**Zach Adair Final Project Write Up**

**Spark Project, Creating an Inverted Index with Stack Overflow Data**

**Regis University – College of Computer & Information Sciences**

**MSDS 610 – Data Engineering**

**February 19, 2019 – March 5th 2019**

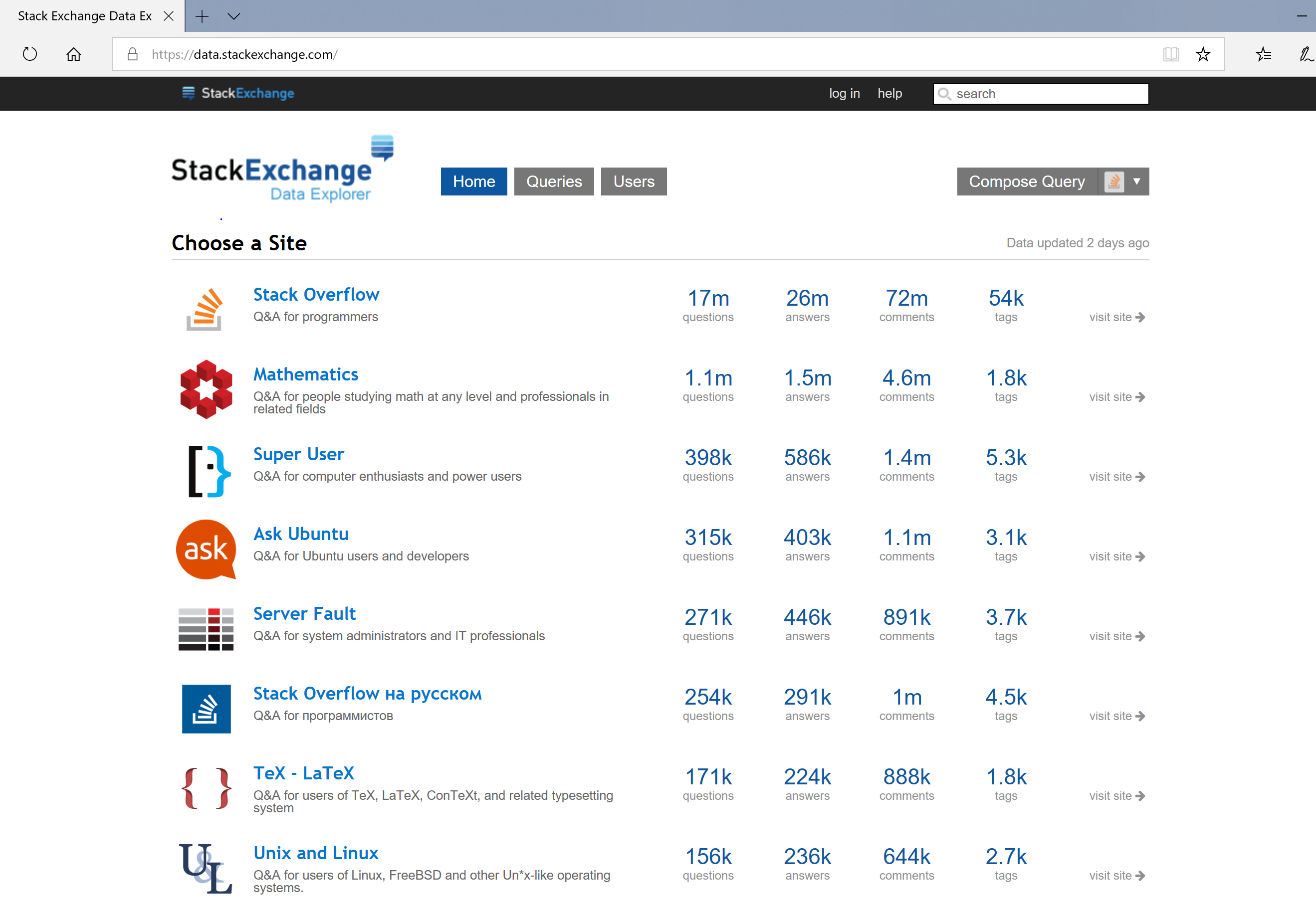
# Lab Overview

We will be creating an inverted index just like the one outlined using the Wikipedia data at the beginning of the Project Notes given from class. The data we will be using instead will be from Stack Overflow.

## Problem:

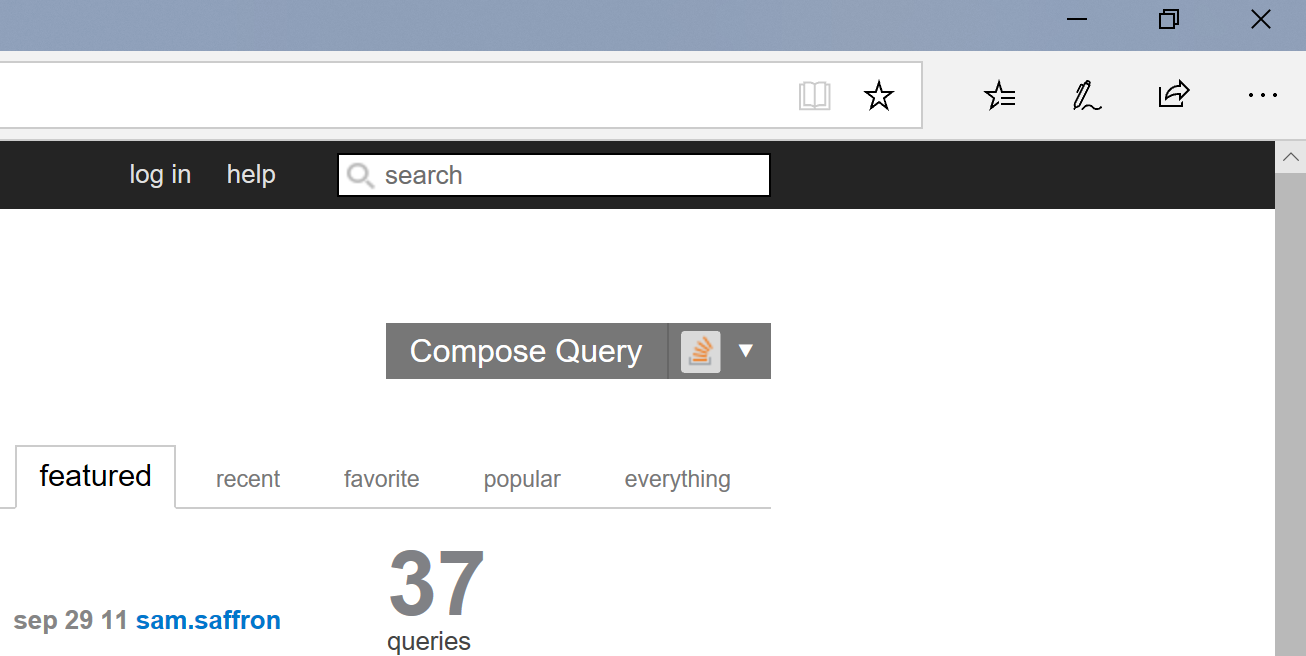
Query Stack Overflow’s data API to create and download a dataset. Next, use Spark, Jupyter notebook, and **your cluster to create an inverted index listing all the Post IDs where each tag appears**. To limit the query time and data transfer, please only use data from one year (Ex. 2018)

Stack Overflow publicly shares their data at <https://data.stackexchange.com/>



Clicking on the Stack Overflow link will show an interface where you can see other peoples’ saved queries. Investigate them to get ideas.

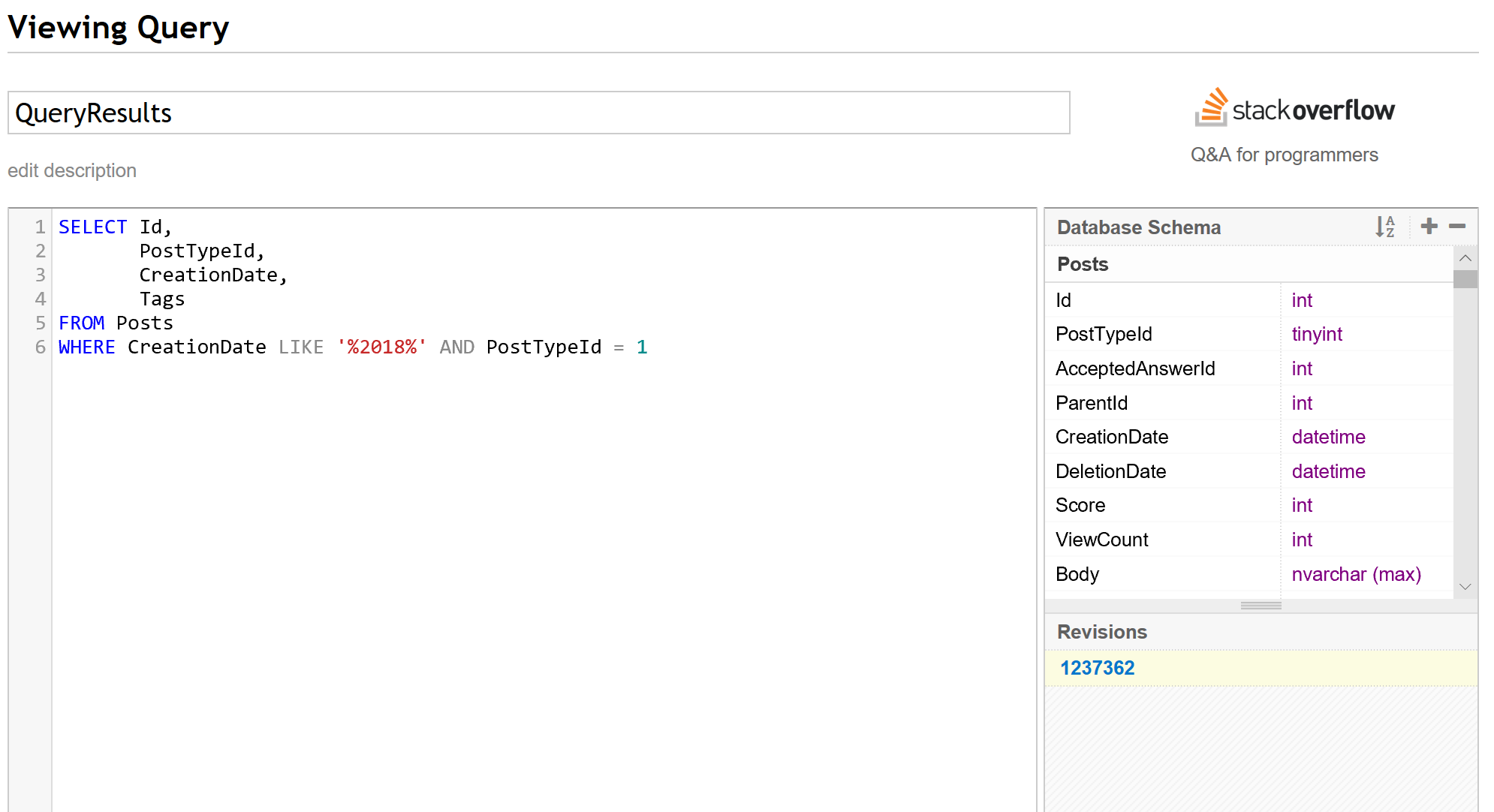
Next, you will click on “Compose Query.” Clicking that will take you to a page where you can see table schemas, enter queries and see results:

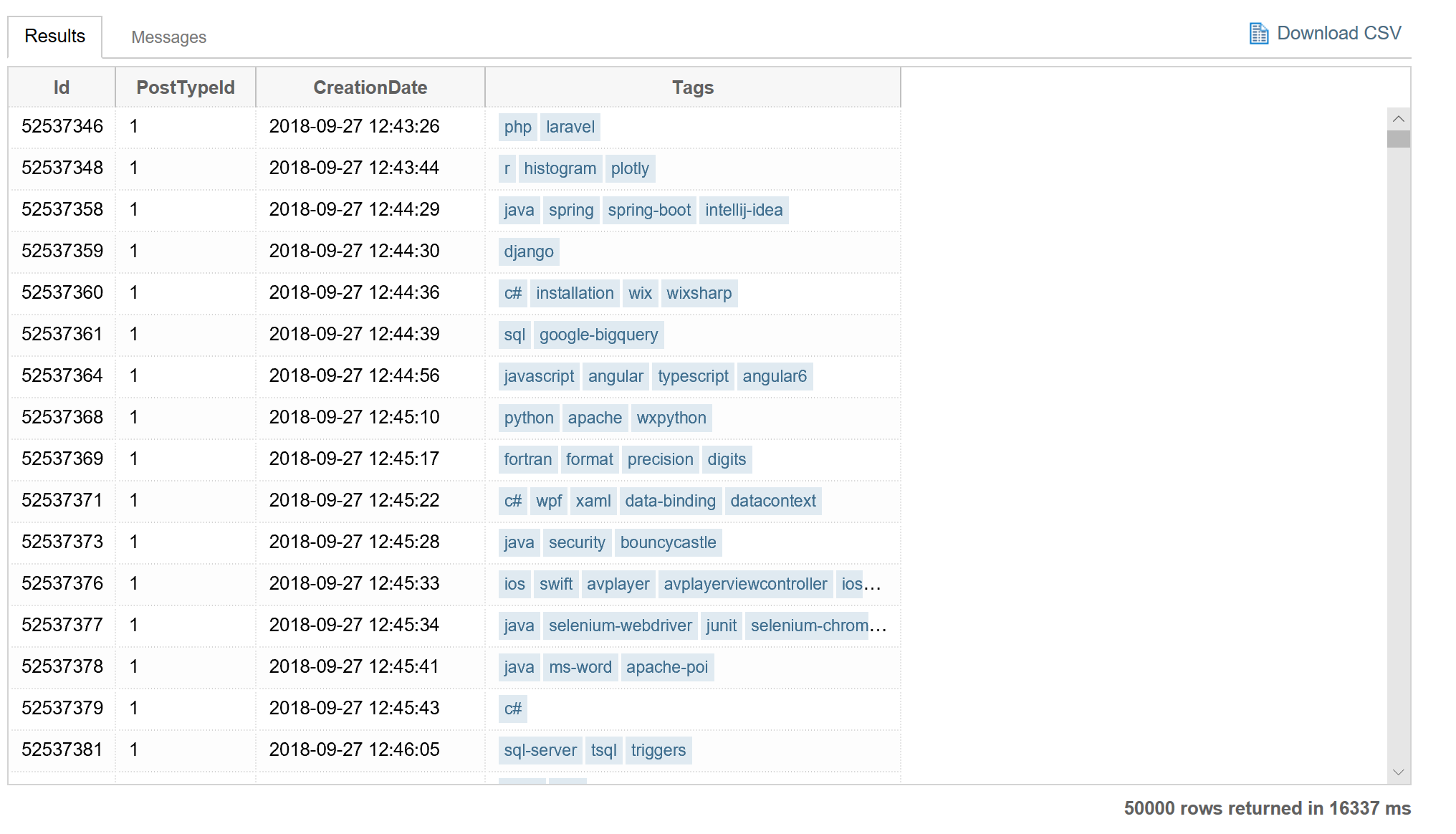


You are encouraged to explore the tables and data using this interface. What tables and columns do you think will help you build the inverted index?

After playing around with the Stack Overflow tables I came up with a question and data set I want to use to make my Inverted Index.

**The question I’m tasked with is create an inverted index listing all the Post IDs where each tag appears.**





Now click the “Download CSV” link and save the data as a text file.

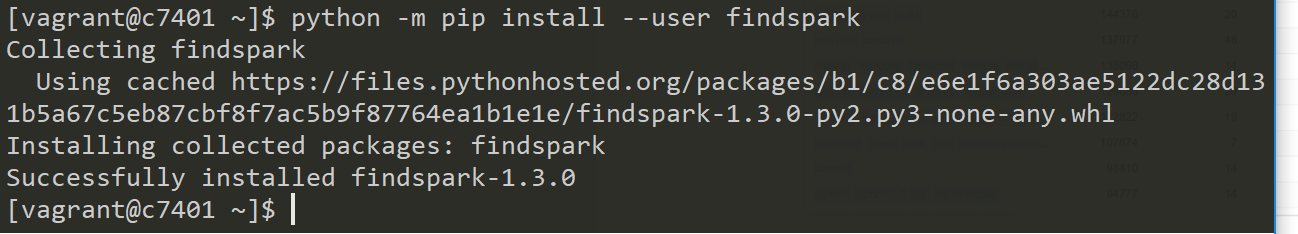
Next with this file we are ready to load the data into Spark and start Mapping and Reducing to get the inverted index. All of your work will be performed in Jupyter Notebooks. Review the Spark-Wordcount.ipynb for an example.

**GOAL: Submit the output file that represents the inverted index as well as a short report that discusses the question, process for answering the question, and reflections or lessons learned in the process.**

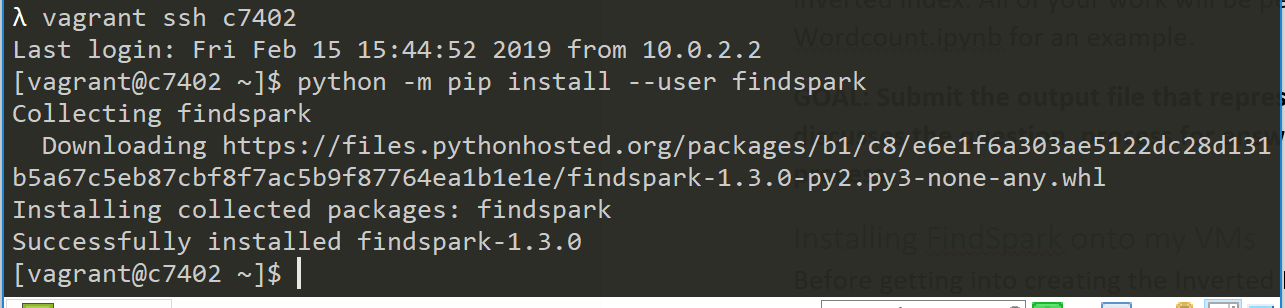
## Installing FindSpark onto my VMs

Before getting into creating the Inverted Index I need to first install findspark on both of my VMs that I have used throughout the duration of this class.

c7401:

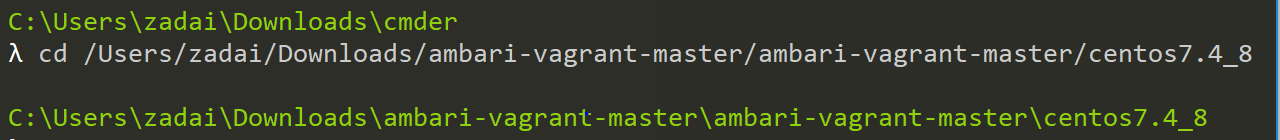


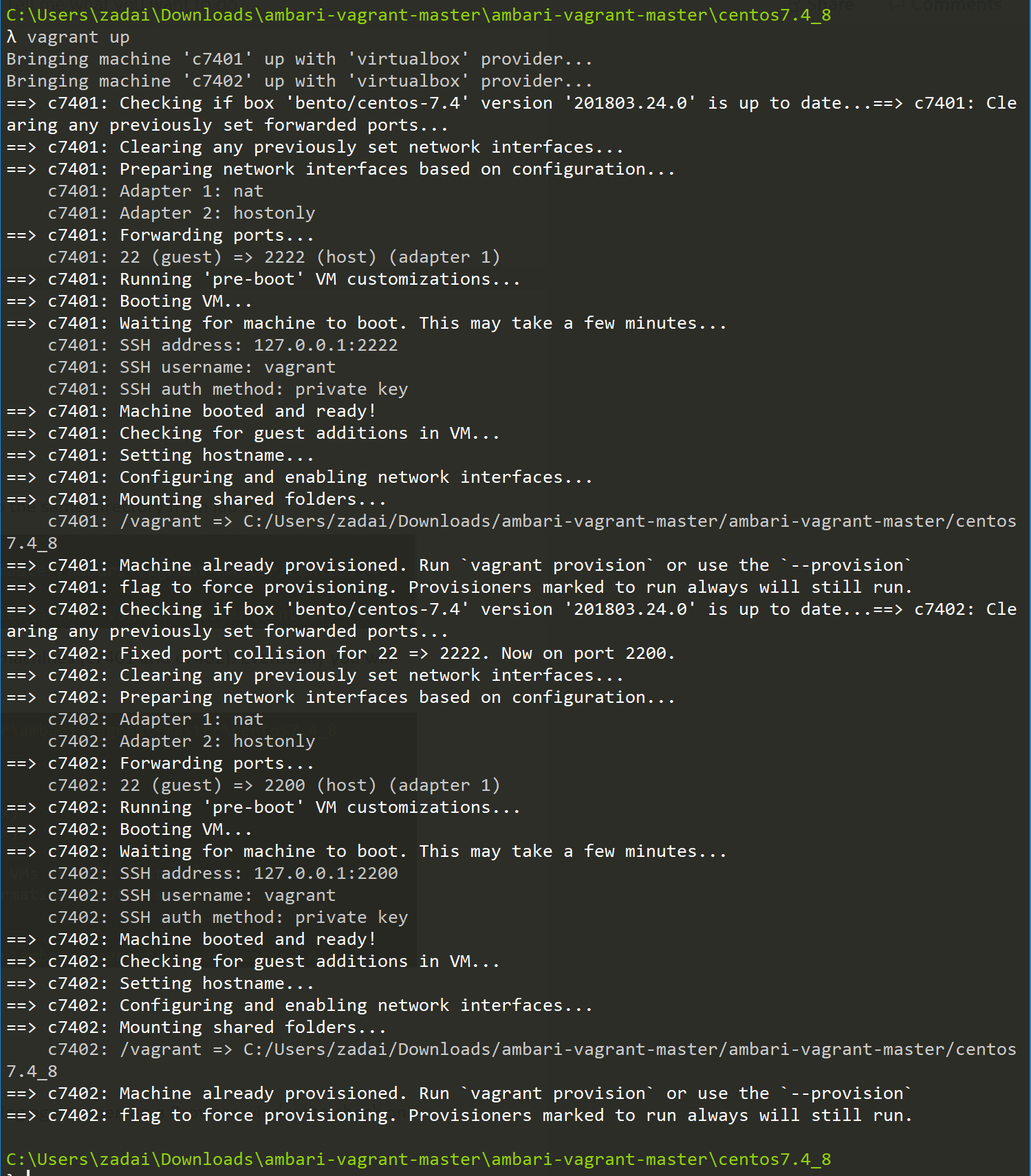
c7402:

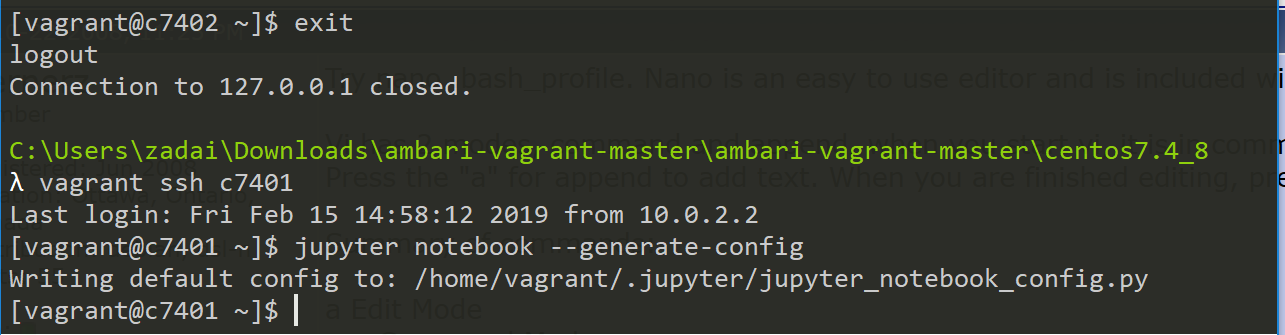


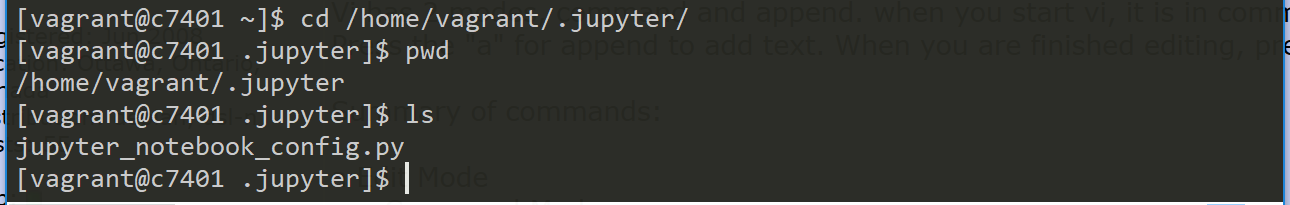
## Go into the Jupyter Notebook

From Lab 5 we learned how to make our own Jupyter Notebook environment from our Ambari-server. Get the server started back up and start up the Jupyter Notebook through it again.

