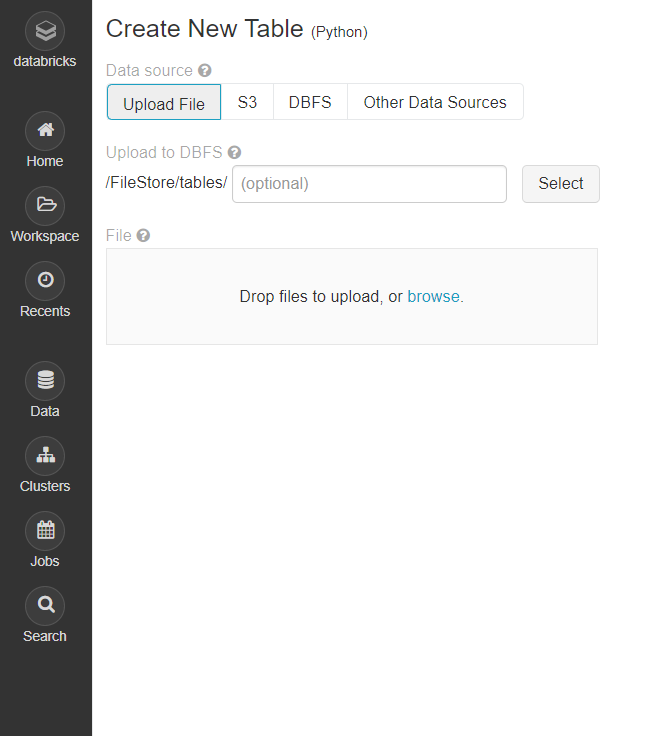
# **Starter Kits**

## **Loading data into Databricks**

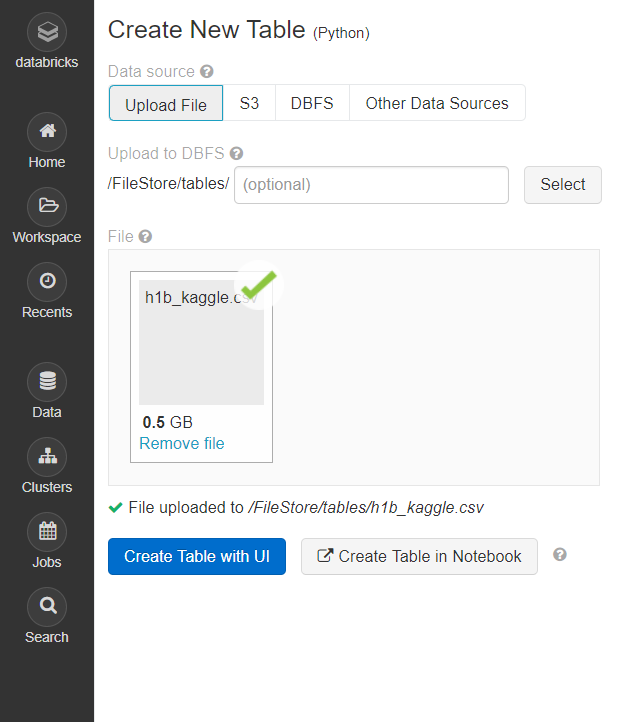


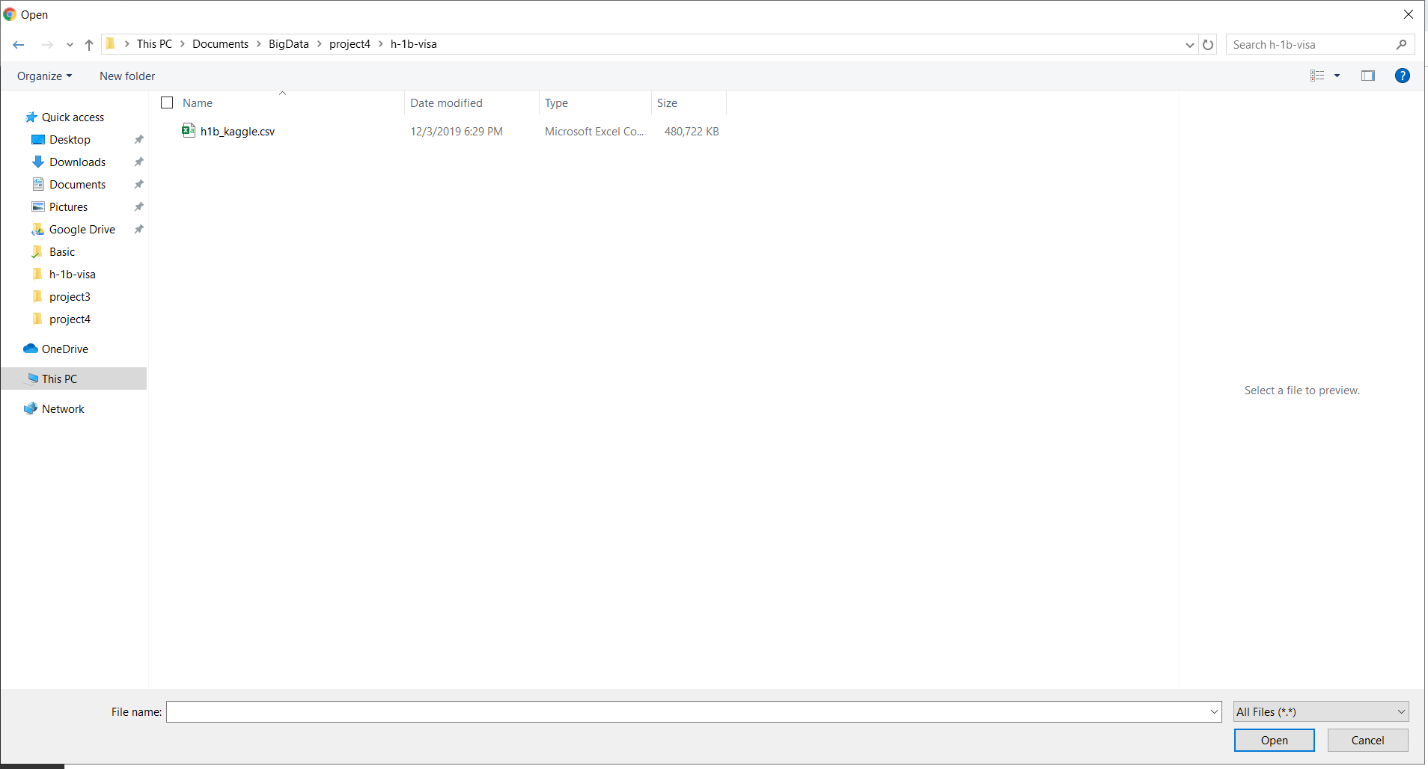
1 – click on the data icon on the left bar

