

# Machine Learning with Pyspark

---

## Part 1: Create a databricks account.

Step 1: Open the link:

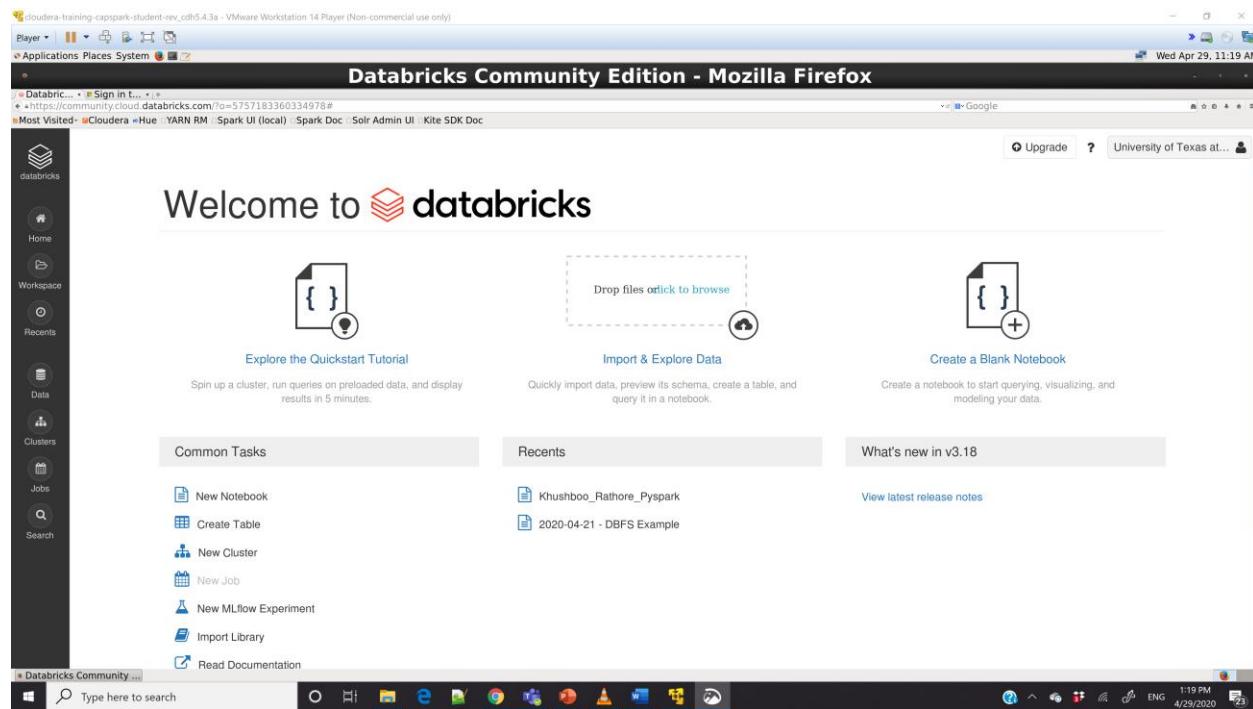
<https://community.cloud.databricks.com/login.html>

Click on sign up.

Step 2: Verify and confirm the databricks account from the email you have received from Databricks.

Go to the link again and click on “Sign In”.

**Q. Login to your databricks account and take a screenshot of the Home page output and paste it below.**

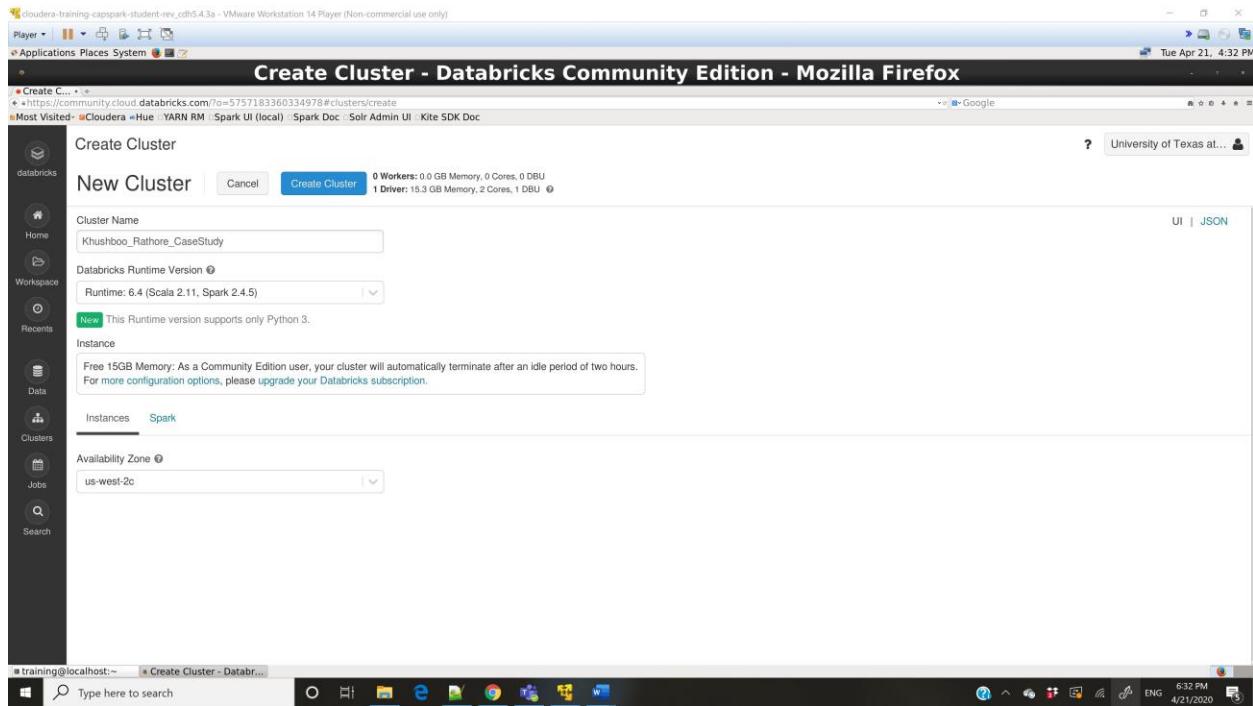


## Part 2: Create a cluster node.

Step 3: Click on the “Clusters” icon as shown in the screenshot below:

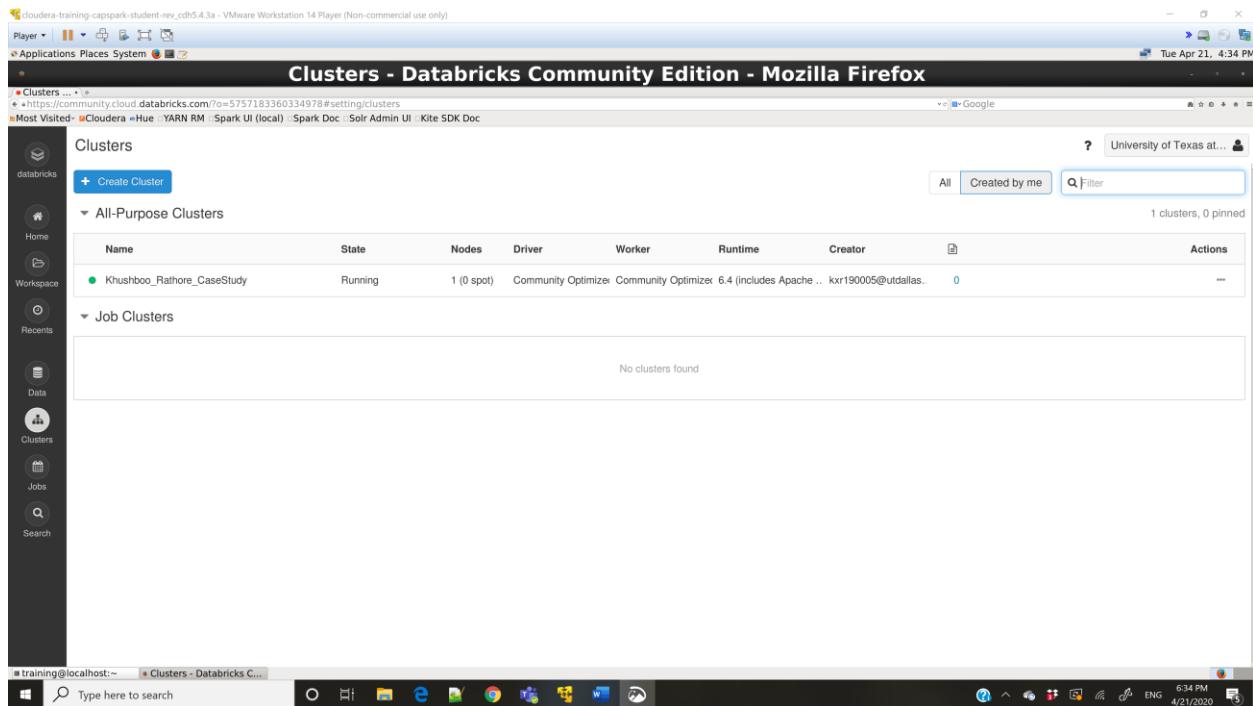
Step 4: Name the cluster as “FirstName\_LastName\_CaseStudy”.

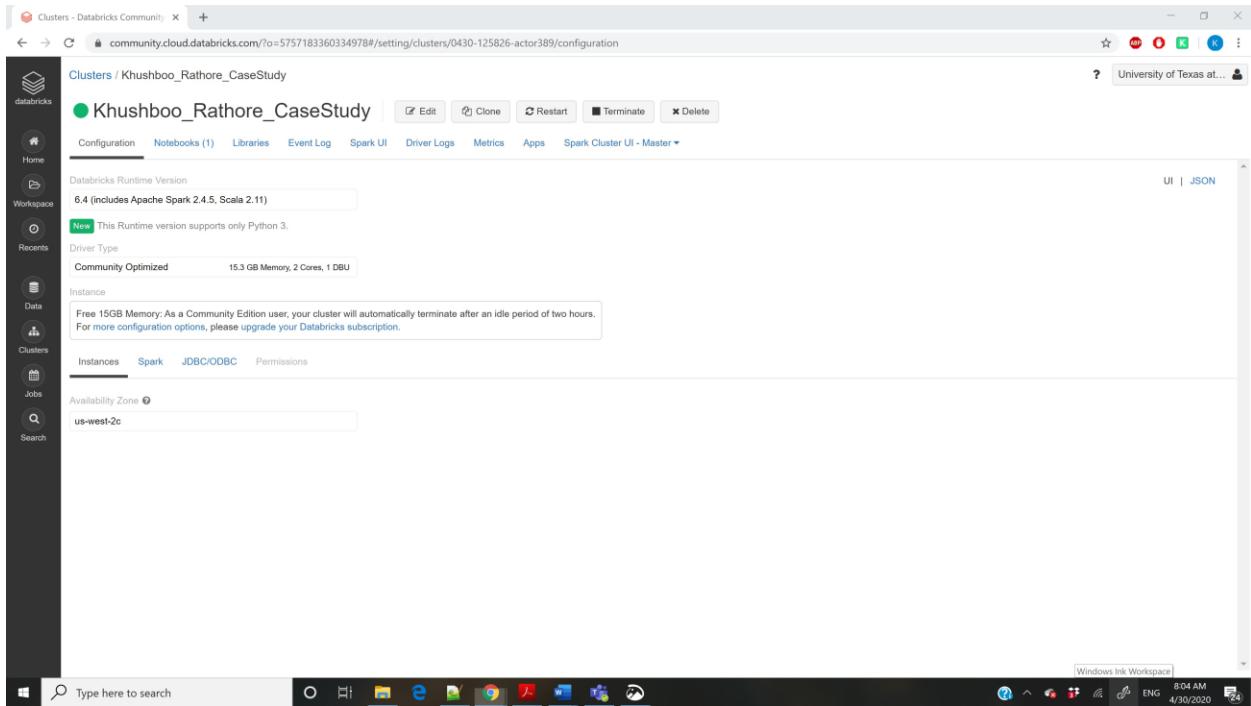
**Q. Take a screenshot of the Create Cluster page and paste it below.**



Step 5: Green sign shows that a cluster is created and running successfully.

