CSC 643 – Big Data & Web Intelligence

Team1.cache(Ahmad, Sean)

Project 4

Submission Date: December 11, 2019

Table of Contents

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

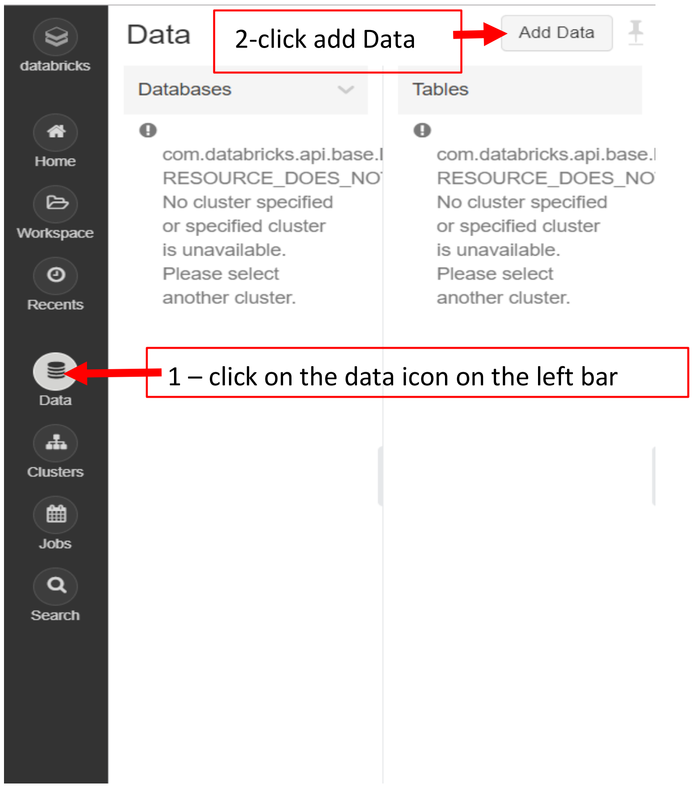
Data science has become an extremely rewarding career choice for people interested in extracting, manipulating, and generating insights out of large volumes of data. To fully leverage the power of data science, scientists often need to obtain skills in databases, statistical programming tools, and data 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.

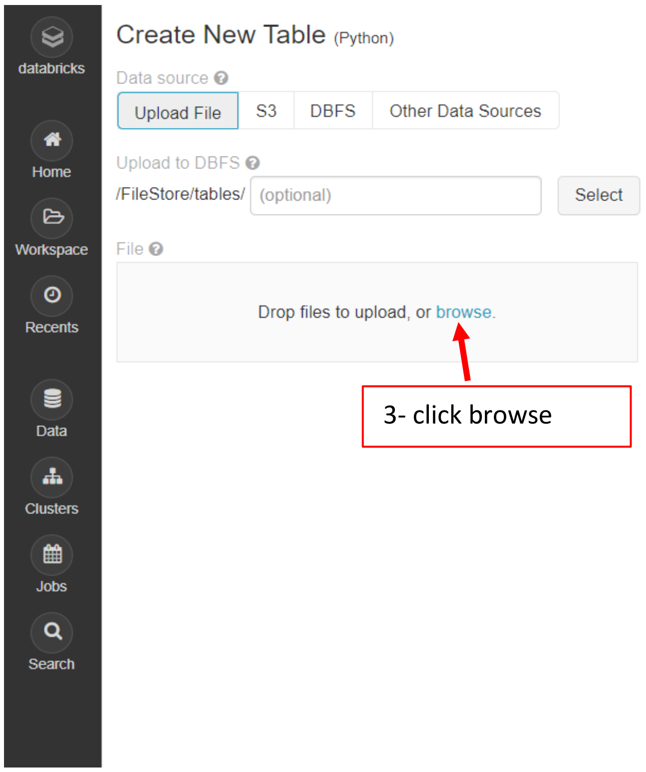
In this project, we will explore our process for discovering on numbers of petition visas base on random cases. It is worth mentioning that this notebook (in Databricks) is intended to be an academically usage, but simply a good example of the work that, a data scientist might be performing using Apache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

