Lab 9.1 - Using Databricks to Run a Recommendation System with MLLib & Spark

## Introduction - Databricks Setup & Installation

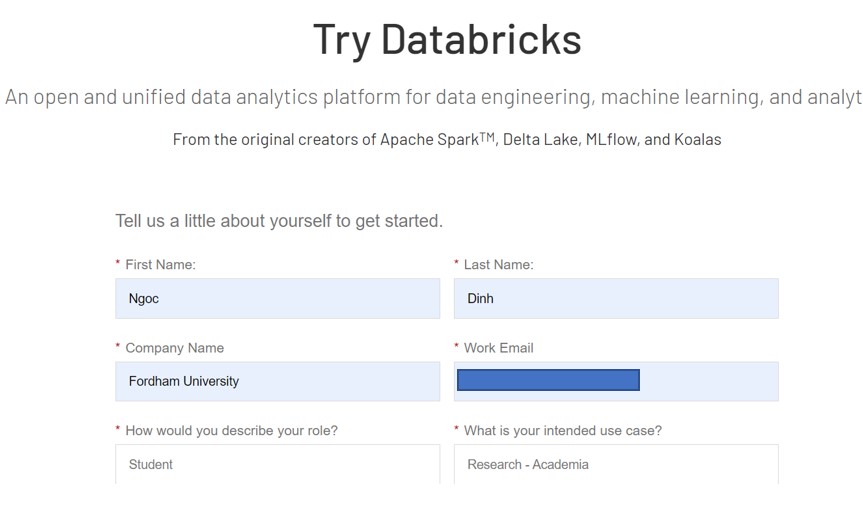
Databricks is a company founded by the original creators of Apache Spark. Databricks grew out of the AMPLab project at University of California, Berkeley that was involved in making Apache Spark, an open-source distributed computing framework built atop Scala. However, despite being built atop Scala, users can use other programming language to work with Databricks.

Databricks is a cloud-enabled platform, therefore many users can collaborate on a project. Databricks is an open and unified platform to collaboratively run all types of analytics workloads, from data preparation to exploratory analysis and predictive analytics, at scale.

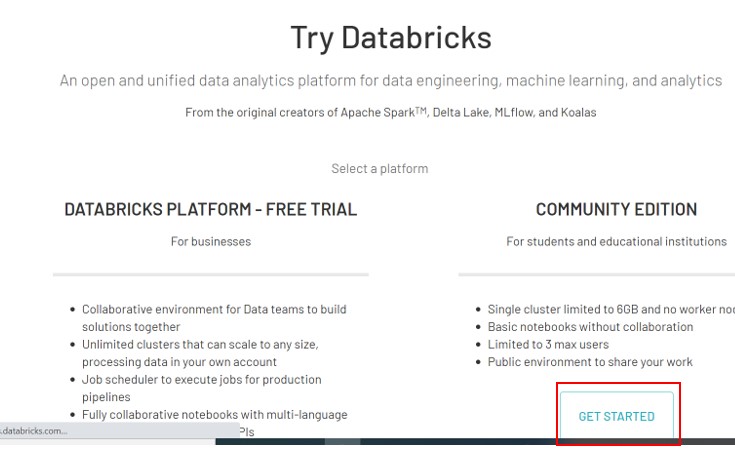
## Setup

You can either setup a trial or a community edition account. However, for the purpose of this workshop let us go with community edition.

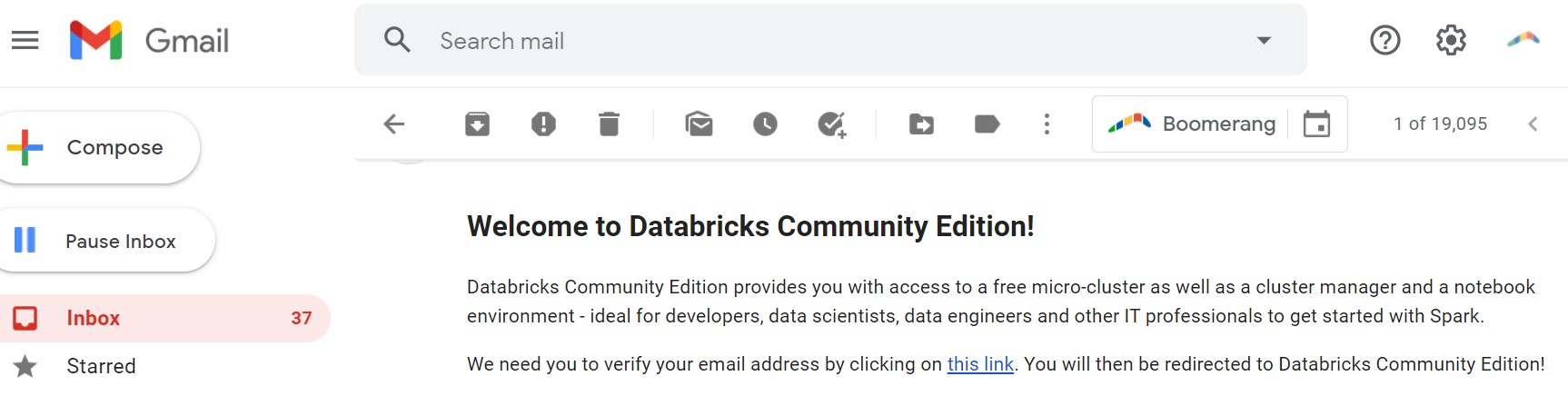
Go to the following link in your browser: <https://databricks.com/try-databricks>Put in all your info to sign up for a databricks account:



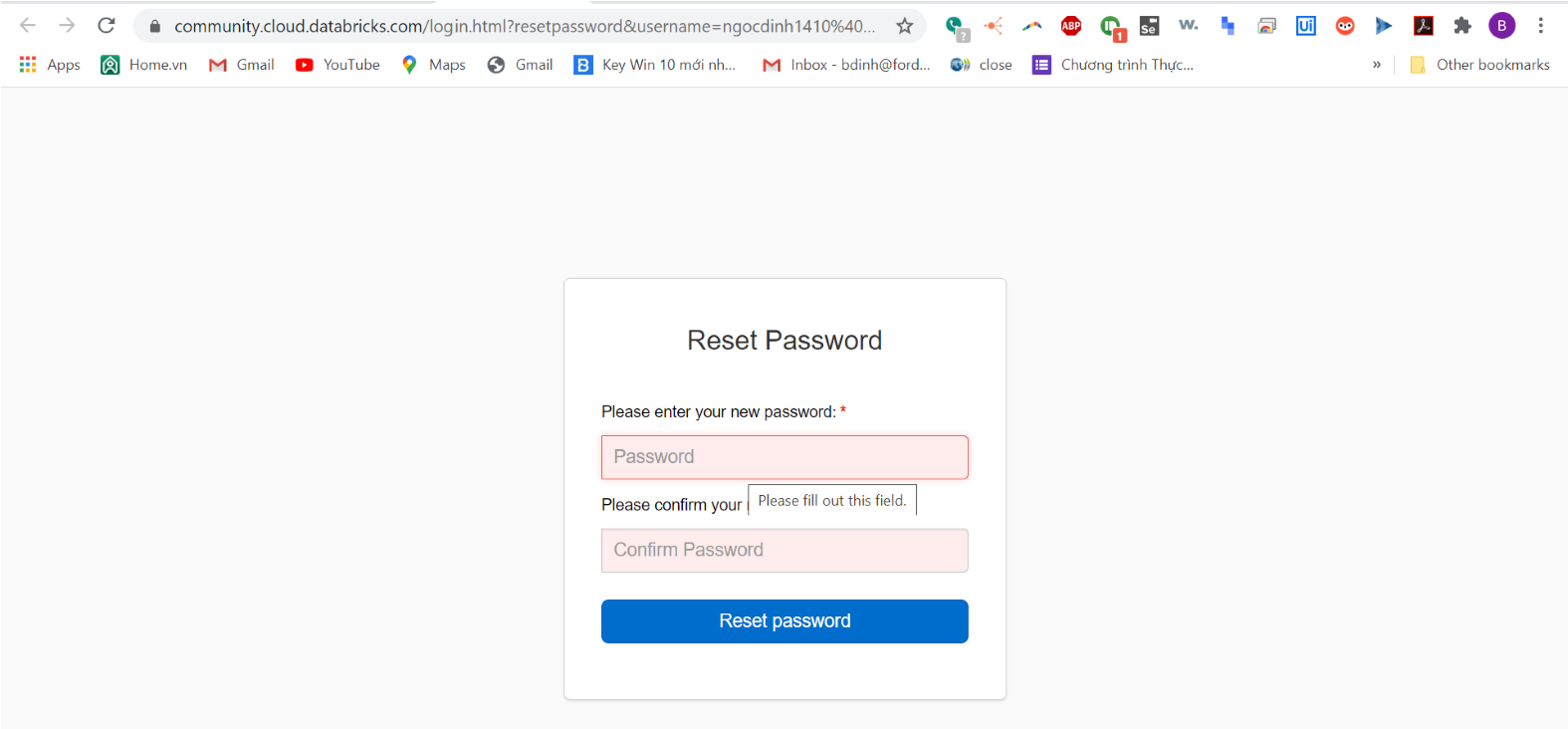
Click on sign up. It should take you to the following page. Click on “Get Started” for Community edition.



Check your email and verify the email address by clicking on the link provided.



Reset your password:



# Part 1 - A Quick Start Tutorial

Requirements

You are logged into a Databricks workspace. See Sign up for a free Databricks trial.