Q. Take a screenshot of the running Cluster Page and paste it below.



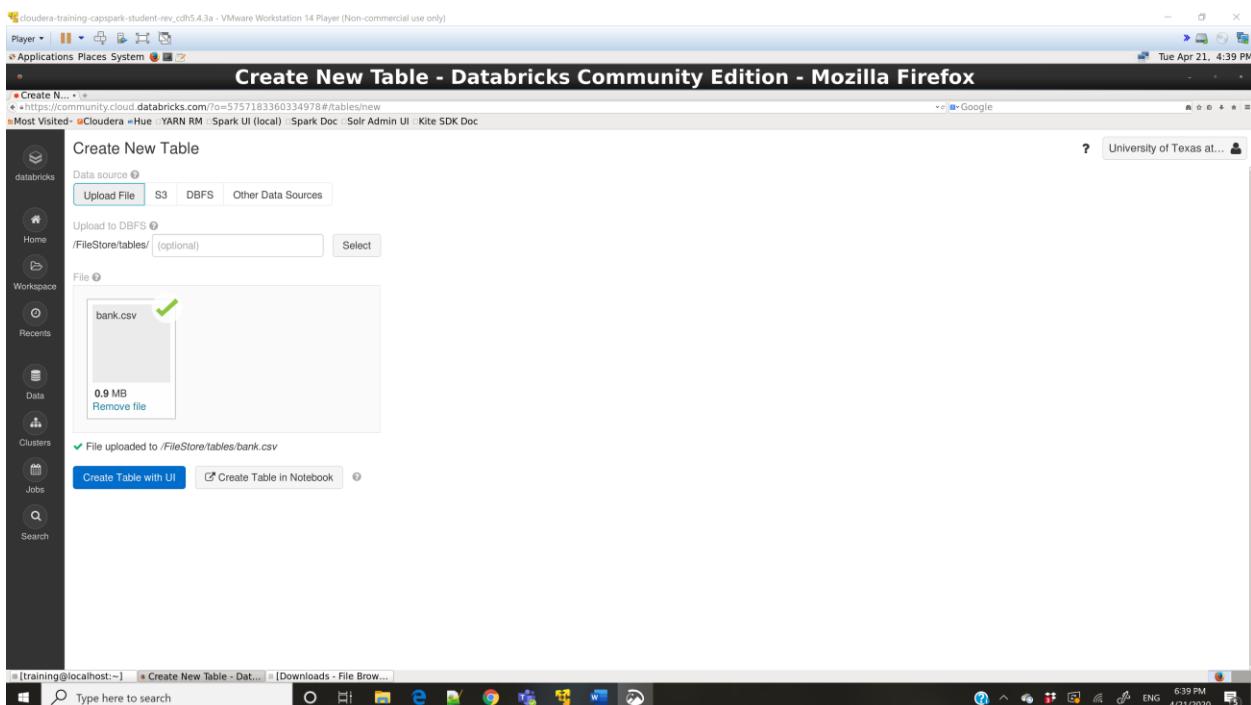


### Part 3: Create Datatable.

Step 6: Download the “Bank.csv” file from eLearning.

Step 7: Click on browse and select the “bank.csv” file that you downloaded from eLearning.

Take a screenshot of the uploaded file and paste it below.



Step 8: Click on “Create Table in Notebook”.

Step 9: The file location can now be seen.

Q. Take a screenshot of the table created and paste it below.

Copy the file location and paste it here.

file\_location = "/FileStore/tables/bank.csv"

The screenshot shows a Databricks notebook interface. On the left, there's a sidebar with icons for Databricks, Home, Workspace, Data, Clusters, Jobs, and Search. The main area has a title bar "2020-04-21 - DBFS Example - Databricks Community Edition - Mozilla Firefox". Below the title bar, the URL is https://community.cloud.databricks.com/#/o=5757183360334978#notebook/3271675534905606/command/3271675534905607. The notebook content starts with a Python cell containing:

```
1 %sql
2
3 /* Query the created temp table in a SQL cell */
4
5 select * from `bank_csv`
```

Below the code, there's a table titled "(1) Spark Jobs" with 16 columns labeled c0 through c16. The table contains 16 rows of data. At the bottom of the table, it says "Showing the first 1000 rows." The table structure is as follows:

c0	c1	c2	c3	c4	c5	c6	c7	c8	c9	c10	c11	c12	c13	c14	c15	c16
age	job	marital	education	default	balance	housing	loan	contact	day	month	duration	campaign	pdays	previous	poutcome	deposit
59	admin.	married	secondary	no	2343	yes	no	unknown	5	may	1042	1	-1	0	unknown	yes
56	admin.	married	secondary	no	45	no	no	unknown	5	may	1467	1	-1	0	unknown	yes
41	technician	married	secondary	no	1270	yes	no	unknown	5	may	1389	1	-1	0	unknown	yes
55	services	married	secondary	no	2476	yes	no	unknown	5	may	579	1	-1	0	unknown	yes
54	admin.	married	tertiary	no	184	no	no	unknown	5	may	673	2	-1	0	unknown	yes
42	management	single	tertiary	no	0	yes	yes	unknown	5	may	562	2	-1	0	unknown	yes
56	management	married	tertiary	no	830	yes	yes	unknown	6	may	1201	1	-1	0	unknown	yes
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	

At the bottom of the notebook, there's a command cell with the following content:

```
1 # With this registered as a temp view, it will only be available to this particular notebook. If you'd like other users to be able to query this table, you can also create a table from the DataFrame.
2 # Once saved, this table will persist across cluster restarts as well as allow various users across different notebooks to query this data.
3 # To do so, choose your table name and uncomment the bottom line.
```

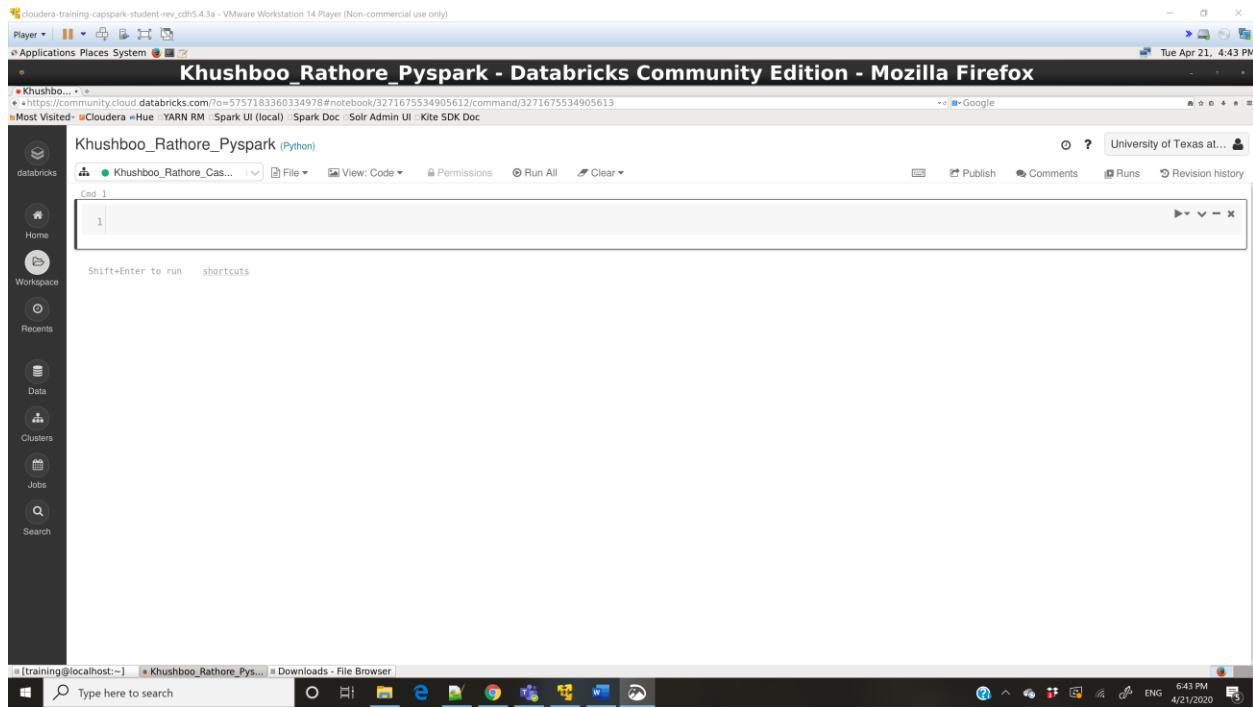
The status bar at the bottom right shows the date as 4/21/2020 and the time as 4:45 PM.

## Part 4 : DataFrame Operations.

Step 10: Create new Notebook.

Step 11: Create the notebook with “FirstName\_LastName\_Pyspark”.

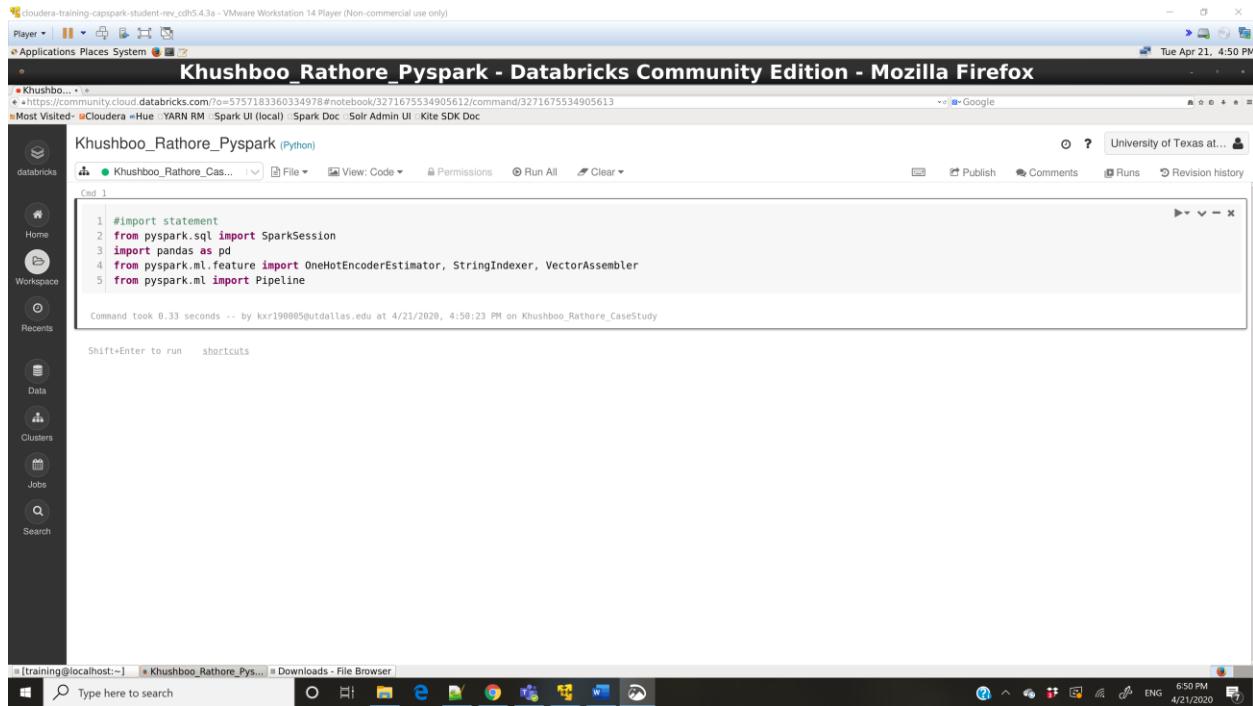
Q. Take a screenshot of the blank notebook and paste it below.



```
1
```

Step 11: Import the required libraries from Spark.