STARTER KITS

1. Loading data into Databricks

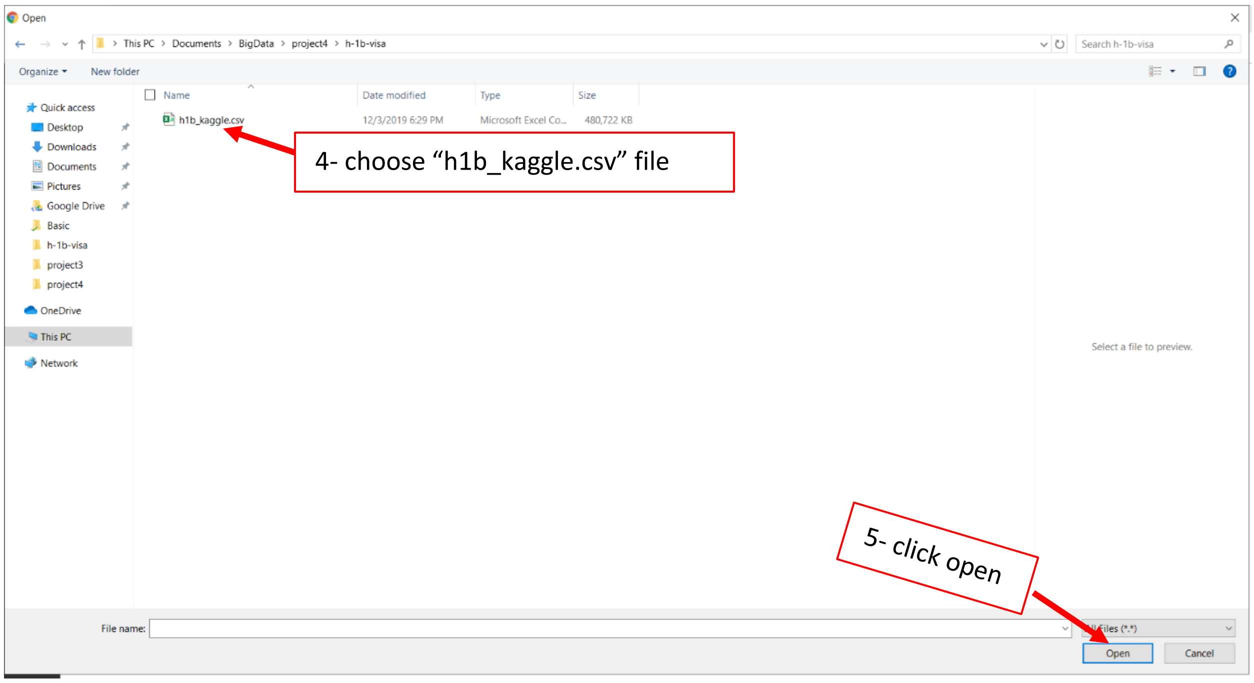


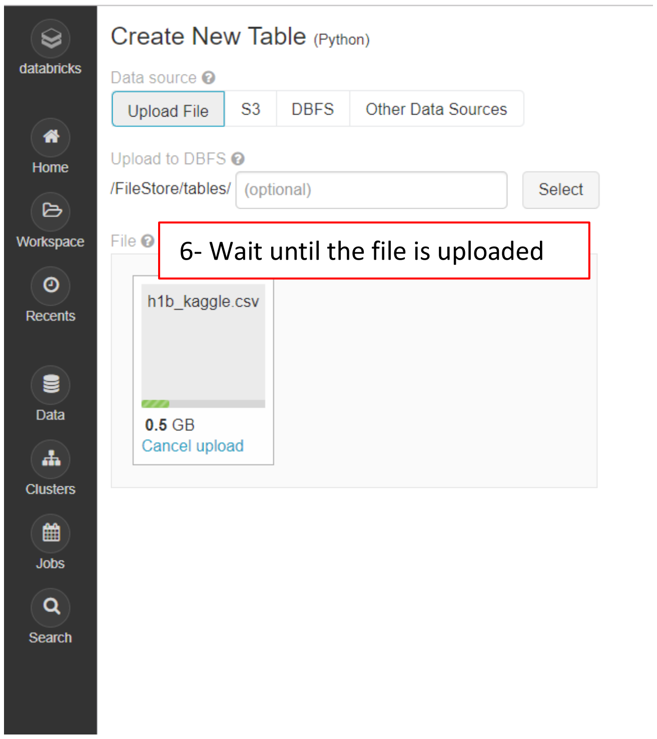
com.databricks.api.base.I
RESOURCE_DOES_NO
No cluster specified
or specified cluster
is unavailable.
Please select
another cluster.

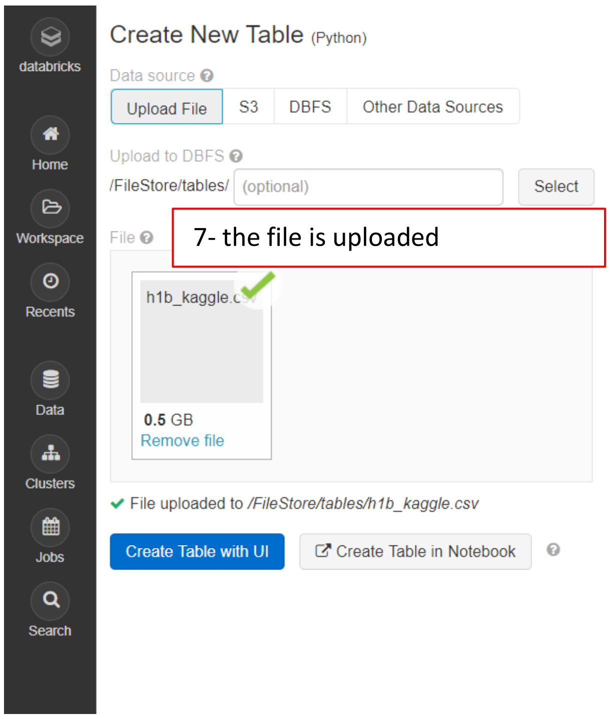


Create New Table (Python)

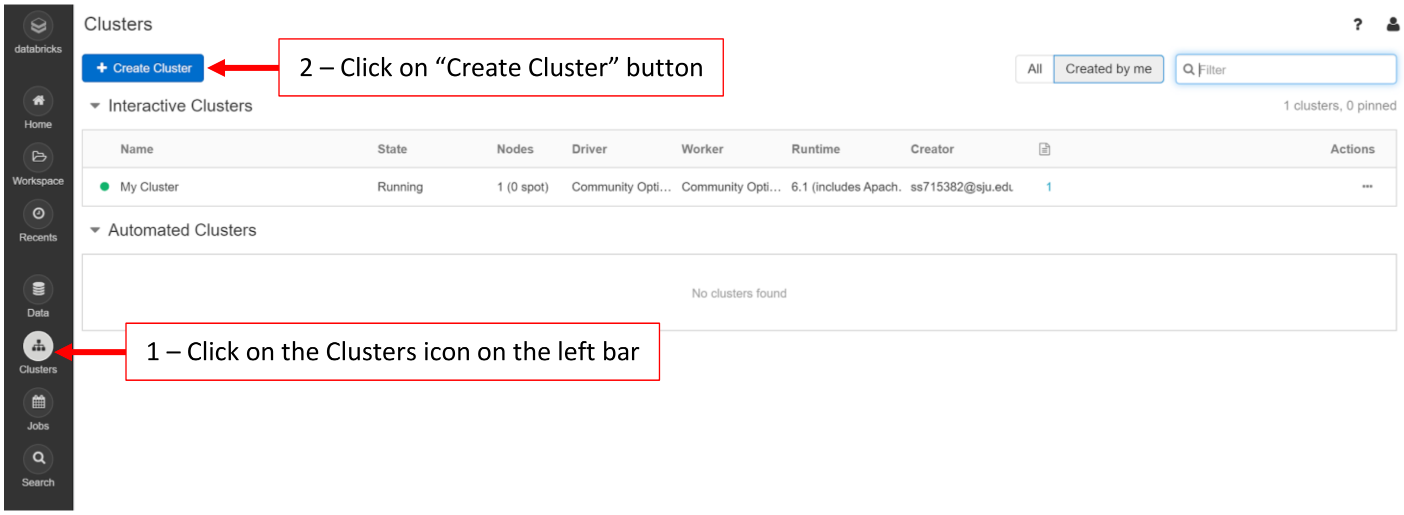
Drop files to upload, or browse.


