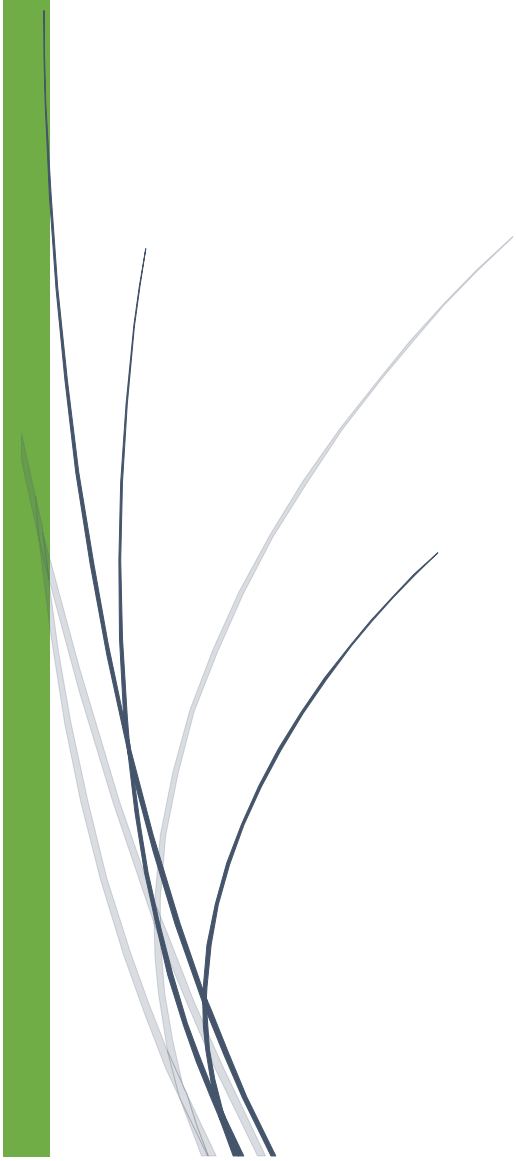
Hortonworks Data Platform

Hands On Labs

Hive , Zeppelin, Spark SQL



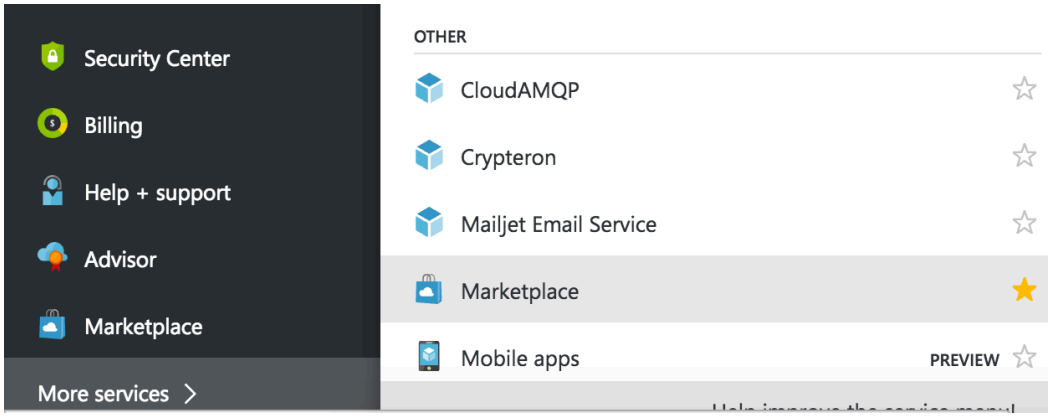
Hortonworks
5470 GREAT AMERICA PKWY, SANTA CLARA, CA, 95054

Table of Contents

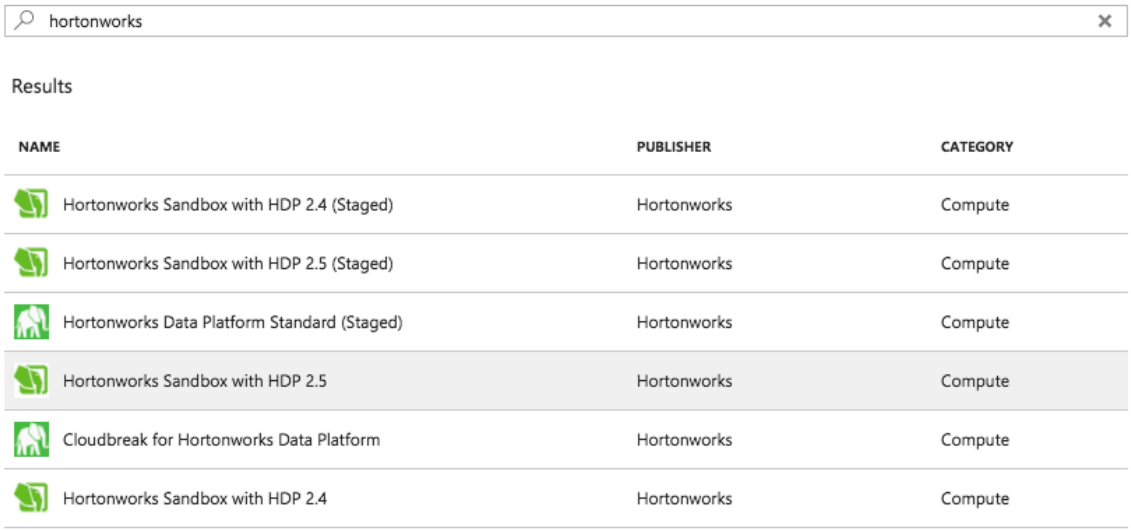
Lab Prerequisite – Setting up the Azure Environment

1. Sign into the Microsoft Azure portal.

2. On the bottom left hand pane, select **More Services** and filter for **Marketplace**.



3. Search/Filter for **Hortonworks** and select **Hortonworks Sandbox with HDP 2.5** than click **Create** in the new pane that opens on the bottom right.

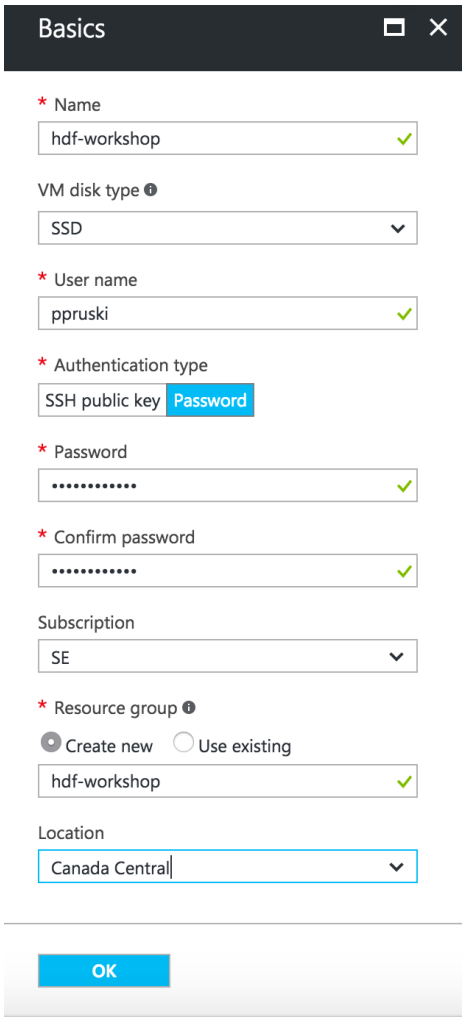


4. Follow instructions and fill in values as directed by the Wizard.

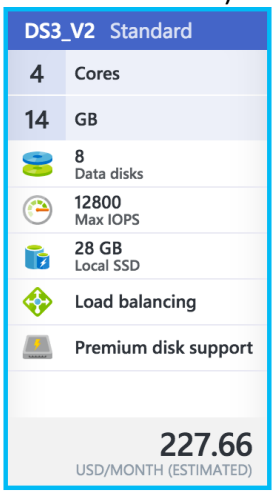
    a. Basics

        Use defaults for all unless otherwise specified below:

        - Name
        - User name
        - Authentication type: Password
        - Password
        - Confirm password
        - Resource group: n*ew or existing*
        - Location



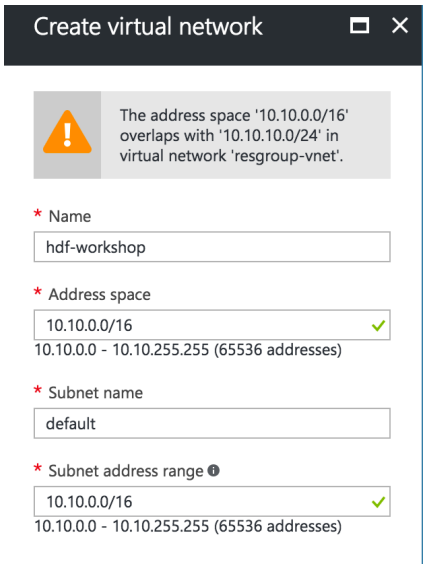
b. For an appropriate sized environment specify DS3_V2. (Note: if the size is unavailable an alternate location can be used)



c. Settings

Use defaults for all unless otherwise specified below:

- Virtual Network: *Select existing or create new*
  - *If new:*



d. Summary

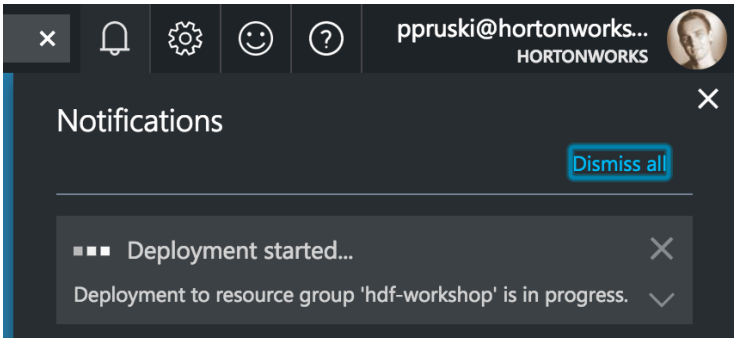
After reviewing, click **OK**.

s

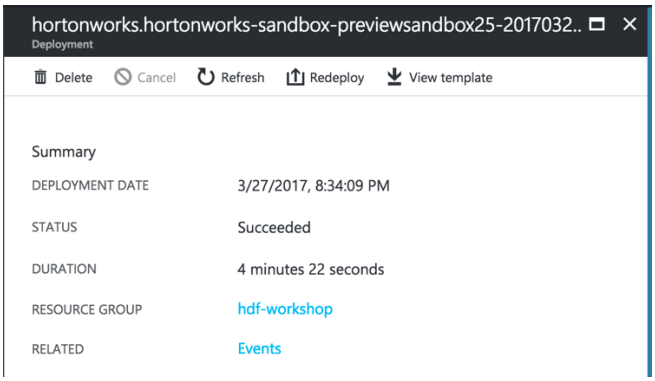
e. Buy

After reviewing, click **Purchase**.

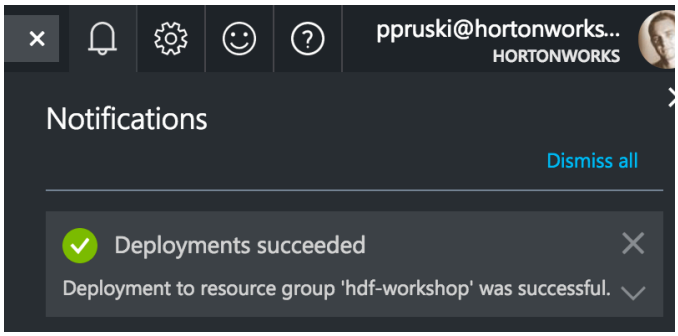
5. You can view the progress of the deployment by clicking the notification icon on the top right of the portal and selecting the deployment.



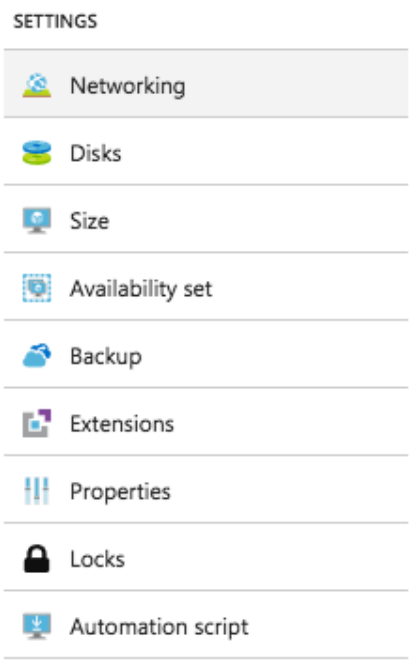
6. The deployment can take a few minutes. Once it is complete, you should see a screen similar to the following:



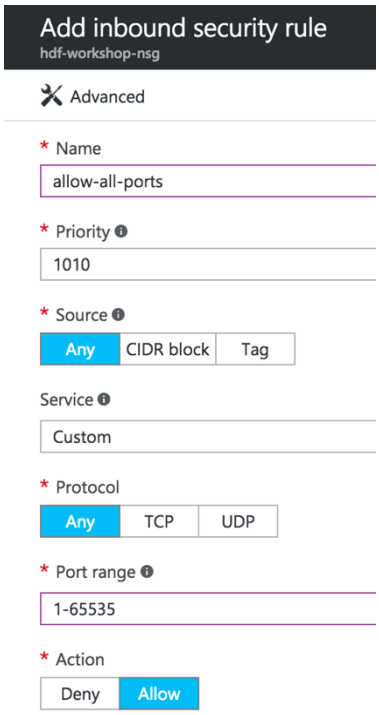
7. Navigate to the Sandbox virtual machine by clicking **Deployments succeeded** from the notifications once again.



8. Click on **Networking** from the Virtual machine navigation pane under SETTINGS and select the network interface for the Sandbox.



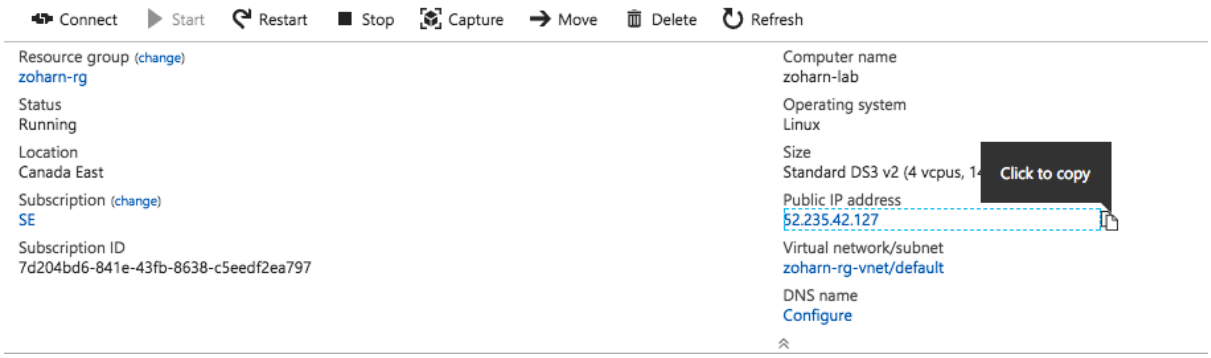
9. Click on the **Add Inbound port rule** button on the right side of the screen and **Add** an inbound rule to allow all ports as the following figure shows and click **OK**.



10. Using a terminal (Mac/Linux) or Putty (Windows), SSH into the VM with the public IP address of the Sandbox and the username/password you set during Sandbox creation.

```
ssh <user_name>@<public_ip_addr>
```

You can find the public IP address of the virtual machine in the Virtual Machine Overview section. You can copy the IP using the copy icon which appears if you hover over the IP address.



11. Change user to sudo and enter your password once again.

```
sudo su -
```

12. Now as the root user SSH into the docker image with the following command.

```
ssh root@sandbox -p 2222
```

13. You will be prompted for the password and to set a new password. The password for root is **hadoop**. You'll be required to enter this password twice before selecting a new password. (HINT: for simplicity it is recommended to change the password to the same as specified above in step 4a)

14. Set the new passwords for Ambari as well. (HINT: for simplicity it is recommended to change the password to the same as specified above in step 4a)

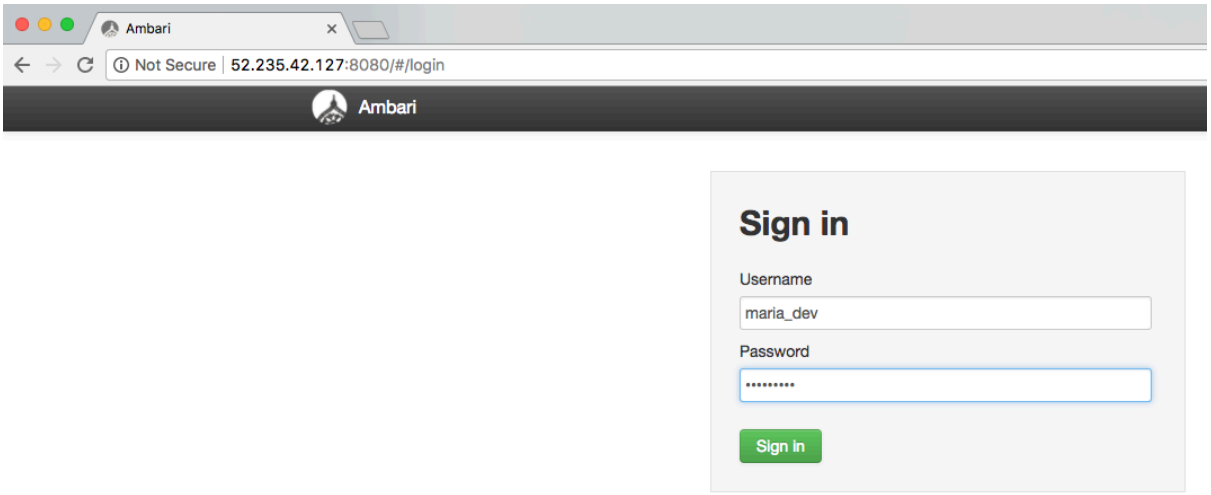
```
ambari-admin-password-reset
```

    This will cause ambari-server to restart.

15. Open a web browser and enter the following URL to access your Ambari console (The public IP address is the same as previous step 10):

```
http://<public_ip_addr>:8080
```

16. Login to Ambari. Use maria_dev/maria_dev as the login/password, and click on Sign In.



Hive Lab – Loading and Querying Data with Hadoop

1. INTRODUCTION

The HDP Sandbox includes the core Hadoop components, as well as all the tools needed for data ingestion and processing. You are able to access and analyze data in the sandbox using any number of Business Intelligence (BI) applications.

In this lab, we will go over how to load and query data for a fictitious web retail store in what has become an established use case for Hadoop: deriving insights from large data sources such as web logs. By combining web logs with more traditional customer data, we can better understand customers and understand how to optimize future promotions and advertising.

We will describe how to ingest data into HDFS, to create tables and perform queries on those tables with Hive to analyze the web logs from that data. By the end of the tutorial, we will have a better understanding of how to perform web log analysis on clickstream data, so we can better understand the habits of our customers.