Q. Take a screenshot of the output of the code and paste it below.



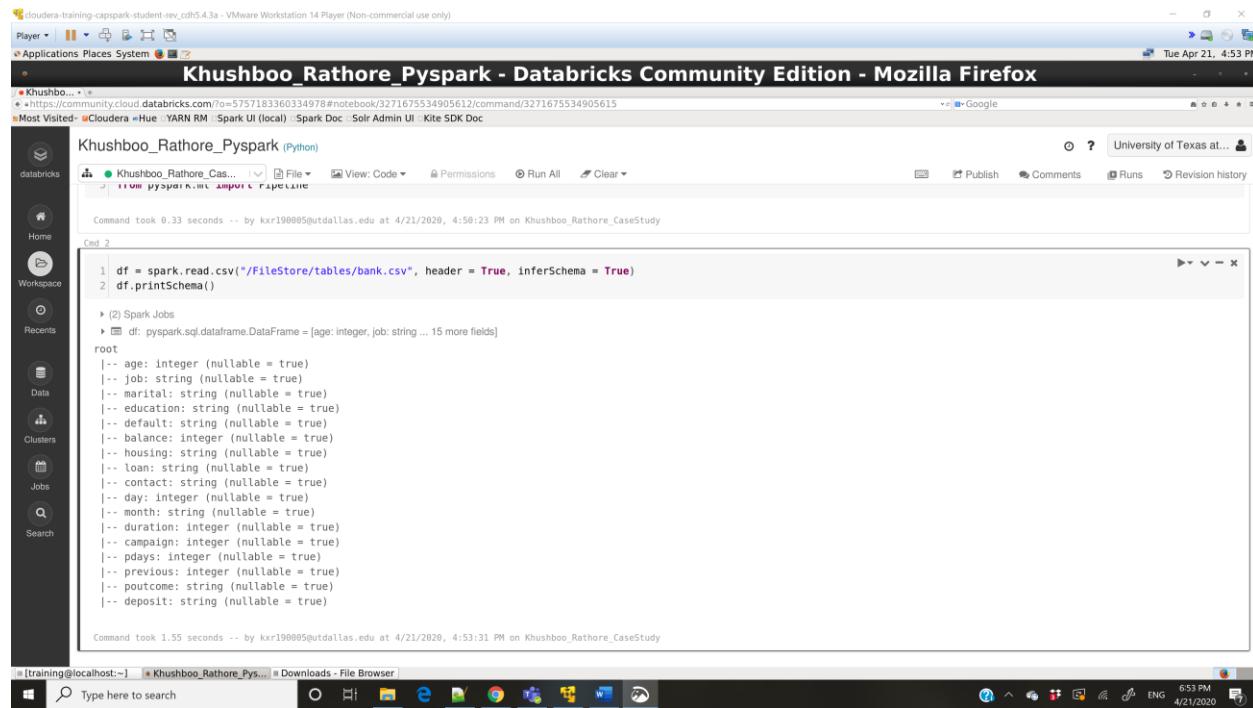
```
1 #import statement
2 from pyspark.sql import SparkSession
3 import pandas as pd
4 from pyspark.ml.feature import OneHotEncoderEstimator, StringIndexer, VectorAssembler
5 from pyspark.ml import Pipeline
```

Command took 0.33 seconds -- by kar190005@utdallas.edu at 4/21/2020, 4:50:23 PM on Khushboo\_Rathore\_CaseStudy

Step 12:

Q. Take a screenshot of the output of the code and paste it below.

The input variables are age, job, marital, education, default, balance, housing, loan, contact, day, month, duration, campaign, pdays, previous, poutcome



```
1 df = spark.read.csv("/FileStore/tables/bank.csv", header = True, inferSchema = True)
2 df.printSchema()

> (2) Spark Jobs
> df: pyspark.sql.dataframe.DataFrame = [age: integer, job: string ... 15 more fields]
root
 |-- age: integer (nullable = true)
 |-- job: string (nullable = true)
 |-- marital: string (nullable = true)
 |-- education: string (nullable = true)
 |-- default: string (nullable = true)
 |-- balance: integer (nullable = true)
 |-- housing: string (nullable = true)
 |-- loan: string (nullable = true)
 |-- contact: string (nullable = true)
 |-- day: integer (nullable = true)
 |-- month: string (nullable = true)
 |-- duration: integer (nullable = true)
 |-- campaign: integer (nullable = true)
 |-- pdays: integer (nullable = true)
 |-- previous: integer (nullable = true)
 |-- poutcome: string (nullable = true)
 |-- deposit: string (nullable = true)
```

Step 13: Type the code below in the next cell and Run the cell.

Q. Take a screenshot of the output of the code and paste it below.

i. Describe “Select”, “Group by” operations in one line. –

Select statement is to show number of records from defined columns (age, balance, and deposit) in dataframe df.

Group by is used to form groups based on values of “age” column for aggregate function “count”. First groupby operation finds count of customers for every age group and sorts them in ascending order and second groupby finds mean of balance for every marital status.

ii. How many customers are there with an age greater than 40? - There are 4967 customers with age > 40, but it is not printed in the output because show() command is not used.

iii. Why are there only 5 records viewable? Due to command show(5)

cloudera-training-cappspark-student-rev\_cdh5.4.3a - VMware Workstation 14 Player (Non-commercial use only)

Player • Applications Places System

**Khushboo\_Rathore\_Pyspark - Databricks Community Edition - Mozilla Firefox**

Tue Apr 21, 7:17 PM

Most Visited: Cloudera Hue YARN RM Spark UI (local) Spark Doc Solr Admin UI Kite SDK Doc

Khushboo Rathore Pyspark (Python)

Detached File View: Code Permissions Run All Clear

```

1 #DF operations
2 df.head(5)
3 df.select('age','balance','deposit').show(5)
4 df.groupby("age").count().sort("count",ascending=True).show()
5 df.describe().show()
6 df.describe('balance').show()
7 df.filter(df.age > 40).count()
8 df.groupby('marital').agg({'balance':'mean'}).show()

* (1) Spark Jobs
+-----+
|age|balance|deposit|
+---+---+---+
| 59| 2343| yes|
| 56|   45| yes|
| 41| 1270| yes|
| 55| 2476| yes|
| 54|  184| yes|
+---+---+---+
only showing top 5 rows

+---+---+
|age|count|
+---+---+
| 95|  1|
| 89|  1|
| 98|  2|
| 92|  2|
| 93|  2|
| 88|  2|
| 87|  4|
+---+---+

```

Command took 17.70 seconds -- by kxr190005@utdallas.edu at 4/21/2020, 7:16:56 PM on Khushboo\_Rathore\_CaseStudy

[training@localhost:~] \* Khushboo\_Rathore\_Pys... \* Downloads - File Browser

Type here to search

9:17 PM 4/21/2020 ENG

Khushboo\_Rathore\_Pyspark - Databricks Community Edition - Mozilla Firefox

community.cloud.databricks.com/?o=5757183360334978#notebook/3271675534905612/command/3271675534905612

University of Texas at...

Khushboo\_Rathore\_CaseStudy (Python)

File Edit View: Code Permissions Run All Clear

Command took 11.29 seconds -- by kxr190005@utdallas.edu at 4/30/2020, 7:58:26 AM on Khushboo\_Rathore\_CaseStudy

Cmd 3

```

1 #DF operations
2 df.head(5)
3 df.select('age','balance','deposit').show(5)
4 df.groupby("age").count().sort("count",ascending=True).show()
5 df.describe().show()
6 df.describe('balance').show()
7 df.filter(df.age > 40).count()
8 df.groupby('marital').agg({'balance':'mean'}).show()

* (1) Spark Jobs
+-----+
|summary| age| job| marital|education|default| balance|housing| loan| contact| day|month| duration| campaign| pdays| prev
|ous|poutcome|deposit|
+-----+
| count| 11162| 11162| 11162| 11162| 11162| 11162| 11162| 11162| 11162| 11162| 11162| 11162| 11162| 11162| 1
|1162| 11162| 11162| | mean|41.231947679627304| null| null| null| 1528.5385235620856| null| null| null| 15.658036194230425| null|371.99381831213043| 2.5088421429851281| 51.33040673714388|0.832556889446
|3358| null| null| | stdDev|11.91336919321518| null| null| null| 3225.413325946149| null| null| null| 8.420739541806462| null|347.12838571630687|2.7220771816614824|168.75828197197717| 2.29208721867
|0588| null| null| | min| 18| admin.|divorced| primary| no| -6847| no| no|cellular| 1| apr| 2| 1| -1|
|0| failure| no| | max| 95|unknown| single| unknown| yes| 81204| yes| yes| unknown| 31| sep| 3881| 63| 854|
|58| unknown| yes| +-----+

```

Command took 17.92 seconds -- by kxr190005@utdallas.edu at 4/30/2020, 7:58:27 AM on Khushboo\_Rathore\_CaseStudy

Cmd 4

1 #Panda Dataframe

Type here to search

8:12 AM 4/30/2020 ENG

```

1 #DF operations
2 df.head(5)
3 df.select('age','balance','deposit').show(5)
4 df.groupBy("age").count().sort("count",ascending=True).show()
5 df.describe().show()
6 df.describe('balance').show()
7 df.filter(df.age > 40).count()
8 df.groupby("marital").agg({'balance':'mean'}).show()

> (1) Spark Jobs
+-----+
| summary | balance |
+-----+
| count | 11162 |
| mean | 1528.5385235620856 |
| stdev | 3225.413325946149 |
| min | -6847 |
| max | 81294 |
+-----+


+-----+
| marital | avg(balance) |
+-----+
| divorced | 1371.8352668213456 |
| married | 1599.9275704613447 |
| single | 1457.2525386969966 |
+-----+


Command took 11.29 seconds -- by kxr190005@utdallas.edu at 4/30/2020, 7:58:26 AM on Khushboo_Rathore_CaseStudy

```

Command took 17.92 seconds -- by kxr190005@utdallas.edu at 4/30/2020, 7:58:27 AM on Khushboo\_Rathore\_CaseStudy