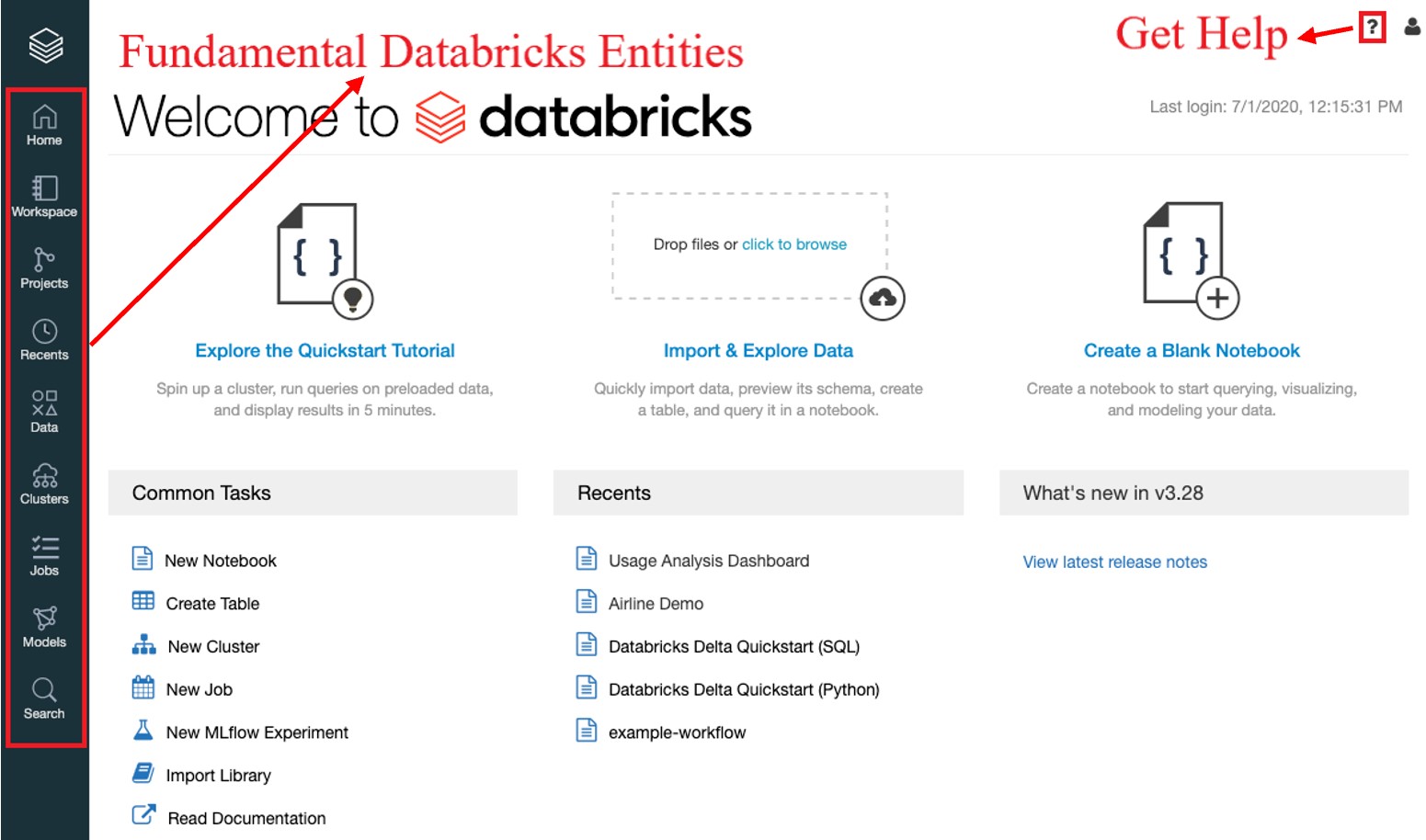
Get started with Databricks

# Step 1: Orient yourself to the Databricks UI

Landing page

From the sidebar at the left and the Common Tasks list on the landing page, you access fundamental Databricks entities: Workspace, clusters, tables, notebooks, jobs, and libraries. The Workspace is the special root folder that stores your Databricks assets, such as notebooks and libraries, and the data that you import.

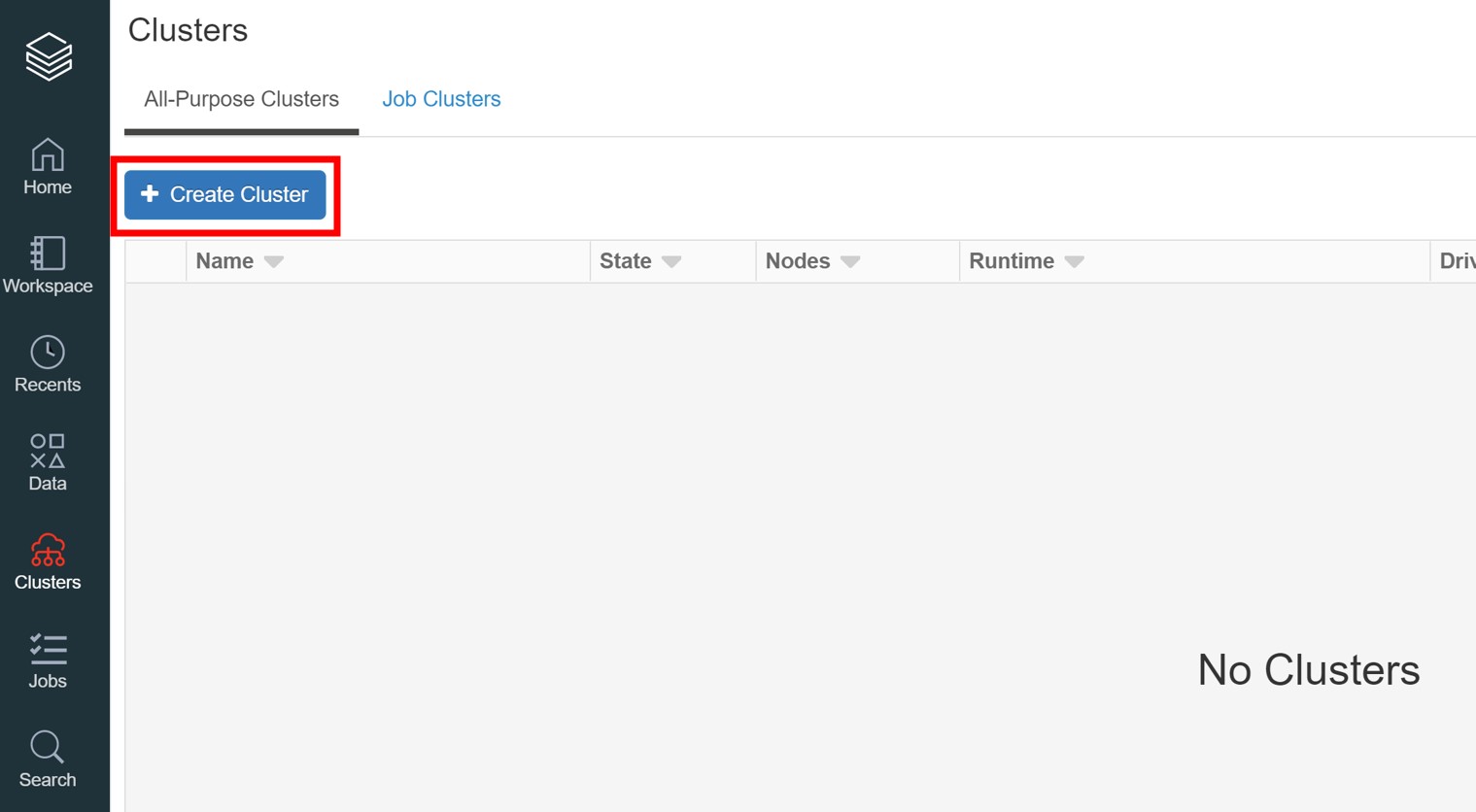
To get help, click the question icon Question Icon at the top right-hand corner.