2. DOWNLOAD SAMPLE DATA

Download the sample data for this tutorial:
https://raw.githubusercontent.com/hortonworks/data-tutorials/master/tutorials/hdp/loading-and-querying-data-with-hadoop/assets/retail-store-logs-sample-data.zip

Extract the archive anywhere you'd like – we will upload the contents into our sandbox in the following step. You should see the following files after the archive is extracted:

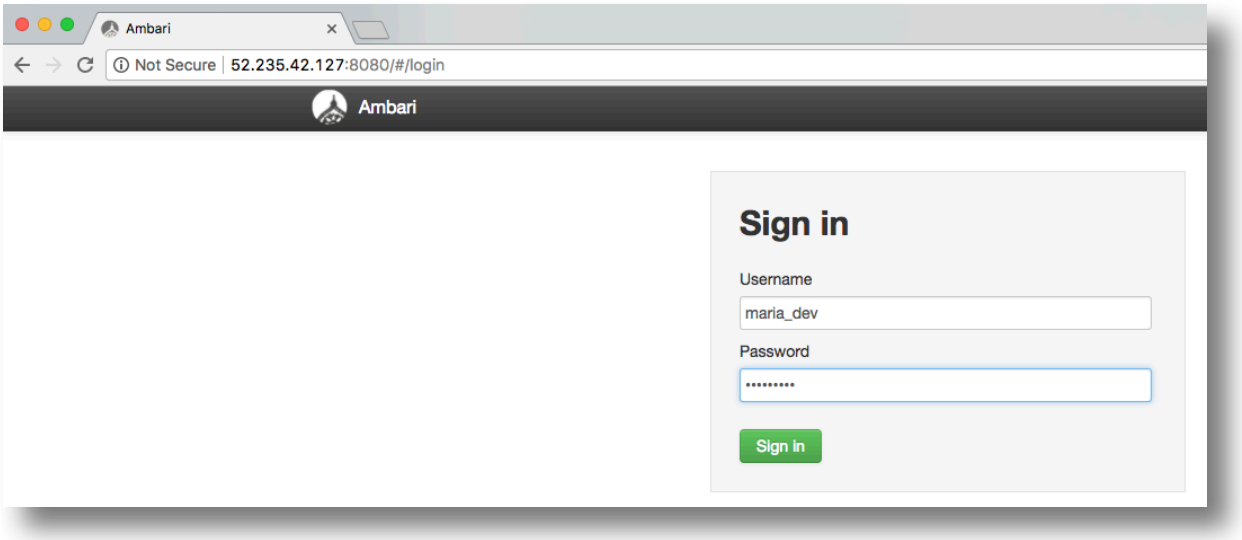
```
omniture-logs.tsv
products.tsv
users.tsv
```

3. UPLOAD DATA INTO HDFS

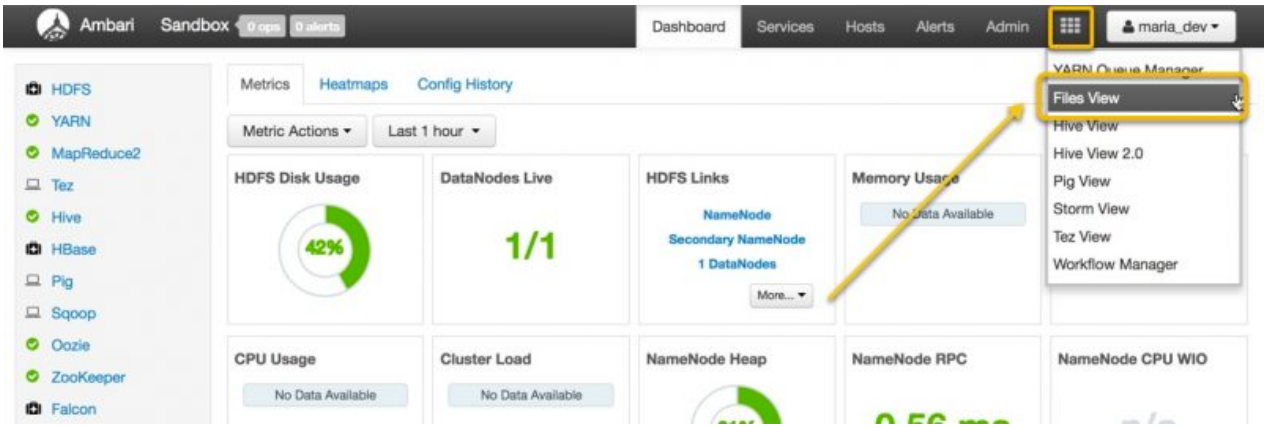
Open a web browser and enter the following URL to access your Ambari console (The public IP address is the same as previous step 10):

```
http://<public_ip_addr>:8080
```

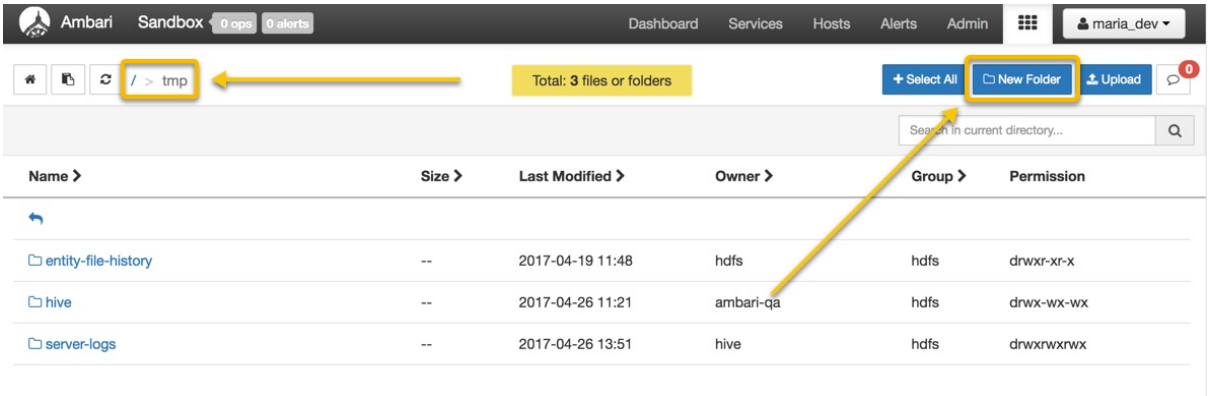
Login to Ambari. Use maria_dev/maria_dev as the login/password, and click on Sign In.



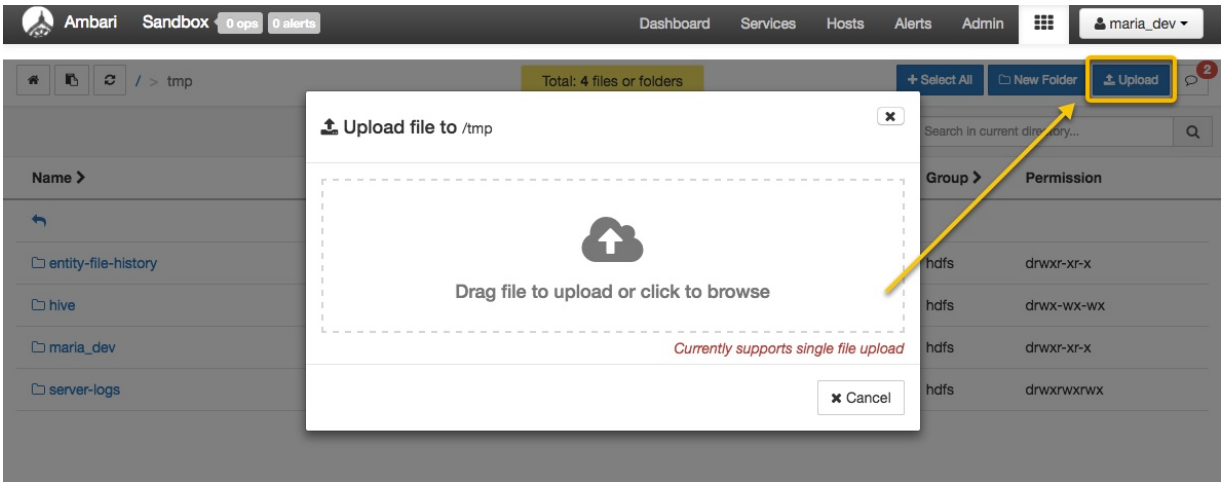
Select the HDFS `Files View` from the Views menu on the navigation bar. The Files View allows you to view the Hortonworks Data Platform (HDP) file store. The HDFS file system is separate from the local file system.



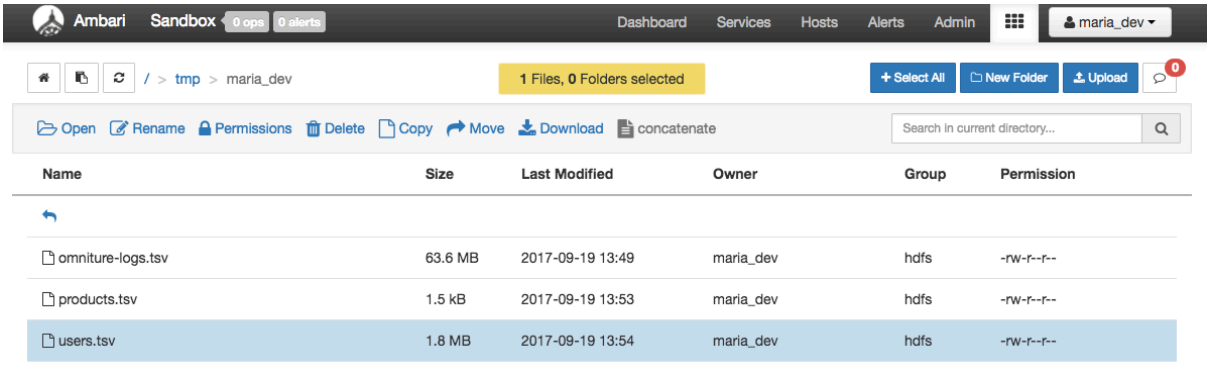
Navigate to `/tmp` by clicking on the "**tmp**" folder. Select "**New Folder**" and name it `maria_dev`.



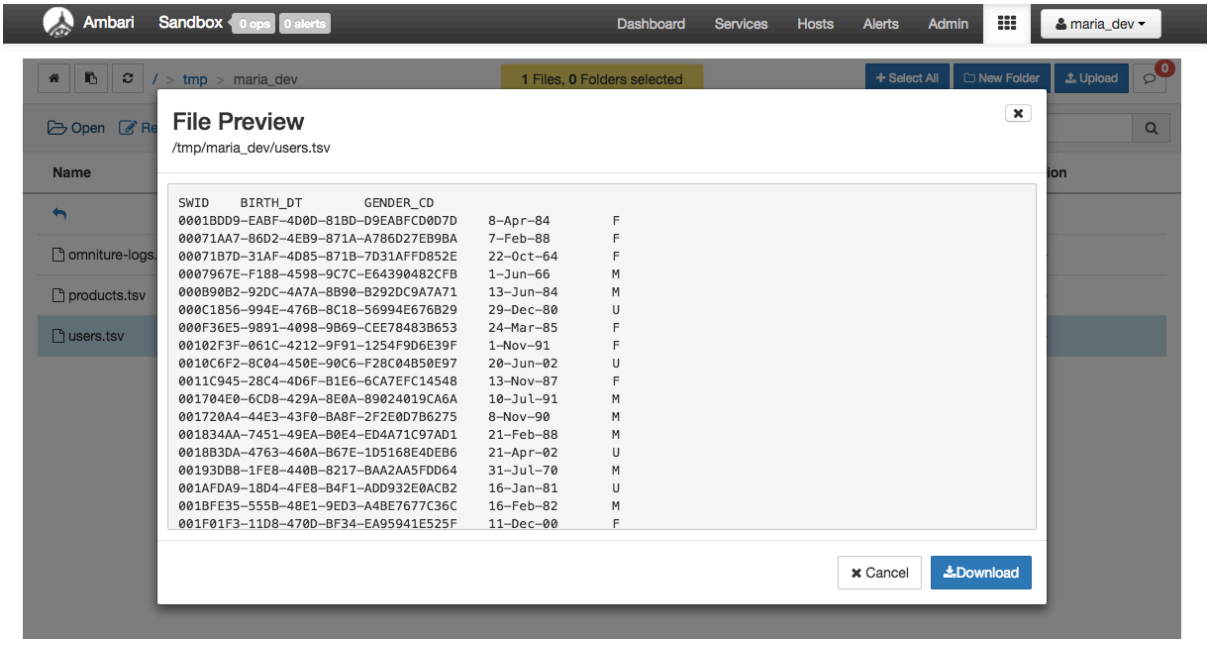
Navigate to `/tmp/maria_dev` by clicking on the "**maria_dev**" folder. Then click on the "**Upload**" button at the top right of the screen. Upload all three files you extracted in the previous step: `omniture-logs.tsv`, `products.tsv` and `users.tsv`.



Explore the contents of the files uploaded on HDFS. Click on the file “users.tsv”, a menu will appear on top, then click on **“Open”**.



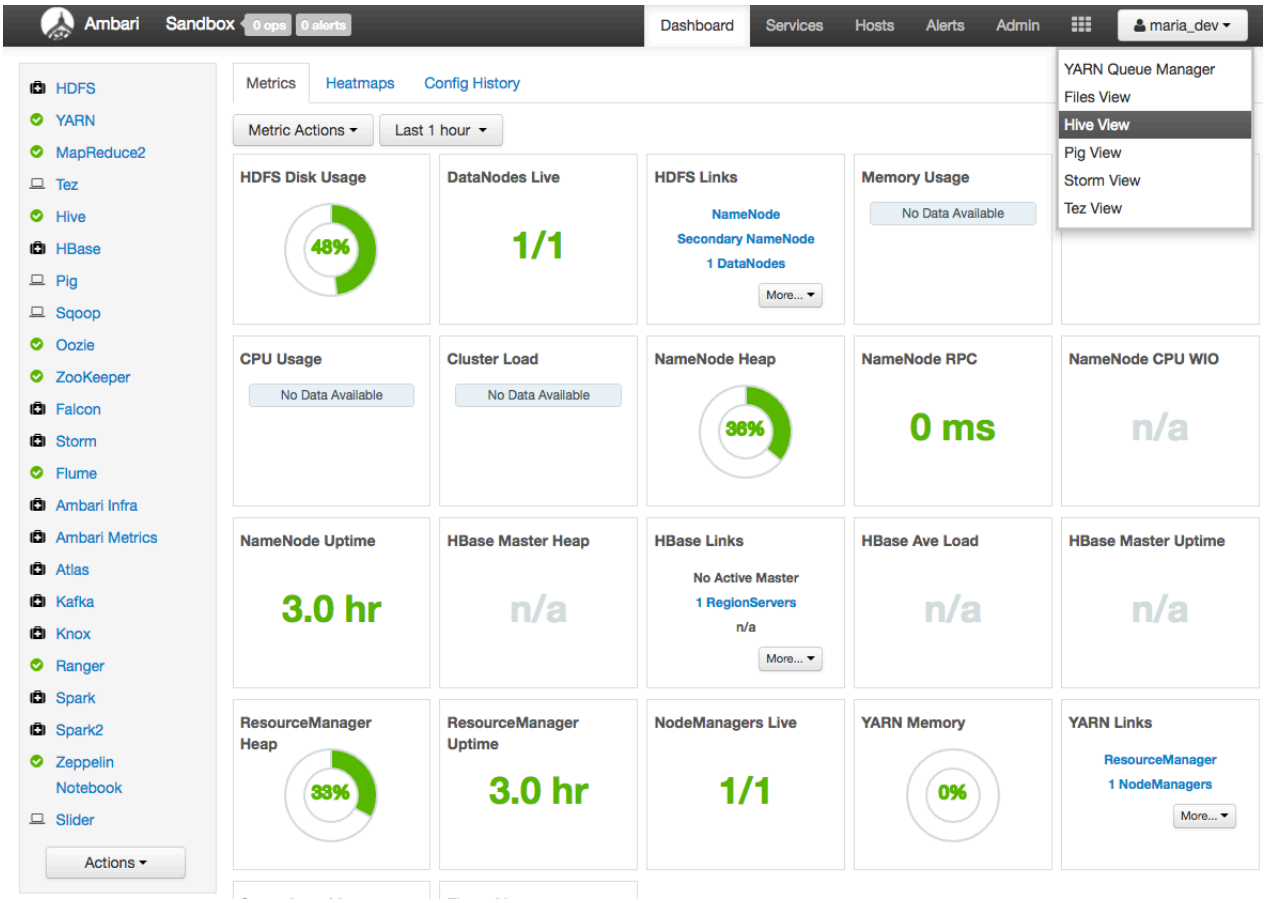
This will open a **“File Preview”** window:



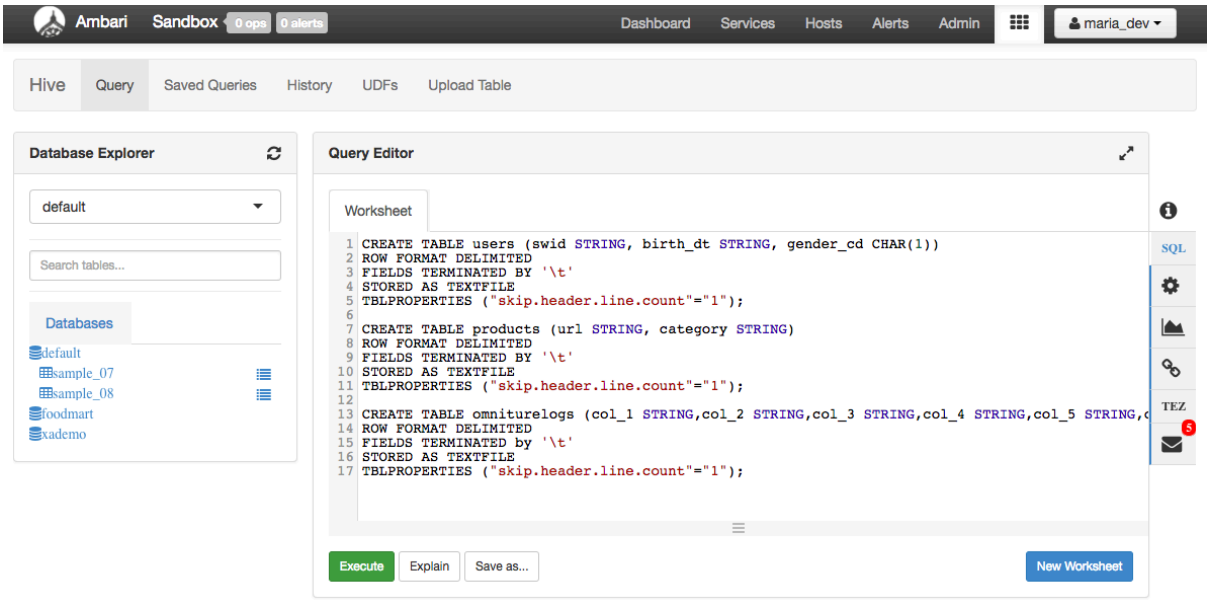
Please repeat the previous steps for the remaining two files. As you can see, these are flat files, tab delimited, containing a data header. In the next section, we are going to load these files on Hive to access them with SQL. Close the window by clicking on **Cancel**.

4. CREATE HIVE TABLES

Let's create some Hive tables for our data. Open the `Hive View` from the Views menu on the navigation bar.



Once there, create three tables by copy/pasting each query (execute the queries one after the other) below into the query editor and clicking "**Execute**".



Create the tables users, products and omniturelogs:

```
CREATE TABLE users (swid STRING, birth_dt STRING, gender_cd CHAR(1))
ROW FORMAT DELIMITED
FIELDS TERMINATED BY '\t'
STORED AS TEXTFILE
TBLPROPERTIES ("skip.header.line.count"="1");
```

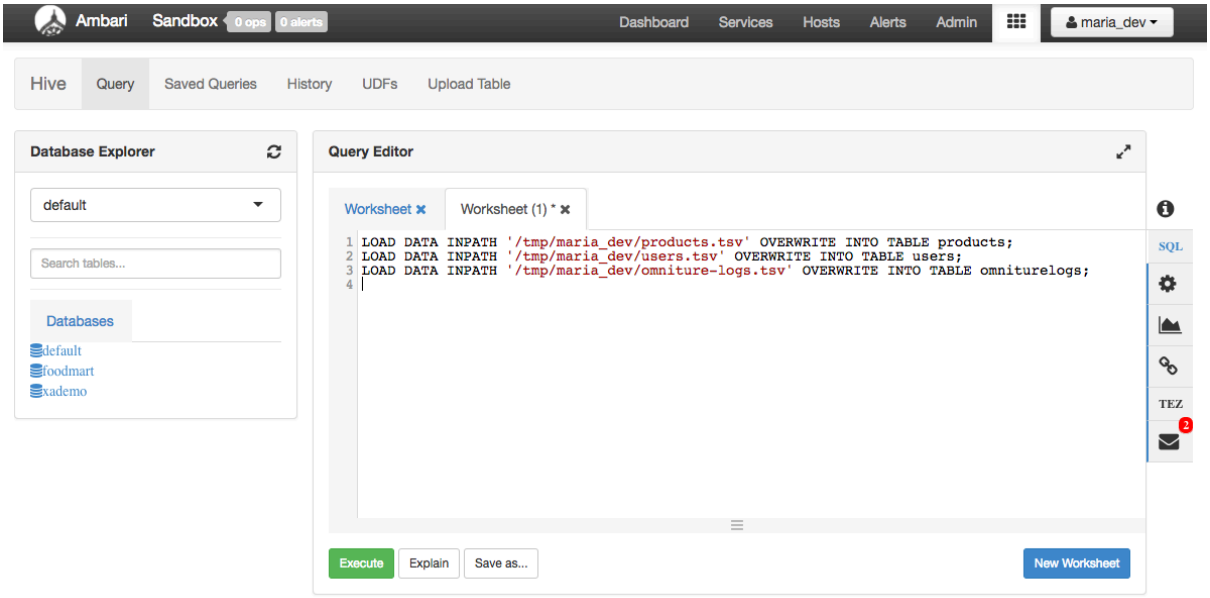
```
CREATE TABLE products (url STRING, category STRING)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY '\t'
STORED AS TEXTFILE
TBLPROPERTIES ("skip.header.line.count"="1");
```

```
CREATE TABLE omniturelogs (col_1 STRING,col_2 STRING,col_3 STRING,col_4
STRING,col_5 STRING,col_6 STRING,col_7 STRING,col_8 STRING,col_9 STRING,col_10
STRING,col_11 STRING,col_12 STRING,col_13 STRING,col_14 STRING,col_15
STRING,col_16 STRING,col_17 STRING,col_18 STRING,col_19 STRING,col_20
STRING,col_21 STRING,col_22 STRING,col_23 STRING,col_24 STRING,col_25
STRING,col_26 STRING,col_27 STRING,col_28 STRING,col_29 STRING,col_30
STRING,col_31 STRING,col_32 STRING,col_33 STRING,col_34 STRING,col_35
STRING,col_36 STRING,col_37 STRING,col_38 STRING,col_39 STRING,col_40
STRING,col_41 STRING,col_42 STRING,col_43 STRING,col_44 STRING,col_45
STRING,col_46 STRING,col_47 STRING,col_48 STRING,col_49 STRING,col_50
STRING,col_51 STRING,col_52 STRING,col_53 STRING)
ROW FORMAT DELIMITED
FIELDS TERMINATED by '\t'
STORED AS TEXTFILE
TBLPROPERTIES ("skip.header.line.count"="1");
```