**Step 14: Execute the following code:**

**Q. Take a screenshot of the output of the code and paste it below.**

**What is seen in the output of the transpose?**

The 5 rows are now converted into columns and the columns are now shown as rows

```

1 #Panda Dataframe
2 pd.DataFrame(df.take(5), columns=df.columns).transpose()

> (1) Spark Jobs
Out[6]:

```

	0	1	2	3	4
age	59	56	41	55	54
job	admin.	admin.	technician	services	admin.
marital	married	married	married	married	married
education	secondary	secondary	secondary	secondary	tertiary
default	no	no	no	no	no
balance	2343	45	1270	2476	184
housing	yes	no	yes	yes	no
loan	no	no	no	no	no
contact	unknown	unknown	unknown	unknown	unknown
day	5	5	5	5	5
month	may	may	may	may	may
duration	1042	1467	1389	579	673
campaign	1	1	1	1	2
pdays	-1	-1	-1	-1	-1
previous	0	0	0	0	0
poutcome	unknown	unknown	unknown	unknown	unknown
deposit	yes	yes	yes	yes	yes

Step 15: Execute the following code:

Q. Take a screenshot of the output of the code and paste it below.

```
1 numeric_features = [t[0] for t in df.dtypes if t[1] == 'int']
2 df.select(numeric_features).describe().toPandas().transpose()

> (2) Spark Jobs
Out[9]:
```

	0	1	2	3	4
summary	count	mean	stddev	min	max
age	11162	41.231947679627304	11.913369192215518	18	95
balance	11162	1528.5385235620856	3225.413325946149	-6847	81204
day	11162	15.658036194230425	8.42073954106462	1	31
duration	11162	371.99381831213043	347.12838571630687	2	3881
campaign	11162	2.508421429851281	2.7220771816614824	1	63
pdays	11162	51.39040673714388	108.75828197197717	-1	854
previous	11162	0.8325568894463358	2.292007218670508	0	58

Part 5: Data Preprocessing:

Step 16: Execute the following code:

Q. Take a screenshot of the output of the code and paste it below.

```

1 categoricalColumns = ['job','marital','education','default','housing','loan','contact','poutcome']
2 stages = []
3 for categoricalCol in categoricalColumns:
4     stringIndexer = StringIndexer(inputCol = categoricalCol, outputCol = categoricalCol + ' Index')
5     encoder = OneHotEncoderEstimator(inputCols=[stringIndexer.getOutputCol()], outputCols=[categoricalCol + "classVec"])
6     stages += [stringIndexer, encoder]
7 label_stringIdx = StringIndexer(inputCol = 'deposit', outputCol = 'label')
8 stages += [label_stringIdx]
9 numericCols = ['age','balance','duration','campaign','pdays','previous']
10 assemblerInputs = [c + "classVec" for c in categoricalColumns] + numericCols
11 assembler = VectorAssembler(inputCols=assemblerInputs, outputCol="features")
12 stages += [assembler]

```

Command took 0.19 seconds -- by kxr190005@utdallas.edu at 4/21/2020, 7:59:42 PM on Khushboo\_Rathore\_CaseStudy

Shift+Enter to run    shortcuts

**Step 17: Execute the following code:**

**Q. Take a screenshot of the output of the code and paste it below.**

```

1 pipeline = Pipeline(stages = stages)
2 pipelineModel = pipeline.fit(df)
3 df = pipelineModel.transform(df)
4 df.printSchema()

```

(8) Spark Jobs

```

root
 |-- age: integer (nullable = true)
 |-- job: string (nullable = true)
 |-- marital: string (nullable = true)
 |-- education: string (nullable = true)
 |-- default: string (nullable = true)
 |-- balance: integer (nullable = true)
 |-- housing: string (nullable = true)
 |-- loan: string (nullable = true)
 |-- contact: string (nullable = true)
 |-- day: integer (nullable = true)
 |-- month: string (nullable = true)
 |-- duration: integer (nullable = true)
 |-- campaign: integer (nullable = true)
 |-- pdays: integer (nullable = true)
 |-- previous: integer (nullable = true)
 |-- poutcome: string (nullable = true)
 |-- deposit: string (nullable = true)
 |-- jobIndex: double (nullable = false)
 |-- jobClassVec: vector (nullable = true)
 |-- maritalIndex: double (nullable = false)

```

Command took 6.53 seconds -- by kxr190005@utdallas.edu at 4/21/2020, 8:05:33 PM on Khushboo\_Rathore\_CaseStudy

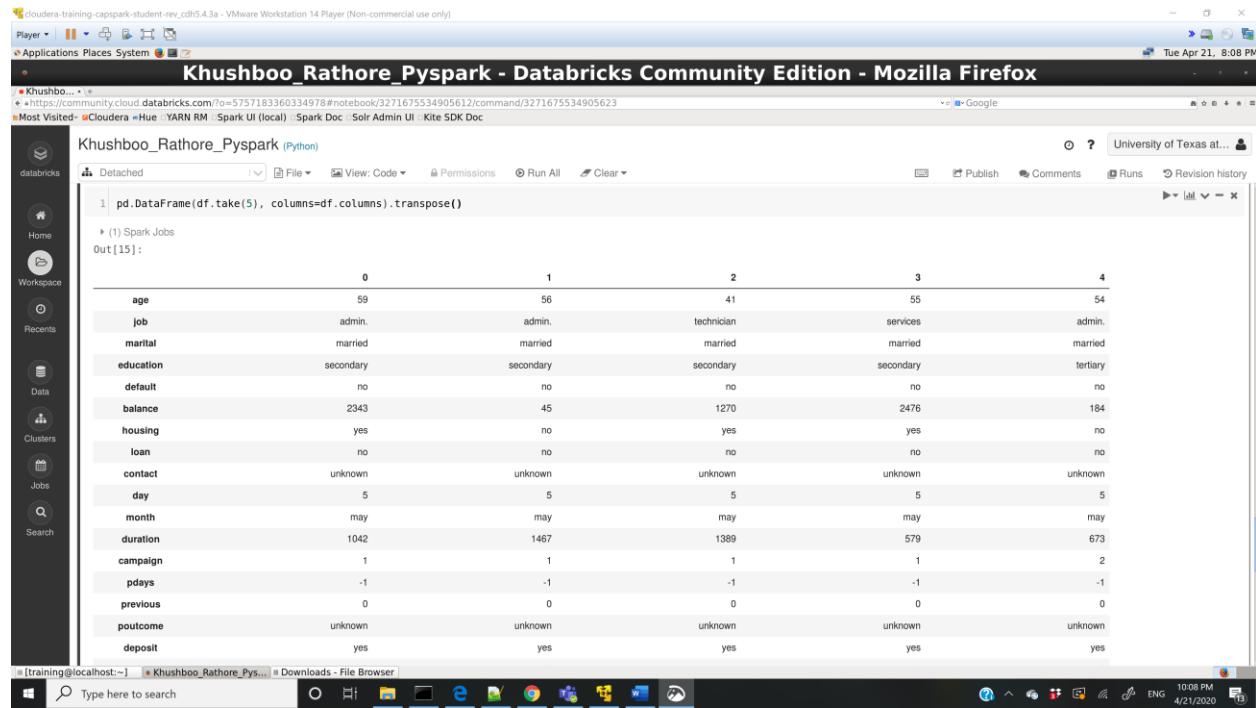
Shift+Enter to run    shortcuts

**Step 18: Execute the following command:**

**Q. Take a screenshot of the output of the code and paste it below.**

## Define the purpose of the transpose function?

To represent the label names of the columns as rows and their corresponding values can be seen as columns for 5 records.



The screenshot shows a Databricks notebook titled "Khushboo\_Rathore\_Pyspark - Databricks Community Edition - Mozilla Firefox". The code cell contains:

```
1 pd.DataFrame(df.take(5), columns=df.columns).transpose()
```

The output, labeled "Out[15]:" is a table with 15 rows and 5 columns:

	0	1	2	3	4
age	59	56	41	55	54
job	admin.	admin.	technician	services	admin.
marital	married	married	married	married	married
education	secondary	secondary	secondary	secondary	tertiary
default	no	no	no	no	no
balance	2343	45	1270	2476	184
housing	yes	no	yes	yes	no
loan	no	no	no	no	no
contact	unknown	unknown	unknown	unknown	unknown
day	5	5	5	5	5
month	may	may	may	may	may
duration	1042	1467	1389	579	673
campaign	1	1	1	1	2
pdays	-1	-1	-1	-1	-1
previous	0	0	0	0	0
poutcome	unknown	unknown	unknown	unknown	unknown
deposit	yes	yes	yes	yes	yes

## Part 6: Implementation of k-means algorithm

Step 19: Execute the following code:

Q. Take a screenshot of the output of the code and paste it below.

```

1 train, test = df.randomSplit([0.8,0.2], seed = 99999)
2 from pyspark.ml.clustering import KMeans
3 import numpy as np
4 cost = np.zeros(10)
5 for k in range(2,10):
6     kmeans = KMeans().setK(k).setSeed(1)
7     model = kmeans.fit(train)
8     cost[k] = model.computeCost(train)

▶ (58) Spark Jobs
▶ train: pyspark.sql.dataframe.DataFrame = [age: integer, job: string ... 33 more fields]
▶ test: pyspark.sql.dataframe.DataFrame = [age: integer, job: string ... 33 more fields]

Command took 1.20 minutes -- by kxr190005@utdallas.edu at 4/21/2020, 8:13:17 PM on Khushboo_Rathore_CaseStudy

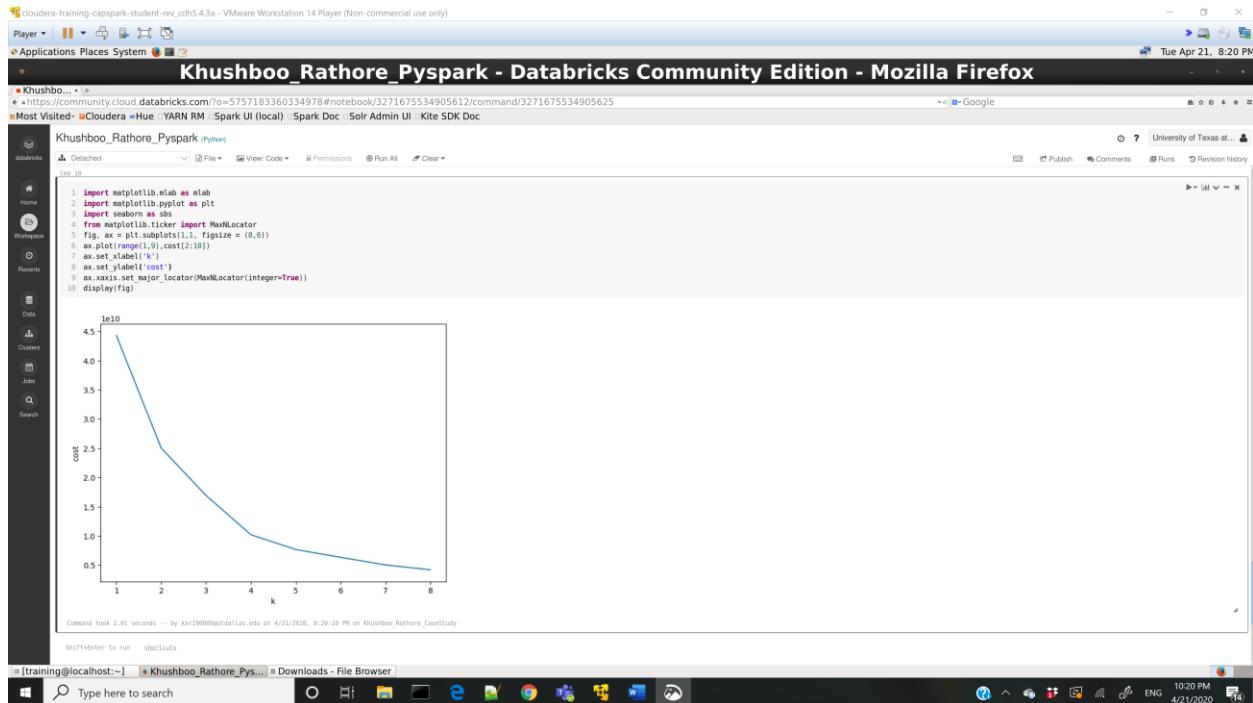
```

Shift+Enter to run    shortcuts

Step 20: Execute the following code:

**Q. Take a screenshot of the output of the code and paste it below.**

The graph plotted is an **Elbow Plot** of number of clusters vs cost which can be used to determine the optimum number of clusters to be used on the dataset to get best result. Here the elbow is at  $k = 2$ , so we should get the best model at 2 clusters.