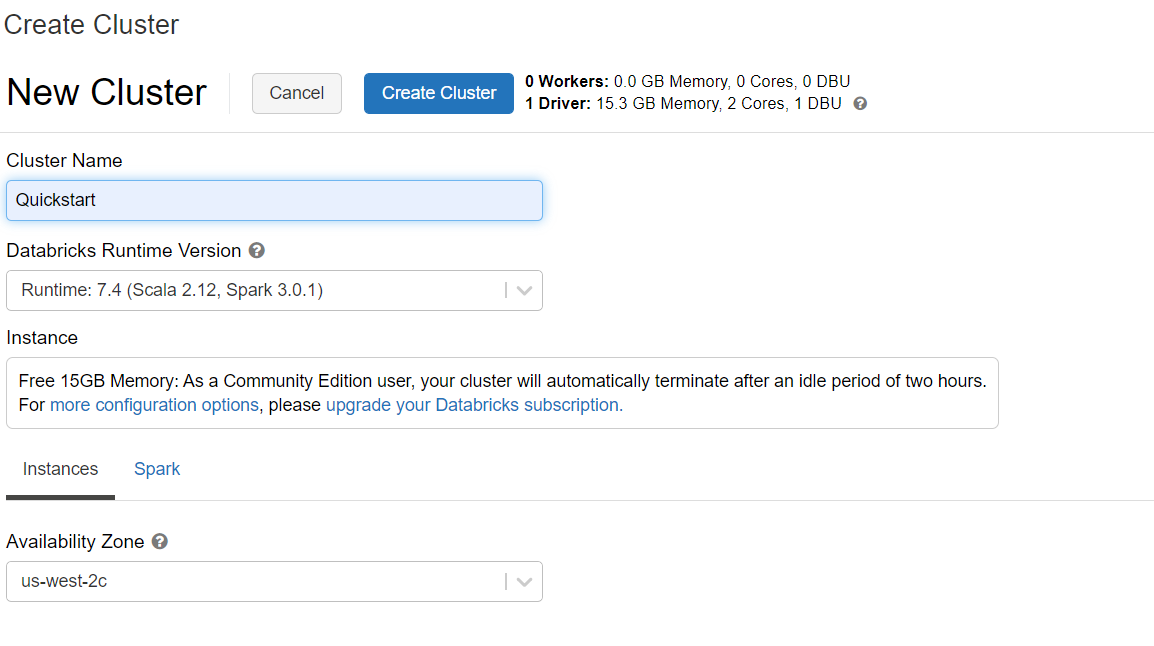
# Step 2: Create a cluster

A cluster is a collection of Databricks computation resources. To create a cluster:

1. In the sidebar, click the **Clusters** button .
2. On the Clusters page, click **Create Cluster**.



1. On the Create Cluster page, specify the cluster name **Quickstart** and select **7.4 (Scala 2.12, Spark 3.0.1)** in the Databricks Runtime Version drop-down.
2. Click **Create Cluster**.

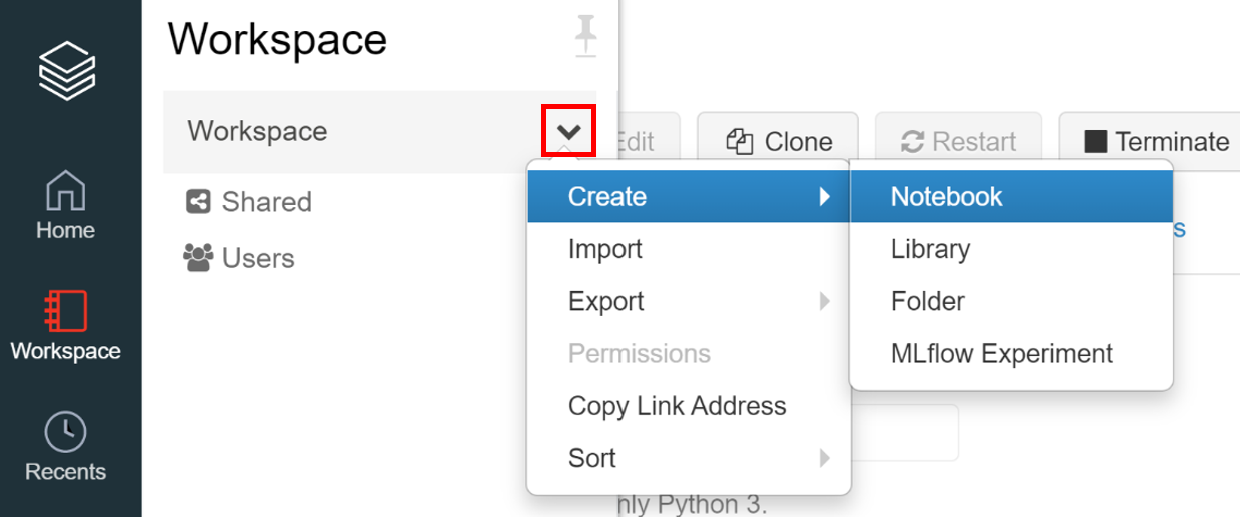


Please notice that for free account, the maximum number of active clusters is 1. You need to terminate an existing cluster or upgrade to a larger plan if you want more than one active cluster.

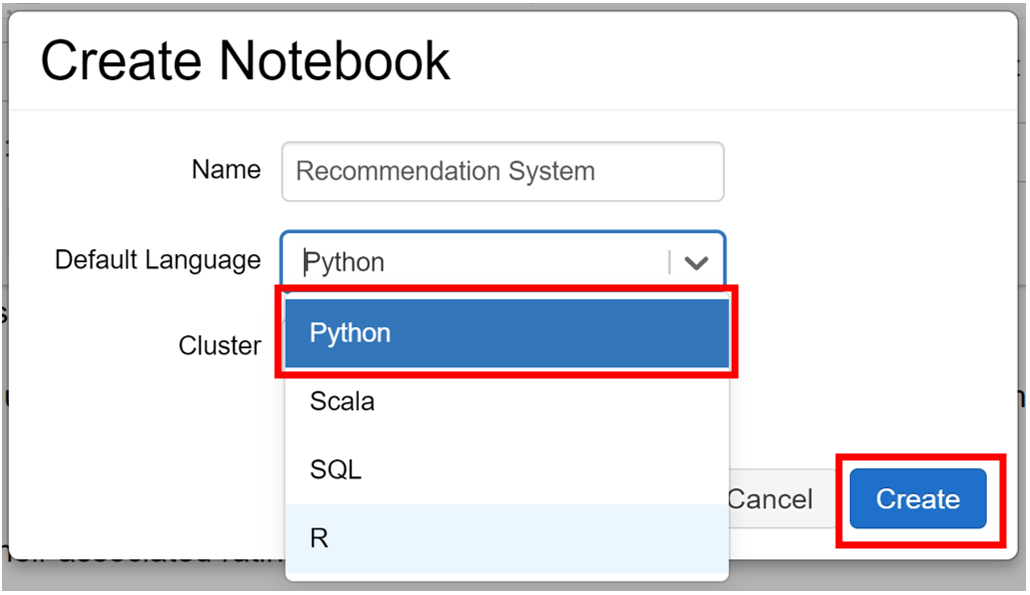
# Step 3: Create a notebook

A notebook is a collection of cells that run computations on an Apache Spark cluster. There can be multiple workspaces in one cluster. To create a notebook in the Workspace:

1. In the sidebar, click the **Workspace** button Workspace Icon.
2. In the Workspace folder, select Down Caret **Create > Notebook**.



1. On the Create Notebook dialog, enter a name and select **Python** in the Language drop-down. This selection determines the default language of the notebook. Make sure the **Cluster** is the **Quickstart** cluster created in our Get Started file.