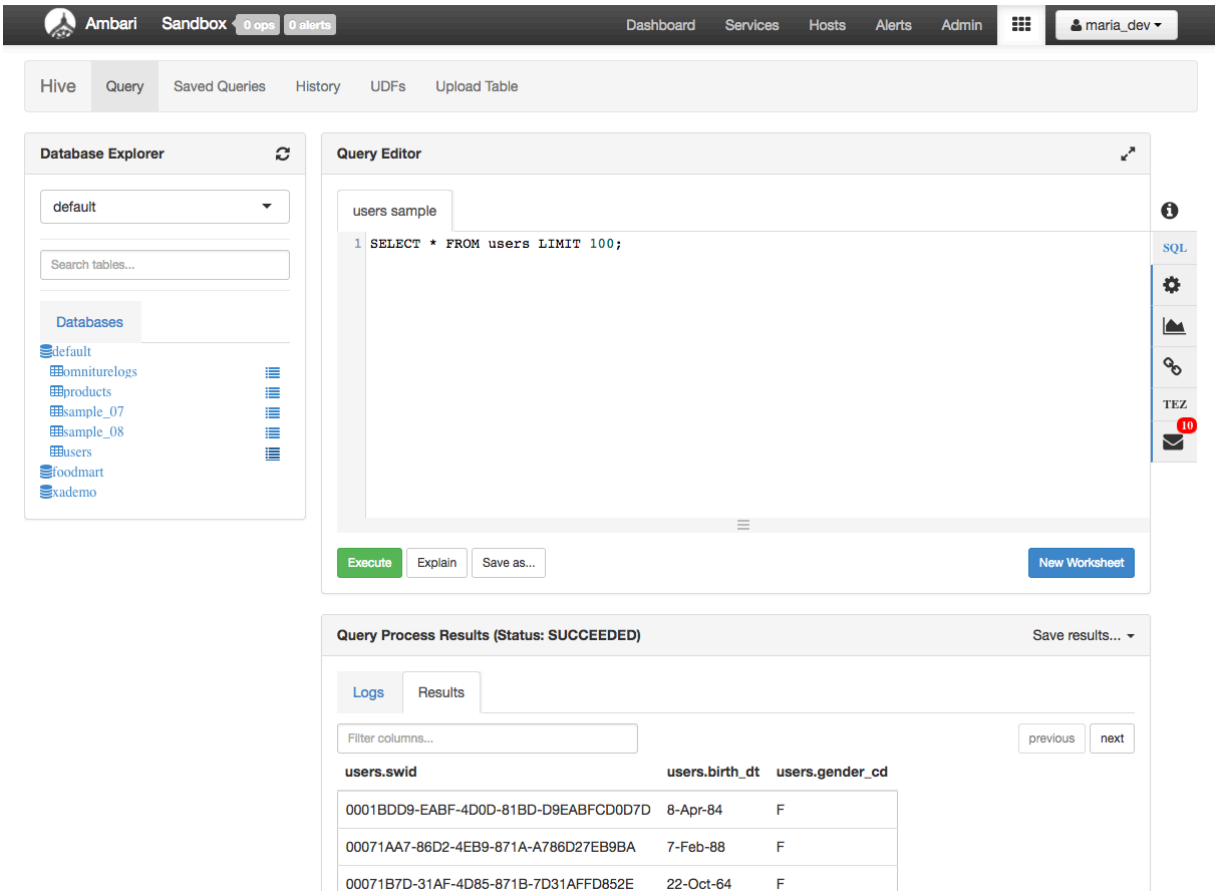
5. LOAD DATA INTO TABLES

Let's run a simple query to take the data we stored in HDFS and populate our new Hive tables. To do so, execute the following query:

```
LOAD DATA INPATH '/tmp/maria_dev/products.tsv' OVERWRITE INTO TABLE products;
LOAD DATA INPATH '/tmp/maria_dev/users.tsv' OVERWRITE INTO TABLE users;
LOAD DATA INPATH '/tmp/maria_dev/omniture-logs.tsv' OVERWRITE INTO TABLE
omniturelogs;
```



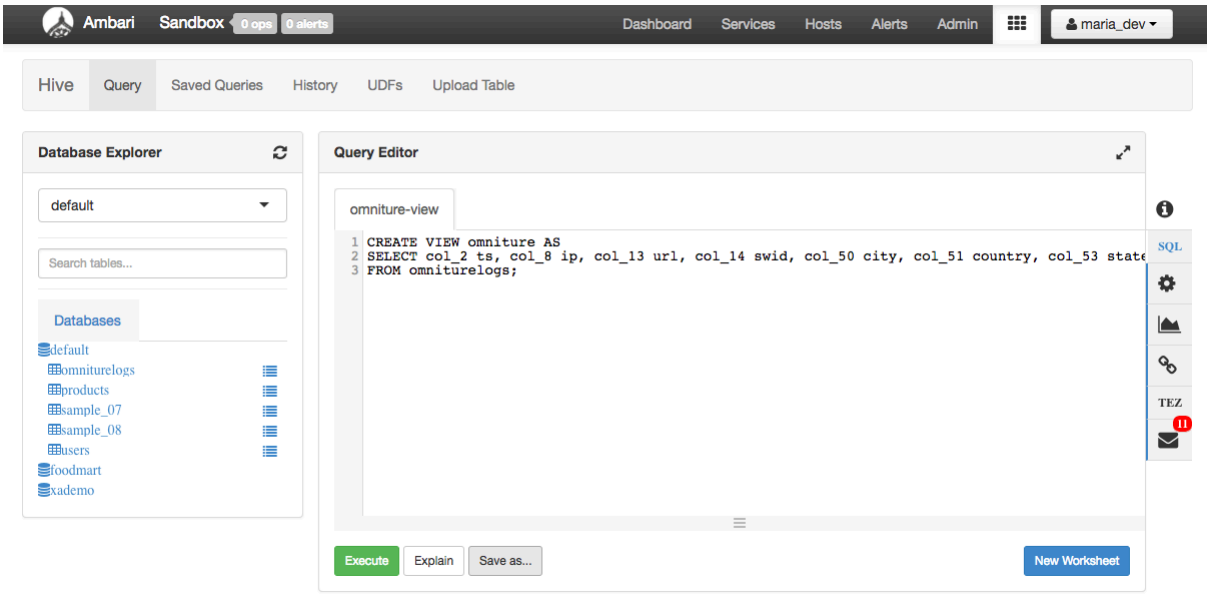
You can verify that the data was loaded properly by browsing information about the different Hive tables in the "**Database Explorer**" tab. Click on the default database. The newly created tables will appear in addition to pre-existing tables on the sandbox. Click on the right-hand side icon to get a sample of 100 rows of the table **"Users"** for example.



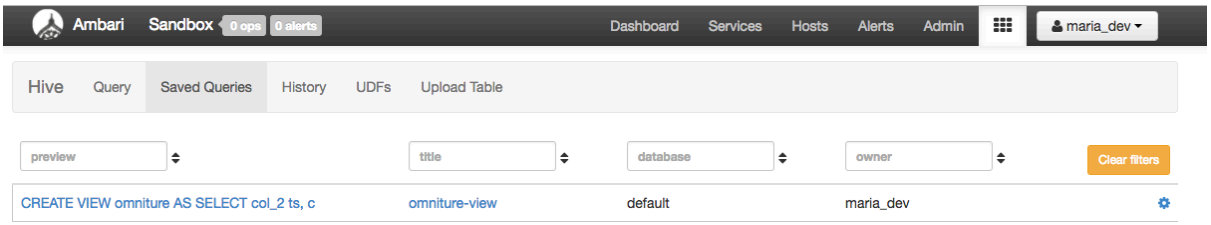
6. SAVE AND EXECUTE A QUERY

Suppose we want to write a query, but not necessarily execute it immediately or perhaps we want to save it for future multiple uses. In the "**QUERY**" tab, copy/paste the following and click "**Save As**". Save the query as omniture-view.

```
CREATE VIEW omniture AS
SELECT col_2 ts, col_8 ip, col_13 url, col_14 swid, col_50 city, col_51 country,
col_53 state
FROM omniturelogs
```

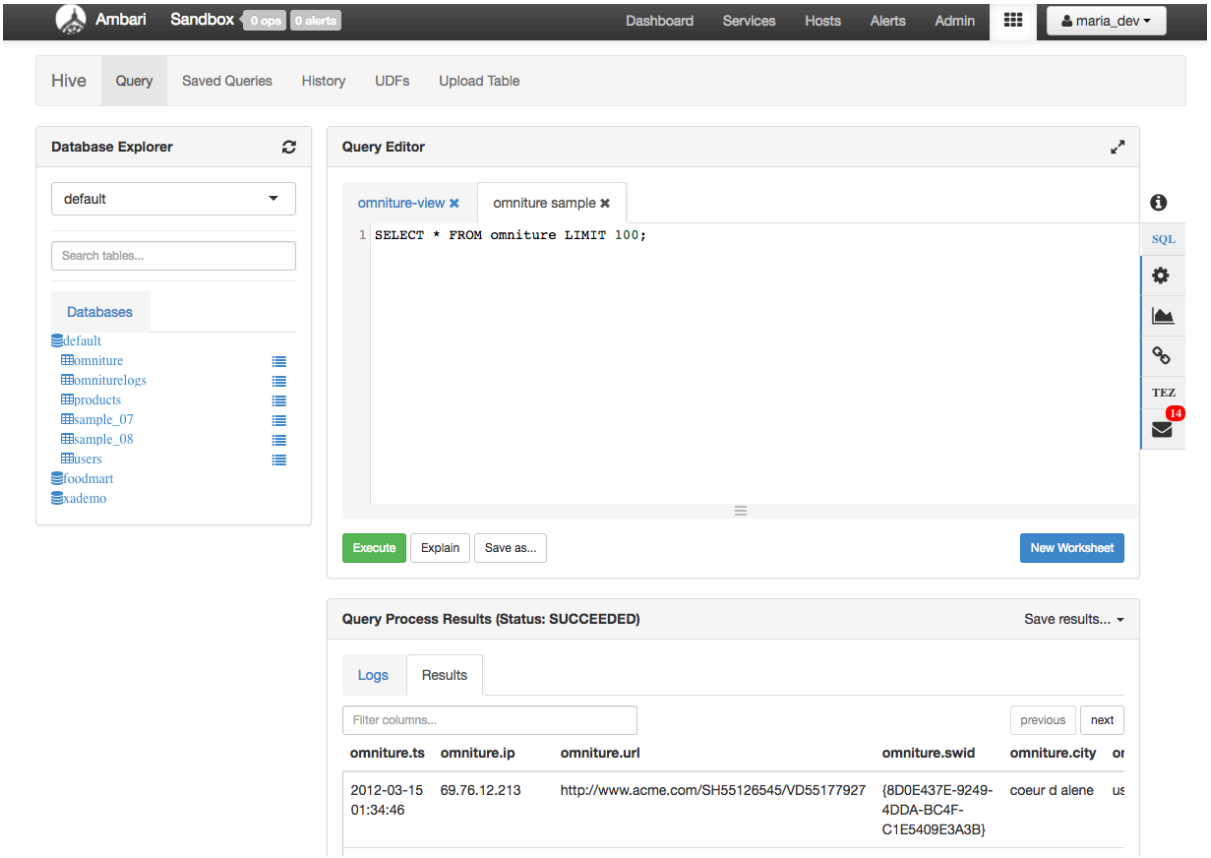


To view your saved queries, navigate to the "**SAVED QUERIES**" tab. For now, let's open our saved query by clicking either on the first or second column:



The "**Query Editor**" should automatically open up, with your saved query preloaded. Click "**Execute**" to run this query, which will create a Hive view named "omniture", a refined subset of data with only a handful of fields.

If you refresh the Database Explorer using the icon, you will see the “omniture” view added in the list of objects. You can click on the icon next to it to see a data sample.

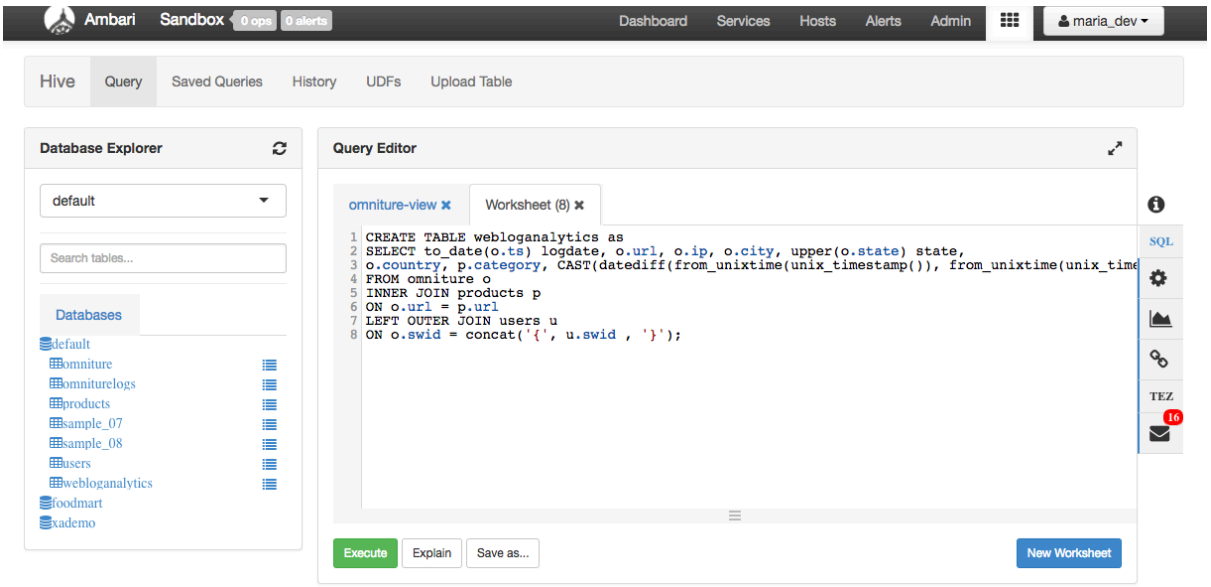


7. JOIN DATA FROM MULTIPLE TABLES

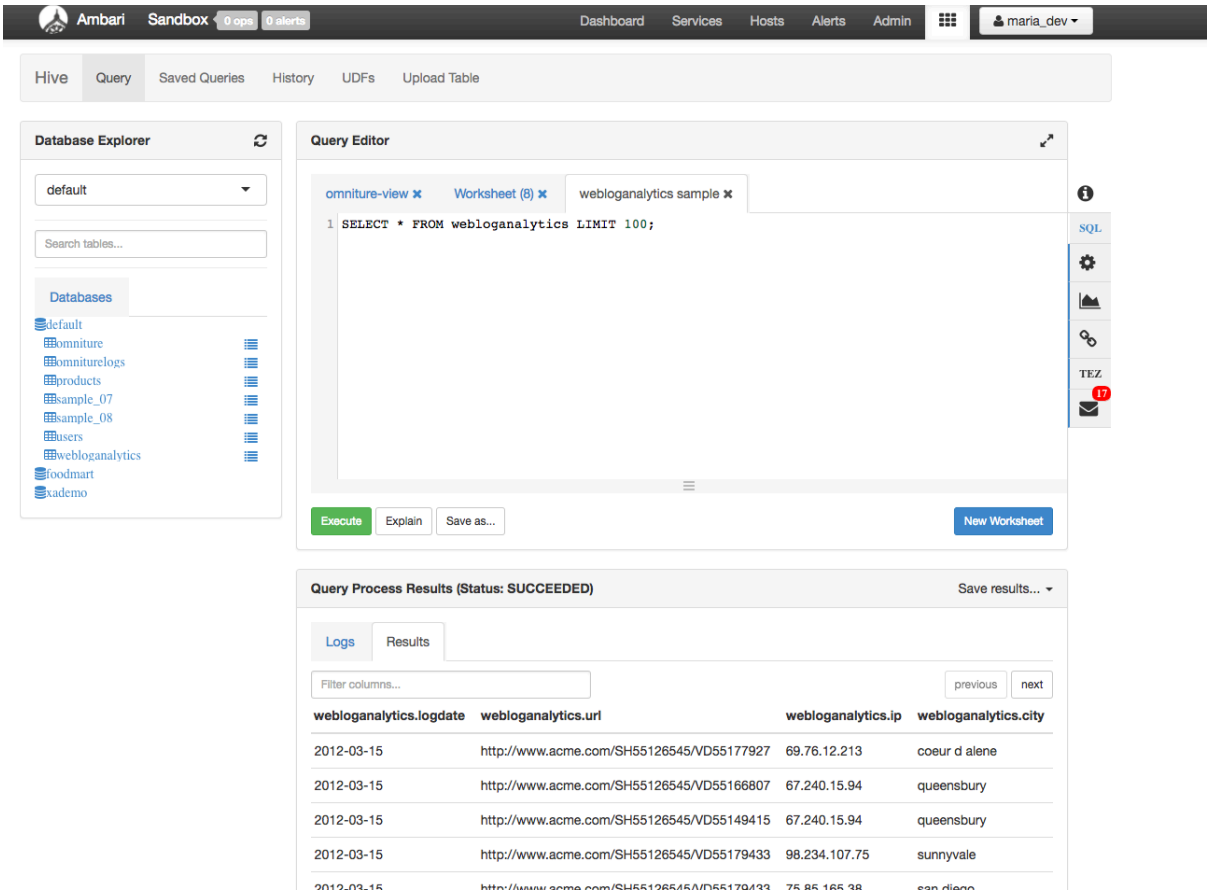
Let's play with our data further, taking specific fields from different tables and creating a custom table from them.

```
CREATE TABLE webloganalytics as
SELECT to_date(o.ts) logdate, o.url, o.ip, o.city, upper(o.state) state,
o.country, p.category, CAST(datediff(from_unixtime(unix_timestamp()),
from_unixtime(unix_timestamp(u.birth_dt, 'dd-MMM-yy'))) / 365 AS INT) age,
u.gender_cd
FROM omniture o
INNER JOIN products p
ON o.url = p.url
LEFT OUTER JOIN users u
ON o.swid = concat('{', u.swid , '}');
```

You can click on the **"New Worksheet"** button and paste the above query there, then click on **"Execute"**:



After refreshing the "Database Explorer", the new "webloganalytics" object will appear there. You can click on the icon next to it to get a data sample and browse the **"Results"** tab at the bottom.



8. SUMMARY

Excellent! We've learned how to upload data into HDFS, create tables and load data into them, and run queries for data refinement and enrichment by using Ambari's convenient and robust Hive View.

Spark Lab – Introduction to Zeppelin

1. INTRODUCTION

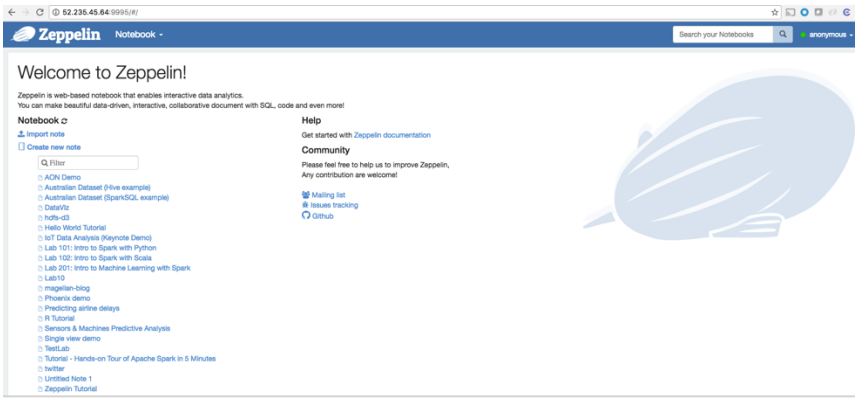
The HDP Sandbox includes the core Hadoop components, as well as all the tools needed for data ingestion and processing. You are able to access and analyze data in the sandbox using any number of Business Intelligence (BI) applications.

In this tutorial, you will access and explore Apache Zeppelin core capabilities.
You will also interact with Hadoop cluster using various language interpreters embedded in zeppelin notebook. Finally, you reuse existing notebook to run an end-to-end prebuilt data pipeline.

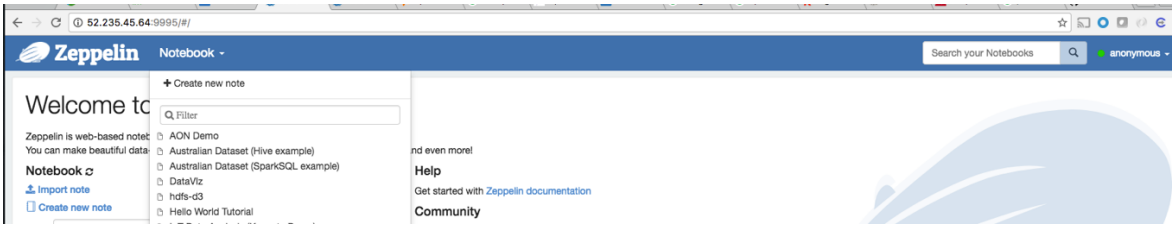
Caution! : Please don't copy and paste in the notebooks, it will generate some errors in your syntax if you do so.