2-click add Data



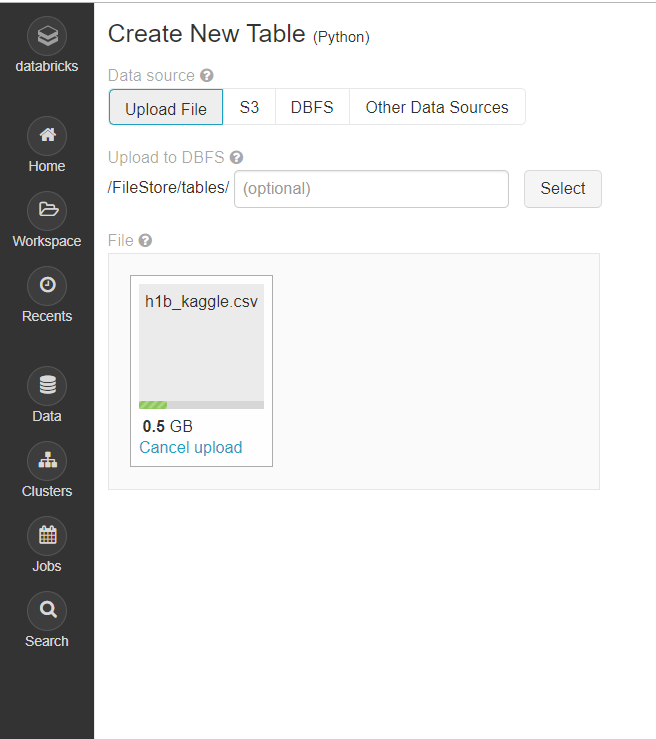
3- click browse





4- choose “h1b\_kaggle.csv” file

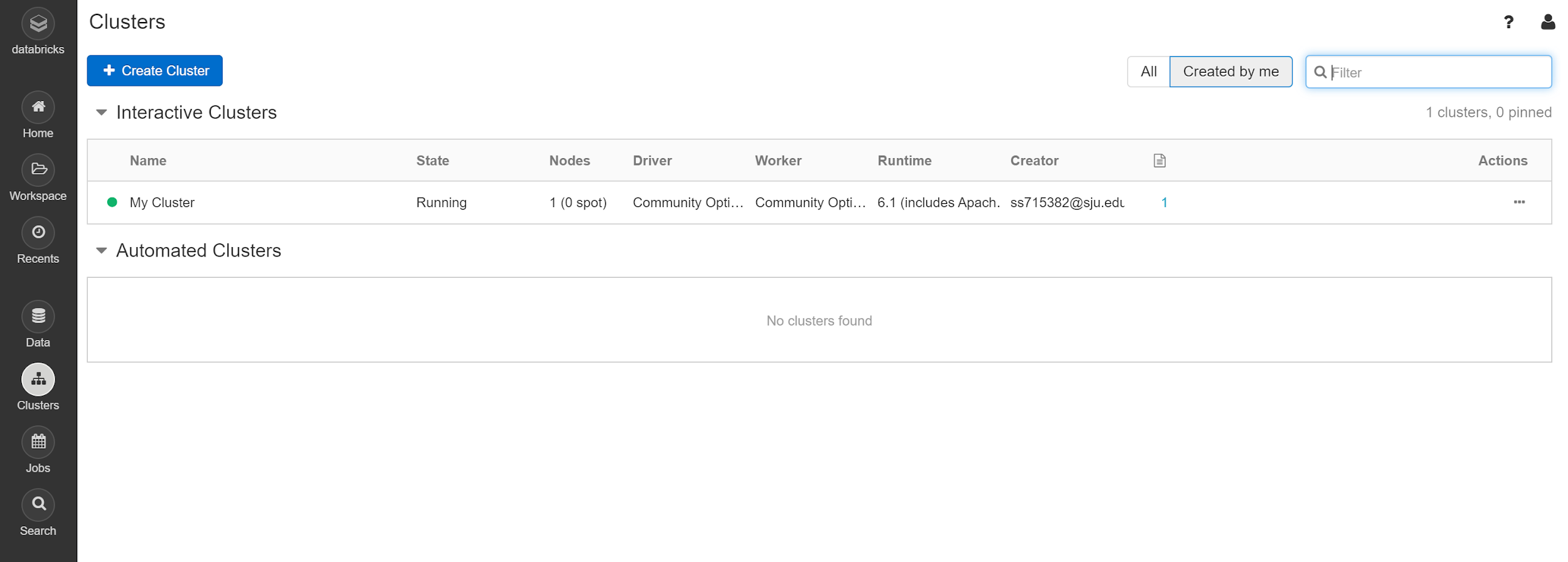
5- click open



6- Wait until the file is uploaded

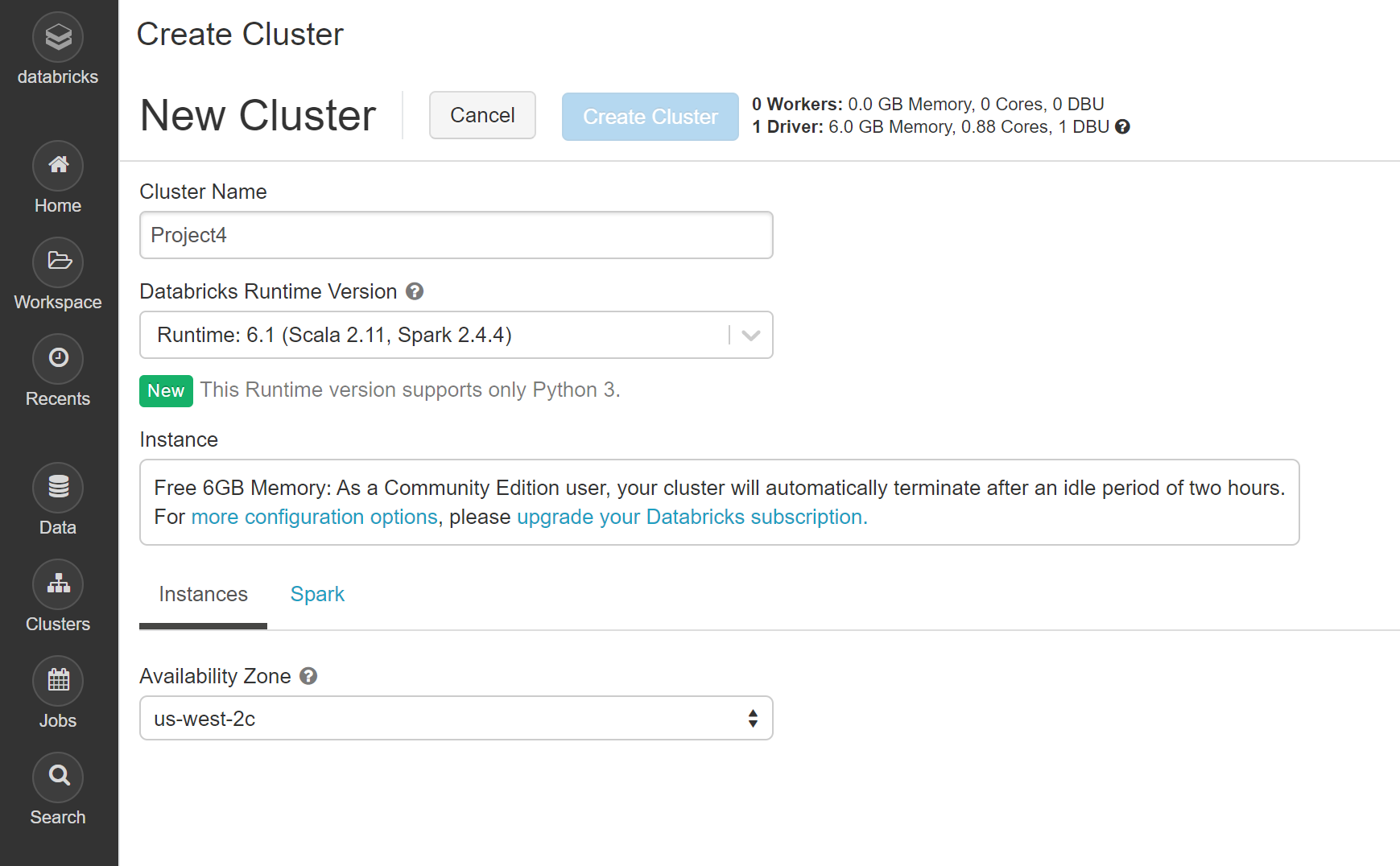
7- the file is uploaded

## **Creating cluster in Databricks**



1 – Click on the Clusters icon on the left bar