Open
This PC > Documents > BigData > project4 > h-1b-visa
Search h-1b-visa
Organize
New folder
Quick access
Desktop
Downloads
Documents
Pictures
Google Drive
Basic
h-1b-visa
project3
project4
OneDrive
This PC
Network
Name
Date modified
Type
Size
h1b_kaggle.csv
12/3/2019 6:29 PM
Microsoft Excel Co...
480,722 KB
4- choose "h1b_kaggle.csv" file
Select a file to preview.
5- click open
File name:
All Files (*.*)
Open
Cancel


databricks
Home
Workspace
Recents
Data
Clusters
Jobs
Search
Create New Table (Python)
Data source
Upload File
S3
DBFS
Other Data Sources
Upload to DBFS
/FileStore/tables/
(optional)
Select
File
6- Wait until the file is uploaded
h1b_kaggle.csv
0.5 GB
Cancel upload


databricks
Home
Workspace
Recents
Data
Clusters
Jobs
Search
Create New Table (Python)
Data source
Upload File
S3
DBFS
Other Data Sources
Upload to DBFS
/FileStore/tables/
(optional)
Select
File
7- the file is uploaded
h1b_kaggle.csv
0.5 GB
Remove file
File uploaded to /FileStore/tables/h1b_kaggle.csv
Create Table with UI
Create Table in Notebook

2. Creating cluster in Databricks



Clusters

+ Create Cluster

2 – Click on “Create Cluster” button

Interactive Clusters

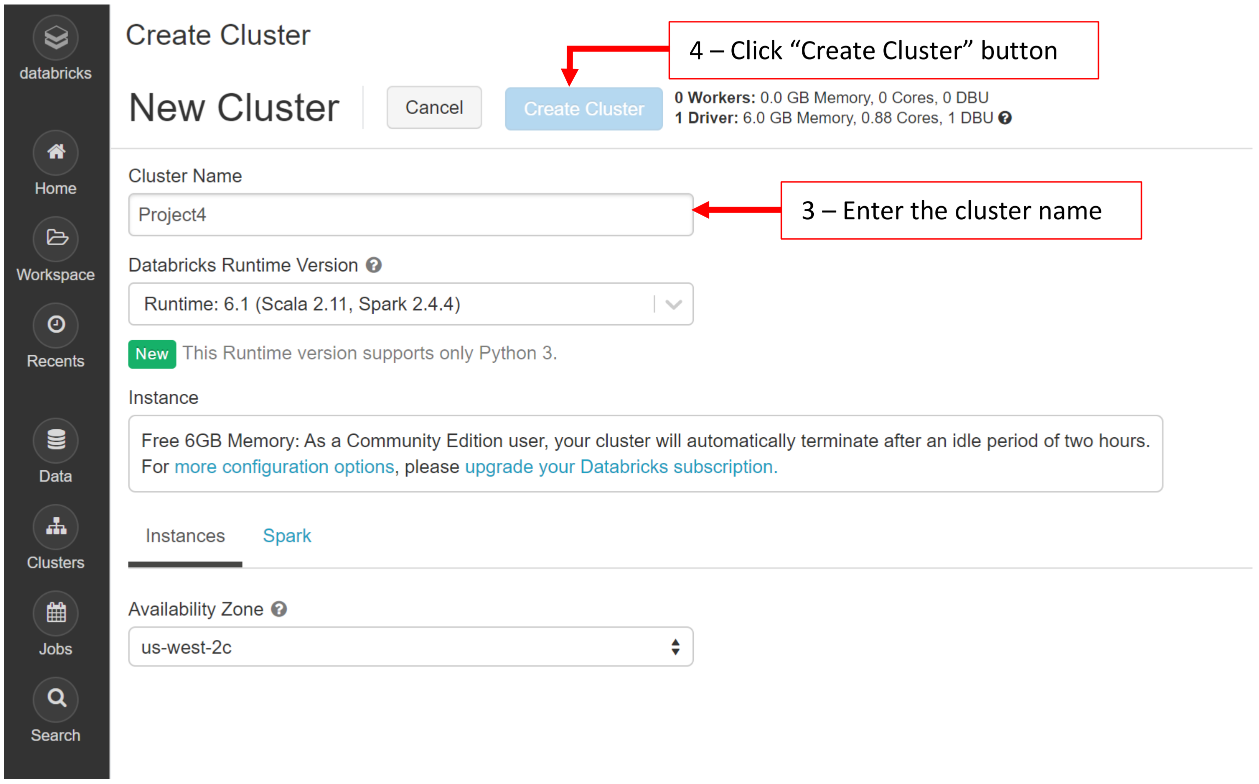
1 clusters, 0 pinned

Name	State	Nodes	Driver	Worker	Runtime	Creator		Actions
● My Cluster	Running	1 (0 spot)	Community Opti…	Community Opti…	6.1 (includes Apach…	ss715382@sju.edu	1	···

Automated Clusters

No clusters found

1 – Click on the Clusters icon on the left bar



Create Cluster

4 – Click “Create Cluster” button

New Cluster

0 Workers: 0.0 GB Memory, 0 Cores, 0 DBU
1 Driver: 6.0 GB Memory, 0.88 Cores, 1 DBU

Cluster Name: Project4

3 – Enter the cluster name

Databricks Runtime Version: Runtime: 6.1 (Scala 2.11, Spark 2.4.4)

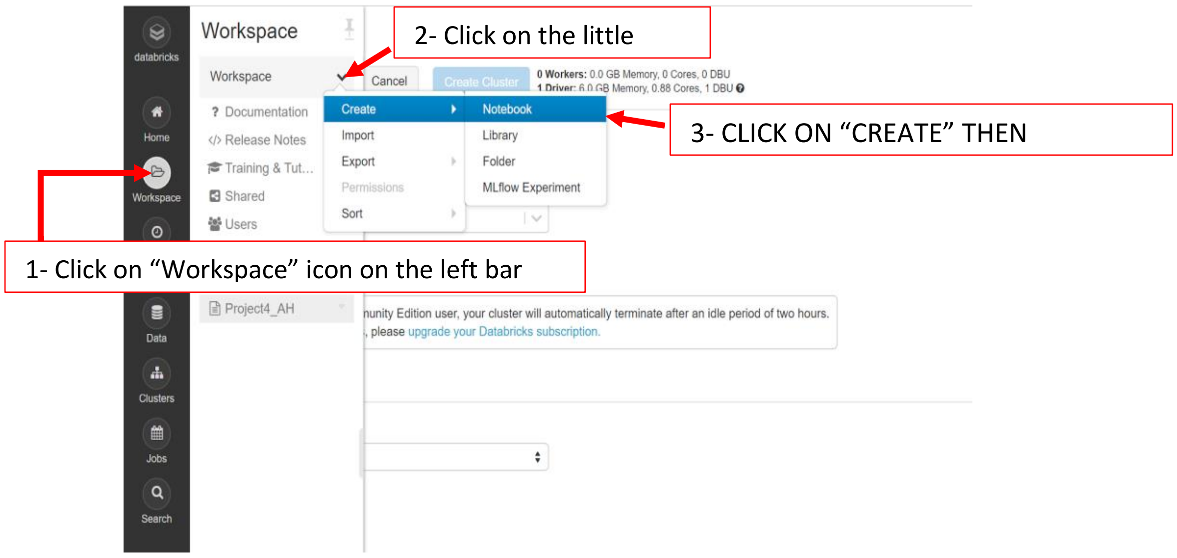
New This Runtime version supports only Python 3.