## Part 7: Evaluate Model

Step 21: Execute the following code:

Q. Take a screenshot of the output of the code and paste it below.

The screenshot shows a Mozilla Firefox browser window titled "Khushboo\_Rathore\_Pyspark - Databricks Community Edition - Mozilla Firefox". The address bar shows the URL: <https://community.cloud.databricks.com/?o=5757183360334978#notebook/3271675534905612/command/3271675534905626>. The main content area displays the following Python code:

```
1 from pyspark.ml.clustering import KMeans
2 from pyspark.ml.evaluation import ClusteringEvaluator
3 kmeans = KMeans().setK(8).setSeed(999)
4 model = kmeans.fit(df)
```

Below the code, the browser shows a detailed log of Spark jobs and stages. The log indicates that the command took 2.01 seconds and was run on a cluster node (krr19000@pdallas.edu) at 8:20:10 PM on Tuesday, April 21, 2020. The log lists numerous jobs (Job 208 to Job 231) and their stages (Stage 1/1 to Stage 2/2), each with a "View" link. At the bottom of the log, it says "Command took 8.12 seconds ... by krr19000@pdallas.edu at 4/21/2020, 8:20:30 PM on Khushboo\_Rathore\_CaseStudy".

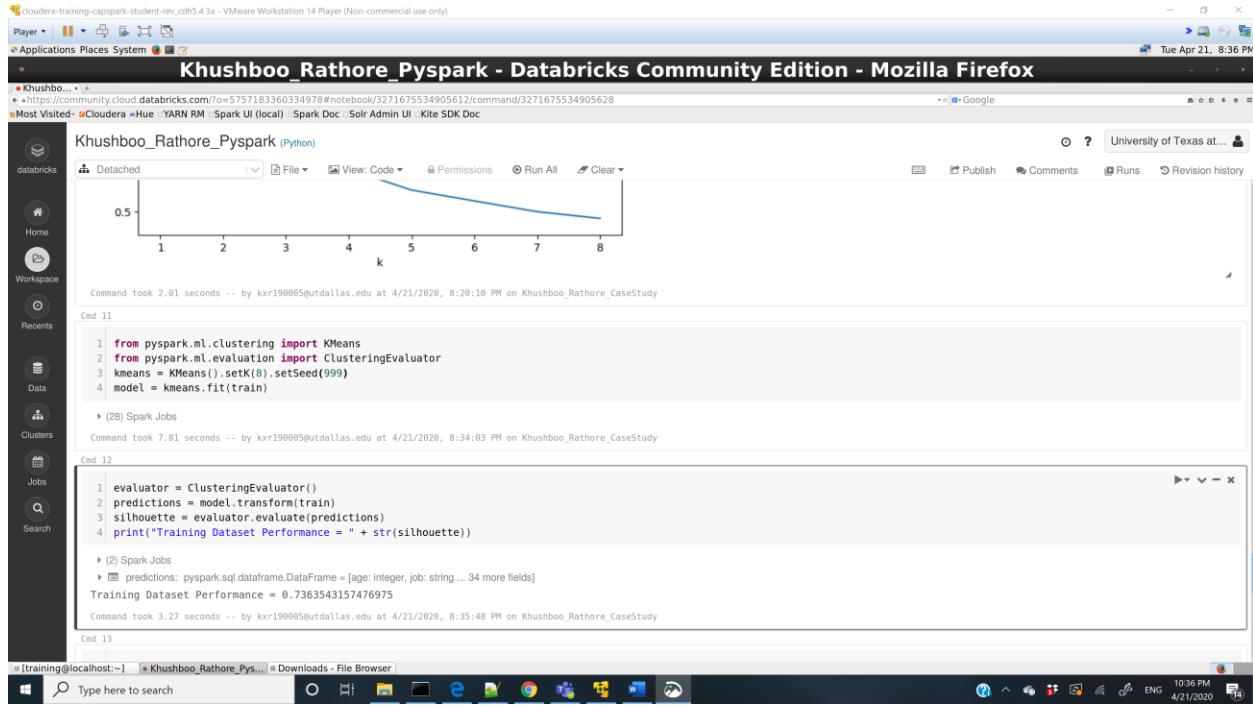
At the bottom of the browser window, there is a terminal-like interface with the prompt "[training@localhost:~]". The terminal shows the command "shift+Enter to run shortcuts" and the path "[training@localhost:~] Khushboo\_Rathore\_Pys... [Downloads - File Browser]". The system tray at the bottom right shows the date as 4/21/2020 and the time as 10:29 PM.

Step 22: Execute the following code:

Q. Take a screenshot of the output of the code and paste it below.

COMBINING STEP 21 AND STEP 22 FOR TRAIN DATASET

**KMeans().setK(8).setSeed(999)** : Performance = 0.74 (approx)



**Step 23:** Execute the following code:

**Q. Take a screenshot of the output of the code and paste it below.**

**What is Centroid and how is it calculated in K-means?**

Centroid is the center of the cluster.

In K Means, as per the value of K (number of clusters) specified by the user, number of clusters are randomly assigned to different points in the dataset and the unassigned points are then “assigned” to a cluster based on their proximity to the previously assigned point using Euclidean distance.

Once assignment completes, the new centroid values of the cluster is computed by calculating the mean values and again the values of Euclidean distance is calculated from the new centroids. This process repeats until we find a constant value for centroids and the latest cluster will be considered as the final cluster solution.

```

1 centers = model.clusterCenters()
2 print("Cluster Centers: ")
3 for center in centers:
4     print(center)

Cluster Centers:
[2.19323794e-01 1.79181968e-01 1.62213499e-01 1.22704825e-01
 8.69950614e-02 6.5467892e-02 3.67228061e-02 3.27972648e-02
 3.35570479e-02 2.91249842e-02 2.51994428e-02 5.62112195e-01
 3.19235153e-01 5.03482335e-01 3.18222119e-01 1.32328732e-01
 9.83031531e-01 5.16272082e-01 8.60073446e-01 7.21666456e-01
 2.13372167e-01 7.46739268e-01 1.12447765e-01 9.40863619e-02
 4.08317082e+00 7.49725972e+02 3.71516145e+02 2.51006711e+00
 5.22369254e+01 8.07648474e+01
[2.75574113e-01 1.31524088e-01 1.79549718e-01 9.60334029e-02
 6.36743215e-02 1.02296451e-01 3.86221294e-02 3.82713987e-02
 2.40983507e-02 2.40983507e-02 2.81837161e-02 5.93945720e-01
 2.96456939e-01 4.00823573e-01 4.16492693e-01 1.37787056e-01
 9.8956159e-01 5.85594990e-01 9.37369520e-01 7.17118998e-01
 1.67891441e-01 7.16075157e-01 1.11691023e-01 1.22129436e-01
 4.40496605e+01 6.46498330e+03 4.03259916e+02 2.42588727e+00
 5.29542797e+01 9.98665428e+01
[3.55932203e-01 5.08474576e-02 1.52542373e-01 6.77966102e-02
 3.38983051e-02 1.86440678e-01 6.77966182e-02 1.69491525e-02
 0.00000000e+00 1.69491525e-02 1.69491525e-02 6.27118644e-01
 3.38983051e-01 3.55932203e-01 5.25423729e-01 5.08474576e-02
 1.00000000e+00 6.94915254e-01 9.66101695e-01 7.45762712e-01
 1.52542373e-01 7.62711864e-01 5.08474576e-02 1.01694915e-01
 4.630580847e+01 2.67298475e+04 4.10000000e+02 2.86440678e+00
 4.03898305e+01 7.28813559e+01]

Command took 0.05 seconds -- by kxr190005@utdallas.edu at 4/21/2020, 8:44:05 PM on Khushboo Rathore CaseStudy

```

## Step 24: Execute the following code:

Q. Take a screenshot of the output of the code and paste it below.

```

1 kmeans = KMeans().setK(8).setSeed(999)
2 model = kmeans.fit(test)

> (17) Spark Jobs
Command took 6.61 seconds -- by kxr190005@utdallas.edu at 4/21/2020, 8:46:44 PM on Khushboo_Rathore CaseStudy

Cmd 14

1 evaluator = ClusteringEvaluator()
2 predictions = model.transform(train)
3 silhouette = evaluator.evaluate(predictions)
4 print("Training Dataset Performance = " + str(silhouette))

> (2) Spark Jobs
>   predictions: pyspark.sql.dataframe.DataFrame = [age: integer, job: string ... 34 more fields]
Training Dataset Performance = 0.5821187999461231

Command took 2.65 seconds -- by kxr190005@utdallas.edu at 4/21/2020, 8:47:50 PM on Khushboo_Rathore CaseStudy

```

## Step 25: Execute the following code:

Q. Take a screenshot of the output of the code and paste it below.

```
[3.55932203e-01 5.08474576e-02 1.52542373e-01 6.77966102e-02
3.38983051e-02 1.86440678e-01 6.77966102e-02 1.69491525e-02
0.00000000e+00 1.69491525e-02 1.69491525e-02 6.27118644e-01
3.38983051e-01 3.55932203e-01 5.25423723e-01 5.08474576e-02
1.00000000e+00 6.94915254e-01 9.66101695e-01 7.45762712e-01
1.52542373e-01 7.62711864e-01 5.08474576e-02 1.01694915e-01
4.63050847e+01 2.67298475e+04 4.10000000e+02 2.86440678e+00
4.03983051e+01 7.28813559e-01]
```

Command took 0.05 seconds -- by kxr190005@utdallas.edu at 4/21/2020, 8:44:05 PM on Khushboo\_Rathore\_CaseStudy

```
1 kmeans = KMeans().setK(8).setSeed(999)
2 model = kmeans.fit(test)
```

(17) Spark Jobs

Command took 6.61 seconds -- by kxr190005@utdallas.edu at 4/21/2020, 8:46:44 PM on Khushboo\_Rathore\_CaseStudy

```
1 evaluator = ClusteringEvaluator()
2 predictions = model.transform(train)
3 silhouette = evaluator.evaluate(predictions)
4 print("Training Dataset Performance = " + str(silhouette))
```

(2) Spark Jobs

[predictions: pyspark.sql.DataFrame] [age: integer, job: string ... 34 more fields]

Training Dataset Performance = 0.5821187999461231

Command took 2.65 seconds -- by kxr190005@utdallas.edu at 4/21/2020, 8:47:50 PM on Khushboo\_Rathore\_CaseStudy

Shift+Enter to run    shortcuts

[training@localhost:~] \* Khushboo\_Rathore\_Pys... Downloads - File Browser

Copy the Train and Test performance results of all the clusters here. Compare the results and comment about the performance of each model. Select your best model and explain why you chose that number of clusters as the optimal level.