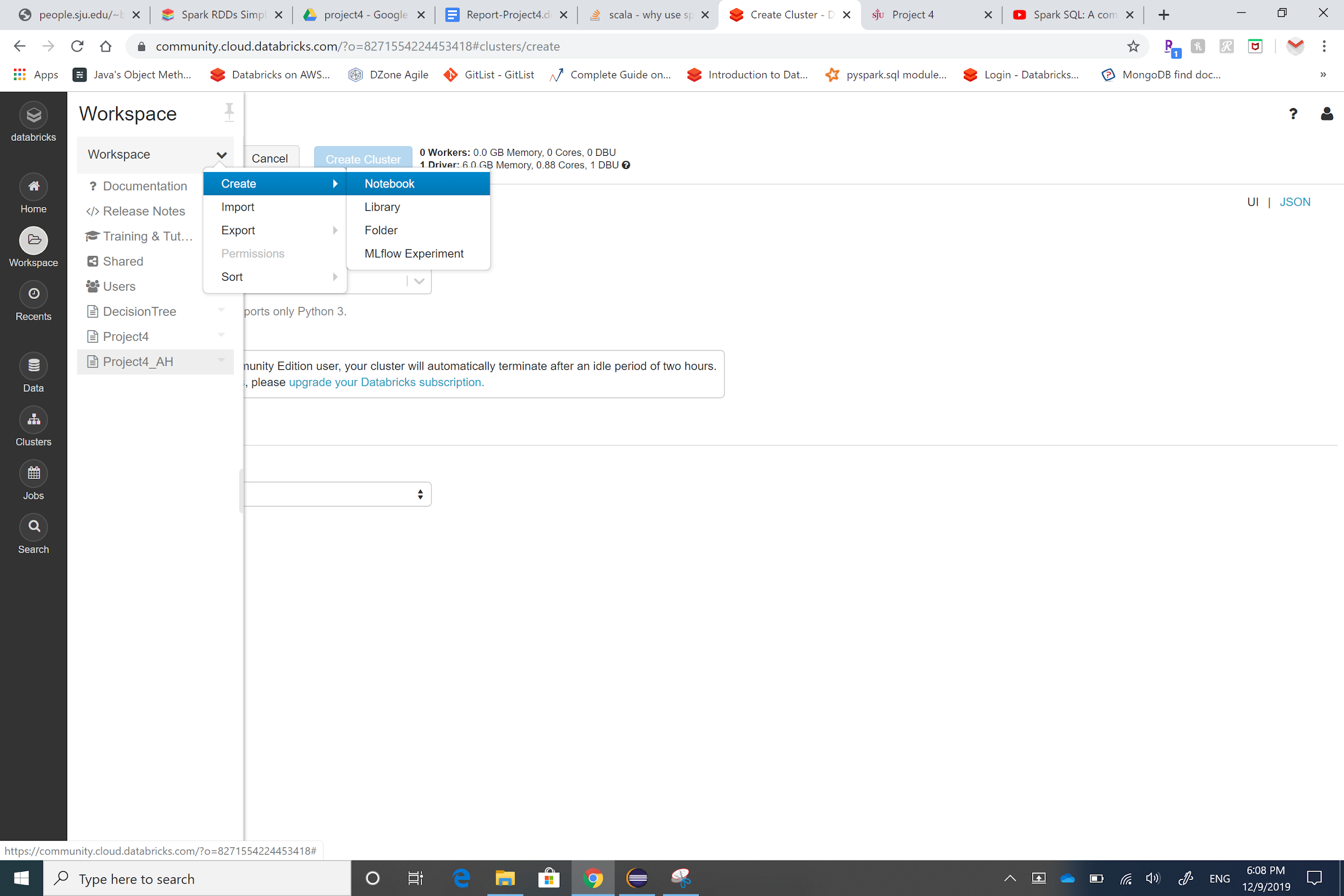
2 – Click on “Create Cluster” button



3 – Enter the cluster name

4 – Click “Create Cluster” button

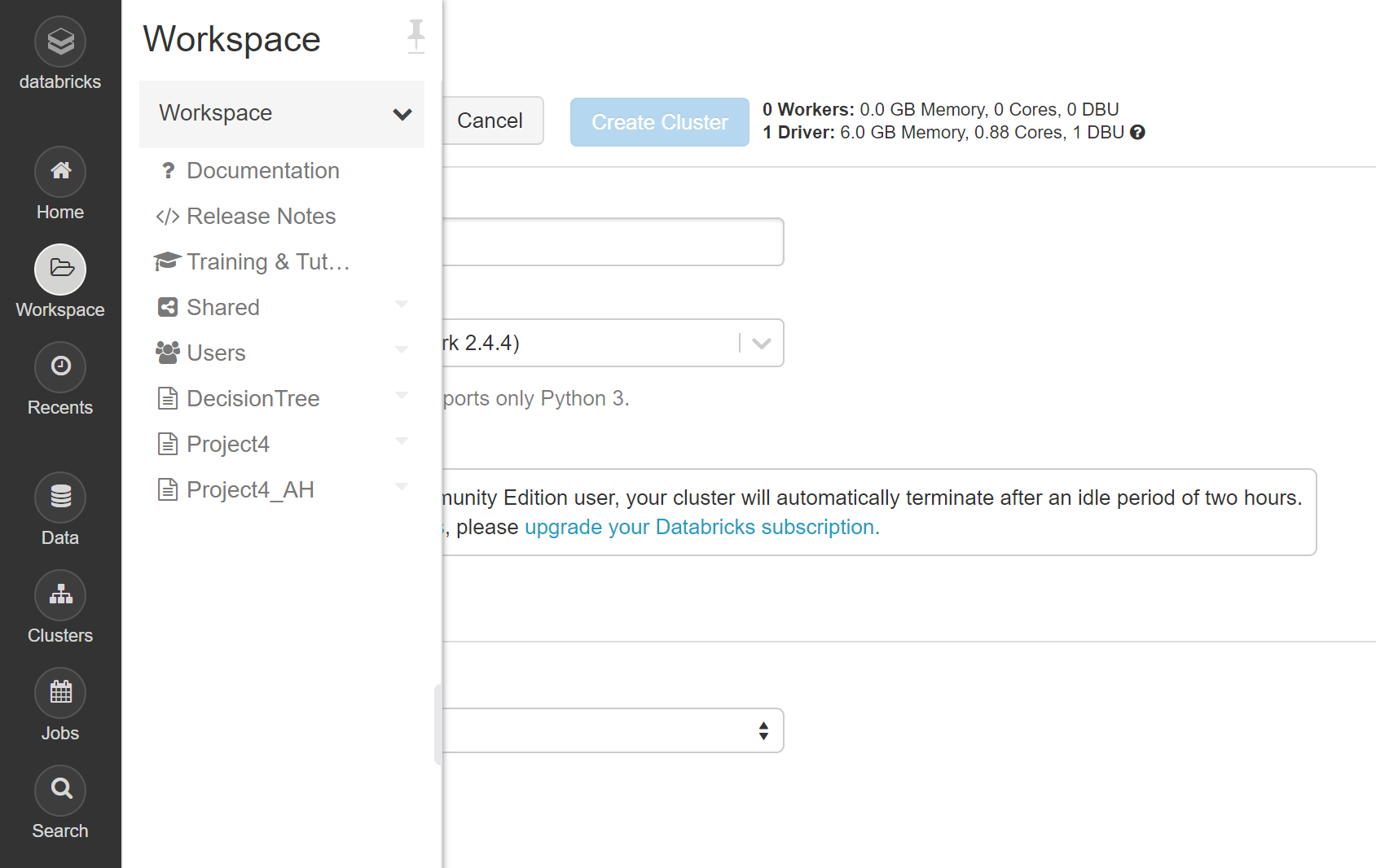
## **Creating notebook in Databricks**



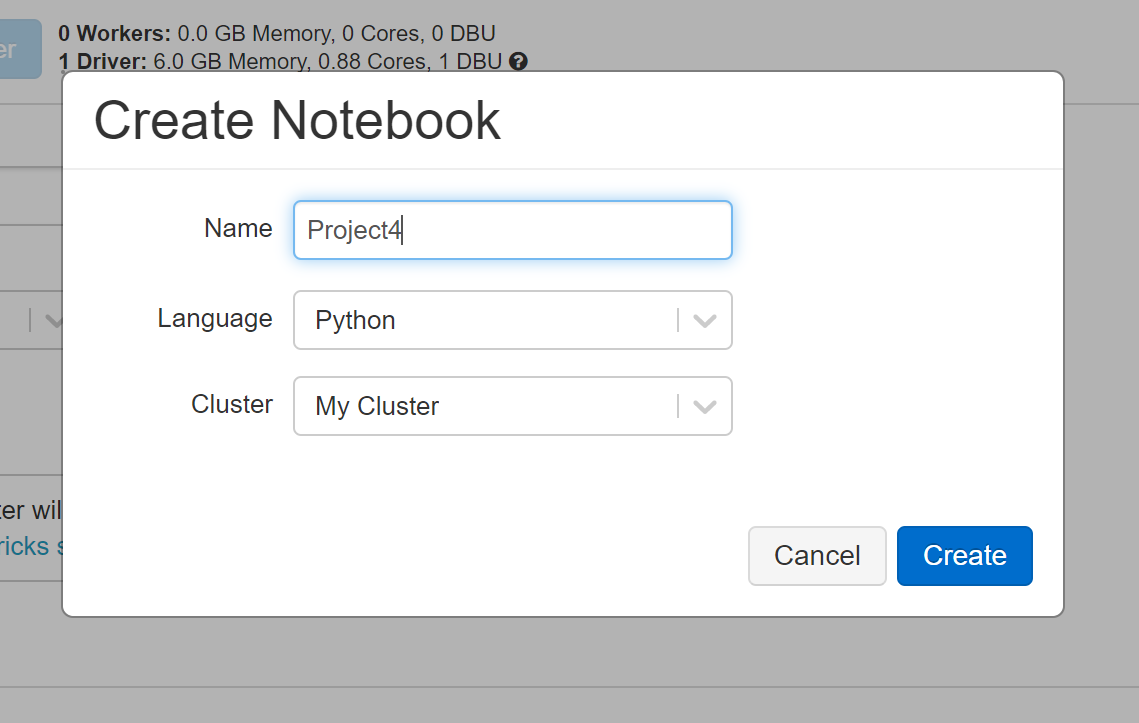
1- Click on “Workspace” icon on the left bar