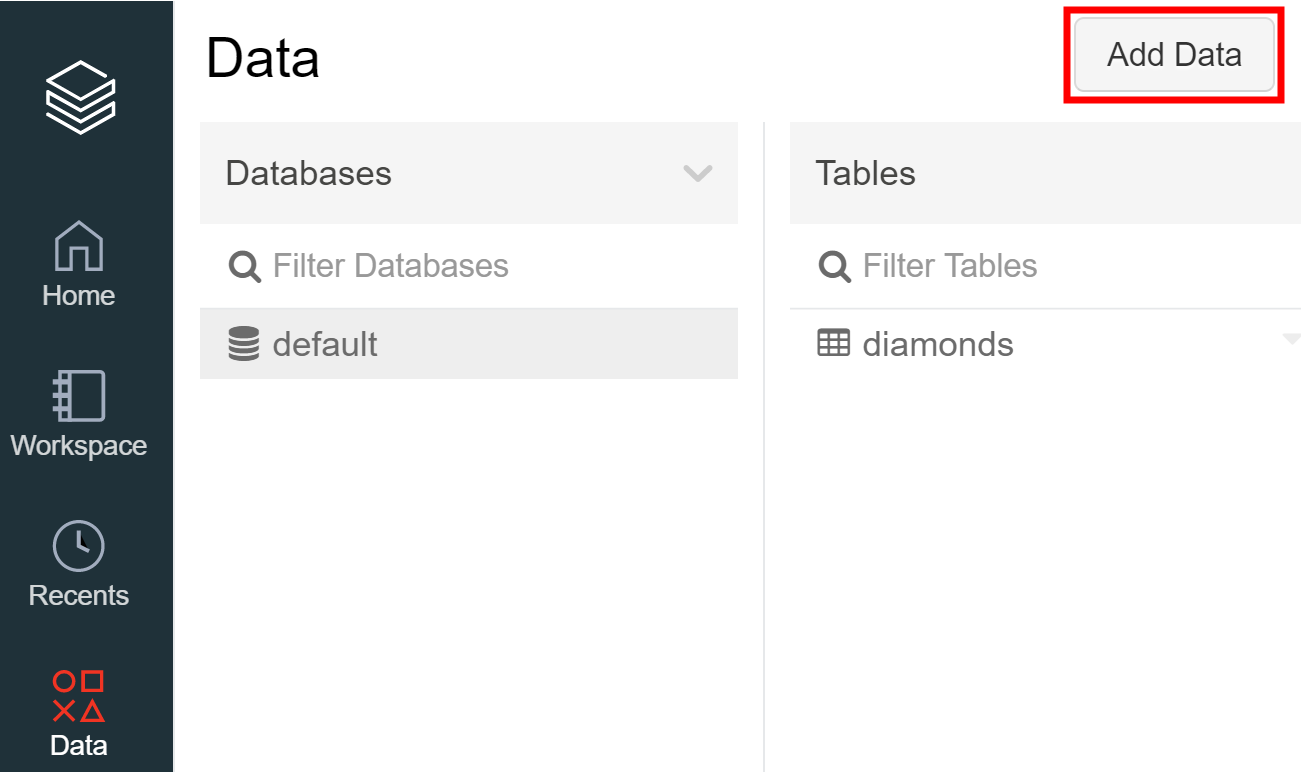


1. Click **Create**. The notebook opens with an empty cell at the top.

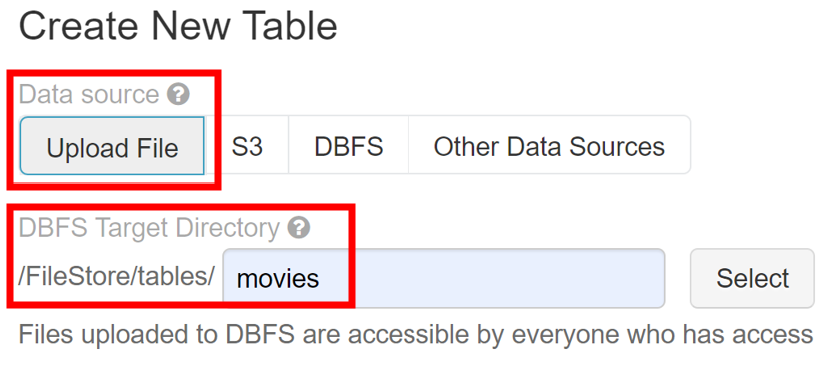
# Step 4: Ingest Movie Data to Notebook

Create the ratings and movies tables using the [Databricks Guide > Create a table using the UI](https://docs.databricks.com/user-guide/tables.html#create-a-table-using-the-ui)

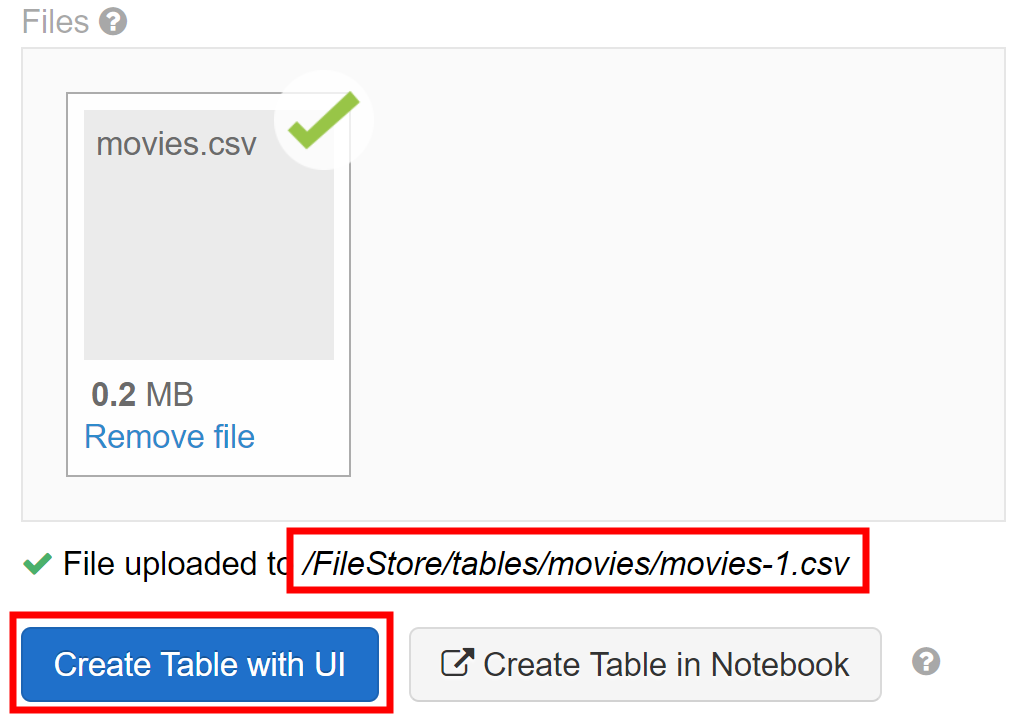
1. In the sidebar, click the **Data** button .
2. In the Data folder, select **Add Data**.