## TRAIN - COMBINING STEP 21 AND STEP 22 FOR TRAIN DATASET

**KMeans().setK(8).setSeed(999)** : Performance = 0.74 (approx)

```
from pyspark.ml.clustering import KMeans
from pyspark.ml.evaluation import ClusteringEvaluator
kmeans = KMeans().setK(8).setSeed(999)
model = kmeans.fit(train)
```

(28) Spark Jobs

[predictions: pyspark.sql.DataFrame] [age: integer, job: string ... 34 more fields]

Training Dataset Performance = 0.736543157476975

Command took 3.27 seconds -- by kxr190005@utdallas.edu at 4/21/2020, 8:35:48 PM on Khushboo\_Rathore\_CaseStudy

## KMeans().setK(3).setSeed(999) : Performance = 0.88 (approx)

The screenshot shows a Databricks notebook interface. At the top, there's a navigation bar with tabs like Player, Applications, Places, System, and a search bar. Below the bar, the title is "Khushboo\_Rathore\_Pyspark - Databricks Community Edition - Mozilla Firefox". The main area has a sidebar with icons for Home, Workspace, Data, Clusters, Jobs, and Search. The workspace contains a line graph titled "Khushboo\_Rathore\_Pyspark (Python)" showing silhouette scores for different values of k (from 1 to 8). The scores decrease as k increases, starting around 0.55 at k=1 and ending near 0.45 at k=8. Below the graph is a code editor with the following Python code:

```
1 from pyspark.ml.clustering import KMeans
2 from pyspark.ml.evaluation import ClusteringEvaluator
3 kmeans = KMeans().setK(3).setSeed(999)
4 model = kmeans.fit(train)

▶ (28) Spark Jobs
```

Below the code editor, a command history shows:

```
Command took 2.01 seconds -- by kxr190005@utdallas.edu at 4/21/2020, 8:20:10 PM on Khushboo_Rathore_CaseStudy
```

```
Cmd 11
1 evaluator = ClusteringEvaluator()
2 predictions = model.transform(train)
3 silhouette = evaluator.evaluate(predictions)
4 print("Training Dataset Performance = " + str(silhouette))

▶ (2) Spark Jobs
▶ (2) predictions: pyspark.sql.dataframe.DataFrame = [age: integer, job: string ... 34 more fields]
Training Dataset Performance = 0.879399553497968

Command took 2.64 seconds -- by kxr190005@utdallas.edu at 4/21/2020, 8:41:28 PM on Khushboo_Rathore_CaseStudy
```

```
Cmd 13
```

The bottom of the screen shows a Windows taskbar with various application icons and the system clock indicating 10:41 PM on April 21, 2020.

## KMeans().setK(4).setSeed(999) : Performance = 0.86 (approx)

This screenshot is similar to the previous one but shows the results for k=4. The line graph shows silhouette scores for k from 1 to 8, with the score for k=4 being approximately 0.55. The Python code in the code editor is identical to the one in the first screenshot:

```
1 from pyspark.ml.clustering import KMeans
2 from pyspark.ml.evaluation import ClusteringEvaluator
3 kmeans = KMeans().setK(4).setSeed(999)
4 model = kmeans.fit(train)

▶ (23) Spark Jobs
```

The command history shows:

```
Command took 2.01 seconds -- by kxr190005@utdallas.edu at 4/21/2020, 8:20:10 PM on Khushboo_Rathore_CaseStudy
```

```
Cmd 11
1 evaluator = ClusteringEvaluator()
2 predictions = model.transform(train)
3 silhouette = evaluator.evaluate(predictions)
4 print("Training Dataset Performance = " + str(silhouette))

▶ (2) Spark Jobs
▶ (2) predictions: pyspark.sql.dataframe.DataFrame = [age: integer, job: string ... 34 more fields]
Training Dataset Performance = 0.865189532832998

Command took 2.33 seconds -- by kxr190005@utdallas.edu at 4/21/2020, 8:39:31 PM on Khushboo_Rathore_CaseStudy
```

```
Cmd 13
```

The bottom of the screen shows a Windows taskbar with various application icons and the system clock indicating 10:40 PM on April 21, 2020.

## KMeans().setK(5).setSeed(999) : Performance = 0.82 (approx)

Databricks Community Edition - Mozilla Firefox

Khushboo\_Rathore\_Pyspark

```

from pyspark.ml.clustering import KMeans
from pyspark.ml.evaluation import ClusteringEvaluator
kmeans = KMeans().setK(5).setSeed(999)
model = kmeans.fit(train)

(19) Spark Jobs
Command took 6.86 seconds -- by kxr190005@utdallas.edu at 4/29/2020, 1:37:47 PM on Khushboo_Rathore_CaseStudy

```

```

evaluator = ClusteringEvaluator()
predictions = model.transform(train)
silhouette = evaluator.evaluate(predictions)
print("Training Dataset Performance = " + str(silhouette))

(2) Spark Jobs
predictions: pyspark.sql.dataframe.DataFrame = [age: integer, job: string ... 34 more fields]
Training Dataset Performance = 0.8208494545136753
Command took 2.96 seconds -- by kxr190005@utdallas.edu at 4/29/2020, 1:37:47 PM on Khushboo_Rathore_CaseStudy

```

Windows Ink Workspace

**KMeans().setK(6).setSeed(999) : Performance = 0.76 (approx)**

Databricks Community Edition - Mozilla Firefox

Khushboo\_Rathore\_Pyspark

```

from pyspark.ml.clustering import KMeans
from pyspark.ml.evaluation import ClusteringEvaluator
kmeans = KMeans().setK(6).setSeed(999)
model = kmeans.fit(train)

(28) Spark Jobs
Command took 7.83 seconds -- by kxr190005@utdallas.edu at 4/29/2020, 1:37:47 PM on Khushboo_Rathore_CaseStudy

```

```

evaluator = ClusteringEvaluator()
predictions = model.transform(train)
silhouette = evaluator.evaluate(predictions)
print("Training Dataset Performance = " + str(silhouette))

(2) Spark Jobs
predictions: pyspark.sql.dataframe.DataFrame = [age: integer, job: string ... 34 more fields]
Training Dataset Performance = 0.7582696258533238
Command took 2.54 seconds -- by kxr190005@utdallas.edu at 4/29/2020, 1:37:47 PM on Khushboo_Rathore_CaseStudy

```

Windows Ink Workspace

**KMeans().setK(7).setSeed(999) : Performance = 0.59 (approx) which is the worst score received in the performance results of all the clusters.**

```

from pyspark.ml.clustering import KMeans
from pyspark.ml.evaluation import ClusteringEvaluator
kmeans = KMeans().setK(7).setSeed(999)
model = kmeans.fit(train)

# (28) Spark Jobs
Command took 7.91 seconds -- by kxr190005@utdallas.edu at 4/29/2020, 1:37:47 PM on Khushboo_Rathore_CaseStudy

```

```

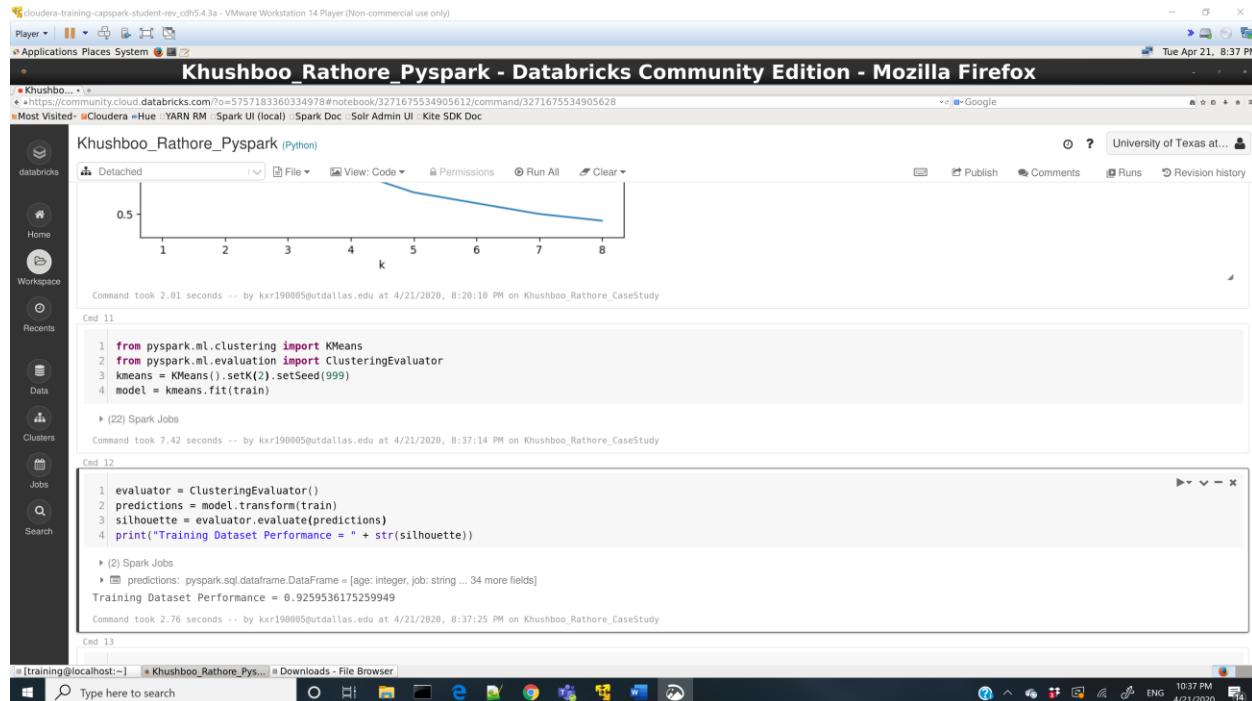
evaluator = ClusteringEvaluator()
predictions = model.transform(train)
silhouette = evaluator.evaluate(predictions)
print("Training Dataset Performance = " + str(silhouette))

# (2) Spark Jobs
# predictions: pyspark.sql.dataframe.DataFrame = [age: integer, job: string ... 34 more fields]
Training Dataset Performance = 0.5874006410765837