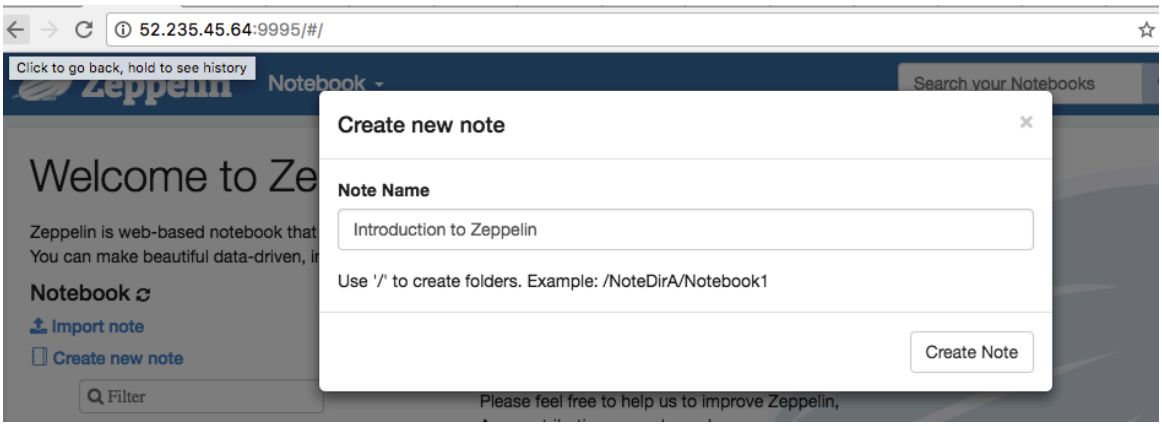
2. ACCESS AND BROWSE ZEPPELIN

Open a web browser and enter the following URL : http://<sandbox ip address>:9995/



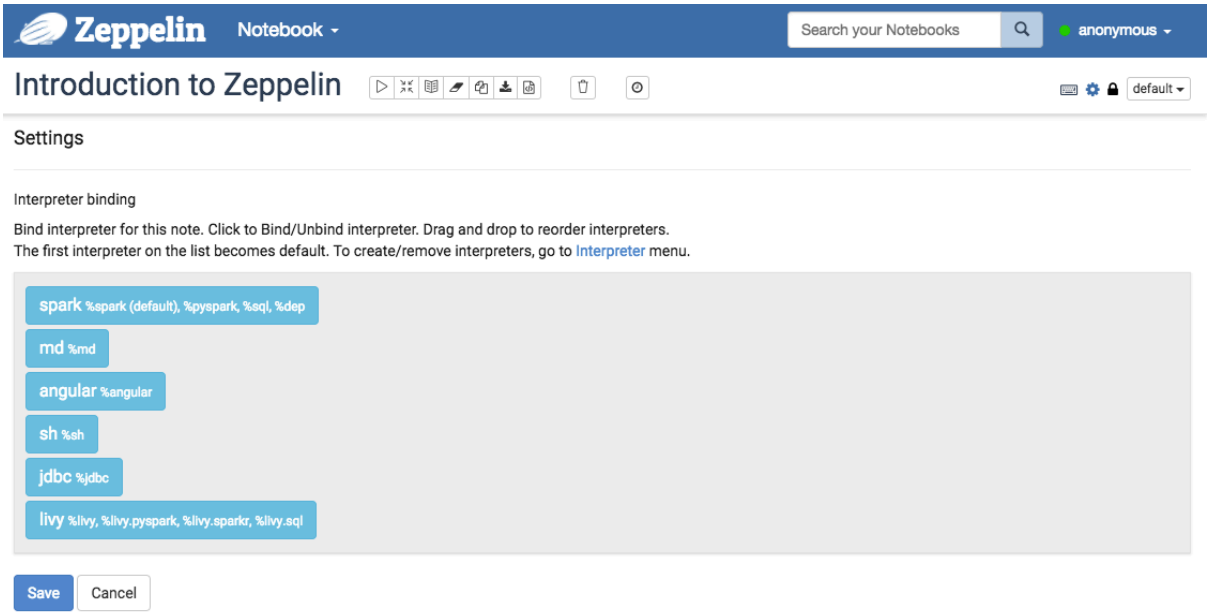
Create a new note named Introduction to Zeppelin

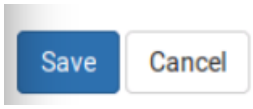




3. EXPLORE INTERPRETER BINDINGS

At the top right click on the gear icon to see interpreter binding. Your administrator has enabled an interpreter called “spark” which is configured for the HDP cluster you are using. Click on Save button to return to your notebook page.





4. WORKING WITH ZEPPELIN INTERPRETERS

Find the values for Spark version and the Spark home directory. When you type the commands, run them either by pressing the Shift + Enter keys, or by clicking on the

Play icon to the right of the word Ready.

```
%spark
sc.version
sc.getConf.get(“spark.home”)
```

While processing, Zeppelin will display a status of RUNNING. It will also display a Pause icon should it become necessary.

The output may vary slightly from the screenshot below, but should look something like this when processing is completed:

Zeppelin can be instructed to use multiple languages in an interactive fashion within the same notebook. Simply specify the desired language prior to the command.

Run the following commands to demonstrate this flexibility using **Shell, Python, Scala, Markdown, and Spark SQL**. Execute each command by clicking on the Play icon or pressing **Shift + Enter** when you are finished typing.

Shell:

```
%sh
echo “Introduction to Zeppelin”
```

```
%sh echo "Introduction to Zeppelin"

Introduction to Zeppelin
```

Python:

```
%pyspark
print “Introduction to Zeppelin”
```

```
%pyspark
print "Introduction to Zeppelin"

Introduction to Zeppelin
```

Scala (default, so no need to specify prior to running command):

```
%spark
val s = “Introduction to Zeppelin”
```

```
val s = "Introduction to Zeppelin"

s: String = Introduction to Zeppelin
```

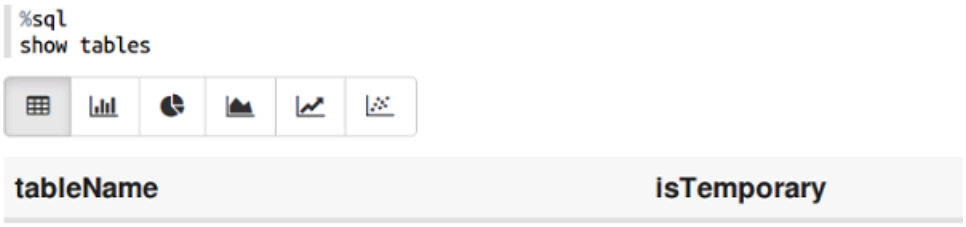
Markdown:

```
%md Introduction to Zeppelin
```

```
%md Introduction to Zeppelin

Introduction to Zeppelin
```

Spark SQL:

```
%sql
show tables
```

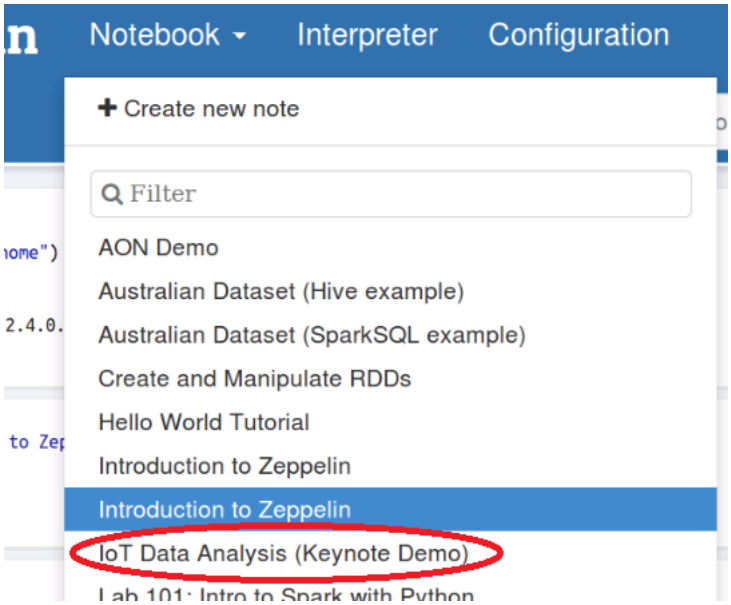


5. USE A PREBUILT NOTEBOOK TO EXPLORE ZEPPELIN CAPABILITIES

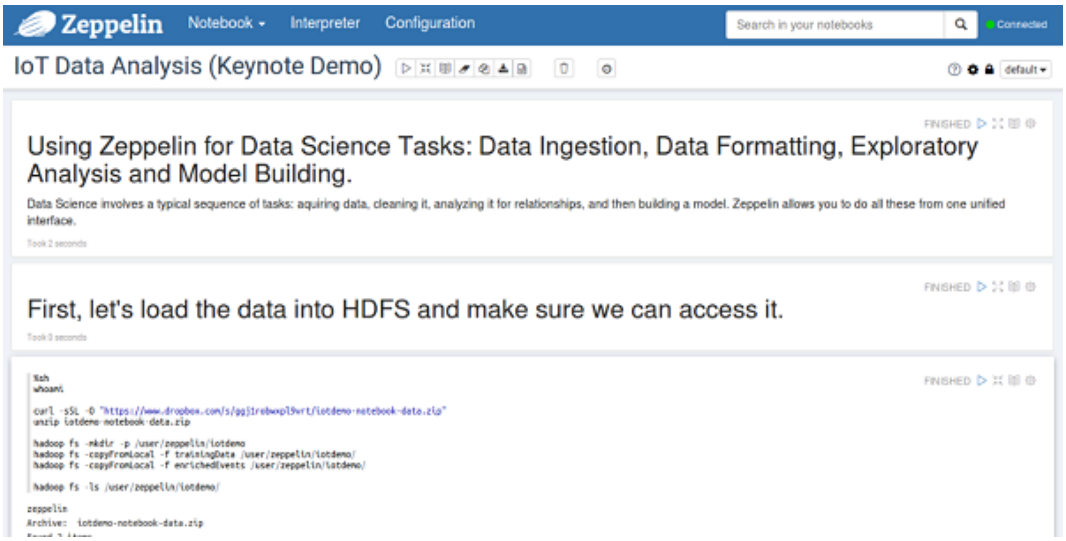
Zeppelin has four major functions:

- data ingestion,
- data discovery,
- data analytics,
- data visualization.

One of the easiest ways to explore these functions is with a preconfigured notebook, many of which are available by default. Click on Notebook at the top of the browser window and find and select the notebook labeled IoT Data Analysis (Keynote Demo) in the resulting drop-down menu.

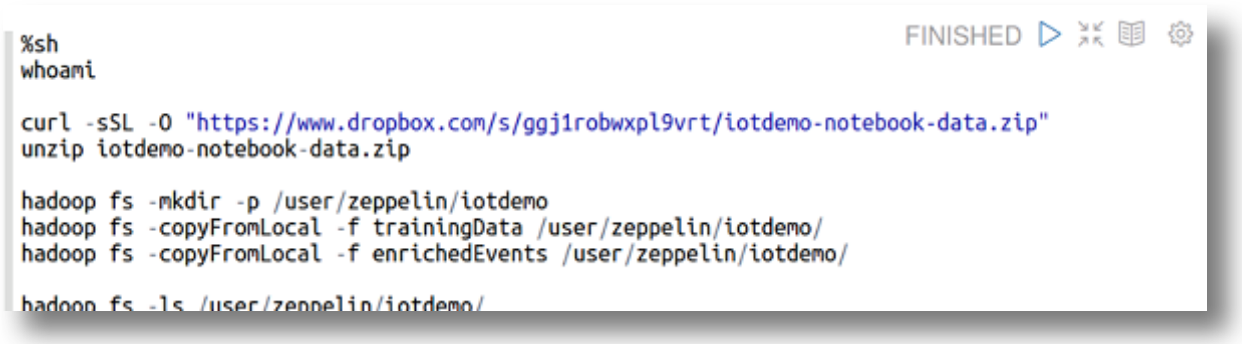


For the purposes of this lab, all necessary code has already been entered for you in the saved notebook. All you have to do is scroll to the appropriate section and click the Play icon or press Shift + Enter.



The first major block of code ingests data from an online source into HDFS and then displays those files using the shell scripting interpreter. Find and run that code.

Note that the label to the left of the Play icon says FINISHED, but this will not prohibit you from running the code again on this machine.
Also note: this notebook uses a deprecated command, hadoop fs, rather than the more updated hdfs dfs command we used in the previous lab. This should not affect the functionality of the demo.



When the code has finished, the output at the bottom should look like this:

The next section of the notebook once again uses the shell scripting interpreter to view some of the raw data in one of the downloaded files. Scroll down and run this code, then view its output.

```
%sh
hadoop fs -cat /user/zeppelin/iotdemo/enrichedEvents | tail -n 10
```

```
Overspeed,"Y","hours",45,2773,-90.07,35.68,0,1,1
Lane Departure,"Y","hours",45,2773,-90.04,35.19,1,1,0
Normal,"Y","hours",45,2773,-90.68,35.12,1,0,0
Normal,"Y","hours",45,2773,-91.14,34.96,0,0,0
Normal,"Y","hours",45,2773,-91.93,34.81,0,0,0
Normal,"Y","hours",45,2773,-92.31,34.78,0,1,0
Normal,"Y","hours",45,2773,-92.09,34.8,0,0,0
Normal,"Y","hours",45,2773,-91.93,34.81,0,0,0
```

The next section of the notebook performs actions necessary to import and use this data with Spark SQL. You may note that the status to the left of the Play icon is shown as ERROR. This is due to the fact that the file being manipulated did not exist at the time the notebook was opened on this system. Run this code and view the output.

```
val sqlContext = new org.apache.spark.sql.SQLContext(sc)

val eventsFile = sc.textFile("hdfs:///user/zeppelin/iotdemo/enrichedEvents")

case class Event(eventType: String,
                 isCertified: String,
                 paymentScheme: String,
                 hoursDriven: Int,
                 milesDriven: Int,
                 lat: Float,
                 long: Float,
                 isFoggy: Int,
                 isRainy: Int
```

The output should look like this:

```
eventsRDD.toDF().registerTempTable("enrichedEvents")

sqlContext: org.apache.spark.sql.SQLContext = org.apache.spark.sql.SQLContext@312d2a12
eventsFile: org.apache.spark.rdd.RDD[String] = MapPartitionsRDD[3] at textFile at <console>:29
defined class Event
eventsRDD: org.apache.spark.rdd.RDD[Event] = MapPartitionsRDD[5] at map at <console>:35
res4: Long = 1359
```

The next block of code utilizes Spark SQL to view this data. Run this code and examine the output.

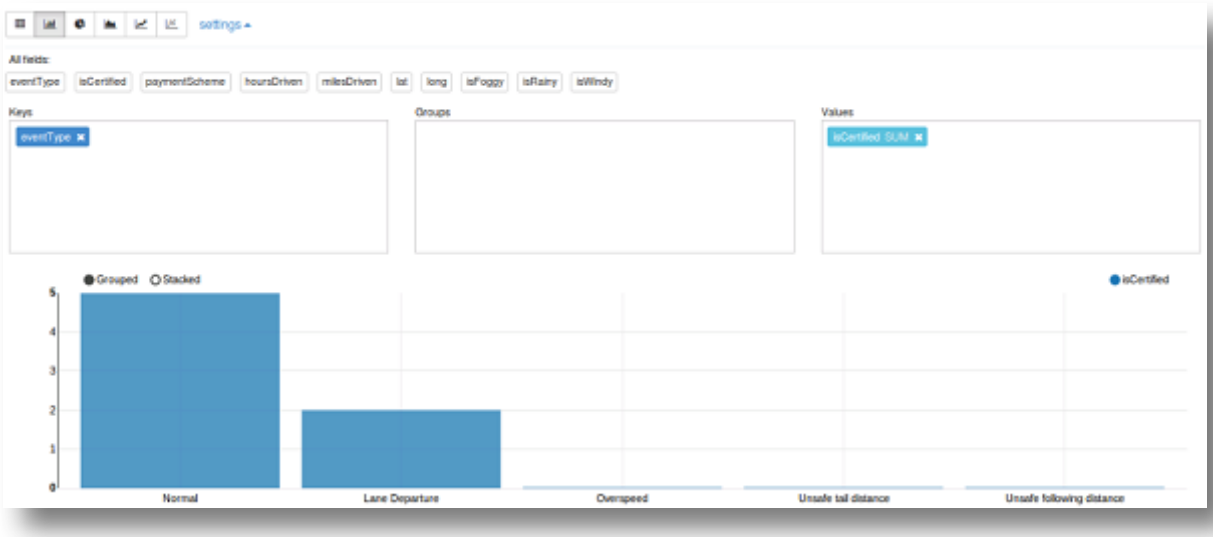
```
%sql
select * from enrichedEvents order by hoursDriven desc limit 10
```

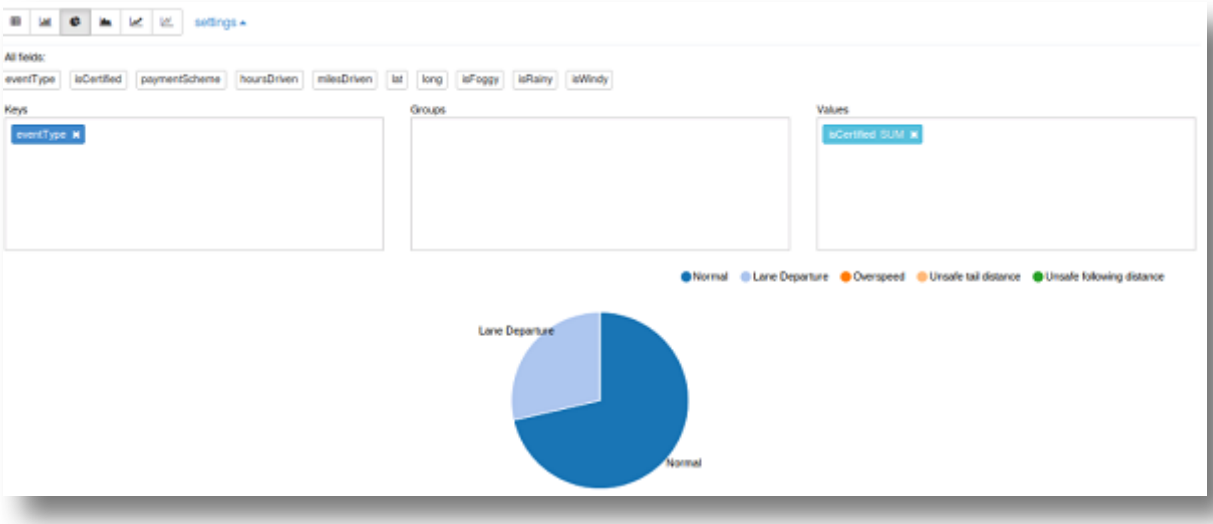
FINISHED

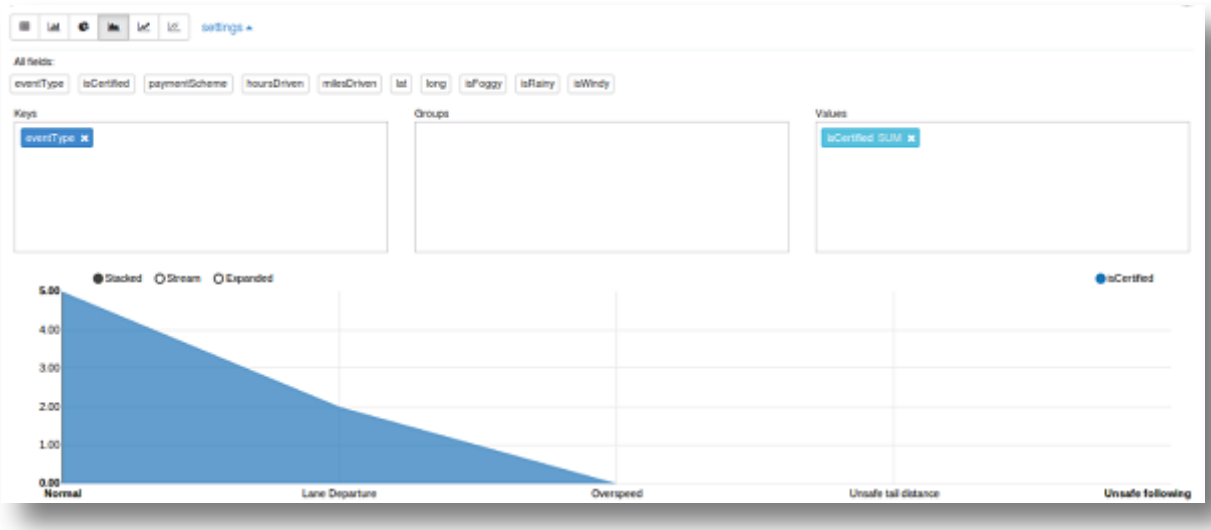
eventType	isCertified	paymentScheme	hoursDriven	milesDriven	lat	long	isFoggy	isRainy
Normal	N	miles	90	4,300	-90.29	40.96	0	0
Lane Departure	N	miles	90	4,300	-88.42	41.11	1	1

Note that at the top of the results there are six buttons that allow you to display the results using six different visualizations. Click on each one to view the differences between them.