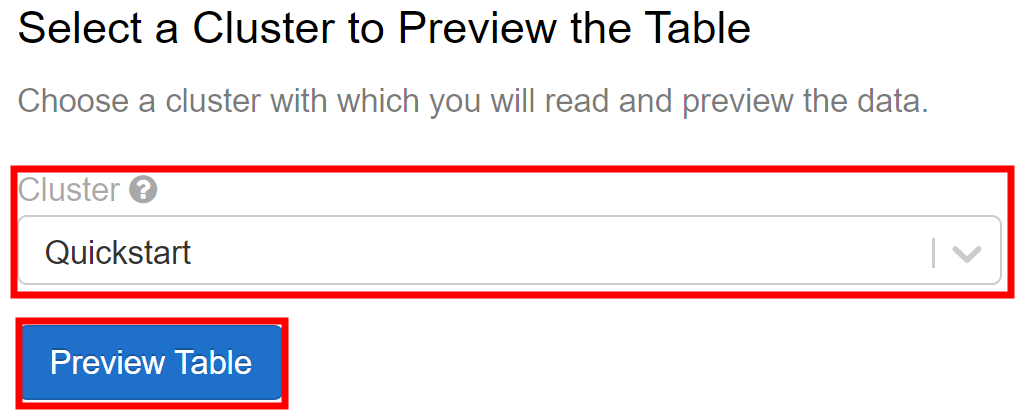
1. Keep the data source as Upload File and name the **DBFS Target Directory** as ‘movies’ (Table names can only contain lowercase alphanumeric characters and underscores and must start with a lowercase letter or underscore).



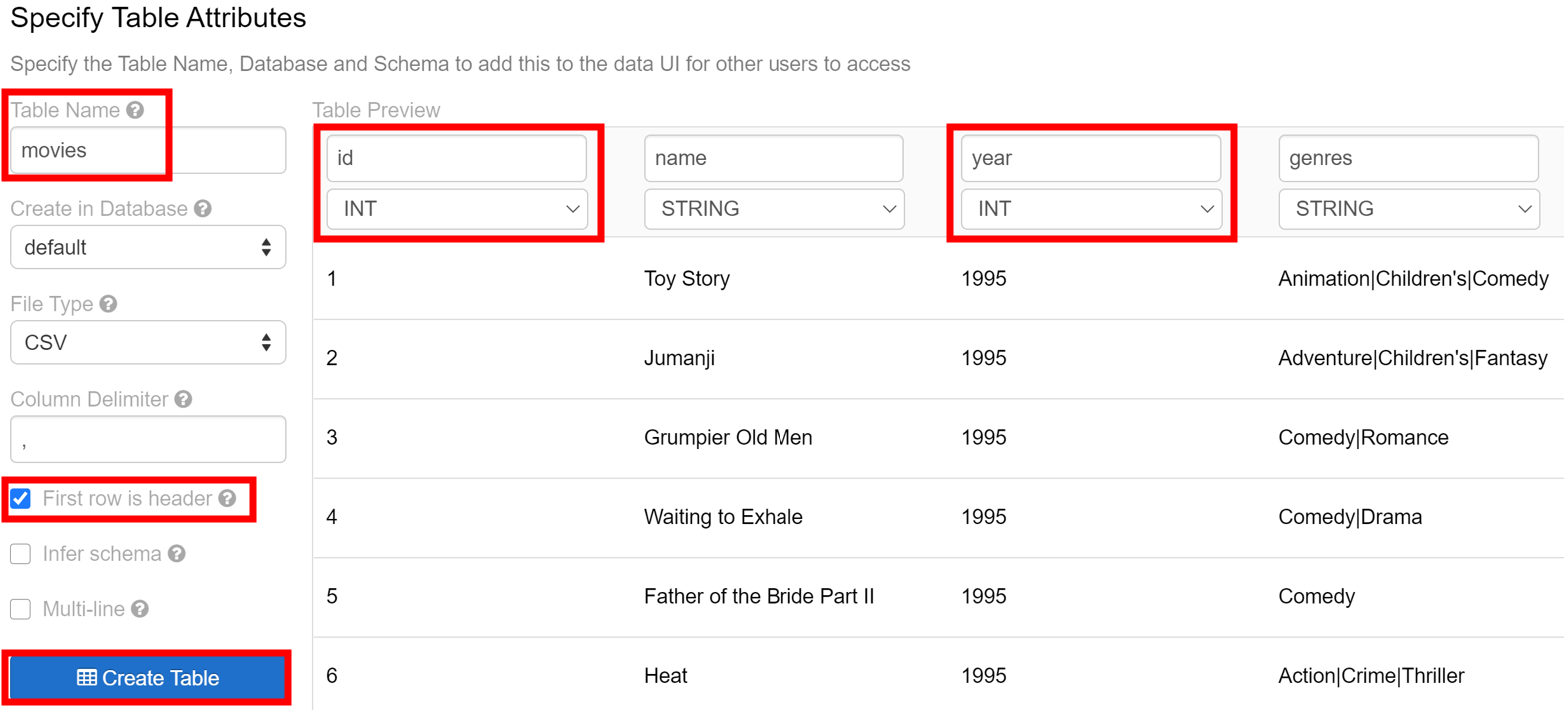
1. Drag file ‘movies.csv’ to the file drop zone or click the dropzone to browse to and choose files. After upload, a path displays for each file. The path will be something like /FileStore/tables/<filename>-<random-number>.<file-type> and you use this path in a notebook to read data.