Instance

Free 6GB Memory: As a Community Edition user, your cluster will automatically terminate after an idle period of two hours. For more configuration options, please upgrade your Databricks subscription.

Availability Zone: us-west-2c

3. Creating notebook in Databricks

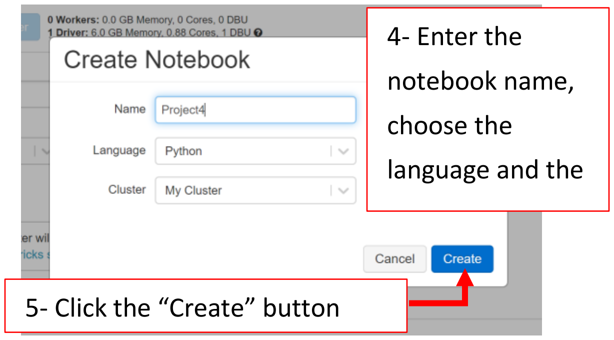

2- Click on the little

3- CLICK ON "CREATE" THEN

1- Click on "Workspace" icon on the left bar

0 Workers: 0.0 GB Memory, 0 Cores, 0 DBU
1 Driver: 6.0 GB Memory, 0.88 Cores, 1 DBU

nunity Edition user, your cluster will automatically terminate after an idle period of two hours.
, please upgrade your Databricks subscription.


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

5- Click the "Create" button

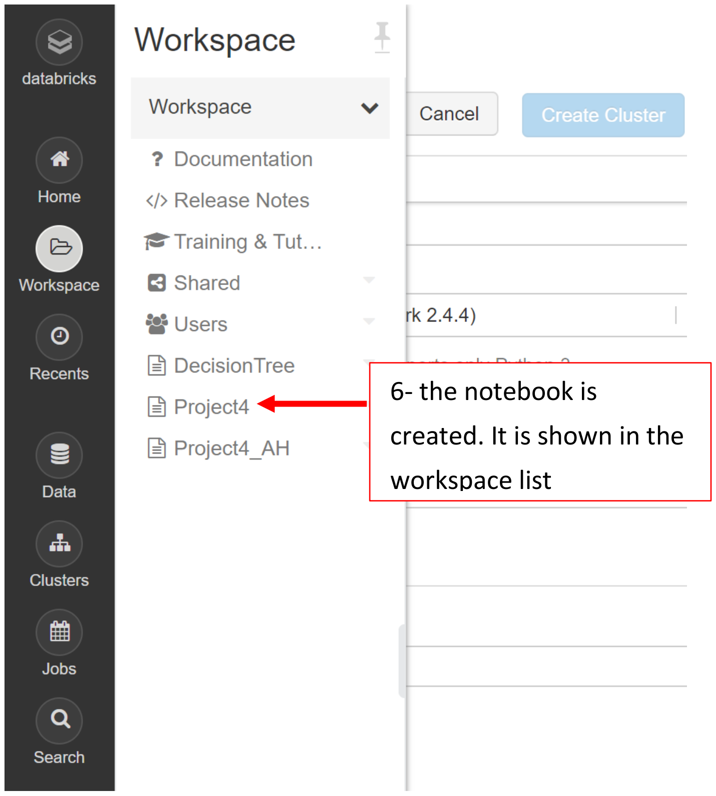
0 Workers: 0.0 GB Memory, 0 Cores, 0 DBU
1 Driver: 6.0 GB Memory, 0.88 Cores, 1 DBU

Create Notebook

Name: Project4

Language: Python

Cluster: My Cluster


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

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
```

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

```
employers = h1bDF.select('EMPLOYER_NAME').distinct()

employers.count()

Out[4]: 236015
```

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

```
employers.orderBy(['EMPLOYER_NAME'], descending=True).show(10)
 +--------------------+
 |       EMPLOYER_NAME|
 +--------------------+
 |"""EXCELLENT COMP...|
 |"""I HAVE A DREAM...|
 | """K"" LINE AMERICA|
 |"""K"" LINE AMERICA"|
 |"""K"" LINE LOGIS...|
 |"""K"" LINE LOGIS...|
 |"""K"" LINE LOGIS...|
 | "CREPE ""N"" TEARIA|
 |"CRISP MEDIA, INC...|
 |"E-2I ""EVOLUTION...|
 +--------------------+
 only showing top 10 rows
```

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.

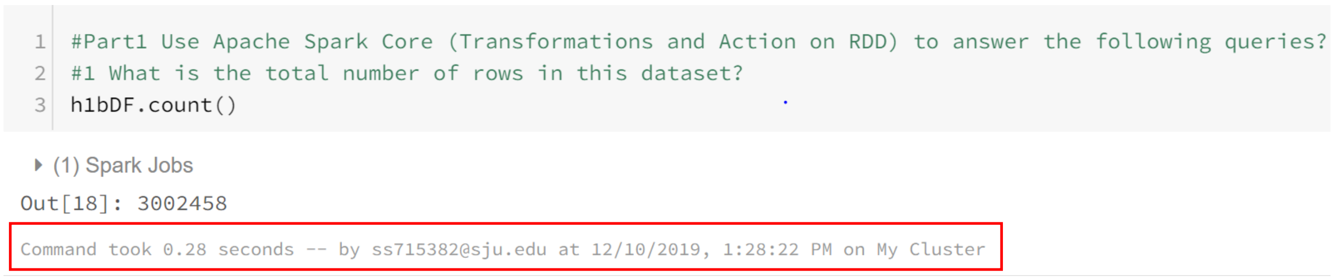
```
h1bDF.groupBy('CASE_STATUS').count().orderBy('count').show()

  +--------------------+-------+
  |         CASE_STATUS|  count|
  +--------------------+-------+
  |         INVALIDATED|      1|
  |            REJECTED|      2|
  |                  NA|     13|
  |PENDING QUALITY A...|     15|
  |           WITHDRAWN|  89799|
  |              DENIED|  94346|
  | CERTIFIED-WITHDRAWN| 202659|
  |           CERTIFIED|2615623|
  +--------------------+-------+
```

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
```