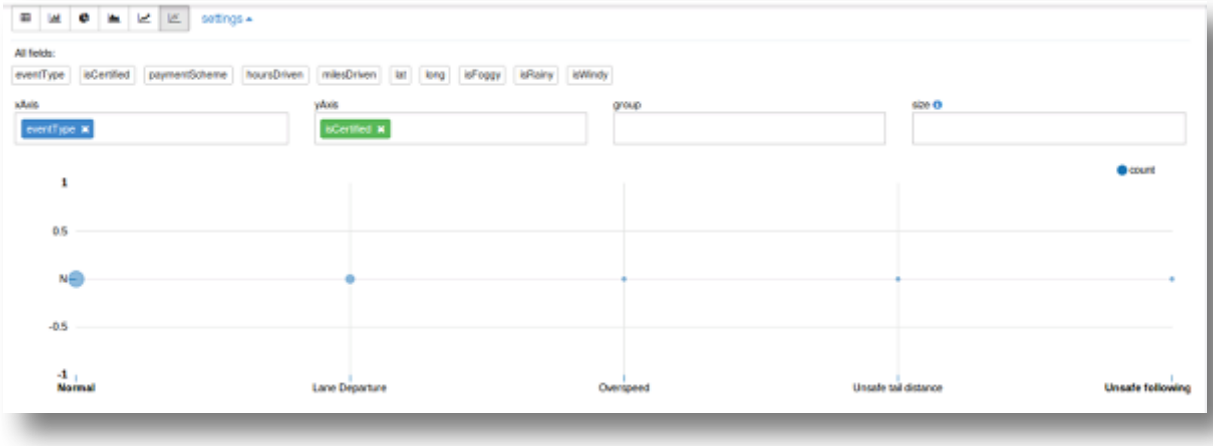
eventType	isCertified	paymentScheme	hoursDriven	milesDriven	lat	long	isFoggy	isRainy	isWindy
Normal	N	miles	90	4,300	-90.29	40.96	0	0	1
Lane Departure	N	miles	90	4,300	-88.42	41.11	1	1	1
Normal	N	miles	90	4,300	-89.91	40.86	0	0	0
Overspeed	N	miles	90	4,300	-93.04	41.71	1	0	0
Unsafe tail distance	N	miles	90	4,300	-87.67	41.87	1	1	1
Normal	N	miles	90	4,300	-89.52	40.7	0	0	0
Normal	N	miles	90	4,300	-91.05	41.72	0	0	1
Normal	N	miles	90	4,300	-91.47	41.74	0	0	0
Lane Departure	N	miles	90	4,300	-91.59	41.7	1	0	0


settings
All fields:
eventType
isCertified
paymentScheme
hoursDriven
milesDriven
lat
long
isFoggy
isRainy
isWindy
Keys
eventType
Groups
Values
isCertified SUM
Grouped
Stacked
isCertified
5
4
3
2
1
0
Normal
Lane Departure
Overspeed
Unsafe tail distance
Unsafe following distance

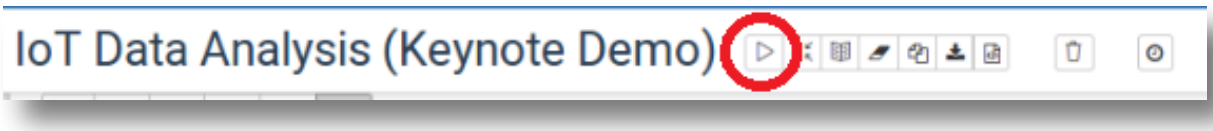

settings
All fields:
eventType
isCertified
paymentScheme
hoursDriven
milesDriven
lat
long
isFoggy
isRainy
isWindy
Keys
eventType
Groups
Values
isCertified SUM
Normal
Lane Departure
Overspeed
Unsafe tail distance
Unsafe following distance
Lane Departure
Normal


settings
All fields:
eventType
isCertified
paymentScheme
hoursDriven
milesDriven
lat
long
isFoggy
isRainy
isWindy
Keys
eventType
Groups
Values
isCertified SUM
Stacked
Stream
Expanded
isCertified
5.00
4.00
3.00
2.00
1.00
0.00
Normal
Lane Departure
Overspeed
Unsafe tail distance
Unsafe following


settings
force Y to 0
show line chart with focus
All fields:
eventType
isCertified
paymentScheme
hoursDriven
milesDriven
lat
long
isFoggy
isRainy
isWindy
Keys
eventType
Groups
Values
isCertified SUM
isCertified
5
4
3
2
1
0
Normal
Lane Departure
Overspeed
Unsafe tail distance
Unsafe following



TIP: In this lab you ran each section of code, known as a paragraph, individually. The entire notebook could have been played at once, however, by clicking the Play icon labeled Run all paragraphs directly to the right of the notebook title at the top of the browser.



6. SUMMARY

Congratulation! You have successfully created a Zeppelin notebook and demonstrated Zeppelin's ability to interpret multiple languages, and used a pre-built Zeppelin notebook to briefly explore Zeppelin's ability to ingest, view, analyze, and visualize data.

Spark Lab – Data Visualization, Reporting and using Zeppelin (Scala)

1. INTRODUCTION

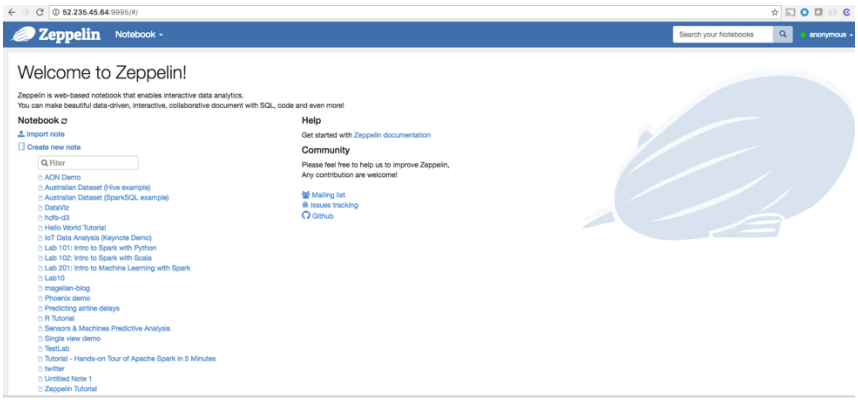
In this lab, we are going to learn to how to use Zeppelin to perform data visualizations, collaborate, and integrate visualization into reports.

We will describe how to ingest data into HDFS, to create table and manipulates tables with apache spark SQL and perform analytic queries on those tables, visualize the query results in real-time. Finally, we will show how collaborate on data project using zeppelin to create dynamic data visualization assets and collaborate with other users.

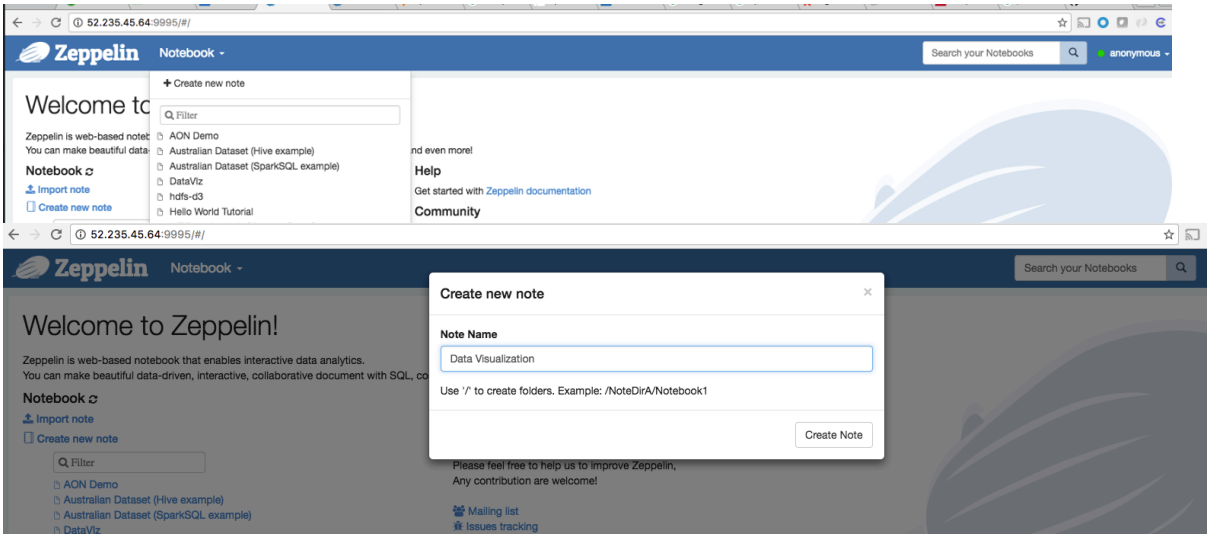
Caution! : Please don't copy and paste in the notebooks, it wil generate errors in the code syntax if you do so.

2. CREATE DATA VISUALIZATIONS NOTEBOOK

Open a web browser and enter the following URL : http://sandbox ip address>:9995/



Create a new note named Data Visualization



3. DOWNLOAD AND INGEST SAMPLE DATA

Use paragraphs in your zeppelin notebook to execute the following shell commands

Download SPARKSQL_LAB git repository :

```
%sh
git clone https://github.com/oascofare/SPARKSQL_LAB.git
```

Create a folder named “bankfile” under the zeppelin user directory on HDFS :

```
%sh
hdfs dfs –mkdir /user/zeppelin/bankfiles
```

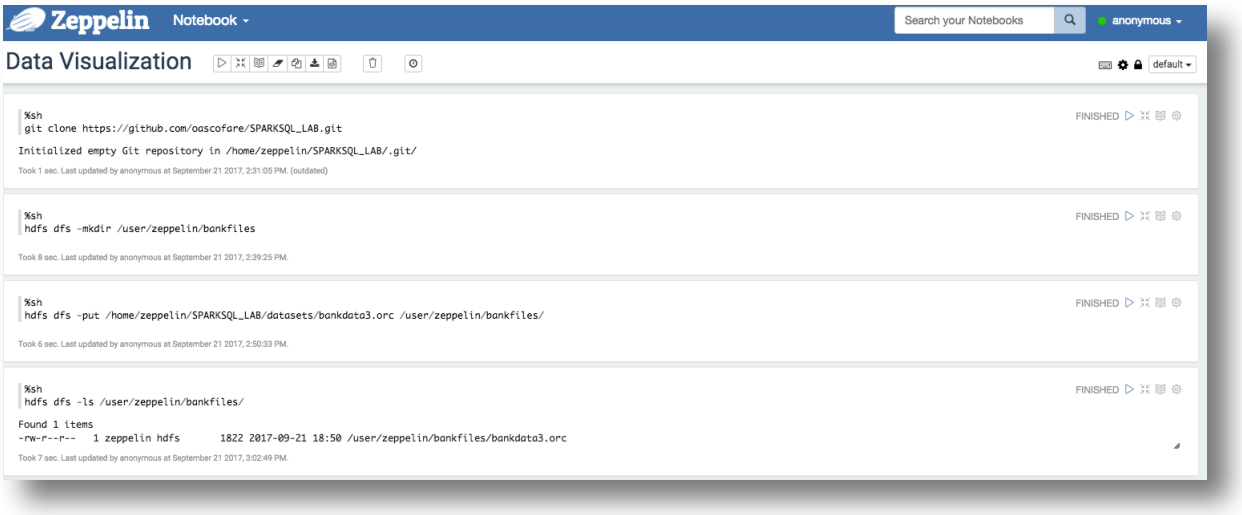
Upload the sample data into HDFS :

```
%sh
hdfs dfs -put /home/zeppelin/SPARKSQL_LAB/datasets/bankdata3.orc /user/zeppelin/bankfiles/
```

Confirm the file was uploaded successfully under HDFS directory `/user/zeppelin/bankfiles/`:

```
%sh
hdfs dfs –ls /user/zeppelin/bankfiles/
```

At this point, your note book should look as follow:



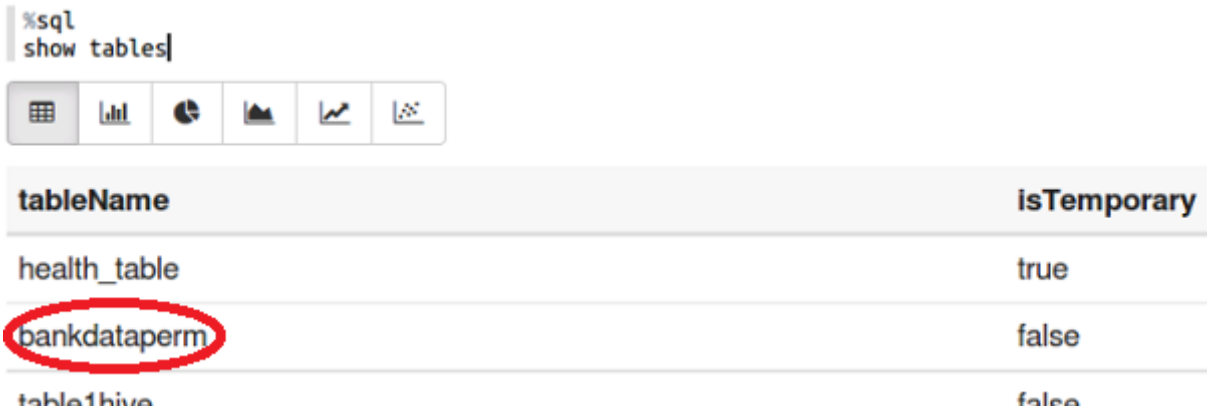
4. CREATE A DATAFRAME FROM HDFS FILE AND SAVE IT AS AN HIVE TABLE

Use the bankdata3.orc file to create a DataFrame named bankdata , a temporary table named banktemp , and a Hive table named bankdataperm .

```
%spark

val sqlContext = new org.apache.spark.sql.hive.HiveContext(sc)

val bankdata = sqlContext.read.format("orc").load("/user/zeppelin/bankfiles/bankdata3.orc")

bankdata.registerTempTable(“banktemp”)

sqlContext.sql(“create table bankdataperm as select * from banktemp”)
```

Use SQL to show the tables available and confirm that bankdataperm is available.

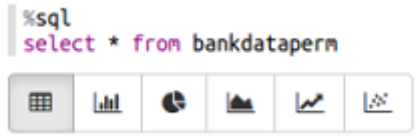
```
%sql
show tables
```



5. ANALYSE YOUR DATA

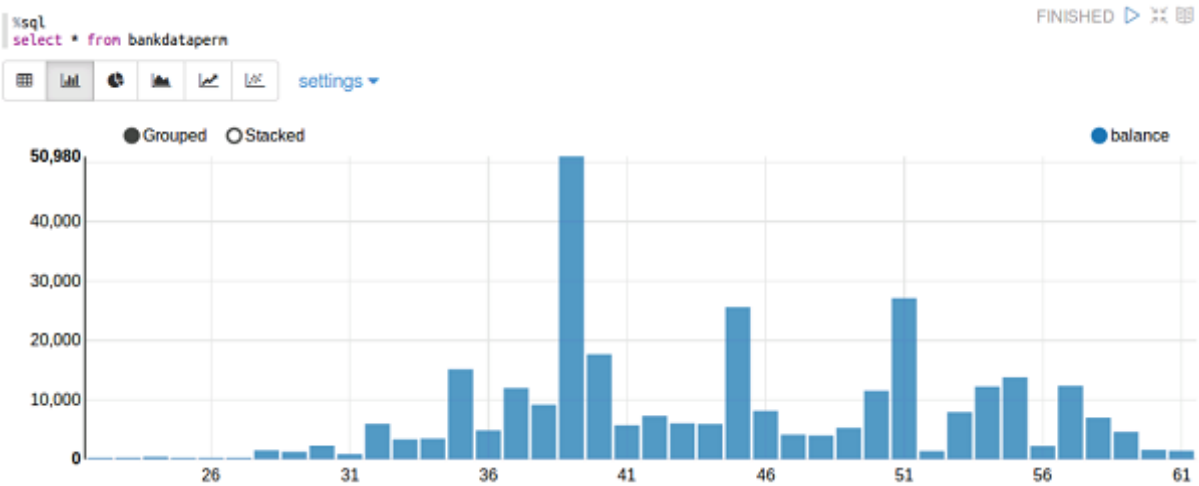
Use SQL to select and display all rows and columns from bankdataperm .

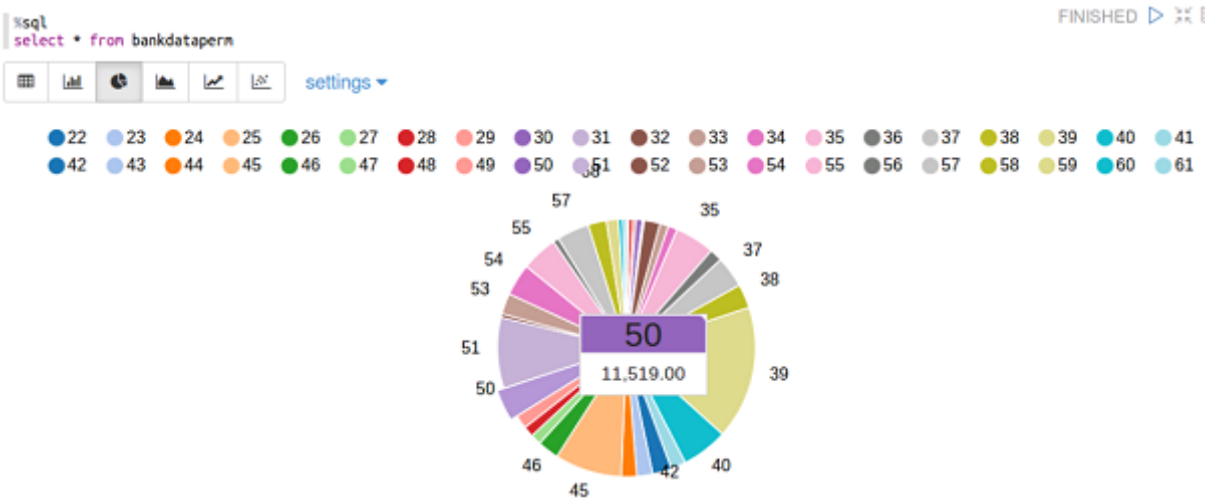
```
%sql
select * from bankdataperm
```

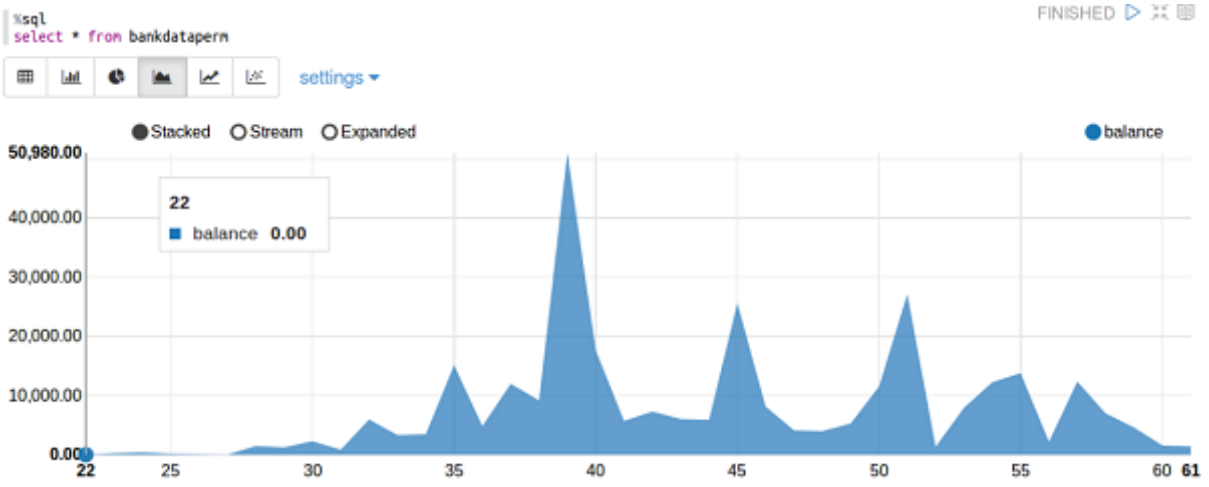


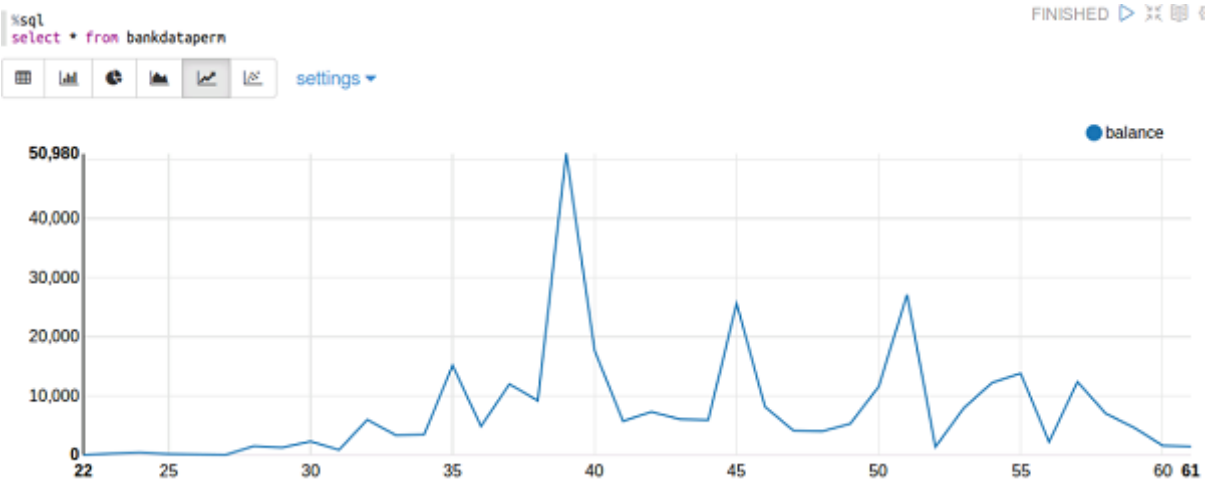
age	balance	marital
58	2,143	married
44	29	single
33	2	married
47	1,506	married
33	1	single