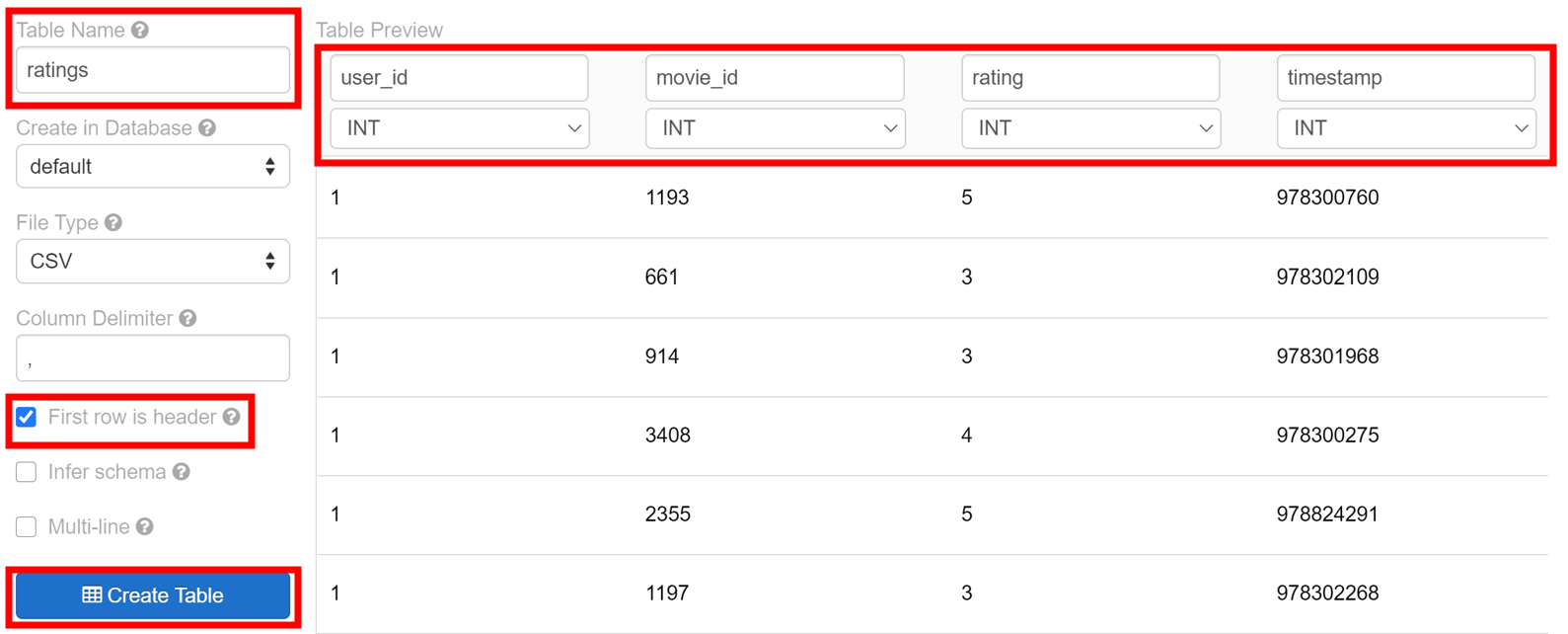
1. Click **Create Table with UI**.
2. In the Cluster drop-down, choose a cluster, then click **Preview Table**.



1. In the specify Table Attributes part, change the **Table Name** to ‘movies’. Check the **First row is header** box. Then change the data type for ‘id’ and ‘year’ to **INT in the drop-down.**



1. Click **Create Table**. (Make sure your cluster is running and allows you to add data)
2. Click the **Data** button in the side bar and select **Add Data** again. Name the DBFS Target Directory as ‘ratings’. Drag ‘ratings.csv’ to the drop zone. Then click **Create Table with UI**, choose a cluster and click **Preview Table**. Change **Table Name** to ‘ratings’, check the **First row is header** box then change all the data type to **INT**. At last, click **Create Table**.



# Step 5: Configure Databricks Widgets

* The code snippet below will select 10 random movies out of the 200 topmost rated movies
* You will use **Databricks Widgets** to personalize your movie ratings. Go back to your databricks notebook and paste the following code chunk:

spark.sql("""

select

movie\_id, movies.name, count(\*) as times\_rated

from

ratings

join

movies on ratings.movie\_id = movies.id

group by

movie\_id, movies.name, movies.year

order by

times\_rated desc

limit

200

"""

).createOrReplaceTempView("most\_rated\_movies")

# Take a sample of 10 movies

most\_rated\_movies\_sample = spark.table("most\_rated\_movies").rdd.takeSample(**True**, 10)

# Update widgets with movies for you to rate

**for** i **in** range(0, len(most\_rated\_movies\_sample)):

dbutils.widgets.dropdown("movie\_%i" % i, "5", ["1", "2", "3", "4", "5"], most\_rated\_movies\_sample[i].name)

# Step 6: Personalize Your Movie Ratings with Databricks Widgets

Change the values on top to be your own personal ratings before proceeding. Right now they’re all 5 but you should change it into arbitrary values.



For example, I change it as follows:



The following code snippet will process your personal movie ratings.

# Create DataFrame based on your own personal ratings

**from** datetime **import** datetime

**from** pyspark.sql **import** Row

ratings = []

**for** i **in** range(0, len(most\_rated\_movies\_sample)):

ratings.append(

Row(user\_id = 0,

movie\_id = most\_rated\_movies\_sample[i].movie\_id,

rating = float(dbutils.widgets.get("movie\_%i" %i))

)

)

myRatingsDF = spark.createDataFrame(ratings)

# Create myRatings DataFrame with specific column order to match `ratings`

myRatings = myRatingsDF.select("user\_id", "movie\_id", "rating")

myRatings.createOrReplaceTempView("myRatings")

%sql

-- Display your ratings

**select** f.movie\_id, m.name, f.rating **from** myRatings f **inner** **join** most\_rated\_movies m **on** m.movie\_id = f.movie\_id

The output should look like this:



# Step 7: Create Training and Test Datasets for our ALS model

The following code snippet will split the dataset into training and testing datasets by 8/2 ratio.

**from** pyspark.sql **import** functions

ratings = spark.table("ratings")

ratings = ratings.withColumn("rating", ratings.rating.cast("float"))

ratings = ratings.drop("timestamp")

# Split our data for our training and test datasets

(training, test) = ratings.randomSplit([0.8, 0.2])

# Step 8: Create ALS Model on Training (and Personalized) Data

The following code snippet will create a ALS collaborative model based on training and personalized data.