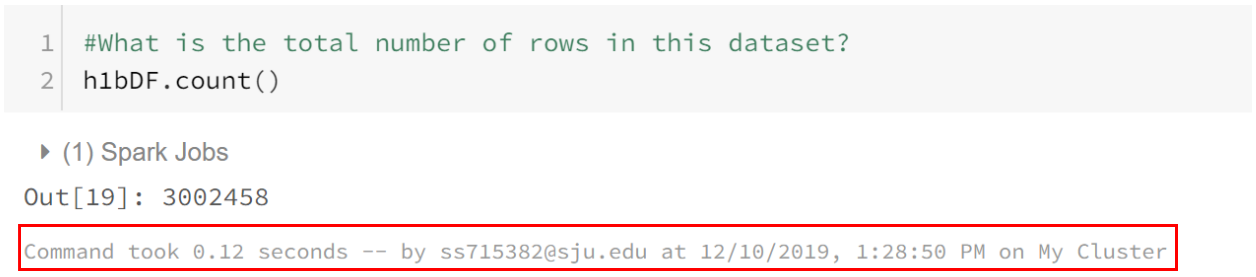


```
#Part1 Use Apache Spark Core (Transformations and Action on RDD) to answer the following queries?
#1 What is the total number of rows in this dataset?
h1bDF.count()
```

(1) Spark Jobs

```
Out[18]: 3002458
Command took 0.28 seconds -- by ss715382@sju.edu at 12/10/2019, 1:28:22 PM on My Cluster
```

Figure 1: Part 1-1 without cache



```
#What is the total number of rows in this dataset?
h1bDF.count()
```

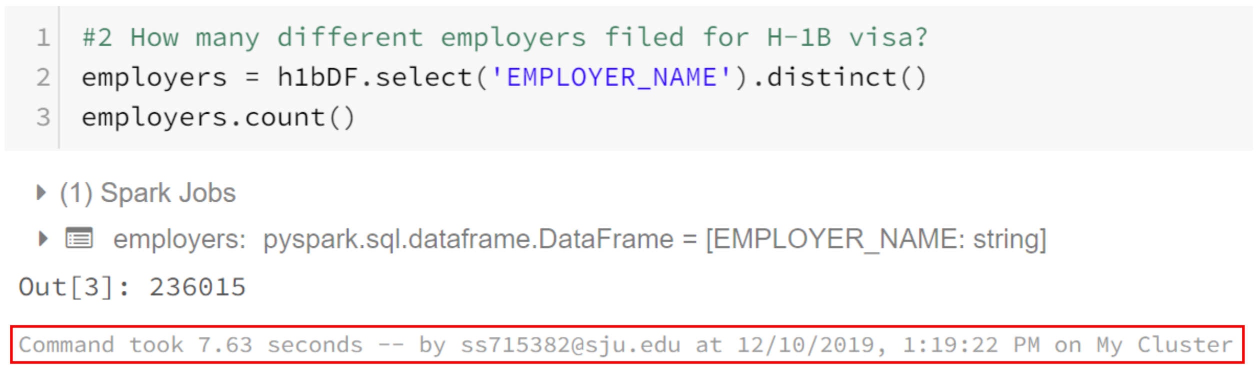
(1) Spark Jobs

```
Out[19]: 3002458
Command took 0.12 seconds -- by ss715382@sju.edu at 12/10/2019, 1:28:50 PM on My Cluster
```

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
```



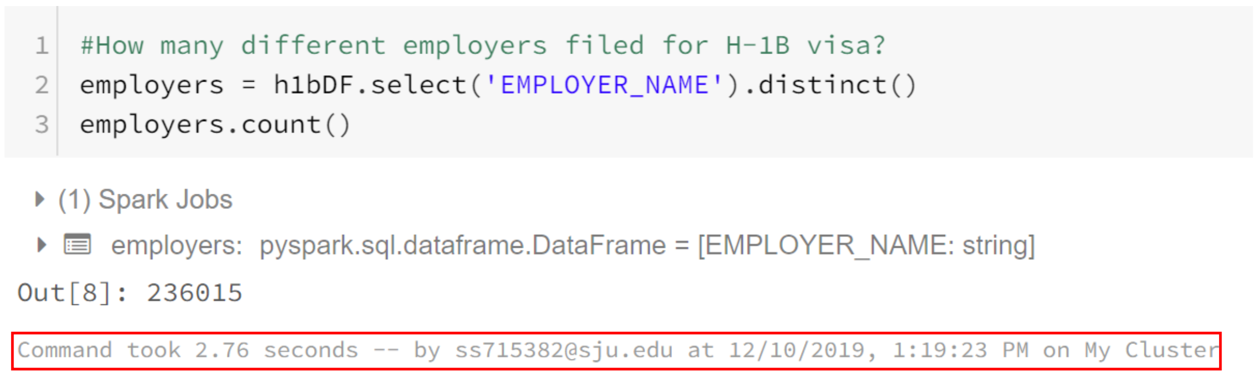
```
#2 How many different employers filed for H-1B visa?
employers = h1bDF.select('EMPLOYER_NAME').distinct()
employers.count()
```

(1) Spark Jobs

employers: pyspark.sql.dataframe.DataFrame = [EMPLOYER_NAME: string]

```
Out[3]: 236015
Command took 7.63 seconds -- by ss715382@sju.edu at 12/10/2019, 1:19:22 PM on My Cluster
```

Figure 4: Part 1-2 without cache