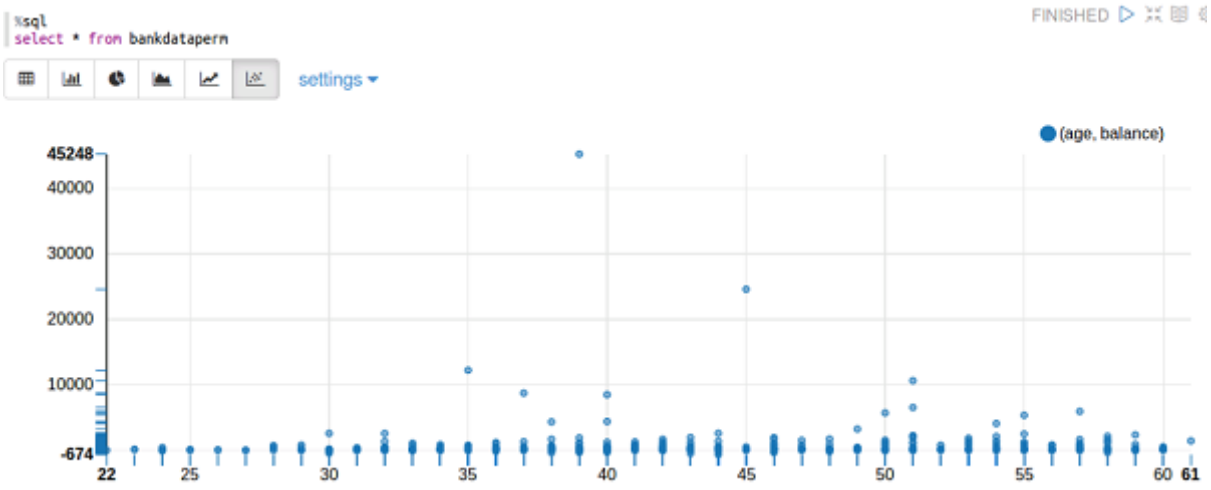
Quickly browse through the five data visualizations available by default in Zeppelin.




%sql
select * from bankdatapern
FINISHED
settings
22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41
42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61
50
11,519.00

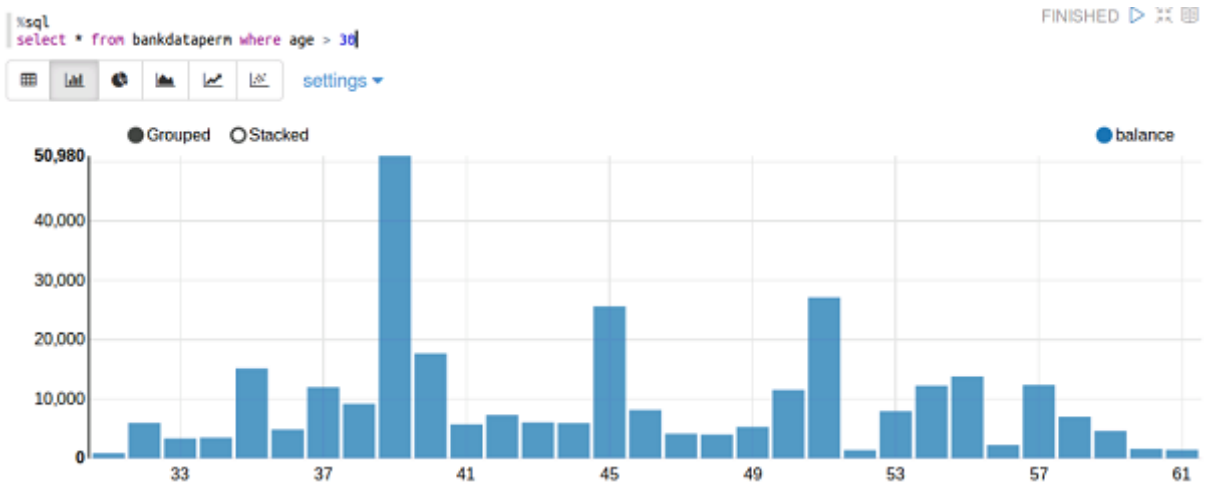

%sql
select * from bankdatapern
FINISHED
settings
Stacked
Stream
Expanded
balance
50,980.00
40,000.00
30,000.00
20,000.00
10,000.00
0.00
22
balance 0.00
22 25 30 35 40 45 50 55 60 61


%sql
select * from bankdatapern
FINISHED
settings
balance
50,980
40,000
30,000
20,000
10,000
0
22 25 30 35 40 45 50 55 60 61

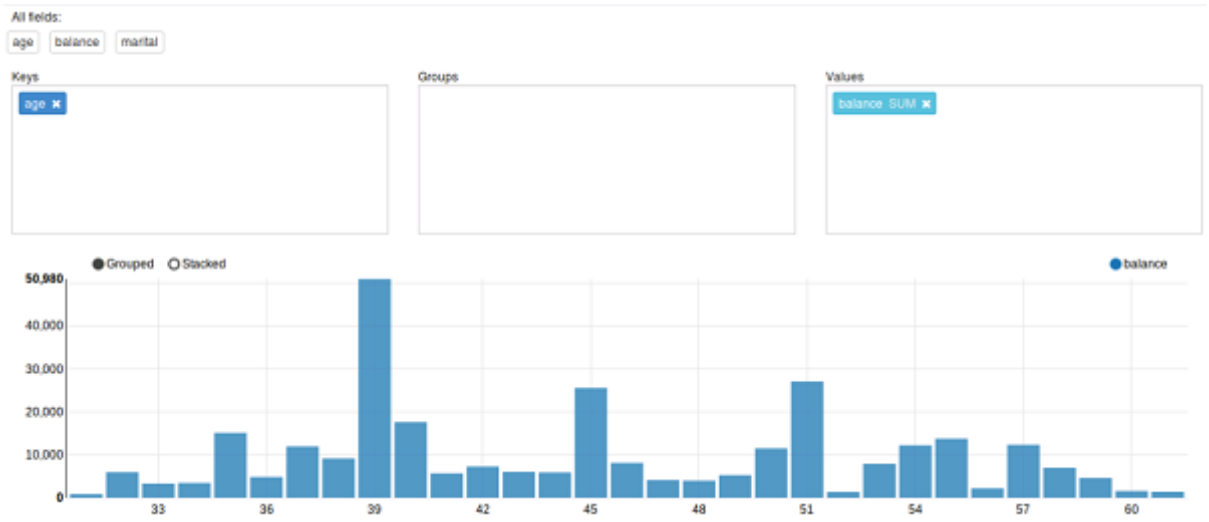


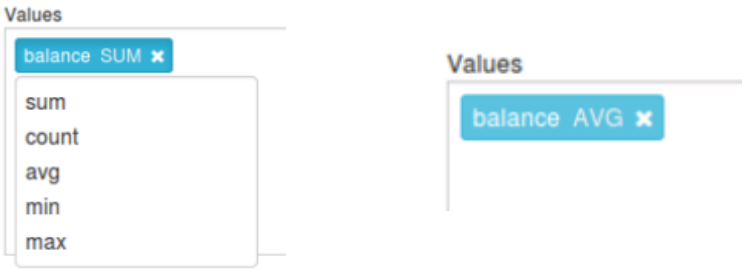
Now, go back to the bar chart view. Then, edit your SQL query so that it only shows data for individuals over the age of 30. Run the query and note the change in the chart.

```
%sql
select * from bankdataperm where age > 30
```

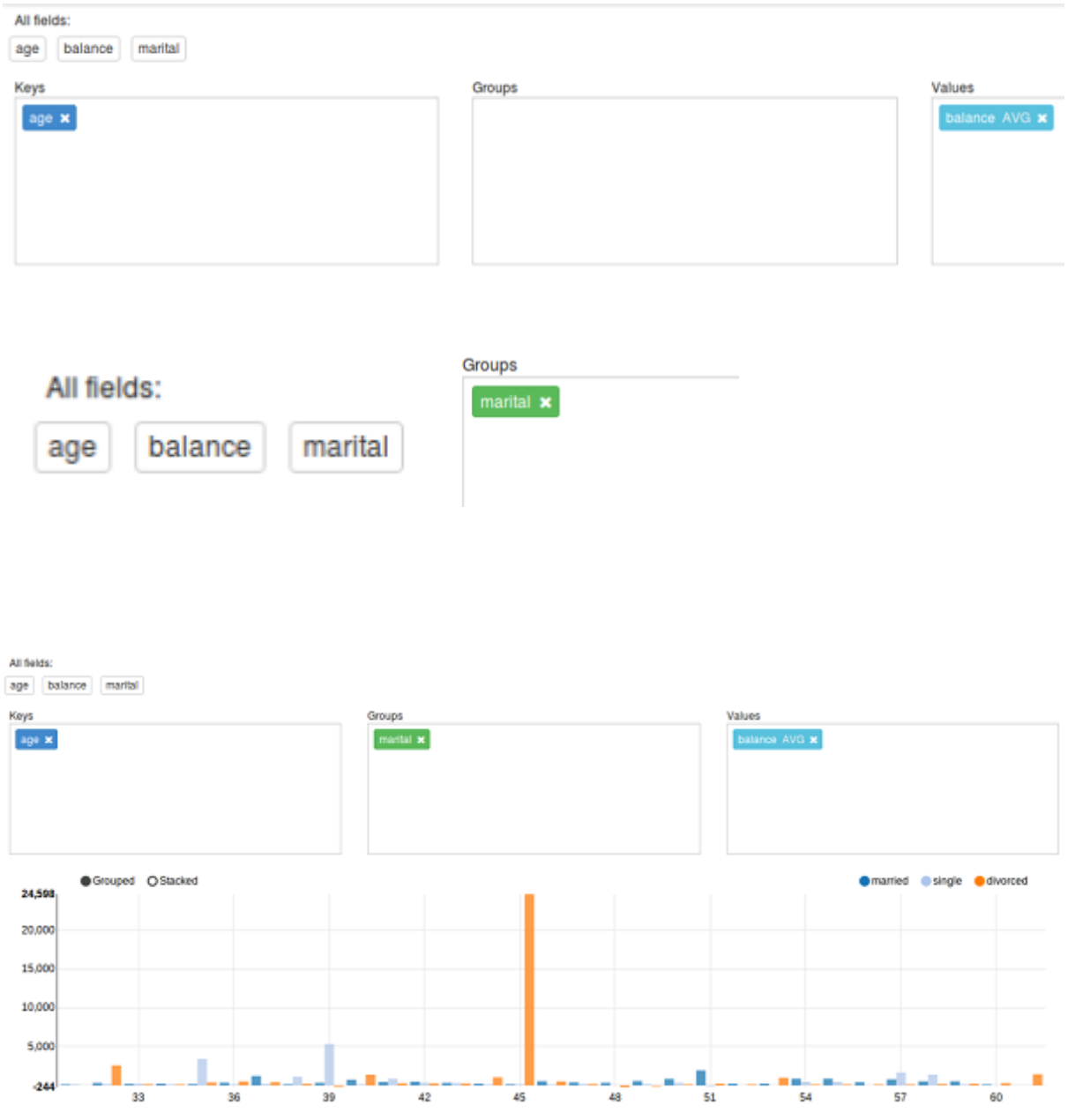


Click on the settings link and notice that Zeppelin has selected the age column as the key column and is showing the sum of the balances for all individuals in each age bracket. Display the average balance instead of the sum of balances.





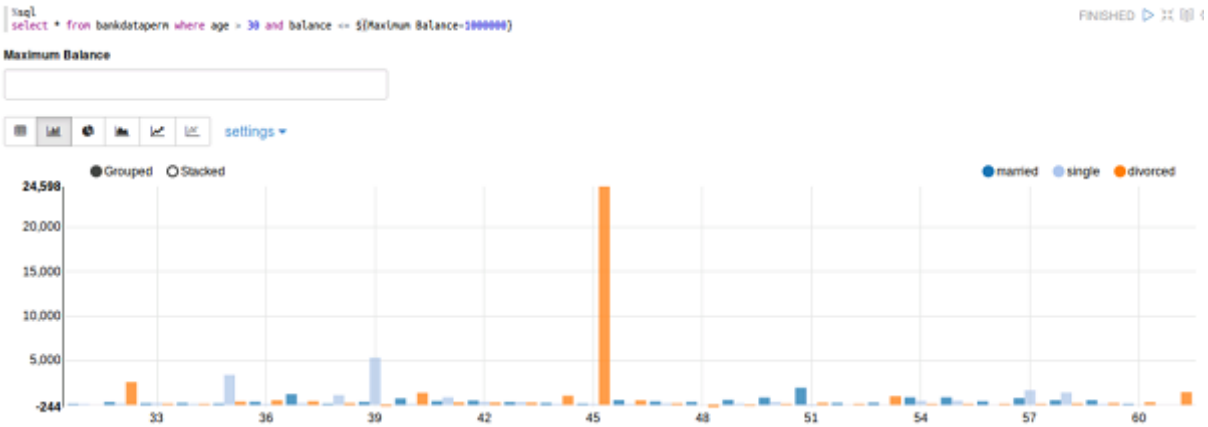
Click and drag the available **marital** field into the **Groups c**ategory to modify the visualization so that data is shown not only by age, but also grouped by **marital status.** When you are finished, click the settings link again to close the pivot chart options.

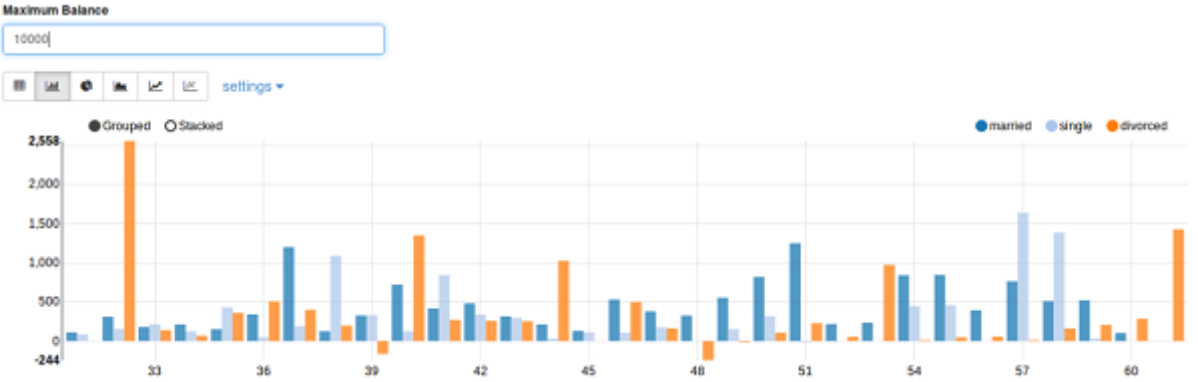


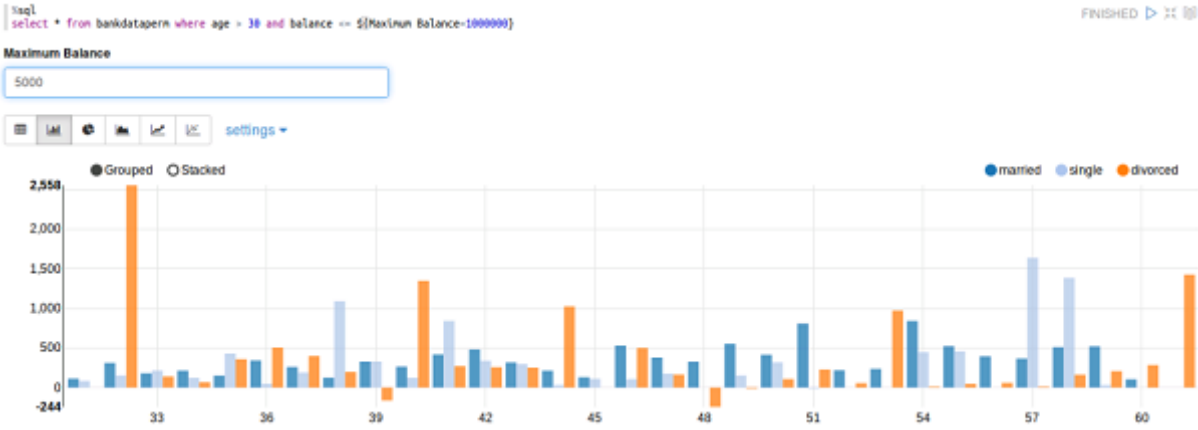
6. USE DYNAMIC VARIABLES IN YOUR ANALYSIS

It appears that we have a single outlier that is skewing the data significantly. We can easily see that the vast majority of average balances are well below $5,000. Add a dynamic form to the SQL query that allows you to filter out data where the maximum balance for any individual exceeds a certain threshold, but set the default to 1,000,000 so that it doesn't immediately modify the chart. Rerun the query with this new code, then use this dynamic form to adjust the maximum balance to $10,000 and $5,000 and note the effects on the visualization.

```
%sql
select * from bankdataperm where age > 30 and
balance <= ${Maximum Balance=1000000}
```







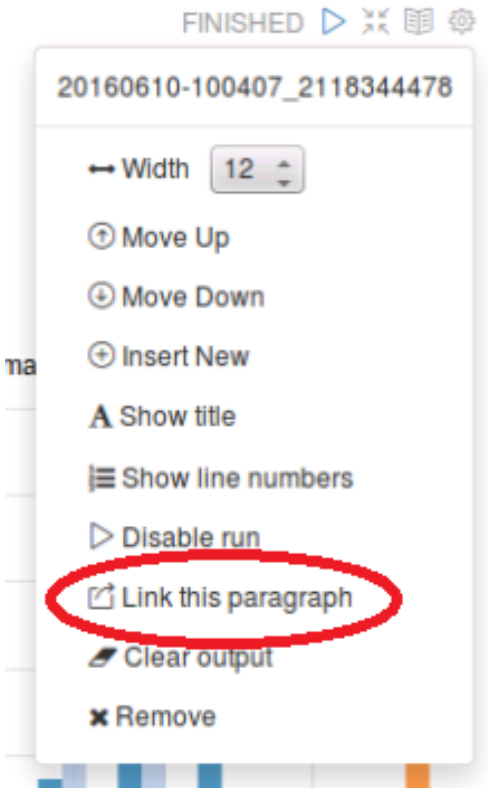
QUESTIONS:

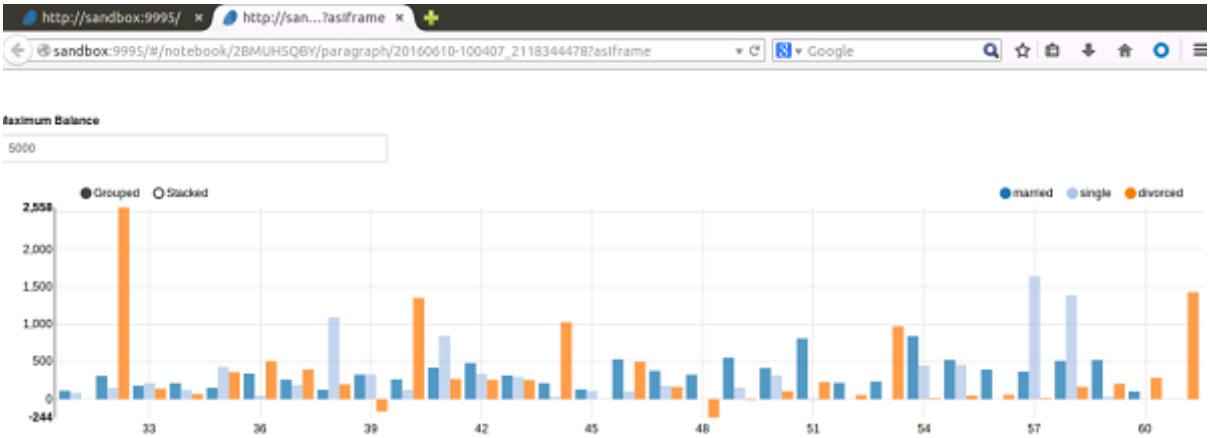
Why do you think changing the maximum balance from $10,000 to $5,000 had so little effect on the chart?