2- Click on the little arrow

3- Click on “Create” then “Notebook”



6- the notebook is created. It is shown in the workspace list



4- Enter the notebook name, choose the language and the cluster

5- Click the “Create” button

# **Answers:**

First, create RDD (Resilient Distributed Datasets) to run on Disk only, as following:

from pyspark import SparkContext

import pyspark

h1bDF = spark.read.csv('/FileStore/tables/h1b\_kaggle.csv', header=**True**,

inferSchema=**True**)

h1bDF.persist(pyspark.StorageLevel.DISK\_ONLY)

## **Part1: Use Apache Spark Core (Transformations and Action on RDD) to answer**

## **the following queries?**

1. What is the total number of rows in this dataset?

h1bDF.count()

Out[3]: 3002458

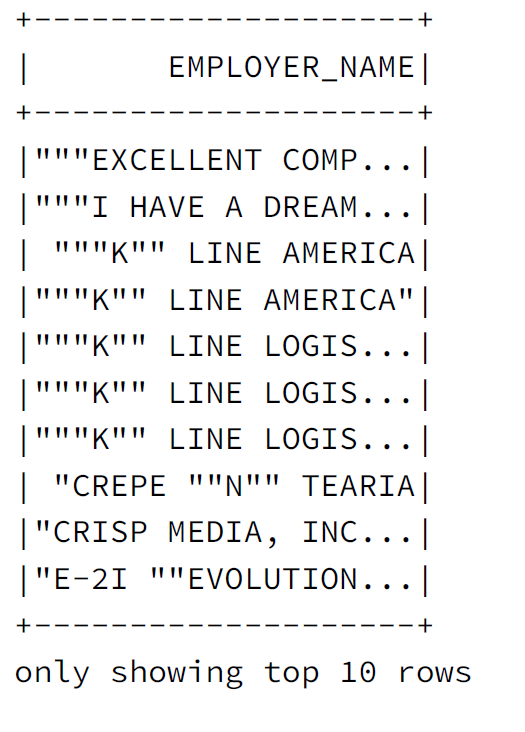
1. How many different employers filed for H-1B visa?

employers = h1bDF.select('EMPLOYER\_NAME').distinct()

employers.count()

Out[4]: 236015

1. Show 10 different employers ordered by employer name in descending order.

employers.orderBy(['EMPLOYER\_NAME'], descending=True).show(10)

## **Part2: Use Apache Spark Core (Transformations and Action on RDD) to answer**

## **the following queries?**

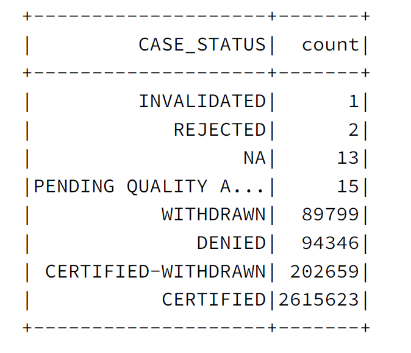
1. How many visa petitions are there for each ‘case status’ (e.g., 1000

withdrawn, 100 denied)? Order results by the number of visa petitions

(i.e., count) in ascending order. Note, ascending order is the default,

thus, there is no need to specify that.

h1bDF.groupBy('CASE\_STATUS').count().orderBy('count').show()



2. Repeat the above query to perform in-memory processing. Databricks will

display execution time for each ‘cell’, take a screenshot of the cell

including the running times for both in-memory and on-disk.

h1bDF.cache()

#What is the total number of rows in this dataset?

h1bDF.count()

Out[8]: 3002458

Figure 1: Part 1-1 without cache

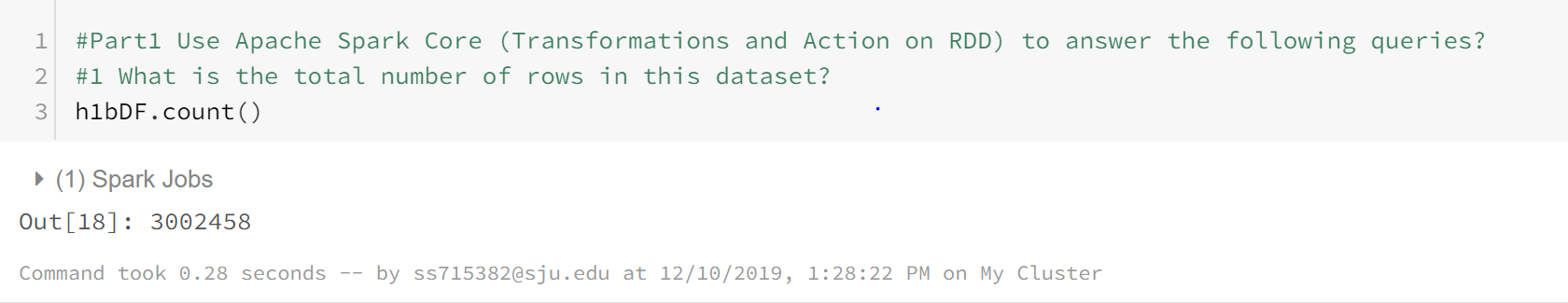
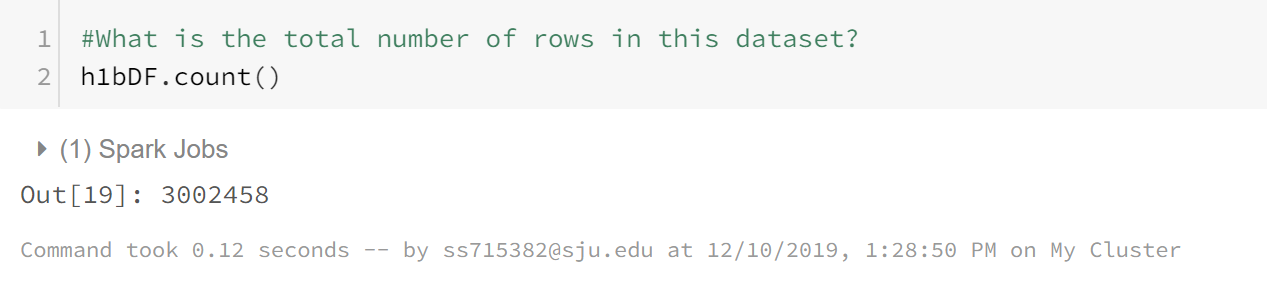


Figure 2: Part 1-1 with cache



#How many different employers filed for H-1B visa?

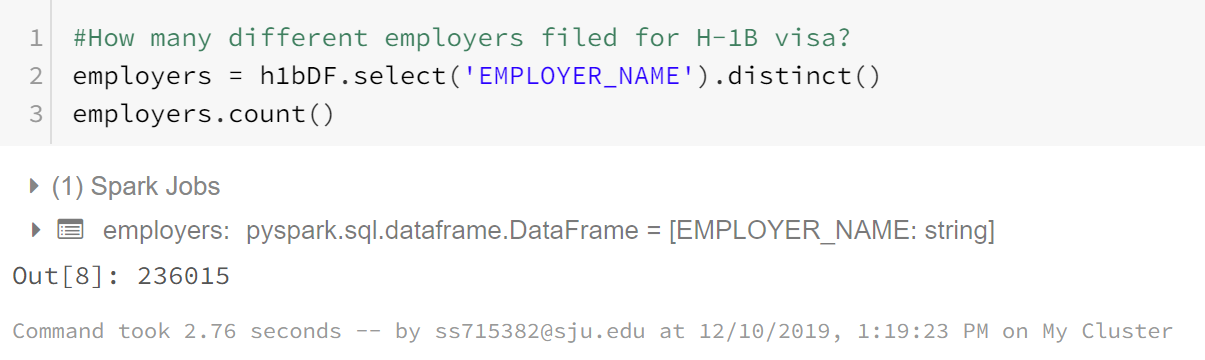
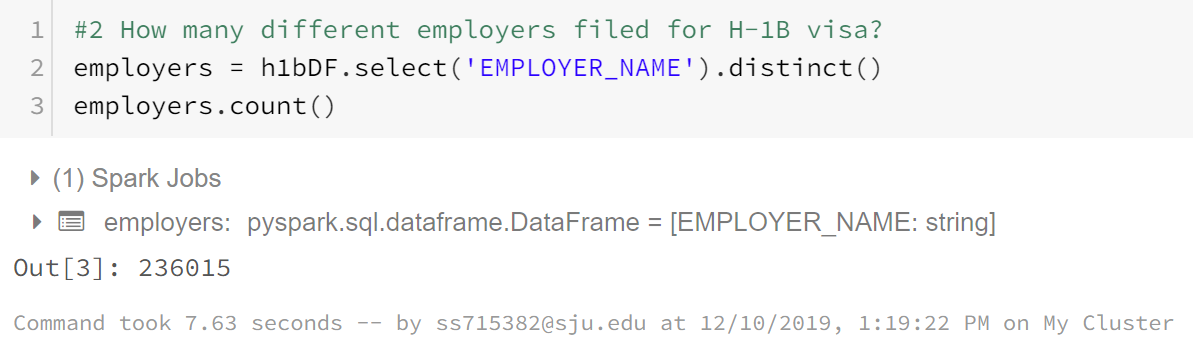
employers = h1bDF.select('EMPLOYER\_NAME').distinct()