```
#How many different employers filed for H-1B visa?
employers = h1bDF.select('EMPLOYER_NAME').distinct()
employers.count()
```

(1) Spark Jobs

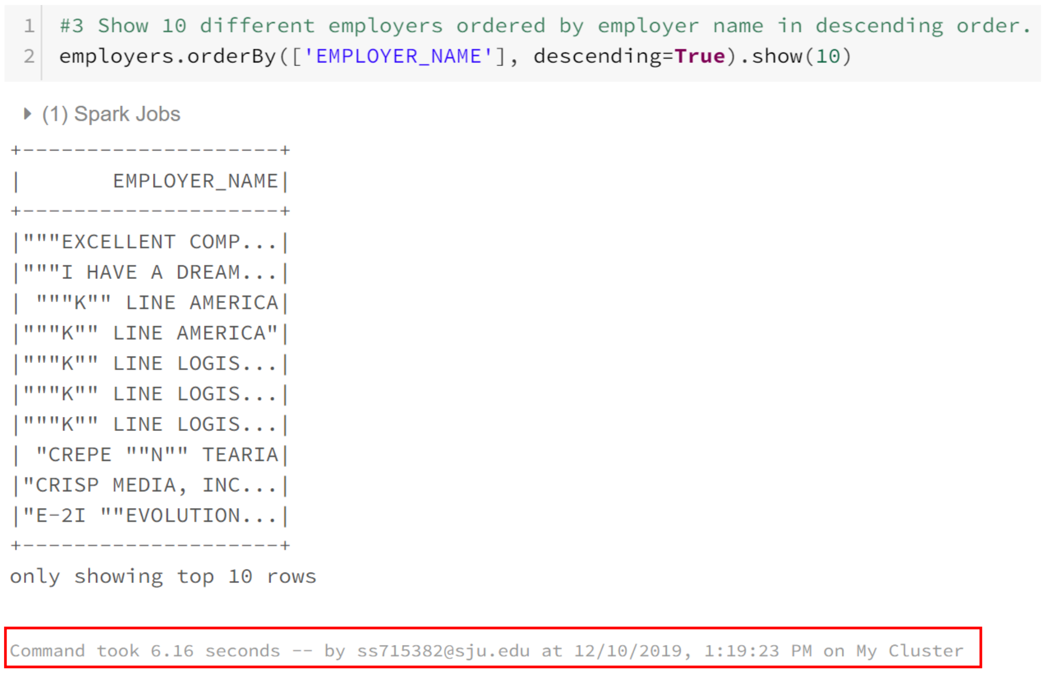
employers: pyspark.sql.dataframe.DataFrame = [EMPLOYER_NAME: string]

```
Out[8]: 236015
Command took 2.76 seconds -- by ss715382@sju.edu at 12/10/2019, 1:19:23 PM on My Cluster
```

Figure 3: Part 1-2 with cache

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

```
employers.orderBy(['EMPLOYER_NAME'], descending=True).show(10)
```



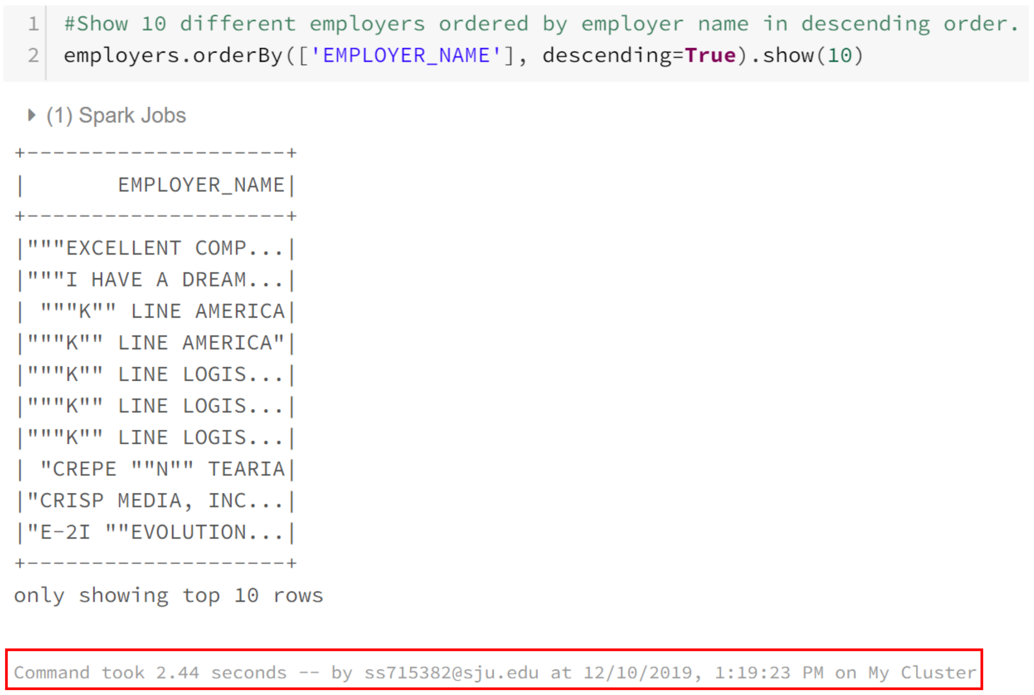
```
#3 Show 10 different employers ordered by employer name in descending order.
employers.orderBy(['EMPLOYER_NAME'], descending=True).show(10)
```

(1) Spark Jobs

```
+--------------------+
|       EMPLOYER_NAME|
+--------------------+
|"""EXCELLENT COMP...|
|"""I HAVE A DREAM...|
| """K"" LINE AMERICA|
|"""K"" LINE AMERICA"|
|"""K"" LINE LOGIS...|
|"""K"" LINE LOGIS...|
|"""K"" LINE LOGIS...|
| "CREPE ""N"" TEARIA|
|"CRISP MEDIA, INC...|
|"E-2I ""EVOLUTION...|
+--------------------+
only showing top 10 rows
```

Command took 6.16 seconds -- by ss715382@sju.edu at 12/10/2019, 1:19:23 PM on My Cluster

Figure 5: Part 1-3 without cache