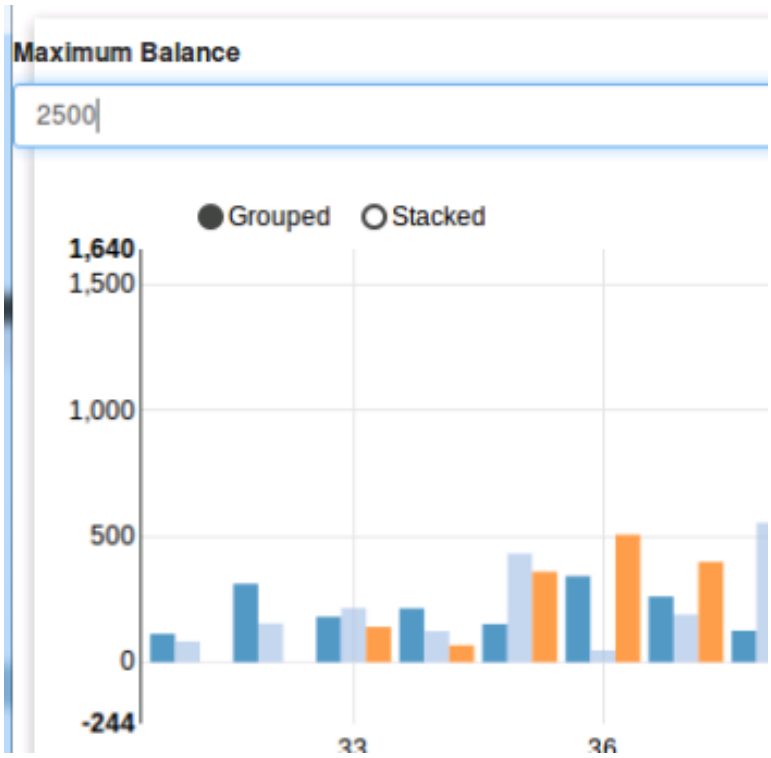
What group (married, single, or divorced) had the most change based on changing the maximum balance?

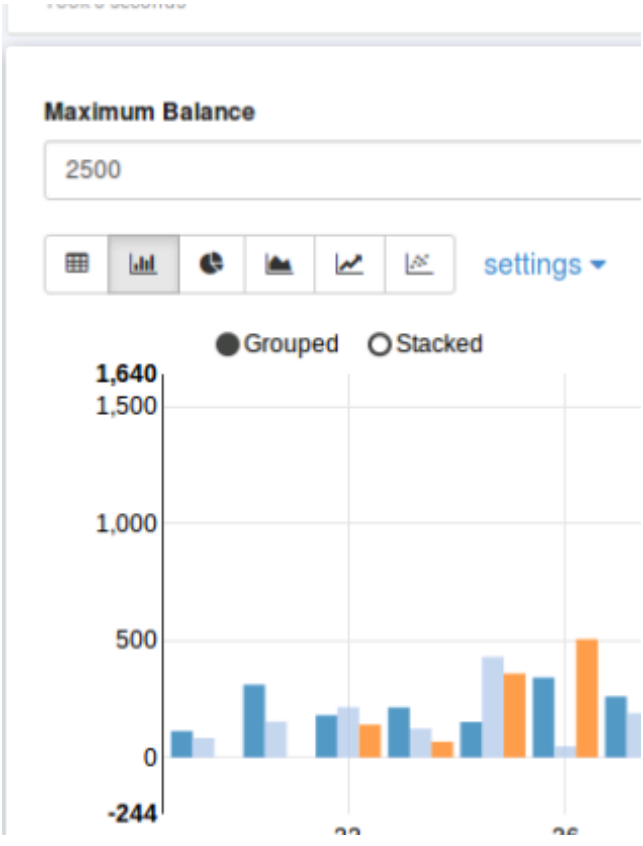
7. SHARE YOUR FINDINGS WITH OTHERS

Create a URL that allows you to share this chart with others without giving them access to the code or the Zeppelin note. Use the linked page to change the maximum balance to $2,500, then return to your note and observe the effects the change had at the source.

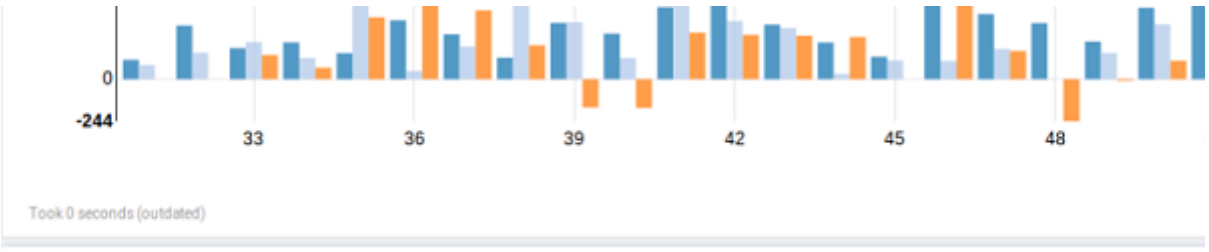





Maximum Balance
2500
Grouped
Stacked
1,640
1,500
1,000
500
0
-244
33
36

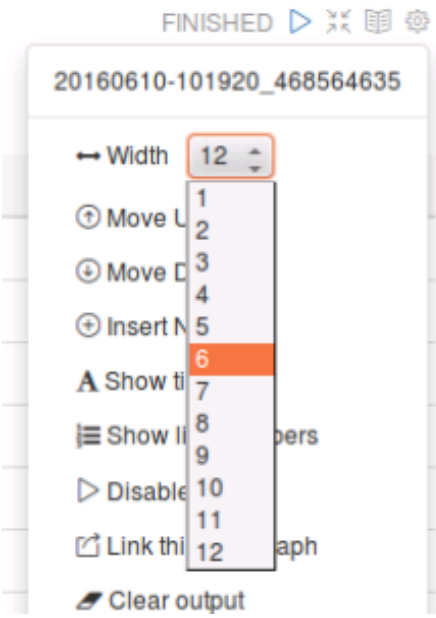

Maximum Balance
2500
settings
Grouped
Stacked
1,640
1,500
1,000
500
0
-244

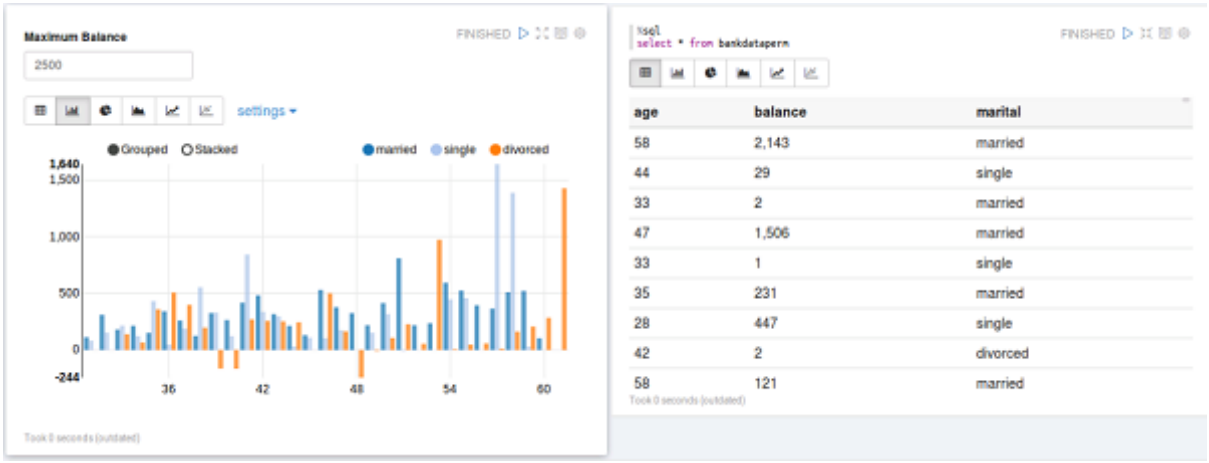
In the paragraph below this one, run the SQL command to read all data from bankdataperm. Then adjust the width of the two paragraphs so that they both appear on the same line.



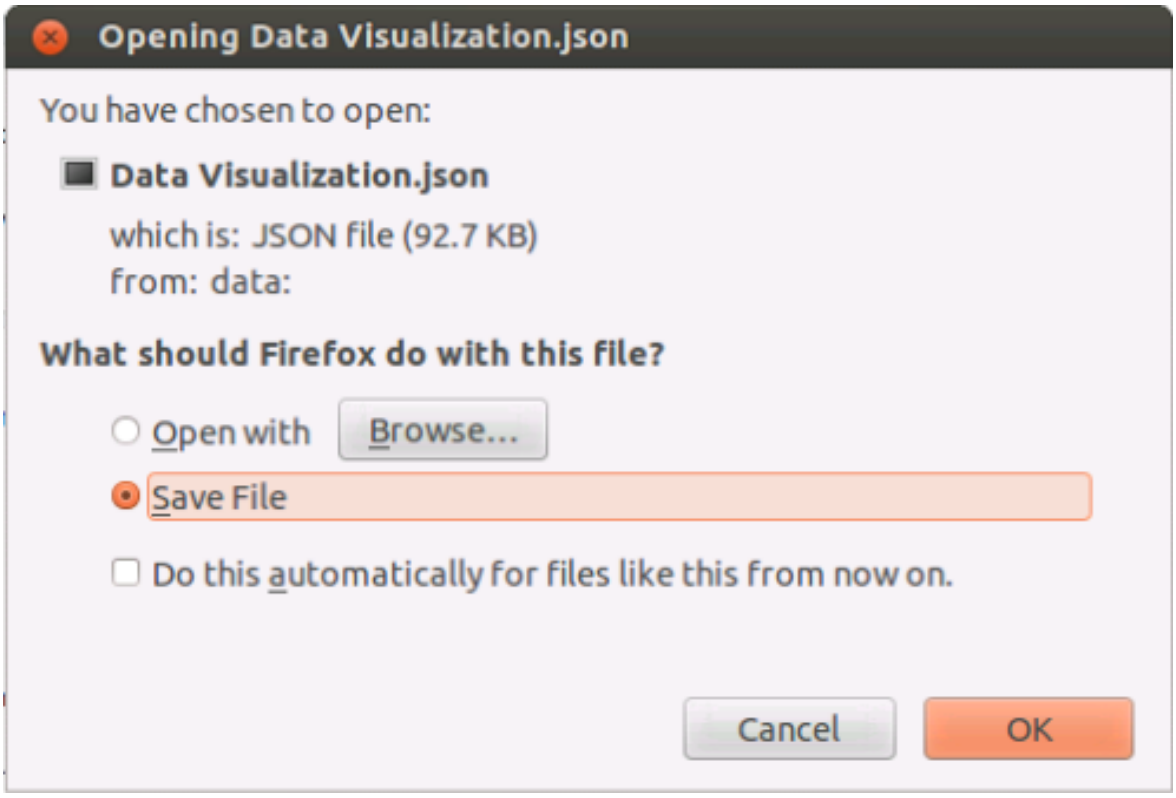

```
%sql
select * from bankdataperm
```

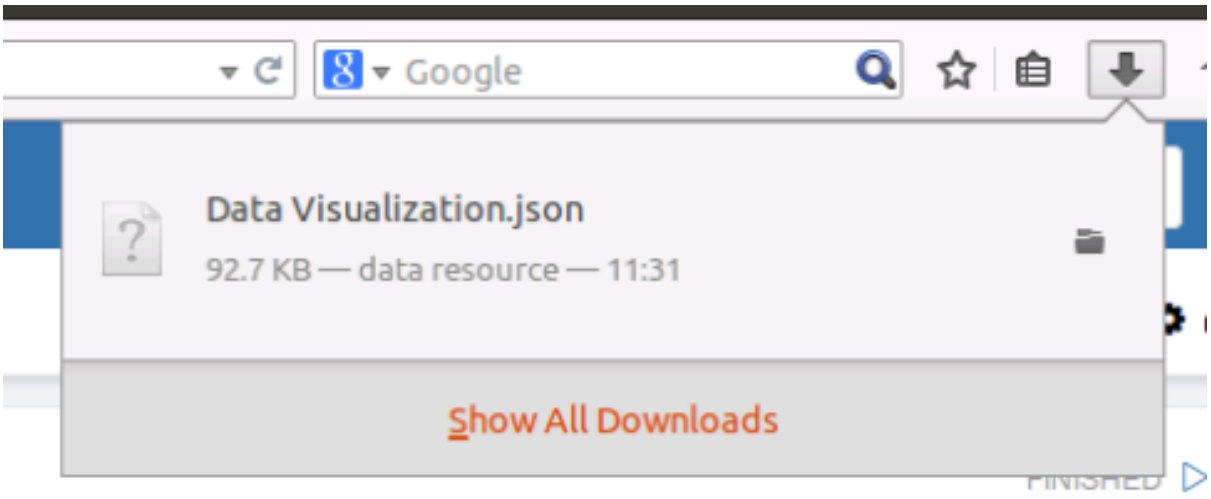
age	balance	marital
58	2,143	married
44	29	single
33	2	married
47	1,506	married



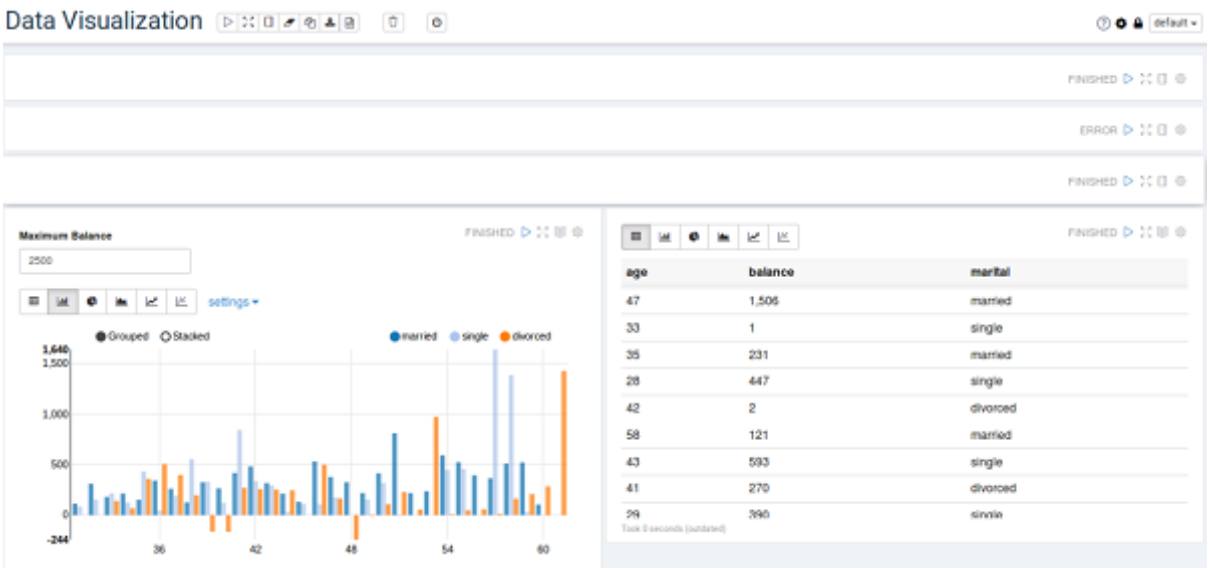


We are now ready to prepare this note for sharing. Create a clone copy of this note named Data Visualization Clone. Also export a copy of the note.

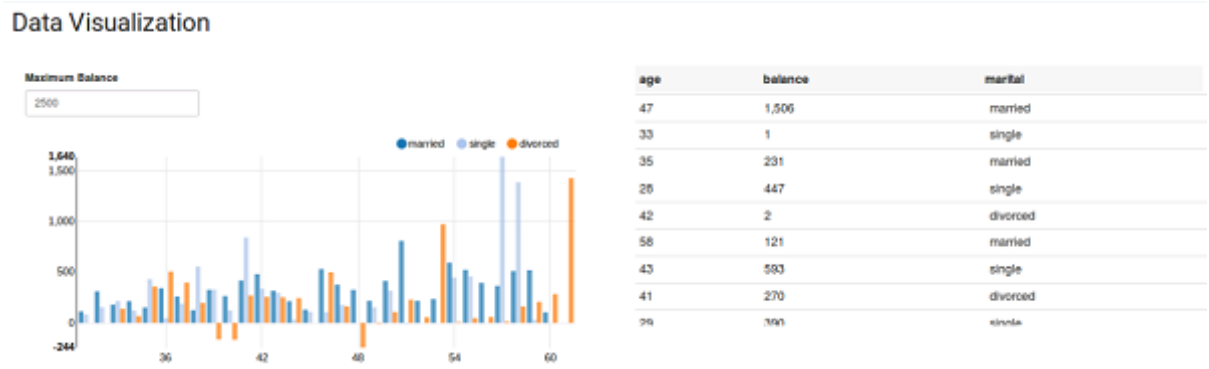

Opening Data Visualization.json
You have chosen to open:
Data Visualization.json
which is: JSON file (92.7 KB)
from: data:
What should Firefox do with this file?
Open with
Browse...
Save File
Do this automatically for files like this from now on.
Cancel
OK


Google
Data Visualization.json
92.7 KB — data resource — 11:31
Show All Downloads

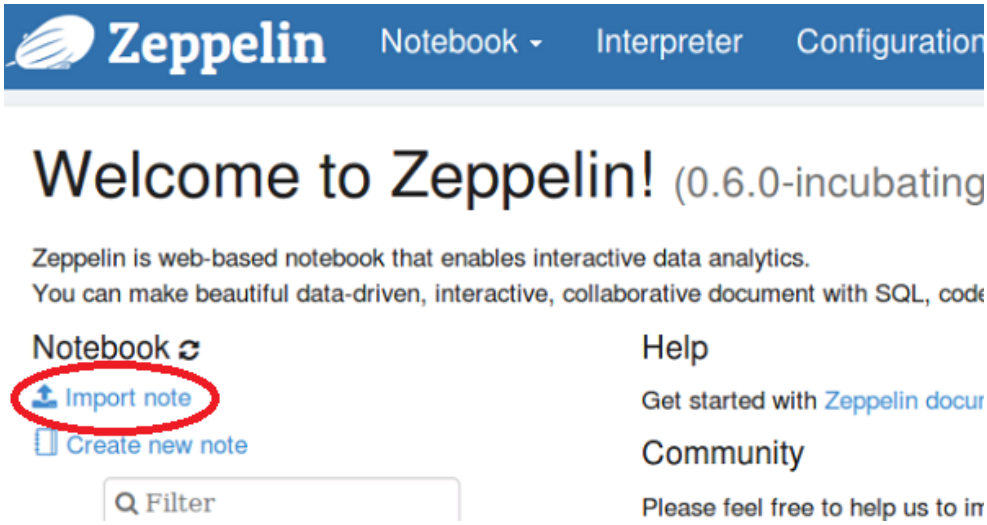
On the Data Visualization note we are going to share, hide the code for all paragraphs. Then hide the output for every paragraph except for the two that are on the same line.