employers.count()

Out[9]: 236015

Figure 3: Part 1-2 with cache

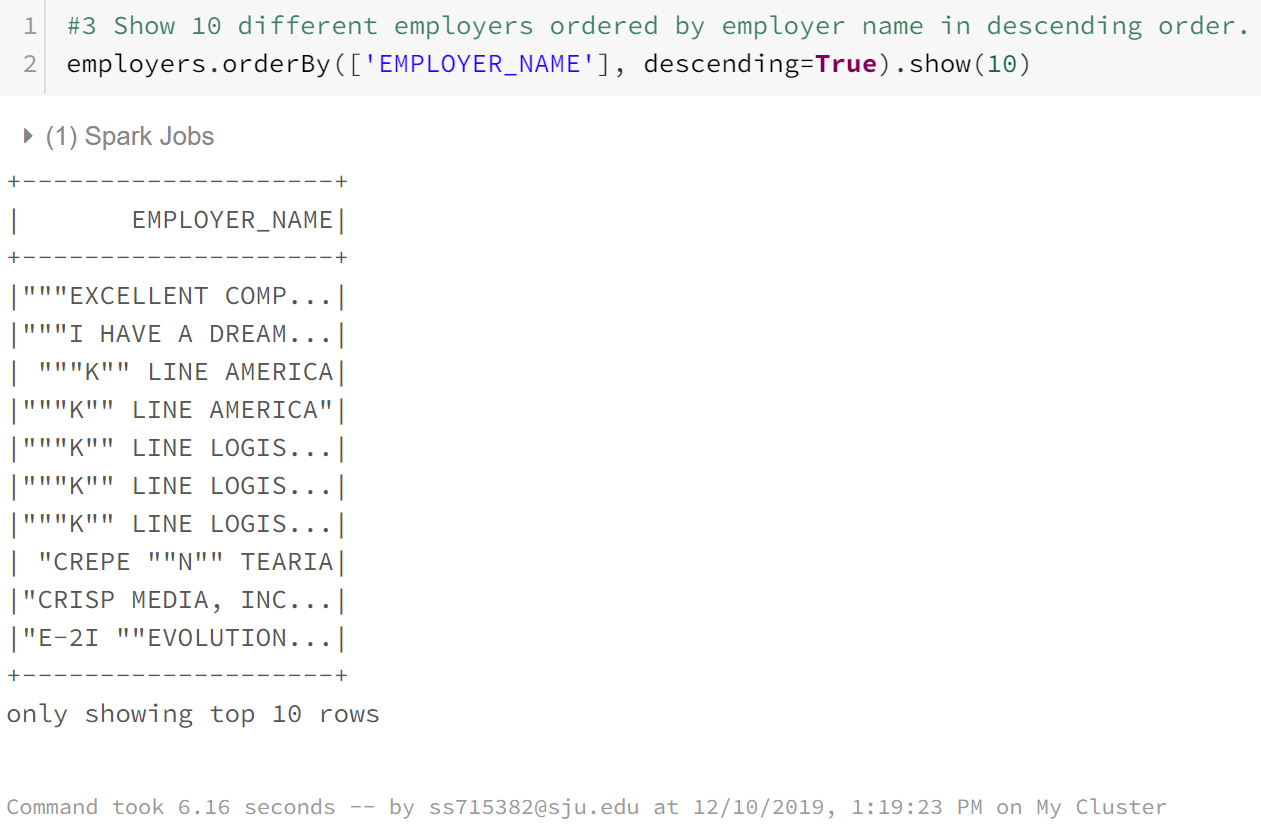
Figure 4: Part 1-2 without cache



#Show 10 different employers ordered by employer name in descending order.

employers.orderBy(['EMPLOYER\_NAME'], descending=**True**).show(10)