```
#Show 10 different employers ordered by employer name in descending order.
employers.orderBy(['EMPLOYER_NAME'], descending=True).show(10)
```

(1) Spark Jobs

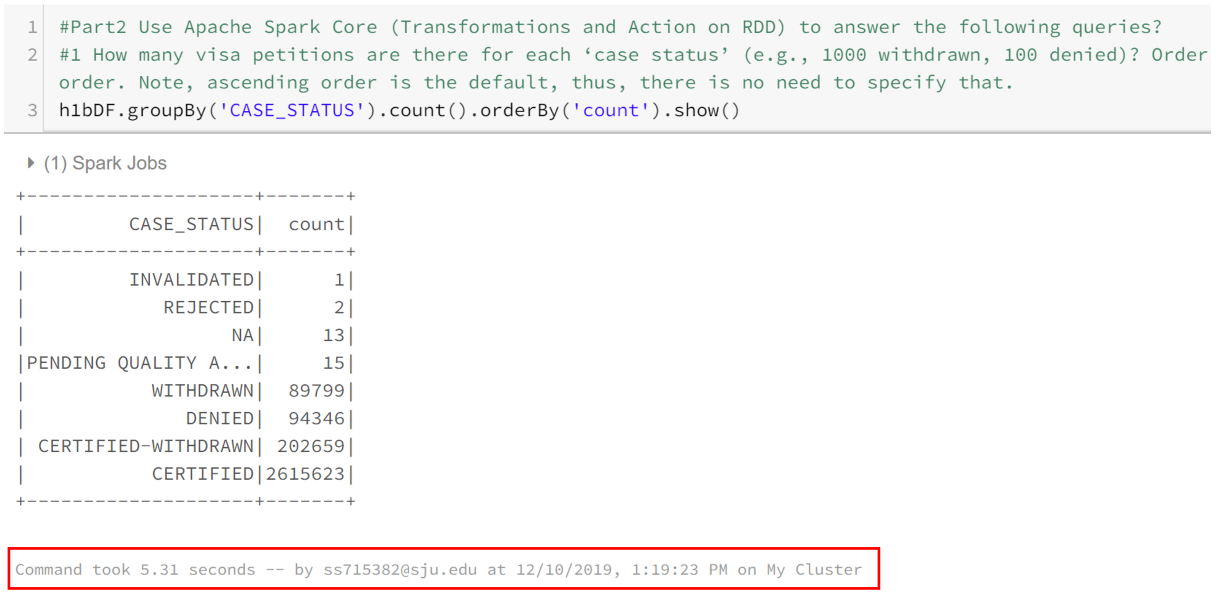
```
+--------------------+
|       EMPLOYER_NAME|
+--------------------+
|"""EXCELLENT COMP...|
|"""I HAVE A DREAM...|
| """K"" LINE AMERICA|
|"""K"" LINE AMERICA"|
|"""K"" LINE LOGIS...|
|"""K"" LINE LOGIS...|
|"""K"" LINE LOGIS...|
| "CREPE ""N"" TEARIA|
|"CRISP MEDIA, INC...|
|"E-2I ""EVOLUTION...|
+--------------------+
only showing top 10 rows
```

Command took 2.44 seconds -- by ss715382@sju.edu at 12/10/2019, 1:19:23 PM on My Cluster

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()
```

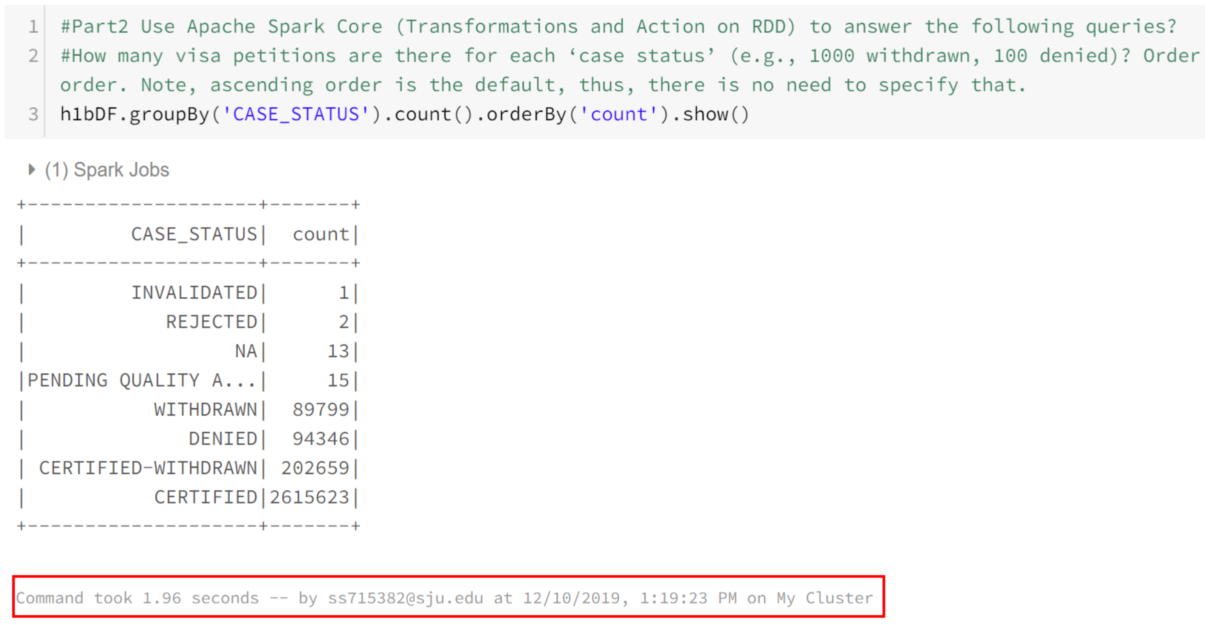


```
#Part2 Use Apache Spark Core (Transformations and Action on RDD) to answer the following queries?
#1 How many visa petitions are there for each ‘case status’ (e.g., 1000 withdrawn, 100 denied)? Order
order. Note, ascending order is the default, thus, there is no need to specify that.
h1bDF.groupBy('CASE_STATUS').count().orderBy('count').show()
```

(1) Spark Jobs

```
+--------------------+-------+
|         CASE_STATUS|  count|
+--------------------+-------+
|         INVALIDATED|      1|
|            REJECTED|      2|
|                  NA|     13|
|PENDING QUALITY A...|     15|
|           WITHDRAWN|  89799|
|              DENIED|  94346|
| CERTIFIED-WITHDRAWN| 202659|
|           CERTIFIED|2615623|
+--------------------+-------+

Command took 5.31 seconds -- by ss715382@sju.edu at 12/10/2019, 1:19:23 PM on My Cluster
```

Figure 7: Part 2-1 without cache



```
#Part2 Use Apache Spark Core (Transformations and Action on RDD) to answer the following queries?
#How many visa petitions are there for each ‘case status’ (e.g., 1000 withdrawn, 100 denied)? Order
order. Note, ascending order is the default, thus, there is no need to specify that.
h1bDF.groupBy('CASE_STATUS').count().orderBy('count').show()
```

(1) Spark Jobs

```
+--------------------+-------+
|         CASE_STATUS|  count|
+--------------------+-------+
|         INVALIDATED|      1|
|            REJECTED|      2|
|                  NA|     13|
|PENDING QUALITY A...|     15|
|           WITHDRAWN|  89799|
|              DENIED|  94346|
| CERTIFIED-WITHDRAWN| 202659|
|           CERTIFIED|2615623|
+--------------------+-------+

Command took 1.96 seconds -- by ss715382@sju.edu at 12/10/2019, 1:19:23 PM on My Cluster
```

Figure 8: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

2. 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)
```

```
Row(EMPLOYER_NAME='UNIVERSITY OF MICHIGAN')
Row(EMPLOYER_NAME='GOODMAN NETWORKS, INC.')
Row(EMPLOYER_NAME='PORTS AMERICA GROUP, INC.')
Row(EMPLOYER_NAME='GATES CORPORATION, A WHOLLY-OWNED SUBSIDIARY OF TOMKINS
PLC')
Row(EMPLOYER_NAME='PEABODY INVESTMENTS CORP.')
Row(EMPLOYER_NAME='BURGER KING CORPORATION')
Row(EMPLOYER_NAME='BT AND MK ENERGY AND COMMODITIES')
Row(EMPLOYER_NAME='GLOBO MOBILE TECHNOLOGIES, INC.')
Row(EMPLOYER_NAME='ESI COMPANIES INC.')
```