```

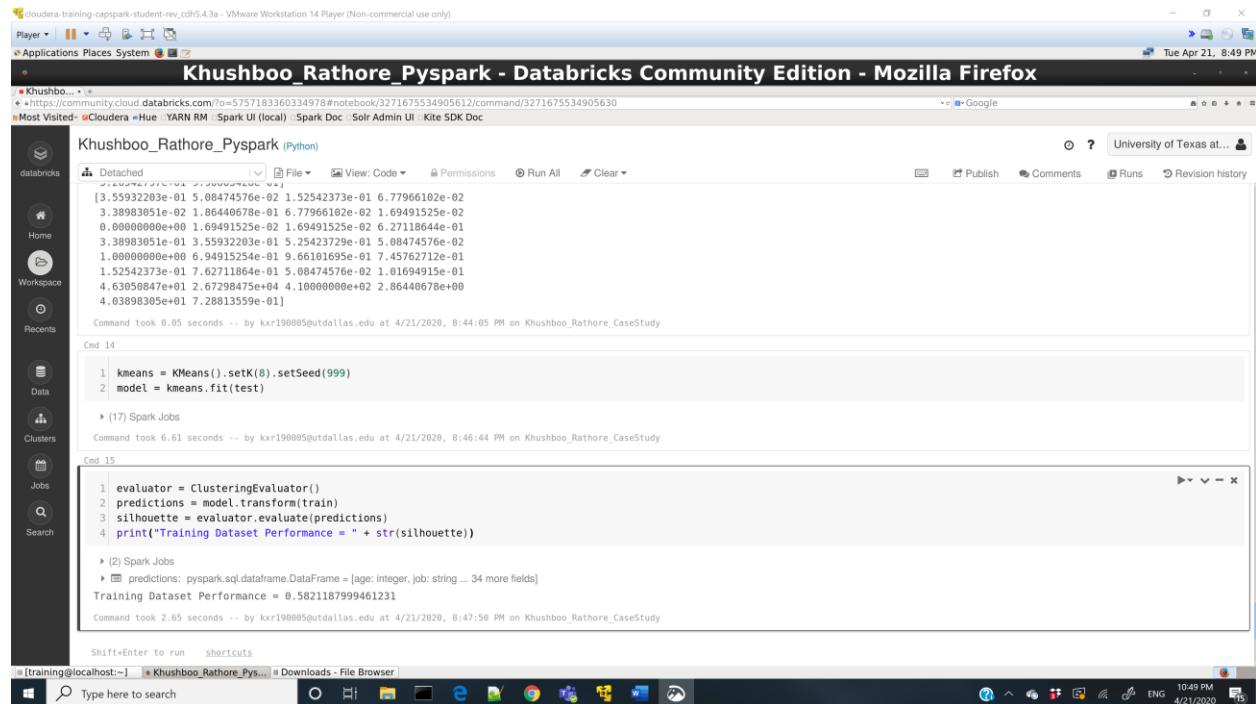
**Best Model** – From the Elbow plot we can see that the best model would be for 2 number of clusters as it best fits the data for Train dataset and has the highest score. By increasing the number of clusters, the training performance of the model is dropping continuously.

**KMeans().setK(2).setSeed(999)** : Performance = 0.93 (approx)



## TEST - COMBINING STEP 24 AND STEP 25 FOR TRAIN DATASET

**KMeans().setK(8).setSeed(999)** : Performance = 0.58 (approx) which is the worst score received in the performance results of all the clusters.



```
[3. 5593203e-01 5.08474576e-02 1.52542373e-01 6.77966102e-02
3. 38983051e-02 1.86440678e-01 6.77966102e-02 1.69491525e-02
0. 00000000e+00 1.69491525e-02 1.69491525e-02 6.27118644e-01
3. 38983051e-01 3.55932203e-01 5.25423729e-01 5.08474576e-02
1. 00000000e+00 6.94915254e-01 9.66101695e-01 7.45762712e-01
1. 52542373e-01 7.62711864e-01 5.08474576e-02 1.01694915e-01
4. 63050847e+01 2.67298475e+04 4.10000000e+02 2.86440678e+00
4. 0398305e+01 7.28813559e-01

Command took 0.05 seconds -- by kxr190005@utdallas.edu at 4/21/2020, 8:44:05 PM on Khushboo_Rathore_CaseStudy

Cmd 14
1 kmeans = KMeans().setK(8).setSeed(999)
2 model = kmeans.fit(test)

> (17) Spark Jobs
Command took 6.61 seconds -- by kxr190005@utdallas.edu at 4/21/2020, 8:46:44 PM on Khushboo_Rathore_CaseStudy

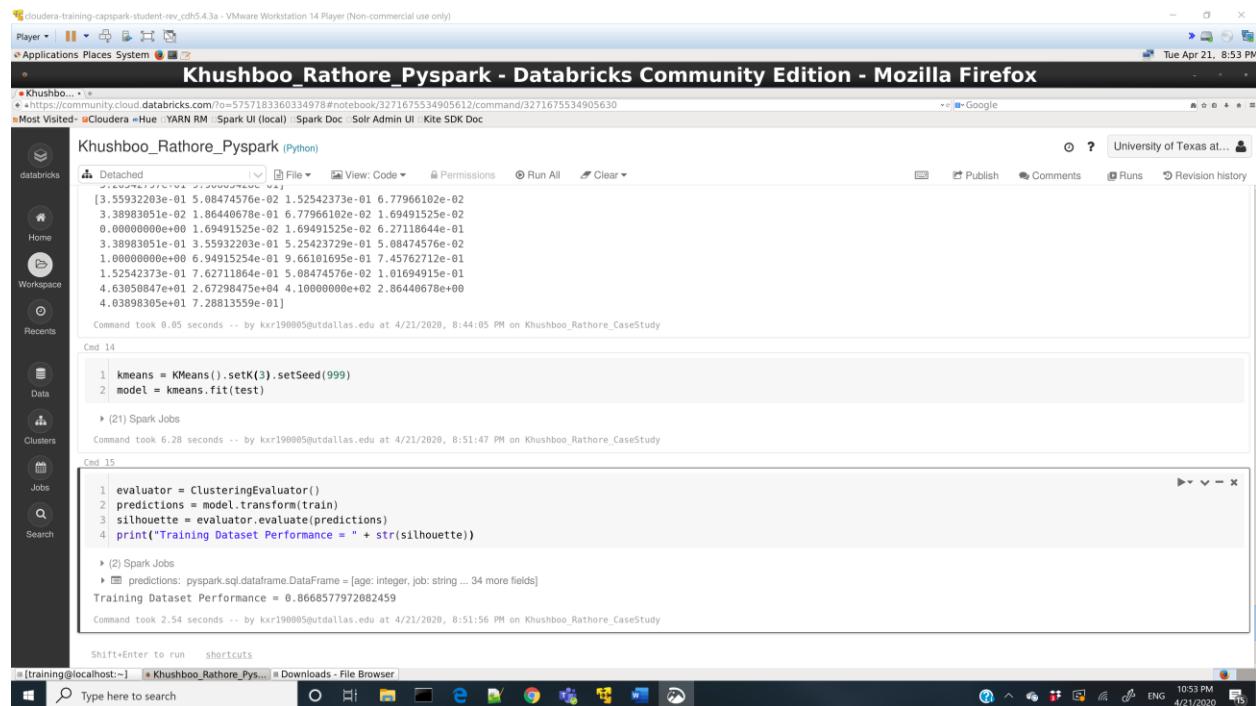
Cmd 15
1 evaluator = ClusteringEvaluator()
2 predictions = model.transform(train)
3 silhouette = evaluator.evaluate(predictions)
4 print("Training Dataset Performance = " + str(silhouette))

> (2) Spark Jobs
> predictions: pyspark.sql.dataframe.DataFrame = [age: integer, job: string ... 34 more fields]
Training Dataset Performance = 0.582118799941231

Command took 2.65 seconds -- by kxr190005@utdallas.edu at 4/21/2020, 8:47:50 PM on Khushboo_Rathore_CaseStudy

Shift+Enter to run shortcuts
```

**KMeans().setK(3).setSeed(999)** : Performance = 0.87 (approx)



```
[3. 5593203e-01 5.08474576e-02 1.52542373e-01 6.77966102e-02
3. 38983051e-02 1.86440678e-01 6.77966102e-02 1.69491525e-02
0. 00000000e+00 1.69491525e-02 1.69491525e-02 6.27118644e-01
3. 38983051e-01 3.55932203e-01 5.25423729e-01 5.08474576e-02
1. 00000000e+00 6.94915254e-01 9.66101695e-01 7.45762712e-01
1. 52542373e-01 7.62711864e-01 5.08474576e-02 1.01694915e-01
4. 63050847e+01 2.67298475e+04 4.10000000e+02 2.86440678e+00
4. 0398305e+01 7.28813559e-01

Command took 0.05 seconds -- by kxr190005@utdallas.edu at 4/21/2020, 8:44:05 PM on Khushboo_Rathore_CaseStudy

Cmd 14
1 kmeans = KMeans().setK(3).setSeed(999)
2 model = kmeans.fit(test)

> (21) Spark Jobs
Command took 6.28 seconds -- by kxr190005@utdallas.edu at 4/21/2020, 8:51:47 PM on Khushboo_Rathore_CaseStudy

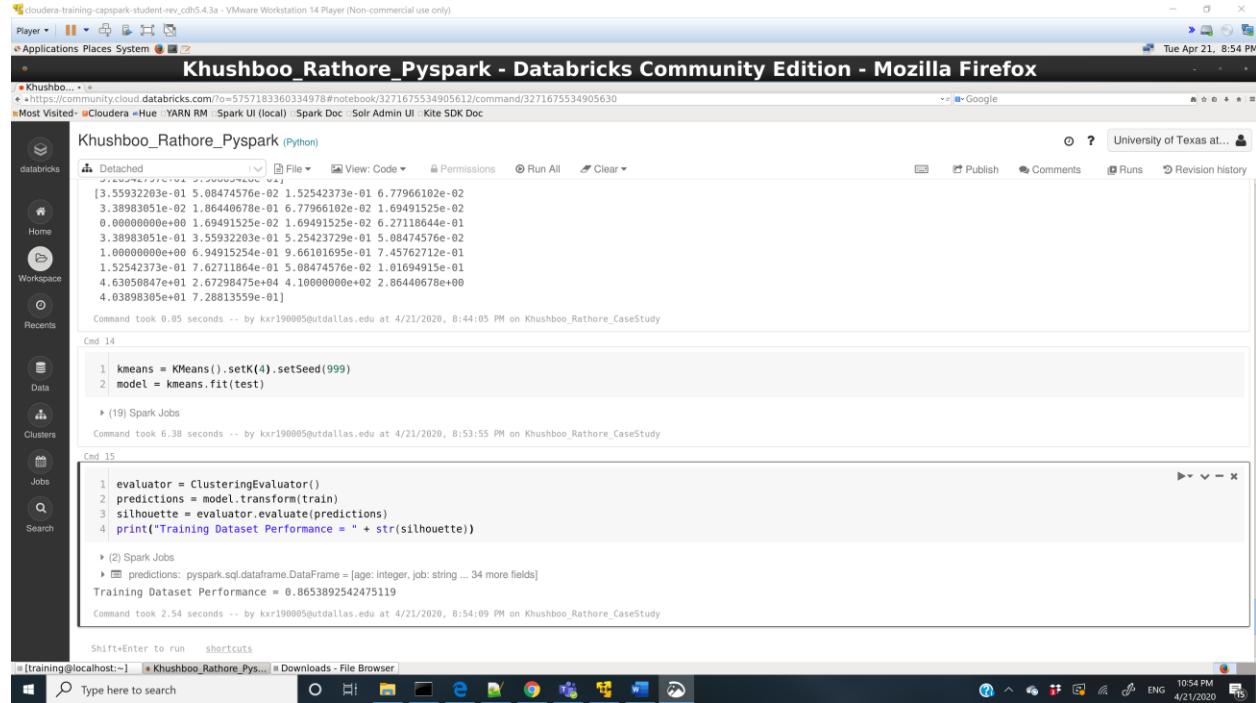
Cmd 15
1 evaluator = ClusteringEvaluator()
2 predictions = model.transform(train)
3 silhouette = evaluator.evaluate(predictions)
4 print("Training Dataset Performance = " + str(silhouette))

> (2) Spark Jobs
> predictions: pyspark.sql.dataframe.DataFrame = [age: integer, job: string ... 34 more fields]
Training Dataset Performance = 0.8668577972082459

Command took 2.54 seconds -- by kxr190005@utdallas.edu at 4/21/2020, 8:51:56 PM on Khushboo_Rathore_CaseStudy

Shift+Enter to run shortcuts
```