Figure 5: Part 1-3 without cache



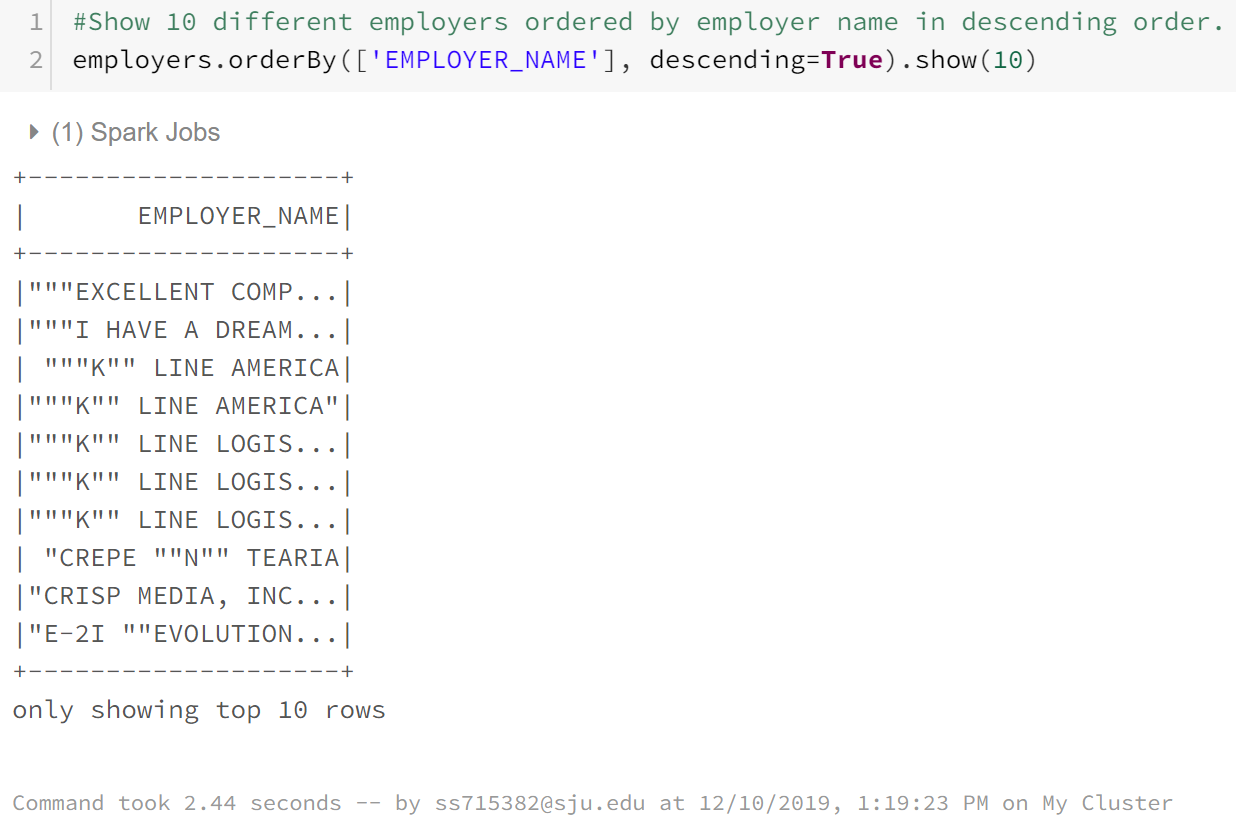


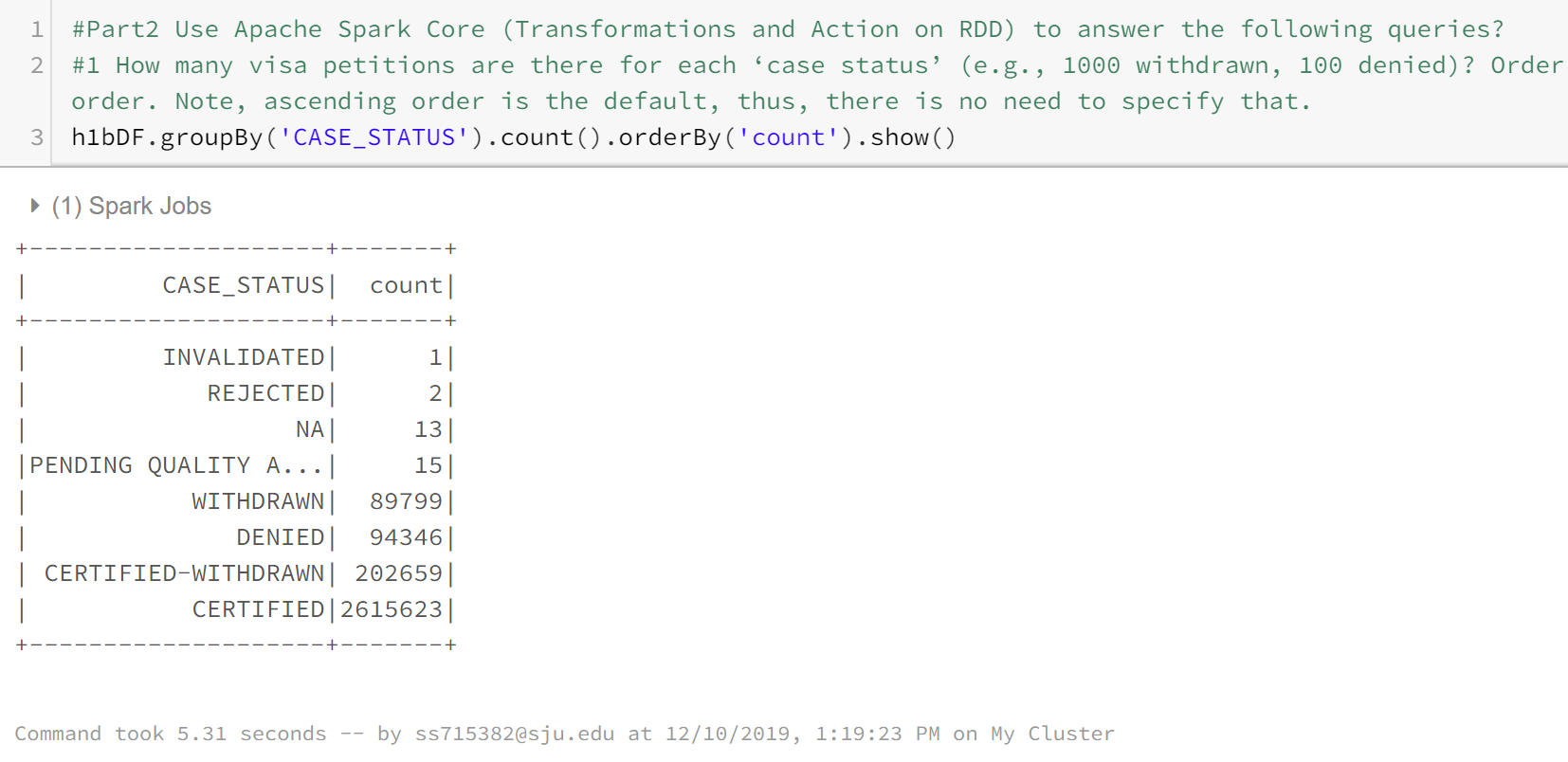
Figure 6: Part 1-3 with cache

#How many visa petitions are there for each ‘case status’ (e.g., 1000

withdrawn, 100 denied)? Order results by the number of visa petitions

(i.e., count) in ascending order. Note, ascending order is the default,

thus, there is no need to specify that.

h1bDF.groupBy('CASE\_STATUS').count().orderBy('count').show()

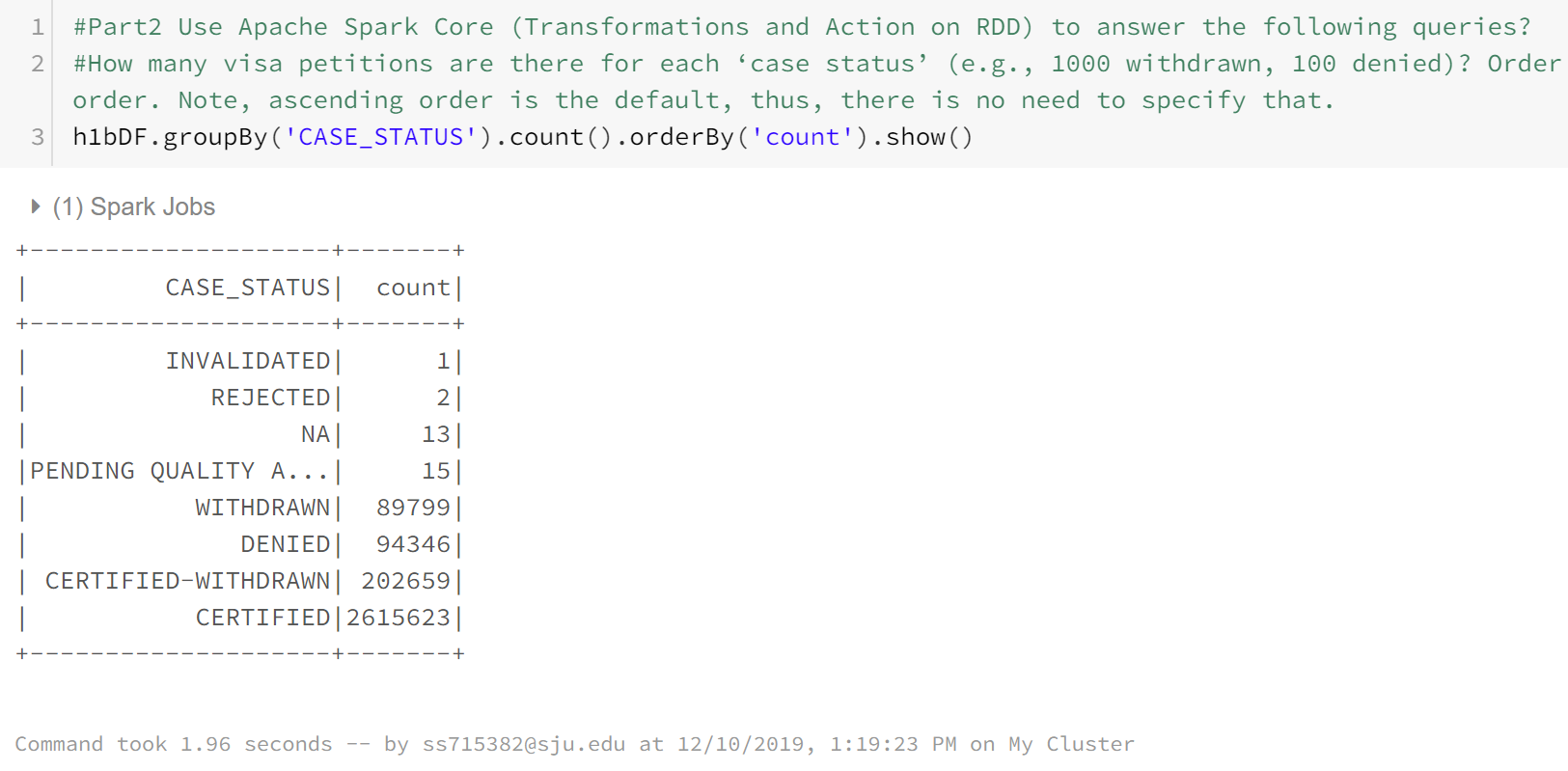


Figure : Part 2-1 without cache

Figure :Part 2-1 with cache

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Command** | **With cache** | **Without cache** |
| **Part 1 - 1** | h1bDF.count() | 0.12 | 0.28 |
| **Part 1 - 2** | employers = h1bDF.select('EMPLOYER\_NAME').distinct() employers.count() | 2.76 | 7.63 |
| **part 1 - 3** | employers.orderBy(['EMPLOYER\_NAME'], descending=True).show(10) | 2.44 | 6.16 |
| **part 2 - 1** | h1bDF.groupBy('CASE\_STATUS').count().orderBy('count').show() | 1.96 | 5.31 |

Table 1: time compression between in-disk and in-memory processing

1. Use Python, R, or Scala to create a new data-frame (RDD) that contains

employer names only. Then, iterate (i.e., loop) through the newly created

data-frame to print all rows of data/employer names, and count/print number

of rows. Hint, a transformation returns a new data-frame. Be patient, this

might take a while.