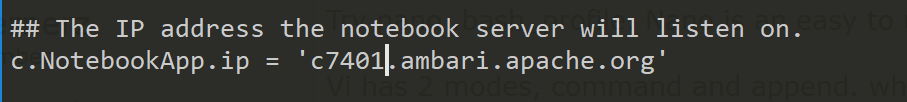


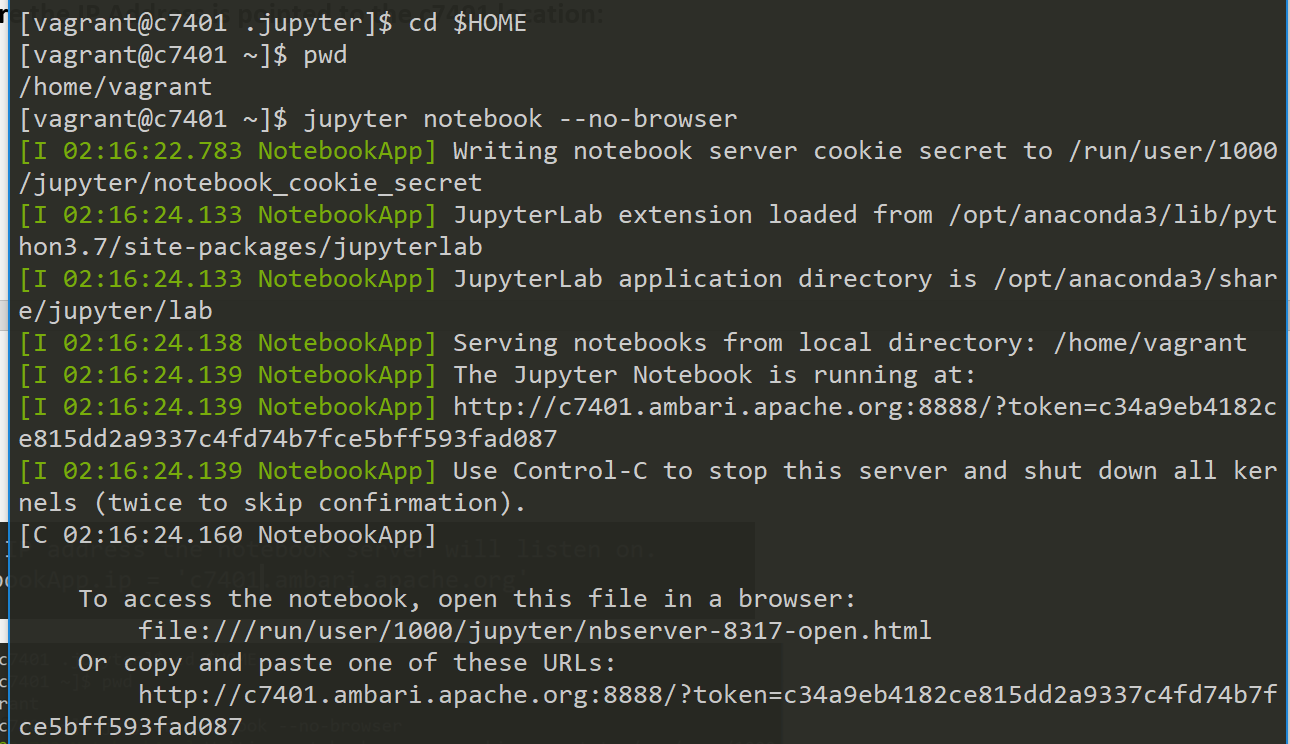




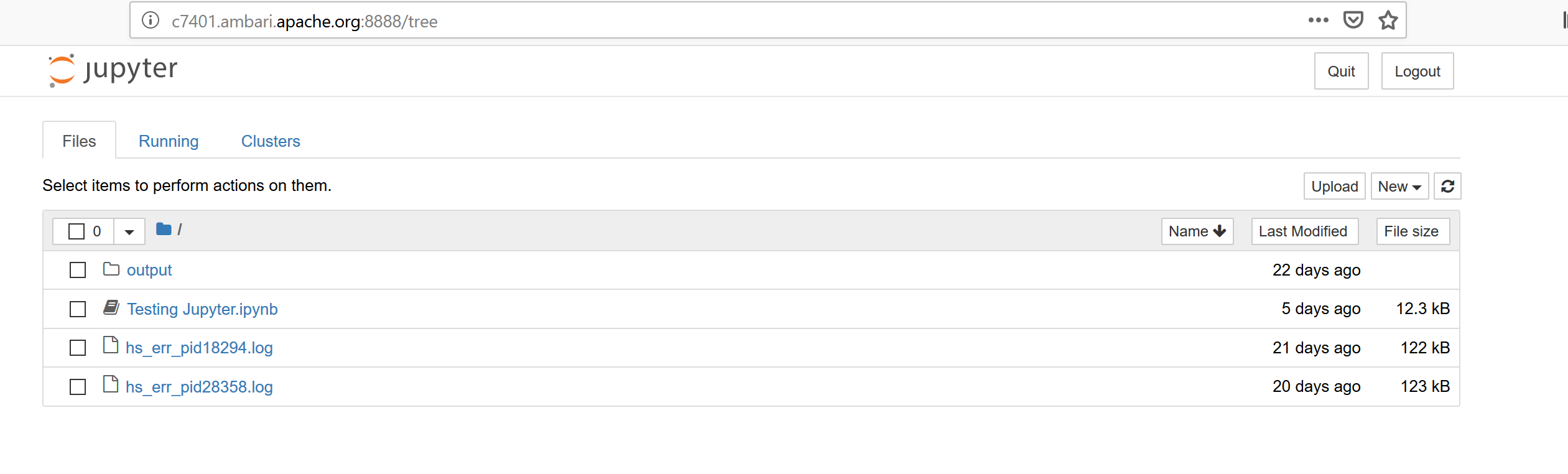


**Make sure the IP Address is pointed to the c7401 location:**





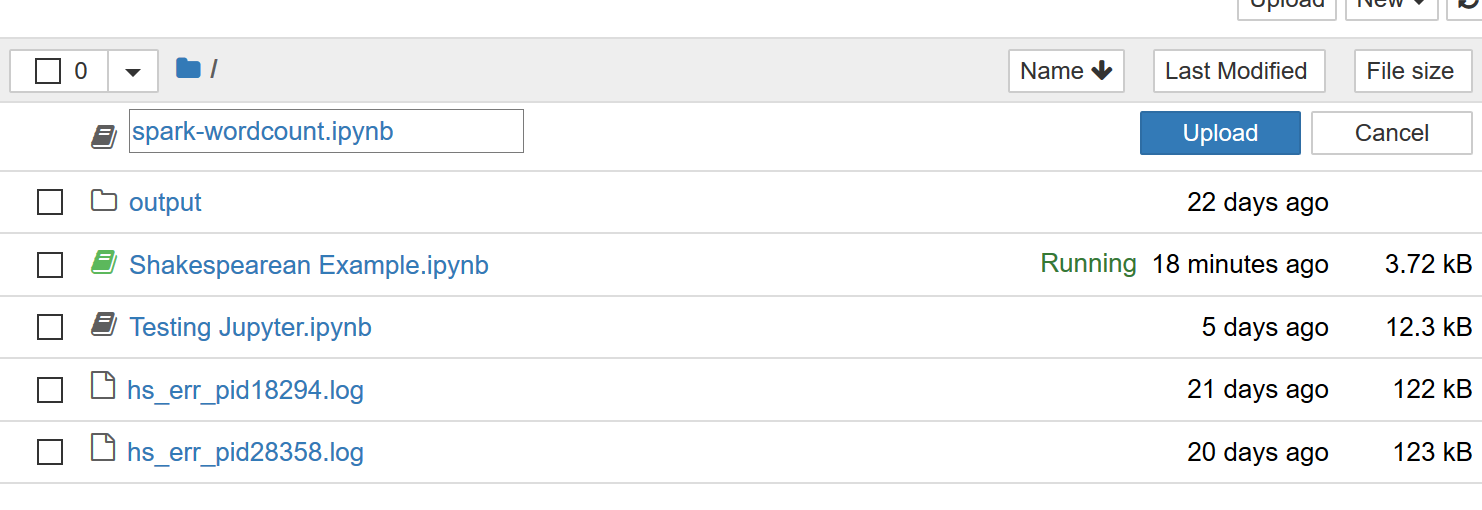
Copy the URL at the bottom and go to that location in a web browser.



Now that we are here, lets upload the Spark-Wordcount.ipynb file from WorldClass. In the Jupyter Notebook home page click Upload next to **New** on the right.



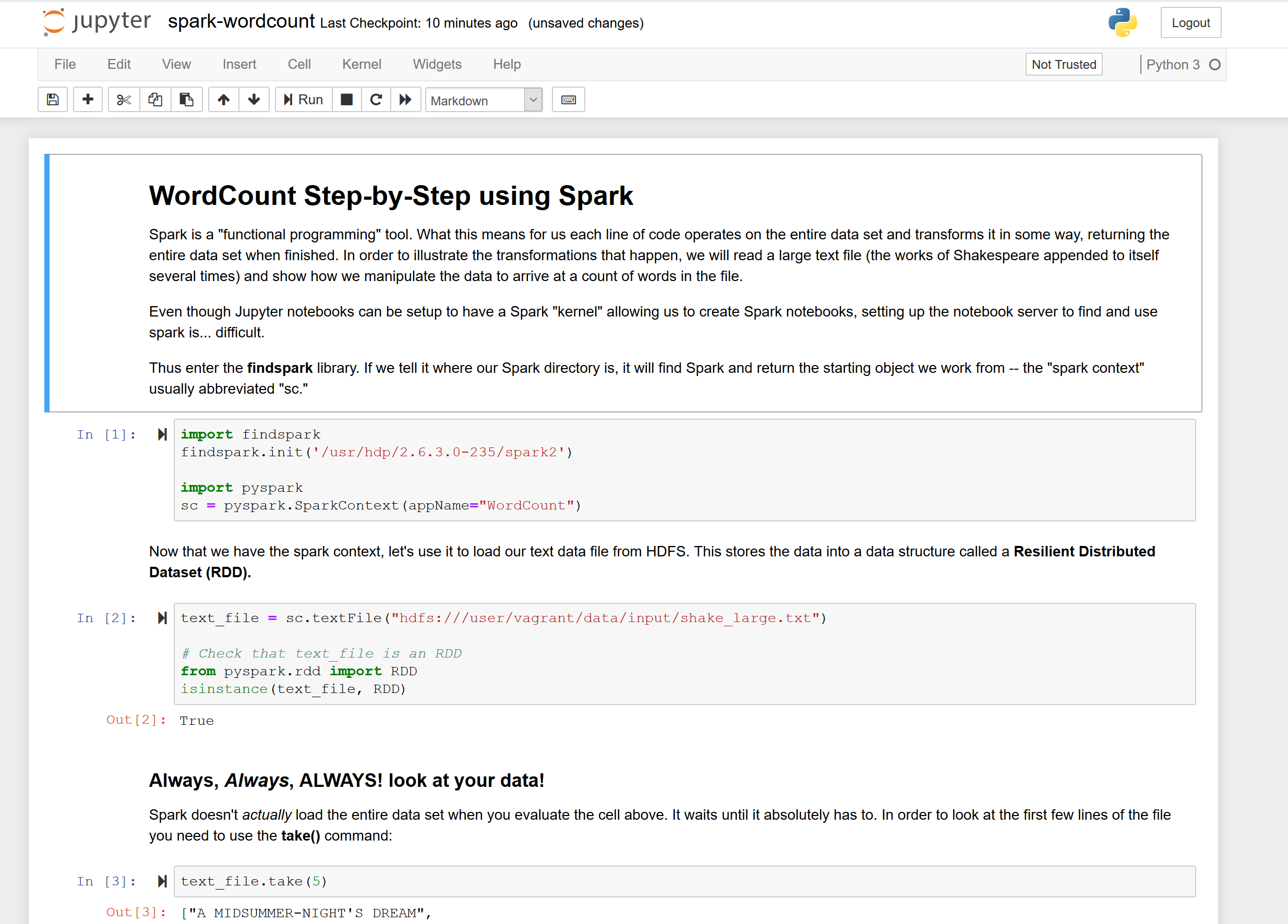
Once you find the file location you will see the file pull up above the different folders and files. Click on **Upload**



Once uploaded it becomes a useable file, click on it and go into the spark-wordcount.ipynb



Once in there you see that the file is a step by step instructions on how to work within Spark.



## Working through the Spark-Wordcount example

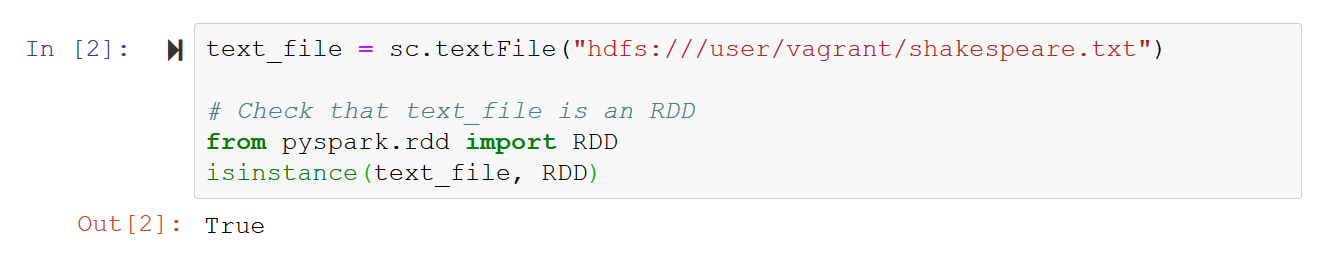
I wanted to personally work through the Spark-Wordcount example and make some notes as I go through it.

Starting from the beginning I need to import findspark and import pyshark. I had some trouble with importing FindSpark originally but I was able to solve it by changing the file location.

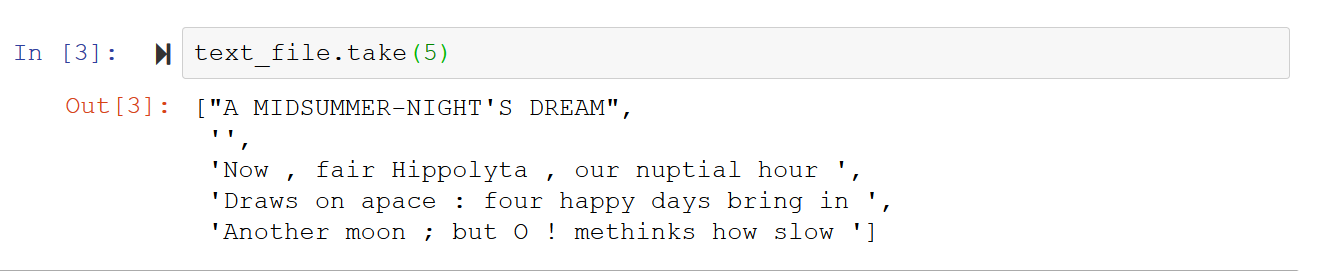


The way I fixed my issue was to go back into the Ambari-server and increase the memory within my Yarn Host to the suggested levels. That corrected the issue I was having and I was able to move onto step 2.