## KMeans().setK(4).setSeed(999) : Performance = 0.87 (approx)



Databricks Community Edition - Mozilla Firefox

Khushboo\_Rathore\_Pyspark - Databricks Community Edition - Mozilla Firefox

Most Visited: Cloudera Hue YARN RM Spark UI (local) Spark Doc Solr Admin UI Kite SDK Doc

Khushboo\_Rathore\_Pyspark (Python)

Detached View: Code Permissions Run All Clear

[3.5593203e-01 5.08474576e-02 1.52542373e-01 6.77966102e-02  
3.38983051e-02 1.86440678e-01 6.77966102e-02 1.69491525e-02  
0.0690000e+00 1.69491525e-02 6.2718644e-01  
3.38983051e-01 3.55932203e-01 5.25423729e-01 5.08474576e-02  
1.0000000e+00 6.94915254e-01 9.66101695e-01 7.45762712e-01  
1.52542373e-01 7.62711864e-01 5.08474576e-02 1.01694915e-01  
4.63059847e+01 2.67298475e+04 4.1000000e+02 2.86440678e+00  
4.0398305e+01 7.28813559e-01]

Command took 0.85 seconds -- by kxr190005@utdallas.edu at 4/21/2020, 8:44:05 PM on Khushboo\_Rathore\_CaseStudy

Cmd 14

```
1 kmeans = KMeans().setK(4).setSeed(999)
2 model = kmeans.fit(test)

▶ (19) Spark Jobs
```

Command took 6.38 seconds -- by kxr190005@utdallas.edu at 4/21/2020, 8:53:55 PM on Khushboo\_Rathore\_CaseStudy

Cmd 15

```
1 evaluator = ClusteringEvaluator()
2 predictions = model.transform(train)
3 silhouette = evaluator.evaluate(predictions)
4 print("Training Dataset Performance = " + str(silhouette))

▶ (2) Spark Jobs
▶ predictions: pyspark.sql.dataframe.DataFrame = [age: integer, job: string ... 34 more fields]
Training Dataset Performance = 0.8653892542475119
```

Command took 2.54 seconds -- by kxr190005@utdallas.edu at 4/21/2020, 8:54:09 PM on Khushboo\_Rathore\_CaseStudy

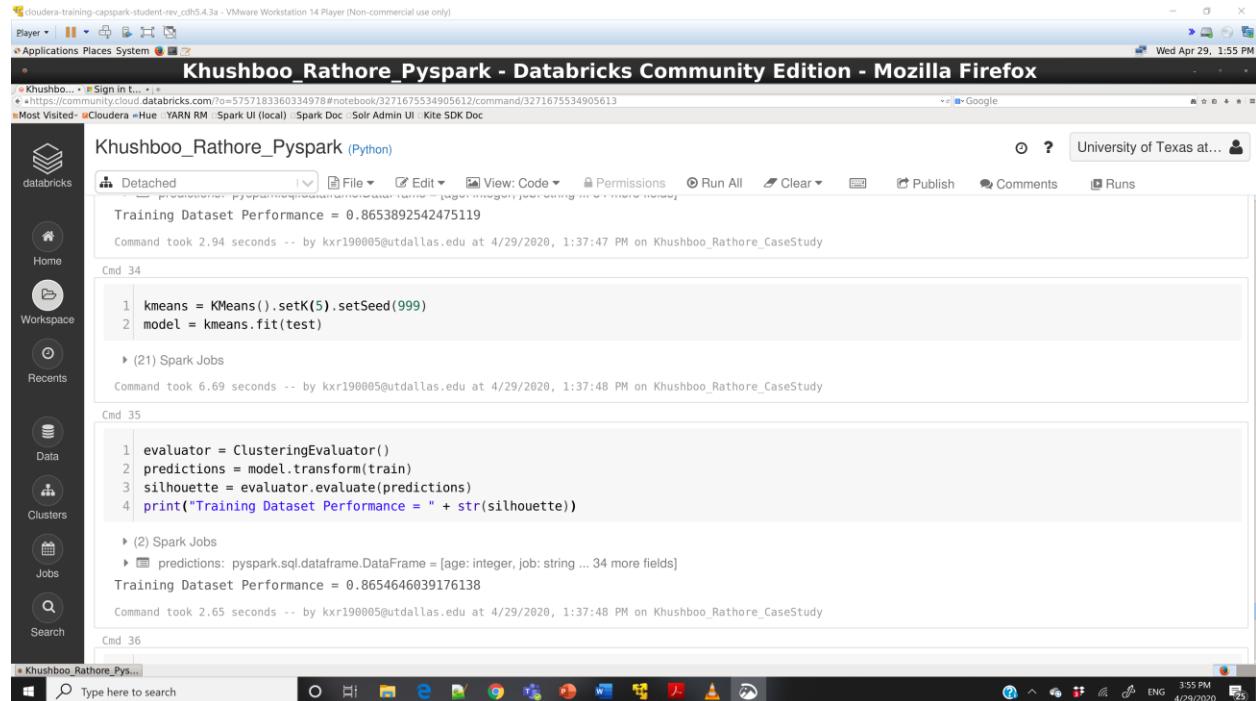
Shift+Enter to run shortcuts

localhost:~ Khushboo\_Rathore\_Pys... Downloads File Browser

Type here to search

10:54 PM ENG 4/21/2020

## KMeans().setK(5).setSeed(999) : Performance = 0.87 (approx)



Databricks Community Edition - Mozilla Firefox

Khushboo\_Rathore\_Pyspark - Databricks Community Edition - Mozilla Firefox

Most Visited: Cloudera Hue YARN RM Spark UI (local) Spark Doc Solr Admin UI Kite SDK Doc

Khushboo\_Rathore\_Pyspark (Python)

Detached View: Code Permissions Run All Clear

Training Dataset Performance = 0.8653892542475119

Command took 2.94 seconds -- by kxr190005@utdallas.edu at 4/29/2020, 1:37:47 PM on Khushboo\_Rathore\_CaseStudy

Cmd 34

```
1 kmeans = KMeans().setK(5).setSeed(999)
2 model = kmeans.fit(test)

▶ (21) Spark Jobs
```

Command took 6.69 seconds -- by kxr190005@utdallas.edu at 4/29/2020, 1:37:48 PM on Khushboo\_Rathore\_CaseStudy

Cmd 35

```
1 evaluator = ClusteringEvaluator()
2 predictions = model.transform(train)
3 silhouette = evaluator.evaluate(predictions)
4 print("Training Dataset Performance = " + str(silhouette))

▶ (2) Spark Jobs
▶ predictions: pyspark.sql.dataframe.DataFrame = [age: integer, job: string ... 34 more fields]
Training Dataset Performance = 0.8654646039176138
```

Command took 2.65 seconds -- by kxr190005@utdallas.edu at 4/29/2020, 1:37:48 PM on Khushboo\_Rathore\_CaseStudy

Cmd 36

Khushboo\_Rathore\_Pys... Type here to search

3:55 PM ENG 4/29/2020

## KMeans().setK(6).setSeed(999) : Performance = 0.82 (approx)

```

KMeans().setK(6).setSeed(999)
model = kmeans.fit(test)

evaluator = ClusteringEvaluator()
predictions = model.transform(train)
silhouette = evaluator.evaluate(predictions)
print("Training Dataset Performance = " + str(silhouette))

KMeans().setK(7).setSeed(999)
model = kmeans.fit(test)

```

**KMeans().setK(7).setSeed(999) : Performance = 0.73 (approx)**

```

KMeans().setK(7).setSeed(999)
model = kmeans.fit(test)

evaluator = ClusteringEvaluator()
predictions = model.transform(train)
silhouette = evaluator.evaluate(predictions)
print("Training Dataset Performance = " + str(silhouette))

```

**Best Model** – From the Elbow plot we can see that the best model would be for 2 number of clusters as it best fits the data for Test dataset and has the highest score. By increasing the number of clusters, the testing performance of the model is dropping continuously.

## KMeans().setK(2).setSeed(999) : Performance = 0.98 (approx)

```
[3.55932203e-01 5.08474576e-02 1.52542373e-01 6.77966102e-02  
3.38983051e-02 1.86440678e-01 6.77966102e-02 1.69491525e-02  
0.0000000e+00 1.69491525e-02 1.69491525e-02 6.27118644e-01  
3.38983051e-01 3.55932203e-01 5.25423732e-01 5.08474576e-02  
1.0000000e+00 6.94915254e-01 9.66101695e-01 7.45762712e-01  
1.52542373e-01 7.62711864e-01 5.08474576e-02 1.01694915e-01  
4.63650847e+01 2.67298475e+04 4.10000000e+02 2.86440678e+00  
4.0398305e+01 7.28813559e-01]  
Command took 0.05 seconds -- by kxr190005@utdallas.edu at 4/21/2020, 8:44:05 PM on Khushboo_Rathore_CaseStudy
```

```
1 kmeans = KMeans().setK(2).setSeed(999)  
2 model = kmeans.fit(test)  
> (1) Spark Jobs  
Command took 5.79 seconds -- by kxr190005@utdallas.edu at 4/21/2020, 8:49:57 PM on Khushboo_Rathore_CaseStudy
```

```
1 evaluator = ClusteringEvaluator()  
2 predictions = model.transform(train)  
3 silhouette = evaluator.evaluate(predictions)  
4 print("Training Dataset Performance = " + str(silhouette))  
> (2) Spark Jobs  
> predictions: pyspark.sql.dataframe.DataFrame = [age: integer, job: string ... 34 more fields]  
Training Dataset Performance = 0.9787389592621479  
Command took 2.74 seconds -- by kxr190005@utdallas.edu at 4/21/2020, 8:50:32 PM on Khushboo_Rathore_CaseStudy
```

Shift+Enter to run    shortcuts

Hence, we can conclude that the Best Model would be when number of cluster k = 2 as it gives the highest dataset performance for both Train and Test Dataset and the model generalizes well. By increasing the number of clusters the train and test set performance is decreasing.

Hence 2 is the optimal number of clusters for this model.

Step 26:

Export the file as “HTML” and save that with your “Firstname\_Lastname” naming convention.

Convert this HTML file to PDF (This is your 2<sup>nd</sup> submission file)

Export as “IPNYB” file. (This is your 3<sup>rd</sup> submission file)