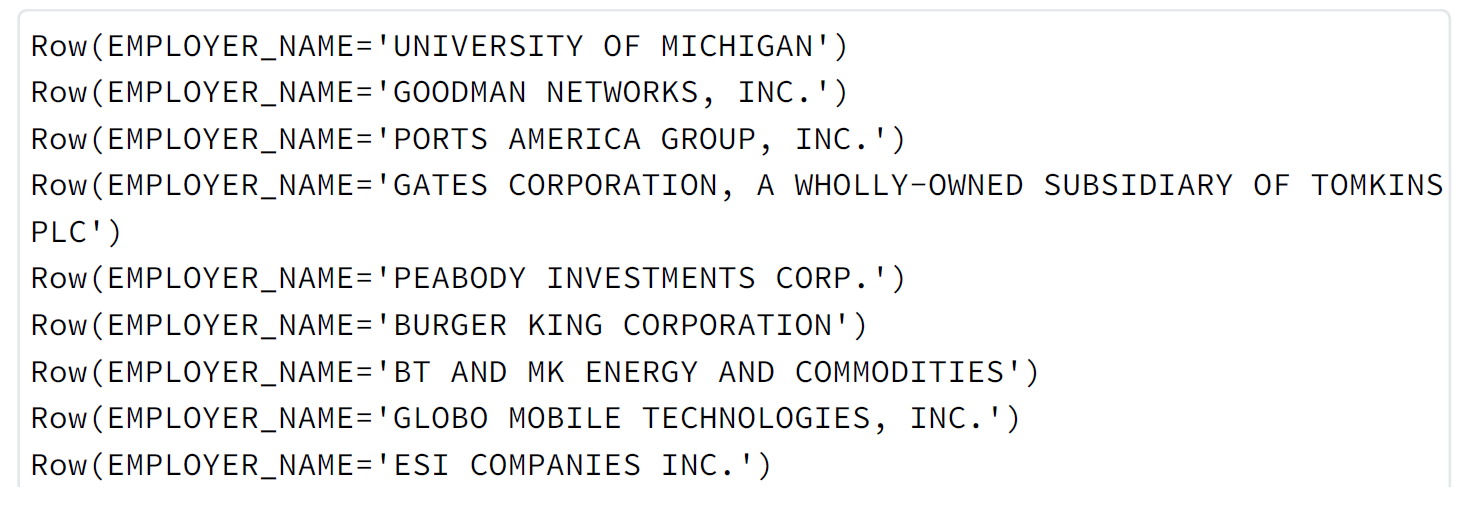
employersDF = h1bDF.select('EMPLOYER\_NAME')

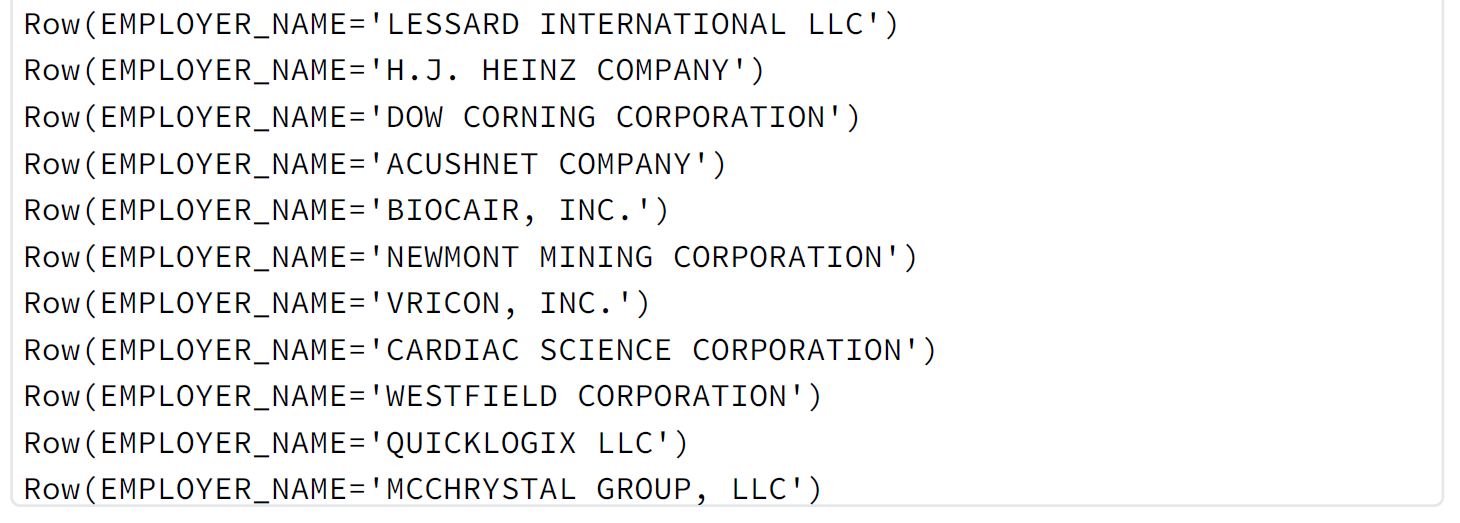
employersDF.count()

Out[91]: 3002458

**for** row **in** employersDF.collect():

print(row)





## **Part3: Use Apache Spark Core and Spark SQL to answer the following queries.**

## **Hint, you might need to create a temporary view/table in order to perform**

## **SQL queries. Cache both RDDs and Views to perform in-memory processing.**

# Create a VIEW to perform SQL queries

#done the RDD cahche in the above part

h1bDF.createOrReplaceTempView("h1bView")

# cache table

spark.catalog.cacheTable("h1bView")

1. How many visa petitions were denied in 2016?

%sql

**select** count(CASE\_STATUS)

**from** h1bView

**where** CASE\_STATUS == 'DENIED' **and** YEAR == '2016';

**count(CASE\_STATUS)**

9175

1. Show different employers in California and Pennsylvania who have

certified visa petitions in 2013. Order results by employer names in

ascending order.

%sql

**select distinct** EMPLOYER\_NAME

**from** h1bView

**where** WORKSITE **regexp** 'CALIFORNIA | PENNSYLVANIA' **and** CASE\_STATUS ==

'CERTIFIED' **and** YEAR == '2013'