```
Row(EMPLOYER_NAME='LESSARD INTERNATIONAL LLC')
Row(EMPLOYER_NAME='H.J. HEINZ COMPANY')
Row(EMPLOYER_NAME='DOW CORNING CORPORATION')
Row(EMPLOYER_NAME='ACUSHNET COMPANY')
Row(EMPLOYER_NAME='BIOCAIR, INC.')
Row(EMPLOYER_NAME='NEWMONT MINING CORPORATION')
Row(EMPLOYER_NAME='VRICON, INC.')
Row(EMPLOYER_NAME='CARDIAC SCIENCE CORPORATION')
Row(EMPLOYER_NAME='WESTFIELD CORPORATION')
Row(EMPLOYER_NAME='QUICKLOGIX LLC')
Row(EMPLOYER_NAME='MCCHRYSTAL GROUP, LLC')
```

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.

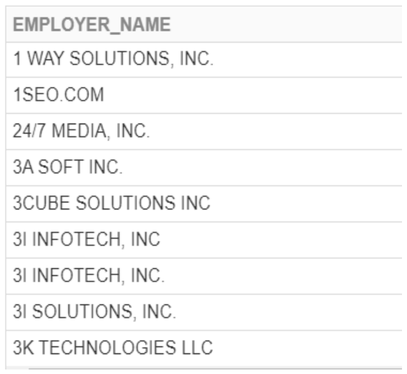
```
# Create a VIEW to perform SQL queries
#done the RDD cahche in the above part
h1bDF.createOrReplaceTempView("h1bView")
# cache table
spark.catalog.cacheTable("h1bView")
```

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
```

2. 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;
```



EMPLOYER_NAME
1 WAY SOLUTIONS, INC.
1SEO.COM
24/7 MEDIA, INC.
3A SOFT INC.
3CUBE SOLUTIONS INC
3I INFOTECH, INC
3I INFOTECH, INC.
3I SOLUTIONS, INC.
3K TECHNOLOGIES LLC

3. 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, are composed 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 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 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:

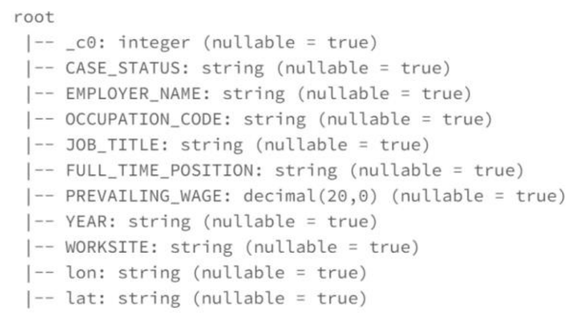
```scala
// One method for defining the schema of an RDD is to make a case class with the desired column
// names and types.
case class Record(key: Int, value: String)

val sc: SparkContext // An existing spark context.
val sqlContext = new SQLContext(sc)

// Importing the SQL context gives access to all the SQL functions and implicit conversions.
import sqlContext._

val rdd = sc.parallelize((1 to 100).map(i => Record(i, s"val_$i")))
// Any RDD containing case classes can be registered as a table.  The schema of the table is
// automatically inferred using scala reflection.
rdd.registerAsTable("records")

val results: SchemaRDD = sql("SELECT * FROM records")
```

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.

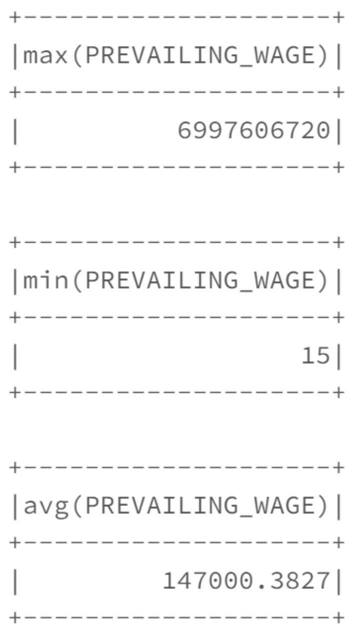
```
h1bDF = h1bDF.withColumnRenamed('SOC_NAME', 'OCCUPATION_CODE')
#convert PREVAILING_WAGE from datatype to decimal
h1bDF = h1bDF.withColumn('PREVAILING_WAGE',
h1bDF.PREVAILING_WAGE.cast(DecimalType(20)))
h1bDF.printSchema()
```



```
root
 |-- _c0: integer (nullable = true)
 |-- CASE_STATUS: string (nullable = true)
 |-- EMPLOYER_NAME: string (nullable = true)
 |-- OCCUPATION_CODE: string (nullable = true)
 |-- JOB_TITLE: string (nullable = true)
 |-- FULL_TIME_POSITION: string (nullable = true)
 |-- PREVAILING_WAGE: decimal(20,0) (nullable = true)
 |-- YEAR: string (nullable = true)
 |-- WORKSITE: string (nullable = true)
 |-- lon: string (nullable = true)
 |-- lat: string (nullable = true)
```

2. 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()
```



```
+--------------------+
|max(PREVAILING_WAGE)|
+--------------------+
|          6997606720|
+--------------------+

+--------------------+
|min(PREVAILING_WAGE)|
+--------------------+
|                  15|
+--------------------+

+--------------------+
|avg(PREVAILING_WAGE)|
+--------------------+
|         147000.3827|
+--------------------+
```

3. 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()
```

```
+---------------+-------------+
|PREVAILING_WAGE|wage_grt80000|
+---------------+-------------+
|         187200|          Yes|
|         241842|          Yes|
|          99986|          Yes|
|          99986|          Yes|
|         187200|          Yes|
|         215862|          Yes|
|         192088|          Yes|
|          95296|          Yes|
|         149594|          Yes|
|         226699|          Yes|
|         187200|          Yes|
|         159370|          Yes|
|          98550|          Yes|
|         130853|          Yes|
|          52416|           No|
|         130853|          Yes|
|          89107|          Yes|
|         130853|          Yes|
|         102190|          Yes|
|         197683|          Yes|
+---------------+-------------+
only showing top 20 rows
```

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")
```

```
▸ (1) Spark Jobs
 Yes, there are wages greater than $80,000 : 789875
```

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