**from** pyspark.ml.recommendation **import** ALS

# Run ALS collaborative filtering

als = ALS(maxIter=5, regParam=0.01, userCol="user\_id", itemCol="movie\_id", ratingCol="rating")

# Run training model which includes your own ratings

model = als.fit(training.unionAll(myRatings))

# Step 9: Review ALS Model Predictions Using Test Dataset

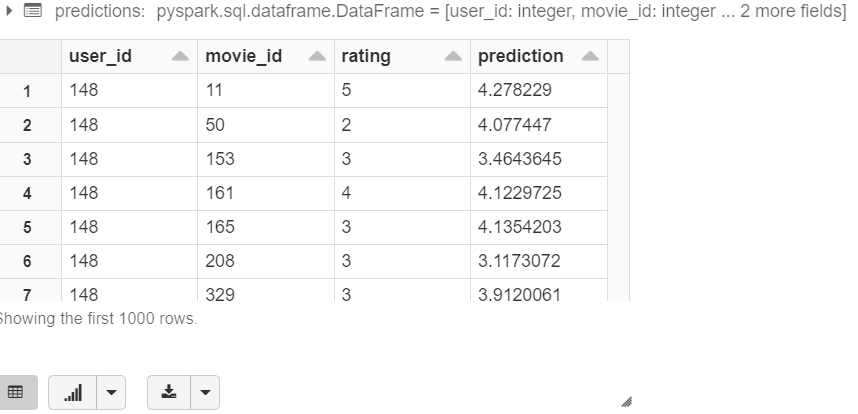
The following code snippet will display the result of the ALS model created in Step 7.

predictions = model.transform(test).dropna()

predictions.createOrReplaceTempView("predictions")

display(predictions)

The output should look like this:



Note that your number might be different from mine, since it depends on how you input the ratings in step 5

# Step 10: Evaluate the Model

The following code snippet will evaluate the ALS model created in Step 7 by its RMSE (Root Mean Squard Error). Small RMSE stands for high accuracy.

# Evaluate the model

**from** pyspark.ml.evaluation **import** RegressionEvaluator

evaluator = RegressionEvaluator(metricName="rmse", labelCol="rating", predictionCol="prediction")

rmse = evaluator.evaluate(predictions)

# Display using `displayHTML`

displayHTML("<span style='font-size:14pt;color:purple'>The Root Mean Square Error is %s</span>" % str(rmse))

The output should look like this:



Note that your number might be different from mine, since it depends on how you input the ratings in step 5

# Step 11: How well does the ALS Model Predict You?

The following code snippet will display how the ALS model predict your personal ratings.

# Ensure the columns are the same order as the model is expecting

myRatings = myRatingsDF.select("user\_id", "movie\_id", "rating")

# Run the same ALS model on my personalized Ratings

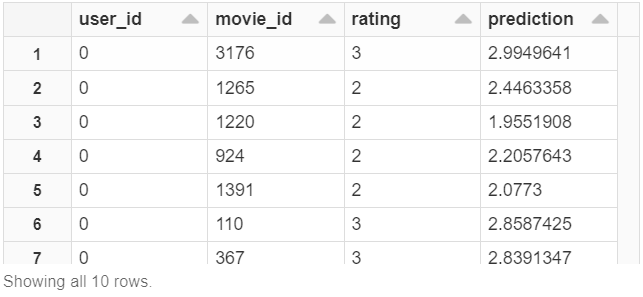
myPersonalizedMovies = model.transform(myRatings).dropna()

myPersonalizedMovies.createOrReplaceTempView("myPersonalizedMovies")

# Display the model's predicted ratings

display(myPersonalizedMovies)

The output should look like this:



Note that your number might be different from mine, since it depends on how you input the ratings in step 5

The following code snippet will show the RMSE of ALS model predicting your personalized ratings.

# Evaluate model for my personalized ratings movies

my\_rmse = evaluator.evaluate(myPersonalizedMovies)

# Display using `displayHTML`

displayHTML("<span style='font-size:14pt;color:purple'>The Root Mean Square Error is %s</span>" % str(my\_rmse))

The output should look like this:



Note that your number might be different from mine, since it depends on how you input the ratings in step 5

# Step 12: Find My Top 10 Movies based on My Ratings

The following code snippet will show the top 10 movie the ALS model picked for you.

# Take the list of the movies you already rated (myRatings), the list of most\_rated\_movies (removing the ones you already rated)

# and assign them to yourself (user\_id = 0)

sqlQuery = """

select null as user\_id, null as movie\_id, null as rating union all

select user\_id, movie\_id, rating from myRatings union all

select cast(0 as bigint) as user\_id, movie\_id, cast(0 as float) as rating from most\_rated\_movies where movie\_id not in (select movie\_id from myRatings)

"""

most\_rated\_movies\_n = spark.sql(sqlQuery)

most\_rated\_movies\_n.createOrReplaceTempView("most\_rated\_movies\_n")

# Applying and then removing NULL to ensure original ratings and most\_rated\_movies\_0 schema match

most\_rated\_movies\_0 = spark.sql("select \* from most\_rated\_movies\_n where user\_id = 0")

# Re-apply our ALS model for all movies

movies\_predicted\_0 = model.transform(most\_rated\_movies\_0).dropna()

movies\_predicted\_0.createOrReplaceTempView("movies\_predicted\_0")

display(movies\_predicted\_0)

%sql

-- Show Your Top 10 movies

**select** m.name, f.prediction

**from** movies\_predicted\_0 f

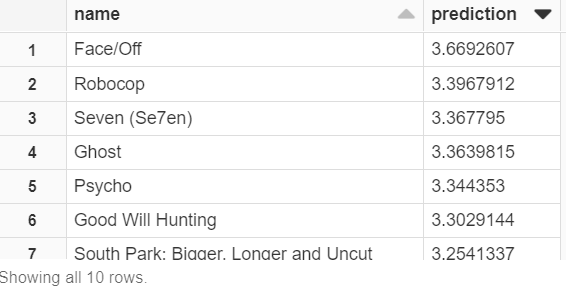
**inner** **join** most\_rated\_movies m

**on** m.movie\_id = f.movie\_id

**order** **by** f.prediction **desc**

**limit** 10

The output should look like this:



Note that your number might be different from mine, since it depends on how you input the ratings in step 5