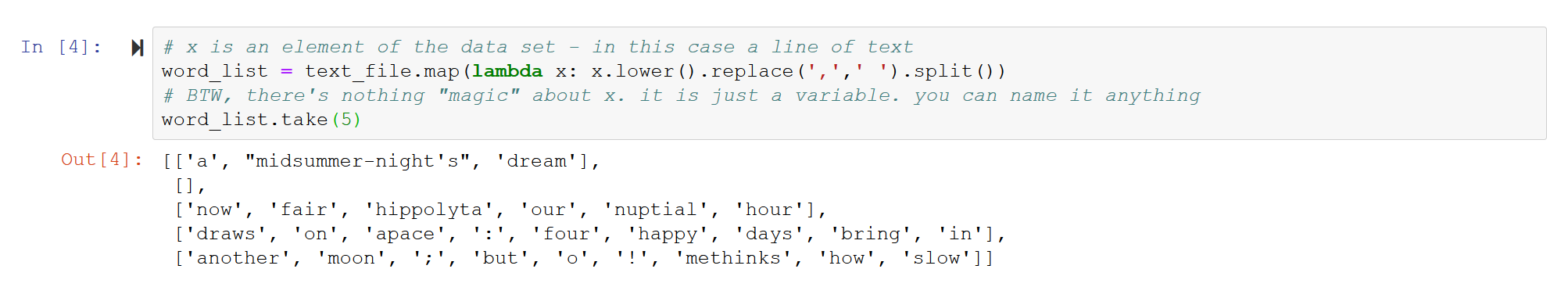
In the 2nd set of code we are setting the text file we’re going to use. The way the data is stored is as an RDD. After some personal research, an RDD is a fundamental data structure of Spark. It is an immutable distributed collection of objects. Each dataset in RDD is divided into logical partitions, which may be computed on different nodes of the cluster. This will be vital to the map-reduce function that we will be doing soon.



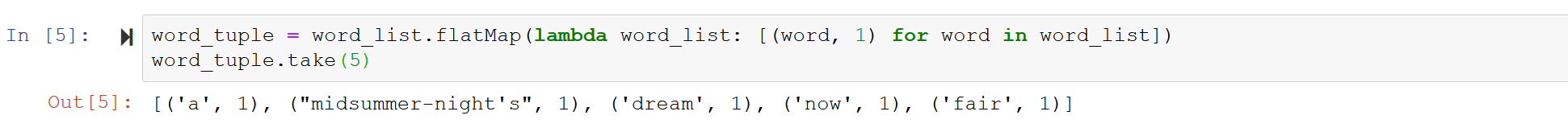
Next we’re going to take a look at our data, just the first 5 lines but make sure it’s in there correctly and no surprises. We are going to do that using a take command, and look at something simple, just the first 5 lines.



Now, we are going to break the lines into individual words. Doing so will make it easier to input our text for analysis and get our final output. Eventually we will have the individual words broken out and counted in a list format. But before that output let’s break up the words individually.



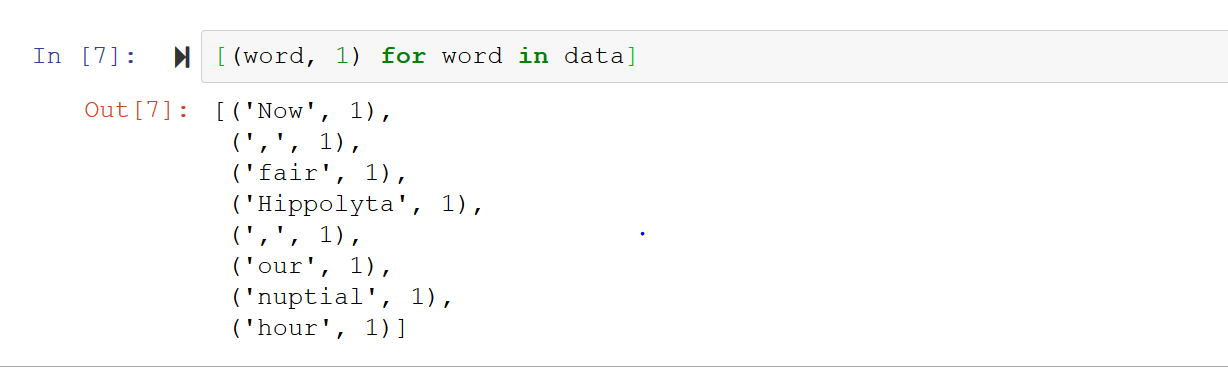
Next what were going to do is take our 5 words and list them out by key words and then individually count each word being used.



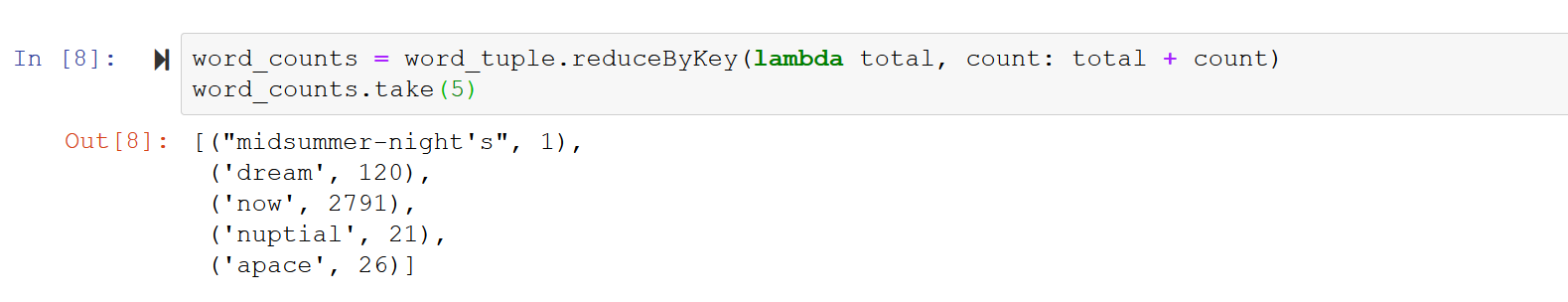
Now we’re going to change the format of how we look at the lines. We’re going to be able to have a line per word and the count of the words use as well.



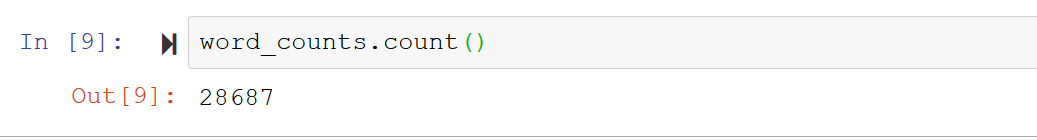
Next, we will simplify the code and see how will be able to make the same list with the data list created previously.



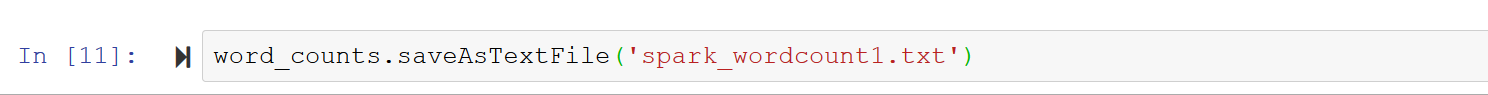
Now we will work on the reduce function with our word\_counts. Doing this will make sum the use of specific words throughout the entire function.



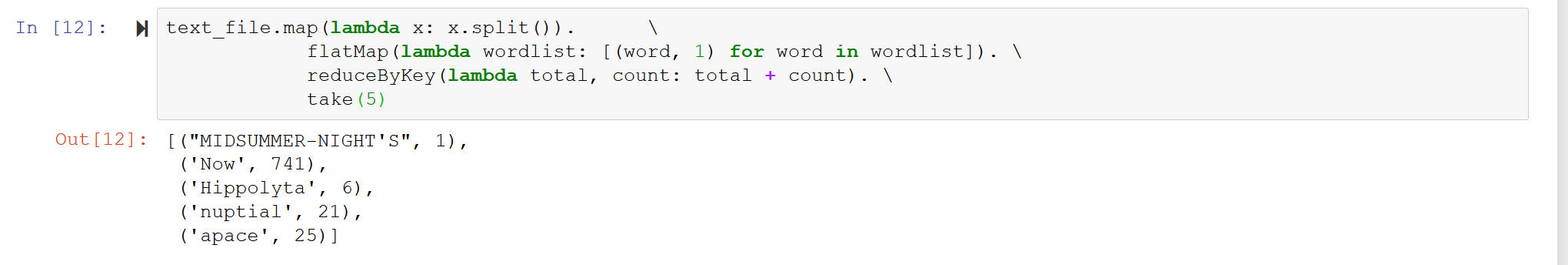
Look at how many times these 5 words were used! That’s pretty cool to see the sum of the use of these 5 words. Next, we will do a simpler line of code which just counts the total number of words.



Now we are going to save as a text file.



Spark are meant to be chained together, since that is the case, our next lines of code will chain multiple commands together and will do all of the commands at one time and not one at a time.



Able to do all the commands that we did one at a time, splitting the words flattening the map and reducing and counting the words. And that is the Spark-Wordcount example, this will be what we do with the query made from Stack Overflow data. What will be different is the use of data frames, the group by action, and the collect() action as well.