**order by** EMPLOYER\_NAME;

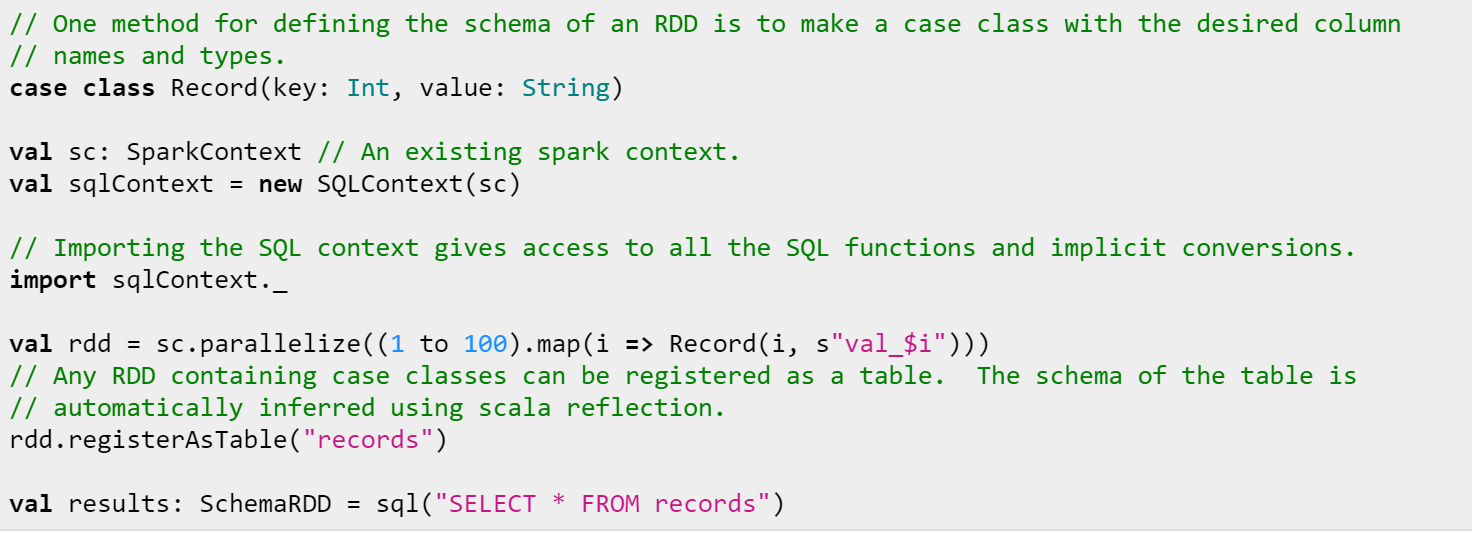


1. Is it possible to perform SQL queries directly on RDD?

It is possible in a way that you create your own version of RDD called schemaRDD. To standard RDD functions, SchemaRDDs can be used in relational queries. At the core of this component, a new type of RDD, [SchemaRDD](https://spark.apache.org/docs/1.0.2/api/scala/index.html" \l "org.apache.spark.sql.SchemaRDD), are composed [Row](https://spark.apache.org/docs/1.0.2/api/scala/index.html#org.apache.spark.sql.catalyst.expressions.Row) objects along with a schema that describes the data types of each column in the row. A SchemaRDD is similar to a table in a traditional relational database. A SchemaRDD can be created from an existing RDD, [Parquet](http://parquet.io/) file, a JSON dataset, etc.

Importing a SQLContext brings an implicit into scope that automatically converts a standard RDD whose elements are scala case classes into a SchemaRDD. A SchemaRDD can be registered as a table in the [SQLContext](https://spark.apache.org/docs/1.0.2/api/scala/org/apache/spark/sql/SQLContext.html) that was used to create it. Once an RDD has been registered as a table, it can be used in the FROM clause of SQL statements.

Here is the example:



## **Part4: Pre-define the Schema manually (inferring the schema) in order to**

**use Data Aggregation functions and compare numeric values (e.g., <, >, =)**

**using Apache Spark Core. You will need to import the following libraries to**

**pre-define/infer schema and use aggregation functions:**

**from** pyspark.sql.functions **import** min, max, avg

**from** pyspark.sql.types **import** StructType, StructField, IntegerType,

StringType

**from** pyspark.sql.types **import** \*

1. Since SOC\_NAME column does not specify what sort of data it contains,

change the column name to OCCUPATION\_CODE for a coherent column name. Take

a screenshot of the Schema, to verify that the column name has changed.

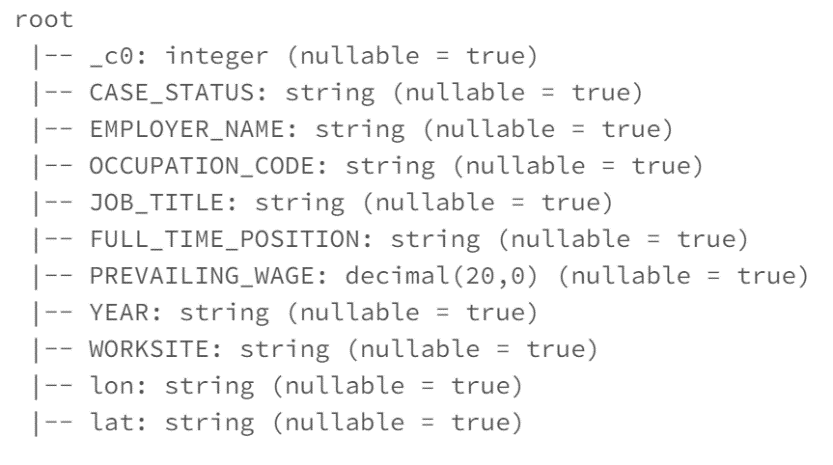
Hint, use DF.printSchema() to view column names and data types.

h1bDF = h1bDF.withColumnRenamed('SOC\_NAME', 'OCCUPATION\_CODE')

#convert PREVAILING\_WAGE from datatype to decimal

h1bDF = h1bDF.withColumn('PREVAILING\_WAGE',

h1bDF.PREVAILING\_WAGE.cast(DecimalType(20)))

h1bDF.printSchema()

1. Show the lowest, highest, and average wage/salary of all filed visa

petitions?

#filter to get only the legit value

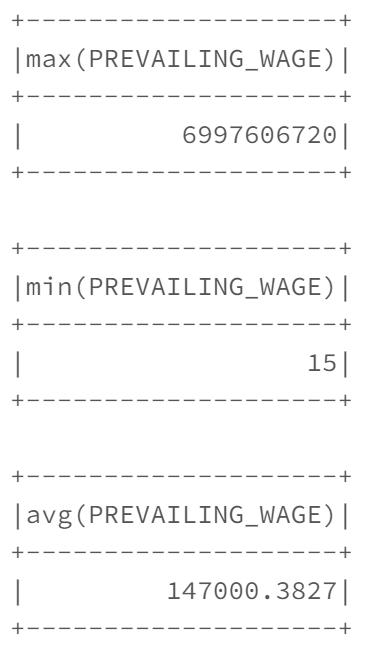
h1bDF\_filter =

h1bDF.select('PREVAILING\_WAGE').filter(h1bDF.PREVAILING\_WAGE>0)

h1bDF\_filter.agg(max("PREVAILING\_WAGE")).show()

h1bDF\_filter.agg(min("PREVAILING\_WAGE")).show()

h1bDF\_filter.agg(avg("PREVAILING\_WAGE")).show()



1. Print an appropriate message if there are wages that are greater than

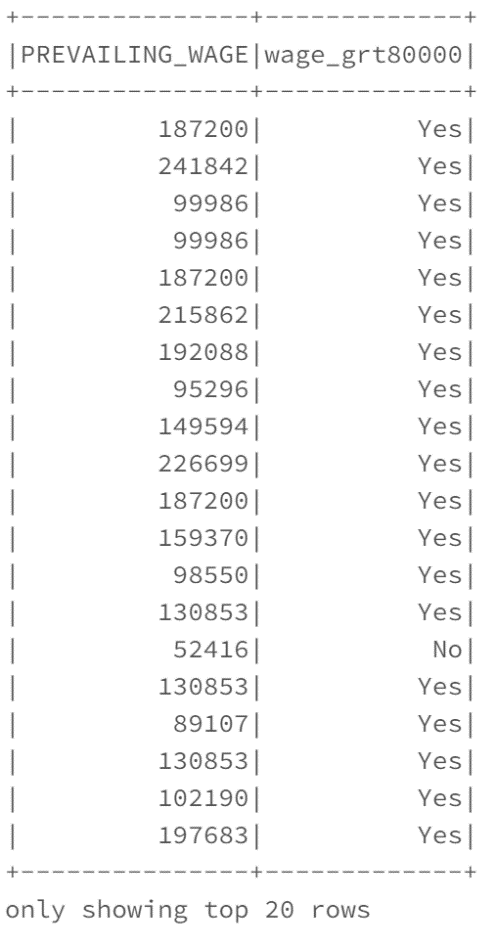
80,000. If yes, print how many, else, print “none”.

**from** pyspark.sql.functions **import** col, when

h1bDF\_filter.withColumn('wage\_grt80000', when(col('PREVAILING\_WAGE') >

80000, "Yes").otherwise("No")).show()

h1bDF\_filter.where(h1bDF\_filter['PREVAILING\_WAGE']>80000).count()



alternative solution:

count = 0

for x in h1bDF\_filter.collect():

if x.PREVAILING\_WAGE > 80000:

count+=1

if count>0:

print("Yes, there are wages greater than $80,000 : " + str(count))

else:

 print("None")

# **References:**

* Data provider:

<https://www.kaggle.com/nsharan/h-1b-visa>

* Databricks Documentation:

<https://docs.databricks.com/>

* SchemaRDD:

<https://spark.apache.org/docs/1.0.2/api/scala/index.html#org.apache.spark.sql.SchemaRDD>

* PySpark – StorageLevel:

<https://www.tutorialspoint.com/pyspark/pyspark_storagelevel.htm>

* CSC643 class’s material:

<http://people.sju.edu/~bforoura/tutorials/databricks/introduction.htm>

* Notebook in HTML:

<file:///C:/Users/seans/Downloads/Project4.html>