## Research Topics to help with creating Inverted Index

### DataFrames in Spark

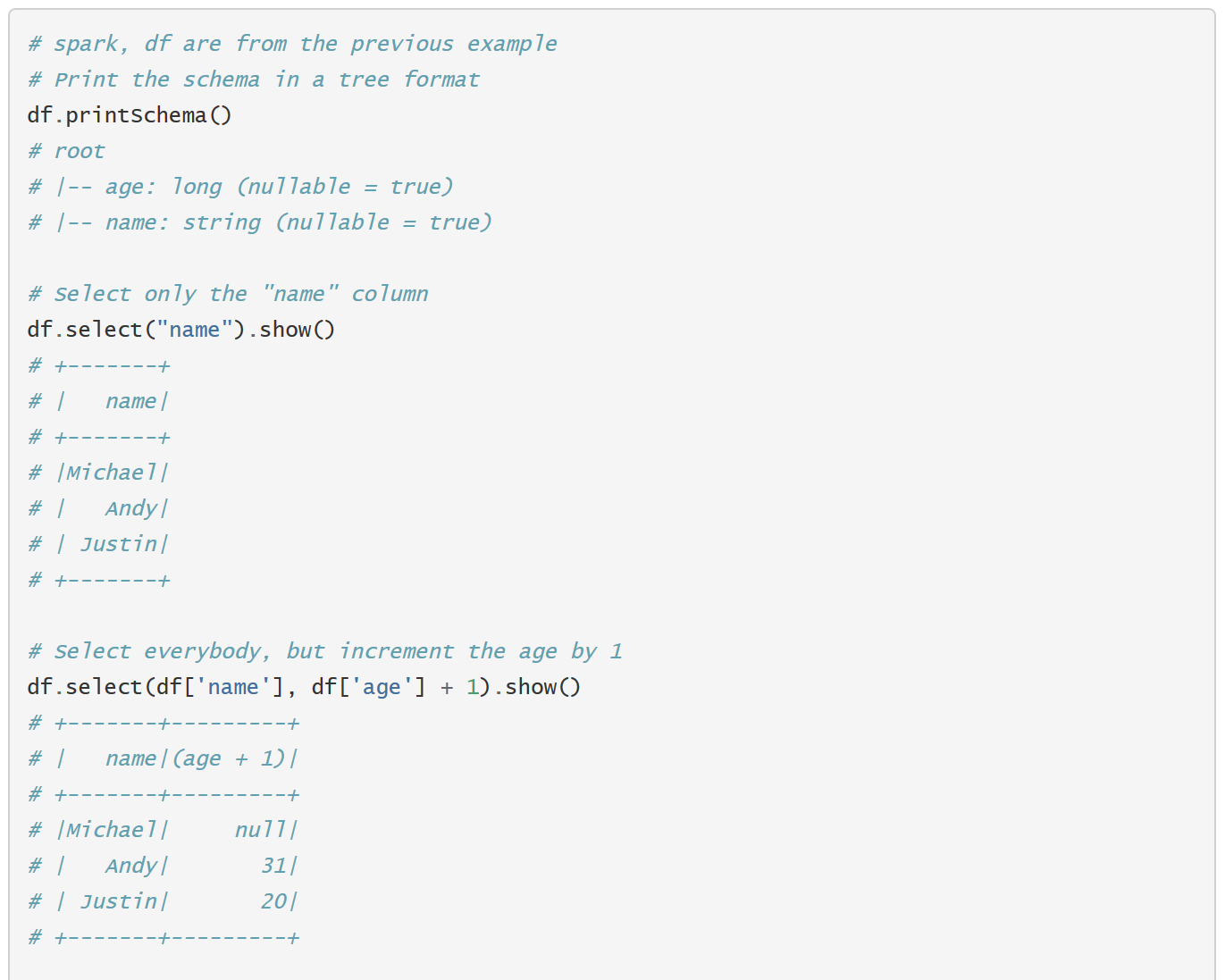
* DataFrames are datasets organized into named columns, it is conceptually equivalent to a table in a relational database or a dataframe in R/Python, but with richer optimization under the hood.
* DataFrames can be constructed froma wide array of sources such as: structured data files, tables in Hive, external databases, or existing RDDs.
* DataFrame API available int Scala, Java, Python and R
  + Scala and Java, dataframes are represented by a dataset of rows
  + Scala API, dataframes is simply a type of alias of dataset
* Creating DataFrames using Python
  + Starting point
    - SparkSession – entry point into all functionality in Spark is the SparkSession class
    - To create a basic SparkSession, just use SparkSession.builder:

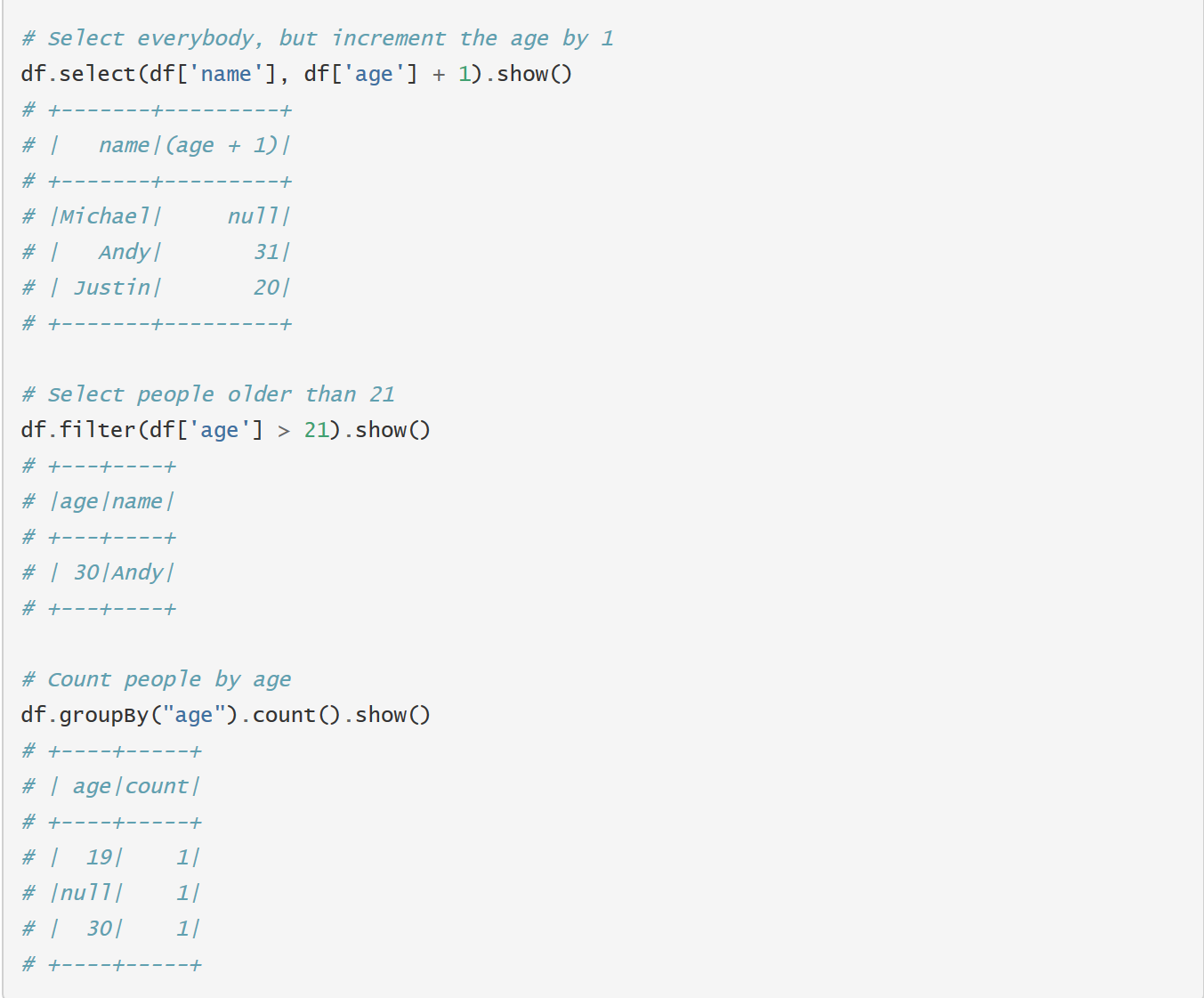


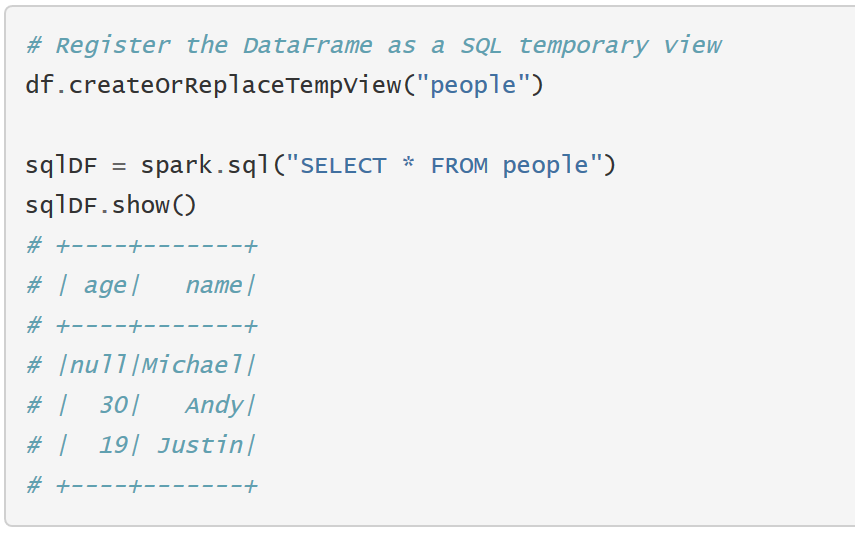
* + - SparkSession in Spark 2.0 provides a built-in support for Hive features including the ability to write queries using HiveQL, to Hive UDFs, and the ability to read data from Hive tables.
  + Creating DataFrames
    - With a SparkSession, applications can create DataFrames from an existing RDD, from a Hive table or from Spark data sources.
    - In the Example below, DataFrames based on the content of a JSON file:



* + - Full example code at “examples/src/main/python/sql/basic.py” in the Spark Repo
  + Untyped Dataset operations (DataFrame Operations)
    - DataFrames provide a domain-specific language for structured data manipulation in Scala, Java, Python, and R
    - As mentioned above, in Spark 2.0, DataFrames are just Dataset of Rows in Scala and Java API. These operations are also referred to as “untyped transformations” in contrast to “typed transformations” come with strongly typed Scala/Java Datasets:
    - In Python it’s possible to access a DataFrame’s columns either by attribute (df.age) or by indexing (df[‘age’])
    - While the former is convenient for interactive data exploration, users are highly encouraged to use the latter form, which is future proof and won’t break with column names that are also attributes on the DataFrame class:

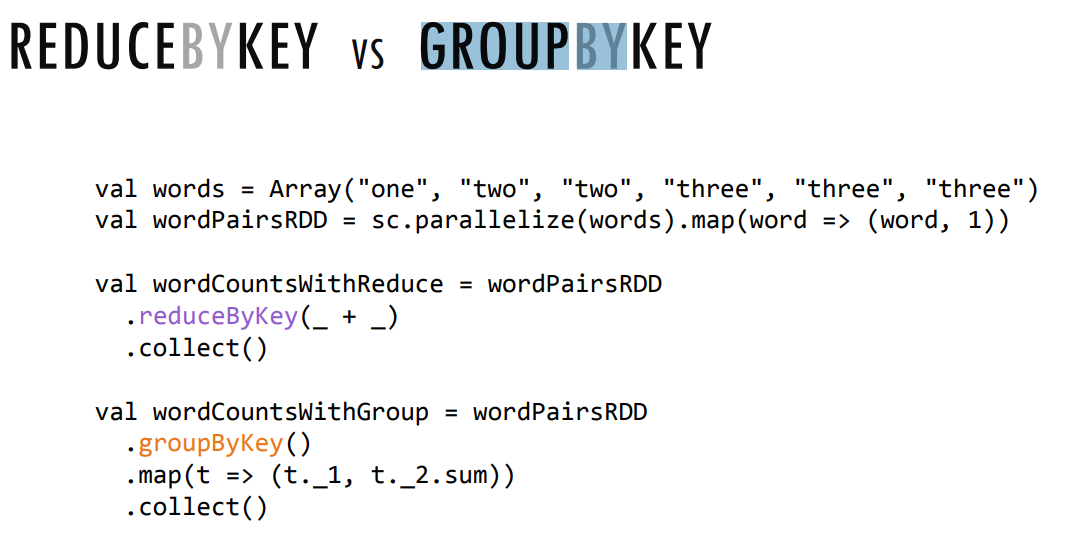




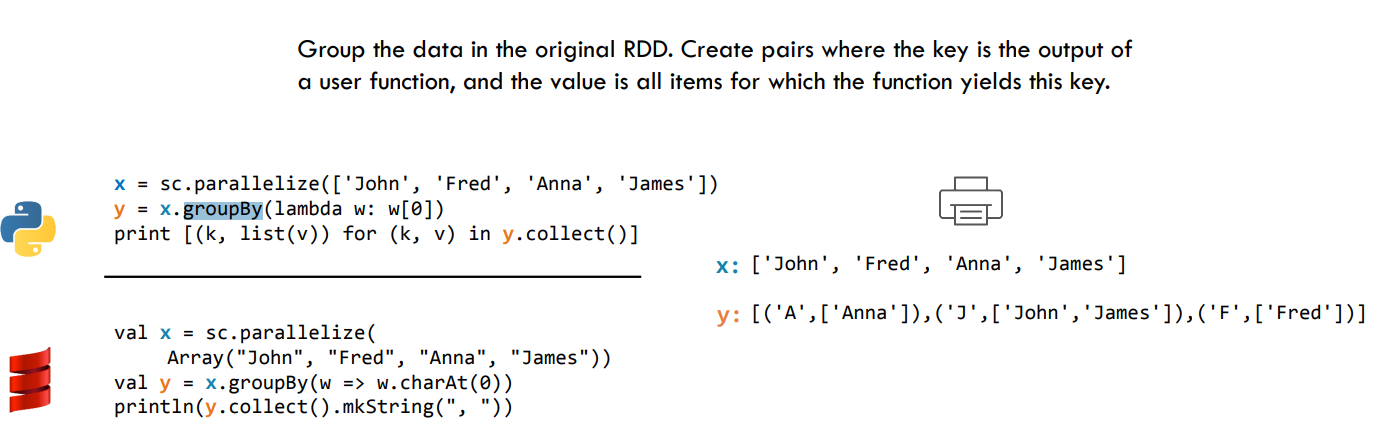
* + - Find the full example at “examples/src/main/python/sql/basic.py” in the Spark repo
    - DataFrames also have a rich library of functions including string manipulation, date arithmetic, common math operations and more
  + Running SQL Queries Programmatically
    - The SQL function on a SparkSession enables applications to run SQL queries programmatically and returns the result as a DataFrame
    - 

### GroupBy Key Action

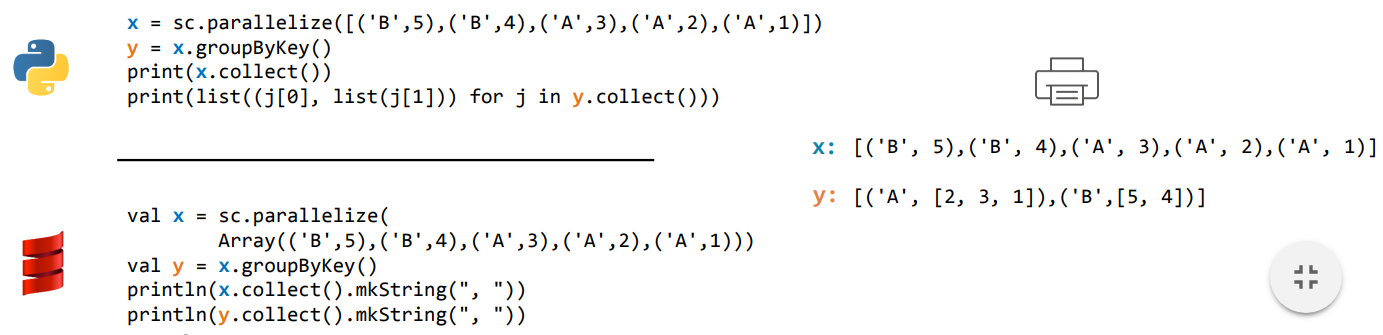
* Difference between GroupBy and ReduceBy
  + GroupByKey can cause out of disk problems as data is sent over the network and collected on the reduce workers
  + ReduceByKey, data is combined at each partition, only one output for one key at each partition to send over network. ReduceByKey required combining all your values into another value with the exact same type



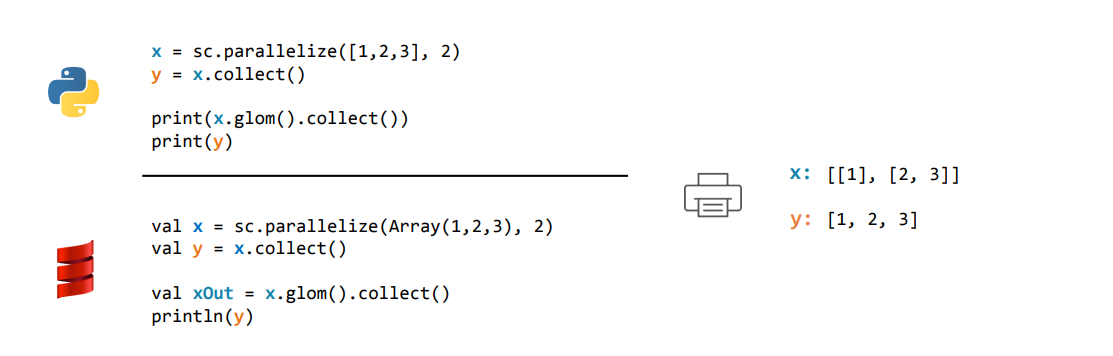
* In a GroupBy, we can take the items in the RDD or DataFrame and put them together
  + Group the values for each key in the original RDD. Create a new pair where the original key corresponds to this collected group of values



* Group the values for each key in the original RDD. Create a new pair where the original key corresponds to this collected group of values



* Collect – Returns all items in the RDD to the driver in a single list
  + Code Examples:



Hopefully some of this research outside of provided text will help with the completion of my inverted index.

# Creating an Inverted Index

Now I’m beginning my crack at creating an Inverted Index and these are the steps I’m going to be taking to its completion. I will also put in my notes for where I failed and where I needed to perhaps go in another direction or where I fix some misplaced code.

Starting from scratch means I’m using a brand new Jupyter Notebook workbook, so start by opening that:

We start by bringing in Spark and Pyspark into the Jupyter notebook:



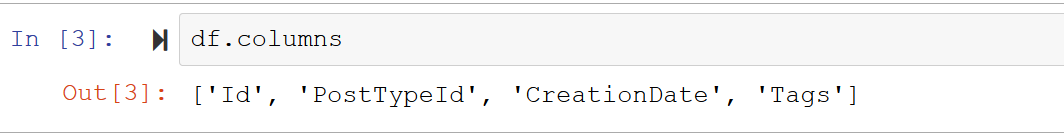
We also assigned the sc or Spark Context, and designated an app name of InvertedIndex because, its an Inverted Index.

Now we will import a few more things like Spark Session and SQL Context and assign spark, and after doing that we will construct our data frame with the simple name of df.

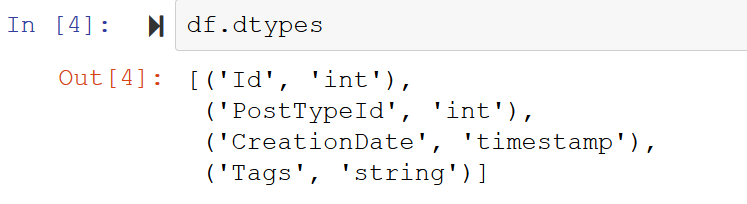


From here I wanted to play around with some commands and take a look at my data so that is what I did.

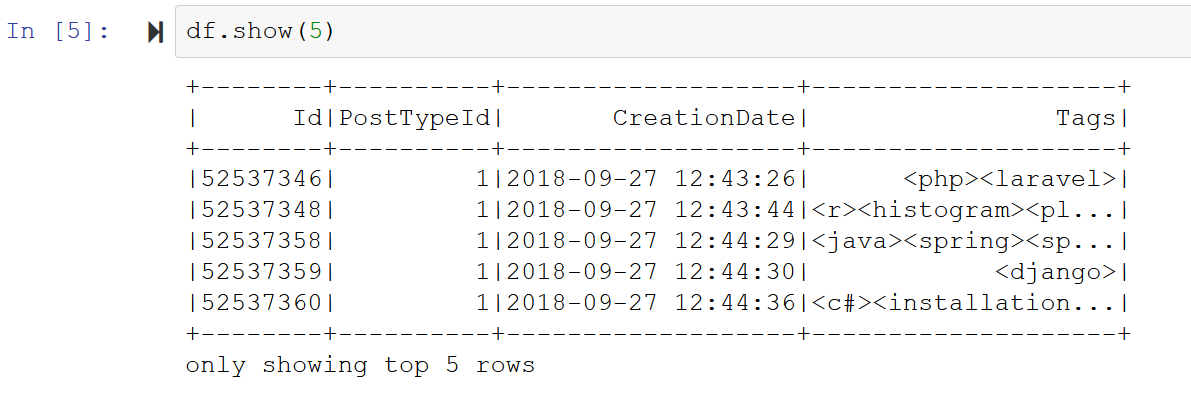
Started with looking at the columns that are in my data frame:



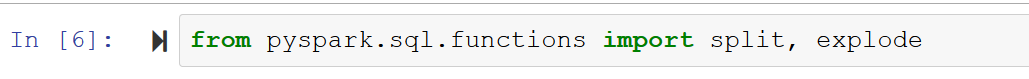
Next I went to look at the data types of each column:



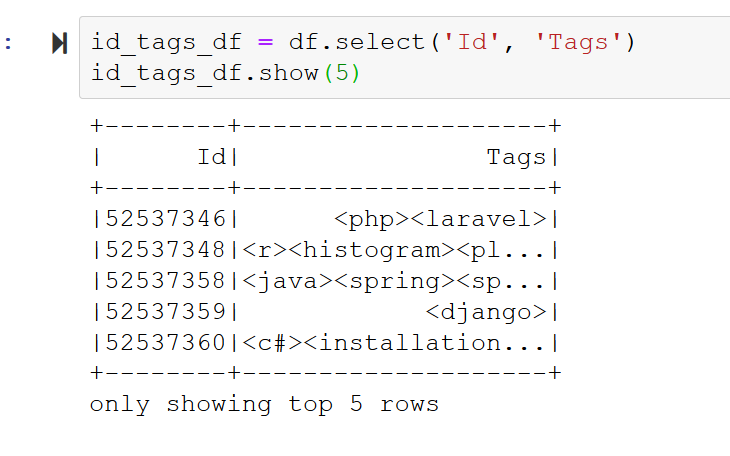
After, I wanted to look at the what was in my data, so I went in and took a look at the first 5 rows:



I was looking at what other people were doing with theirs through the class discussion and saw that people brought in split and explode from the pyspark.sql.functions and thought it would be a good thing to bring in myself and possibly play around with at the very least, if not use them to help create my inverted index:

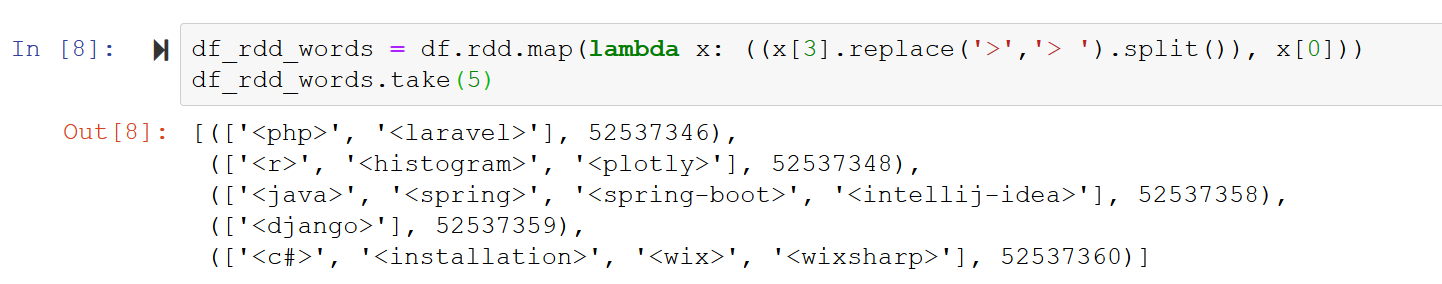


I also decided to play around with some data frame manipulation, so I chose just my Id’s and Tags fields and took a look at how to only bring in those two fields of my data frame:



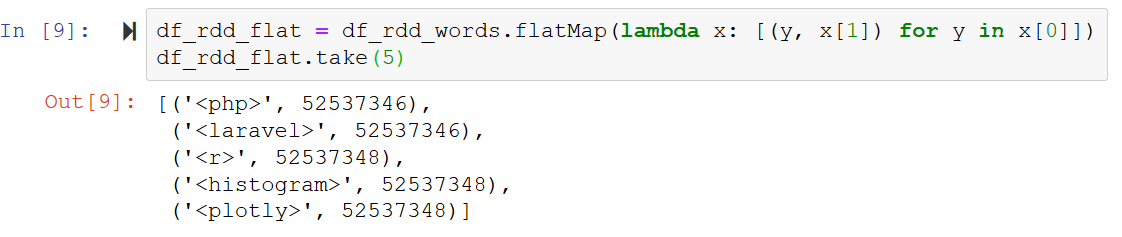
I’m not going to use the new dataframe above but it was fun just to see some of the different things Pyspark can do.

From here I’m going to work on how to construct my Inverted Index. I’ll start by mapping my rows, the tags per Id:

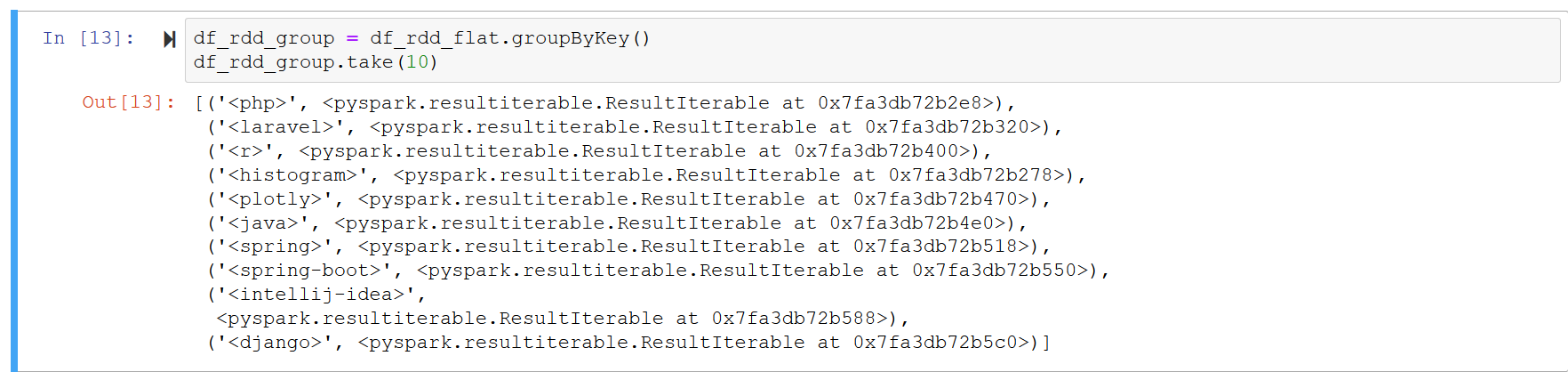


You will also notice in the command above I transformed my data frame into an RDD. Doing this will make it easier to make the transformations necessary to create my inverted index.

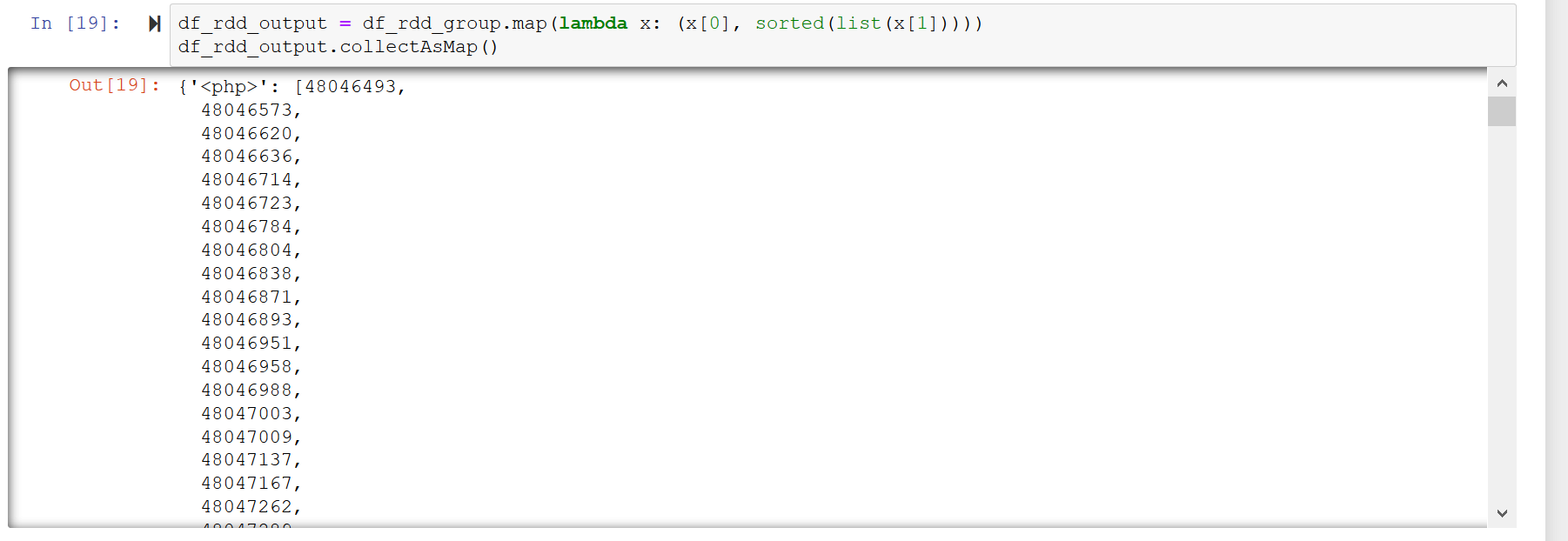
Now I want to try to make each row have just one tag and then the corresponding Id that goes along with it. This will cause my Id’s to duplicate but each tag will have one row for every time it was used. To do this I’m going to flat map my RDD:



From here I need to group my tags together and make sure they have been put together to then pull all the Ids together after.



Finally, we need to run the group by key transformation on our RDD, along with that we will run a map function and a collect function so that way our index is in the desired format for this project.



And above is that final format desired, grouped by each tags and listed out every Id associated with it.

## Reflection:

I found the Inverted Index Problem quite a trial by fire for me. I feel like I made constant mistakes and the amount research I did could make a weaker humans eyes bleed, but I found the end result very gratifying to complete. I found what I learned was a lot to do not only with Python, Jupyter Notebooks and the power of Spark but also I learned a lot about researching and problem solving my answers by researching the errors how they are given to you. The application all together is amazingly powerful and can do a whole lot of neat and interesting things. Piecing it all together and going out and finding new types of code was a challenge though and then going in and piecing it all together as well. All together I was really impressed with how the tool is able to analyze and organize textual data like this, and it also gives me excitement of what else I can learn as I push past this class and further into the data science program at Regis University.