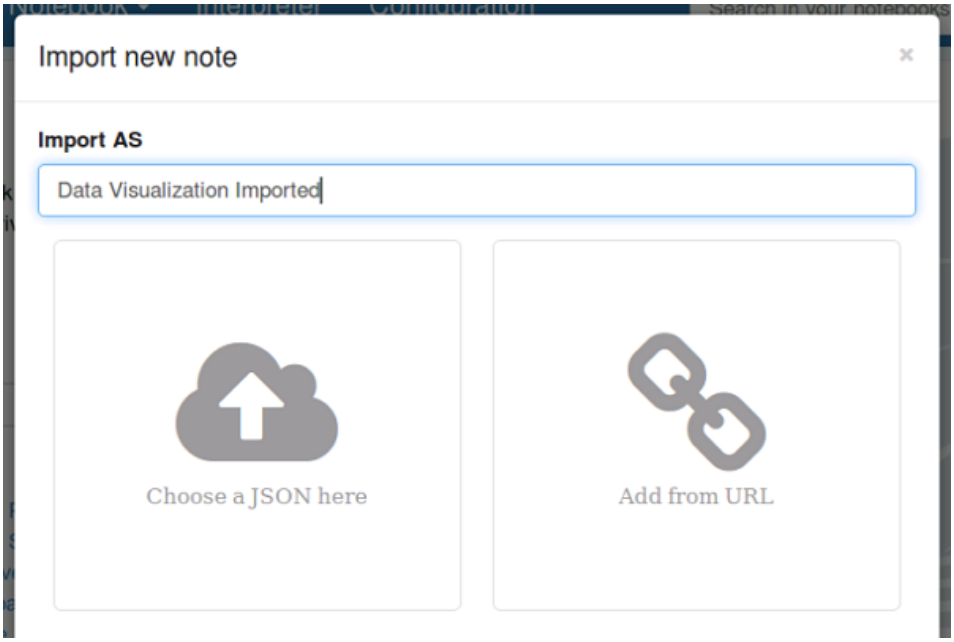


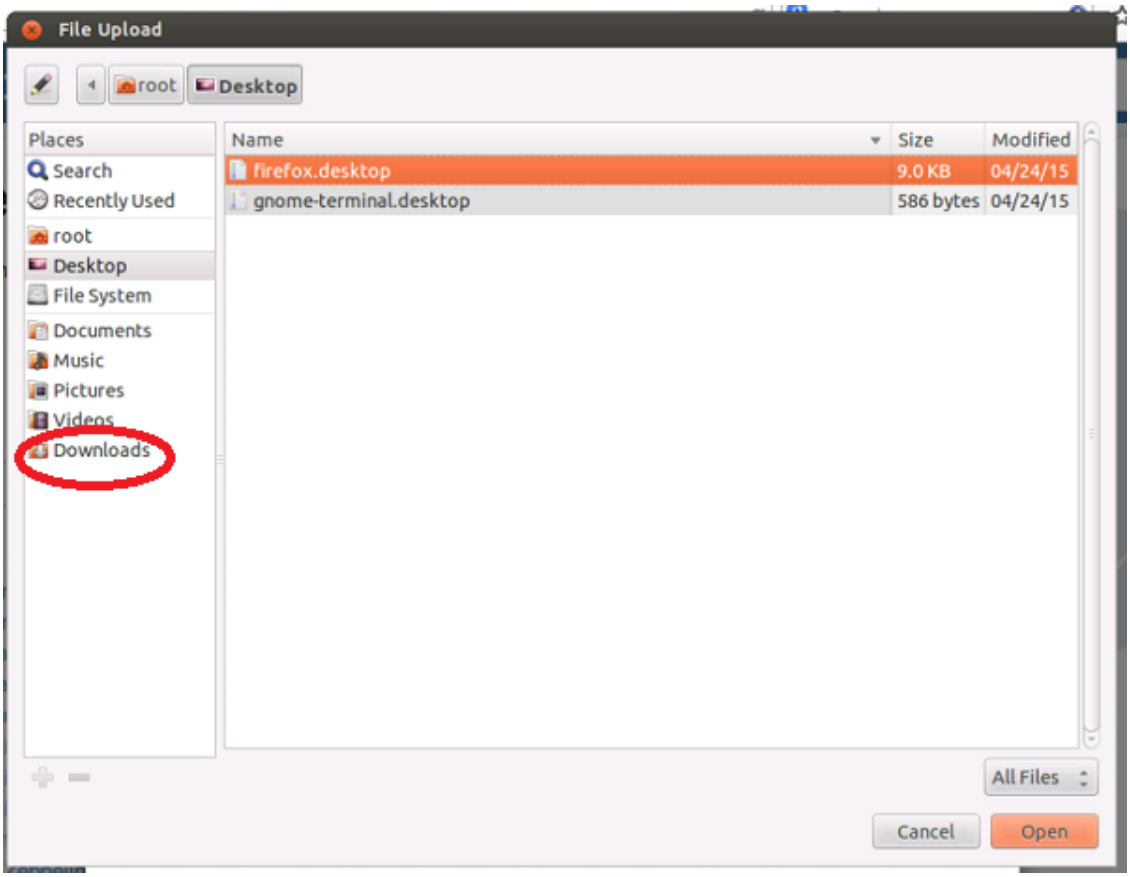
Next, convert this from the default view to report view. Now the URL to this note is ready to share with your stakeholders.

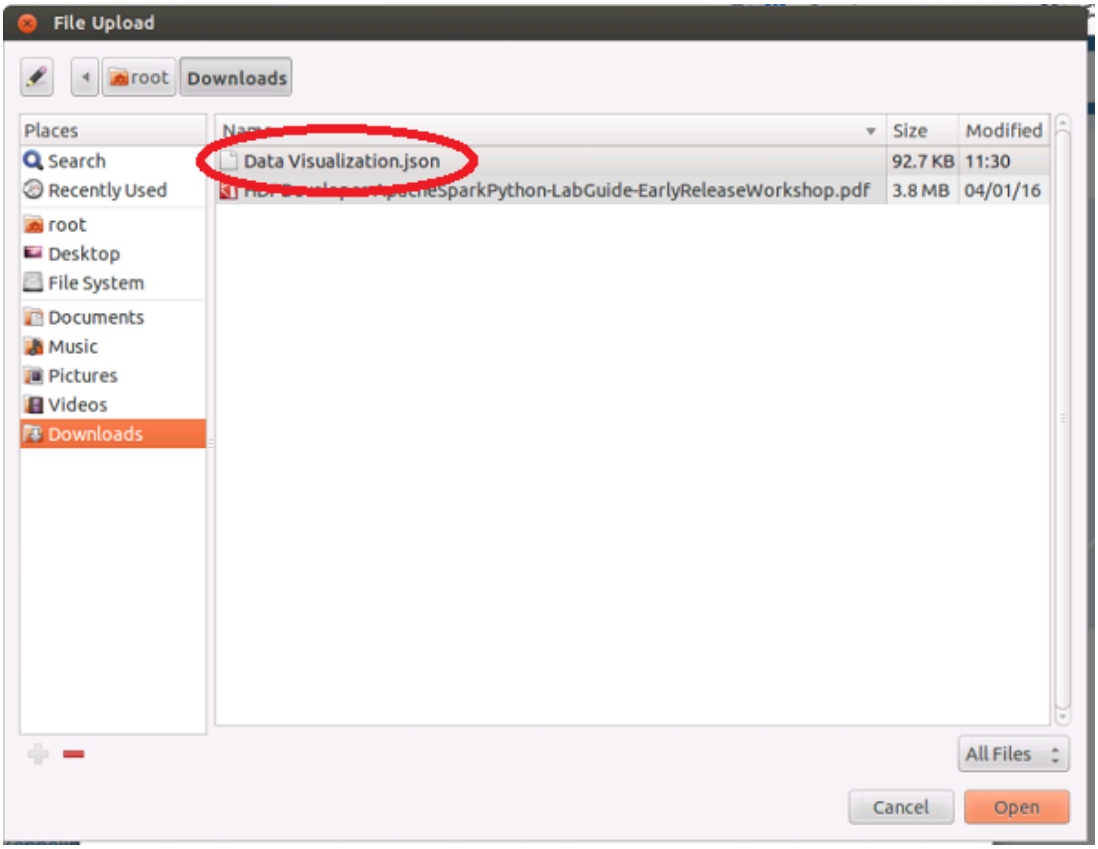


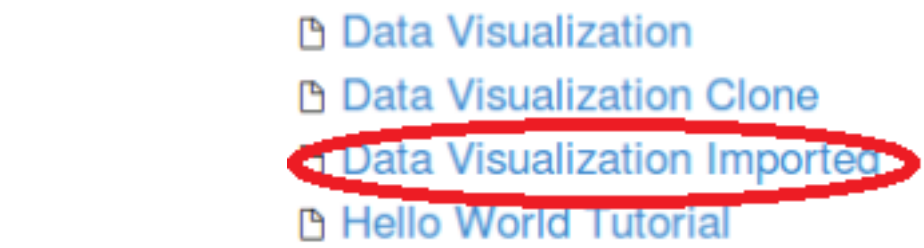
Import the copy of this note you made earlier and name the new note Data Visualization Imported. Confirm that the copy contains all original code and formatting.

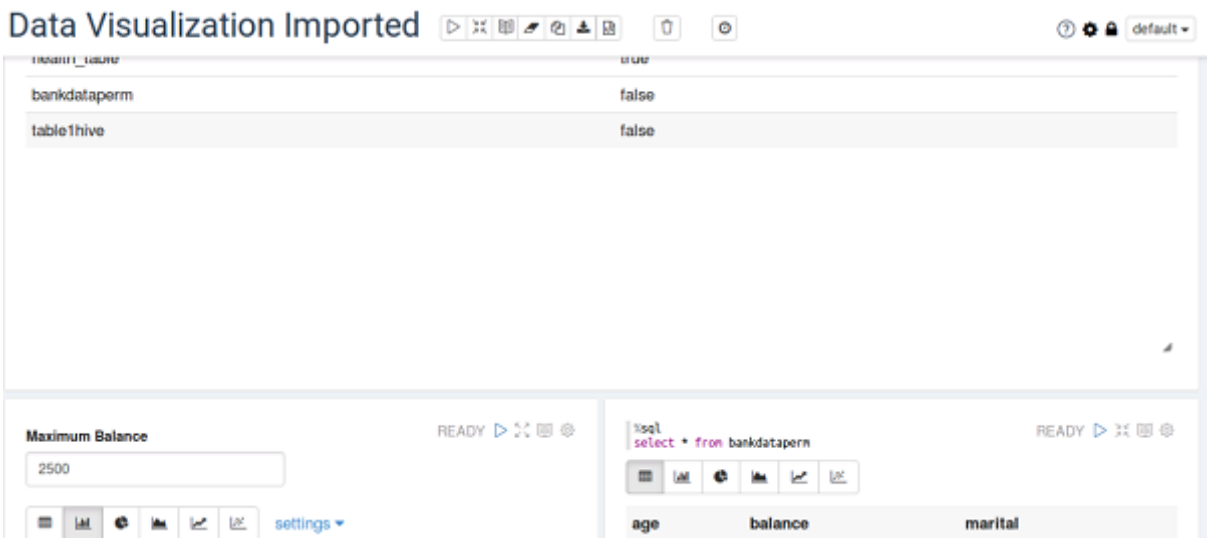





File Upload
root
Desktop
Places
Search
Recently Used
root
Desktop
File System
Documents
Music
Pictures
Videos
Downloads
Name
Size
Modified
firefox.desktop
9.0 KB
04/24/15
gnome-terminal.desktop
586 bytes
04/24/15
All Files
Cancel
Open


File Upload
root
Downloads
Places
Search
Recently Used
root
Desktop
File System
Documents
Music
Pictures
Videos
Downloads
Size
Modified
Data Visualization.json
92.7 KB
11:30
3.8 MB
04/01/16
All Files
Cancel
Open





8. SUMMARY

Congratulation! You have successfully created and manipulated Zeppelin visualizations, made them available for collaboration, and used Zeppelin to create a shareable report.

SparkSQL Lab – Using SparkSQL on Hive Tables

1. INTRODUCTION

In this lab, you will use Zeppelin to query data using Spark SQL on Hive Table. You will learn to visualize data, and write data after processing with SparkSQL back to a Hive Table or a text file on HDFS.

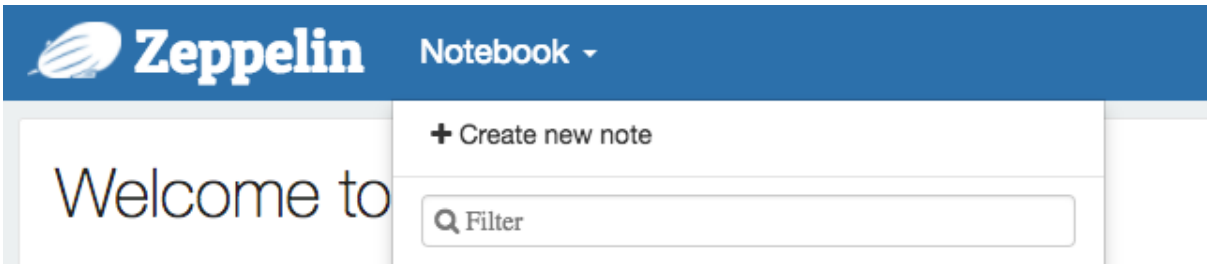
Caution! : Please don't copy and paste in the notebooks, it will generate errors in the code syntax if you do so.

2. CREATE A NEW ZEPPELIN NOTEBOOK

Navigate to the Zeppelin UI on your sandbox:

```
http://<public_ip_addr>:9995
```

Expand the Notebook drop down and click on "+Create new note":



Create a new note called for example: **"Hive Data Exploration with SparkSQL".**

3. READING A HIVE TABLE IN A SPARK DATAFRAME

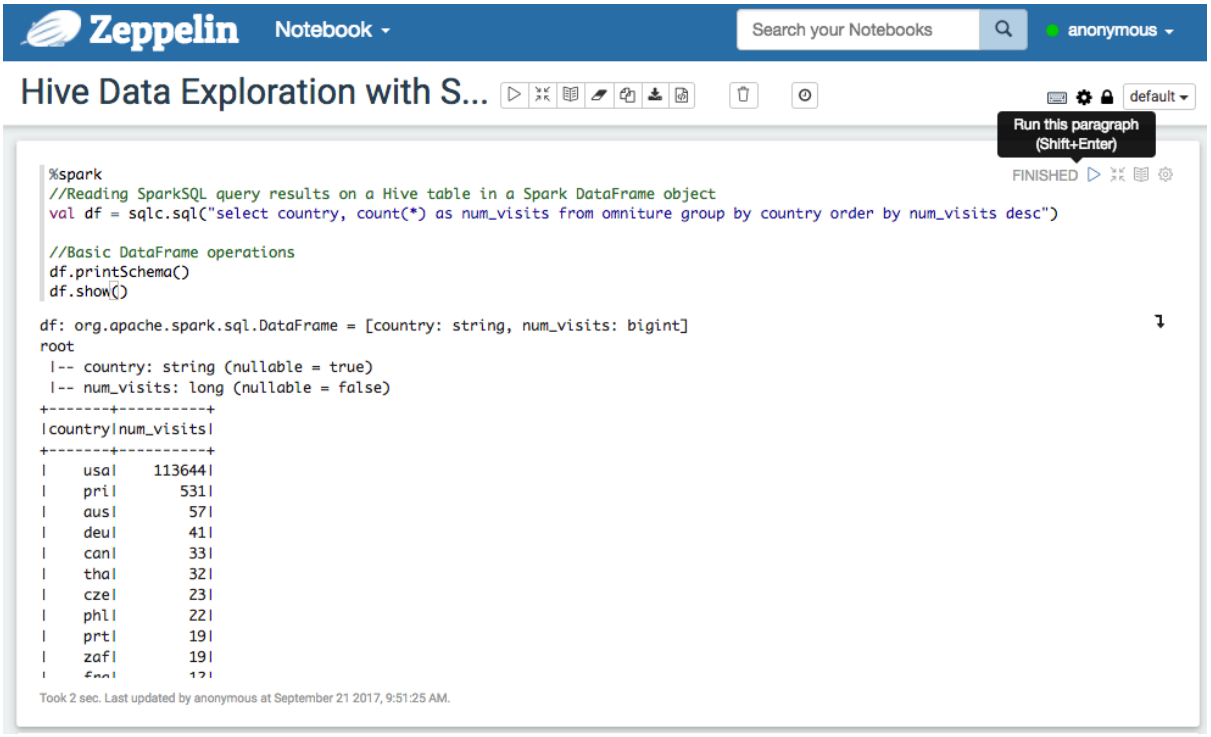
In the first paragraph, we are going to invoke the Spark Hive Context sqlc to query the omniture Hive table created in the Hive lab using the spark interpreter, and read the data into a Spark DataFrame object:

```
%spark

val df = sqlc.sql("select country, count(*) as num_visits from omniture group by
country order by num_visits desc")
```

We can make basic Spark Dataframe APIs to print the schema of the dataframe object, and display the dataset (still in the same paragraph:

```
//Basic DataFrame operations
df.printSchema()
df.show()
```

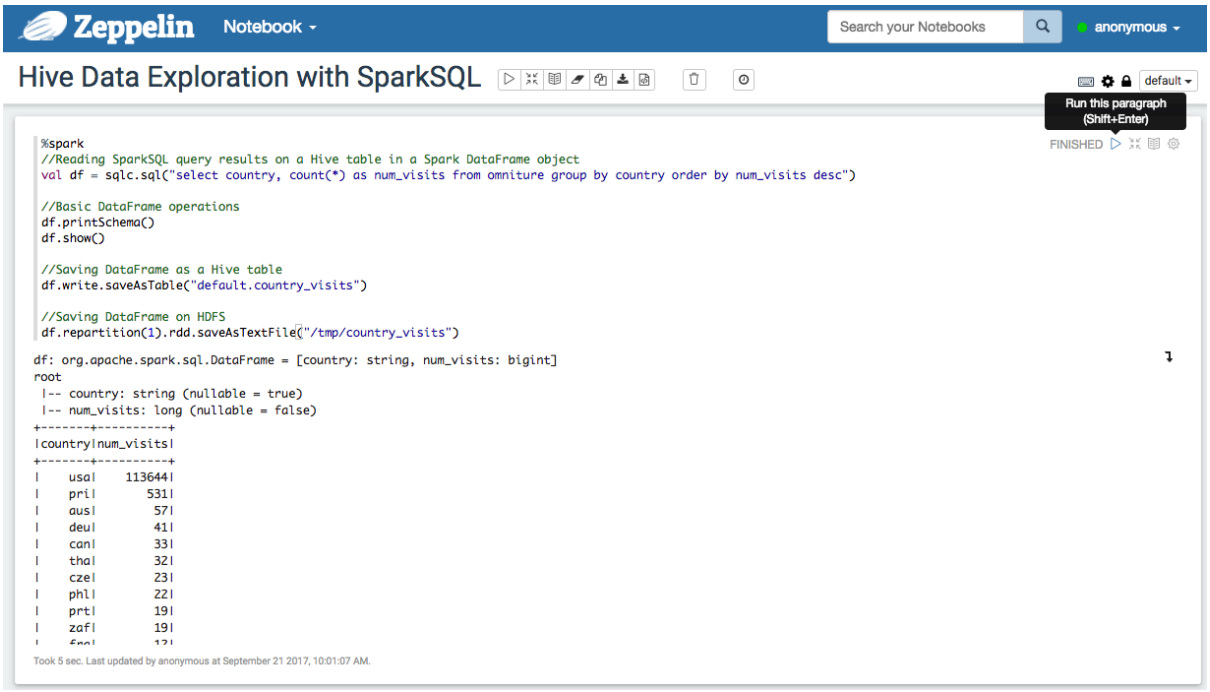
At this point, you can run the paragraph by clicking the play icon (Blue triangle) on the top right side of the paragraph. This should show you the dataframe schema as well as the dataframe contents:



After the processing was done in SparkSQL, we can now write the results back as a Hive table, as well as save the results in a text file on HDFS using the following commands. On the same paragraph, type:

```
//Saving DataFrame as a Hive table
df.write.saveAsTable("default.country_visits")

//Saving DataFrame on HDFS
df.repartition(1).rdd.saveAsTextFile("/tmp/country_visits")
```

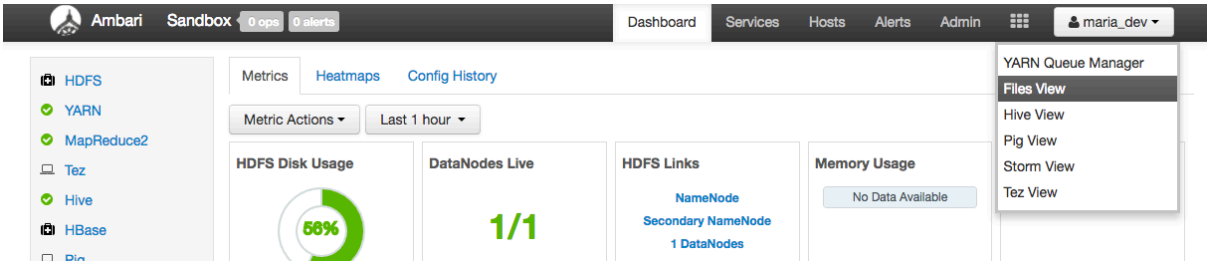
Execute the paragraph again by clicking on the Run icon.



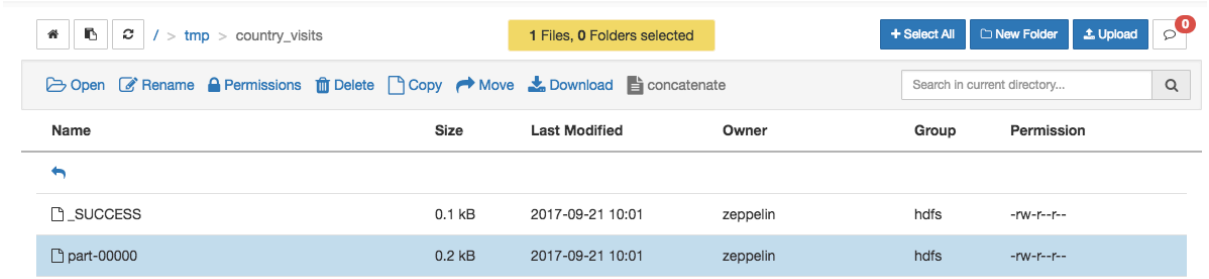
If the paragraph ran successfully, let's check if the file was written on HDFS. Navigate to Ambari on a separate tab on your browser, and sign in as user/password maria_dev/maria_dev

```
http://<public_ip_addr>:8080
```

Click on File View:

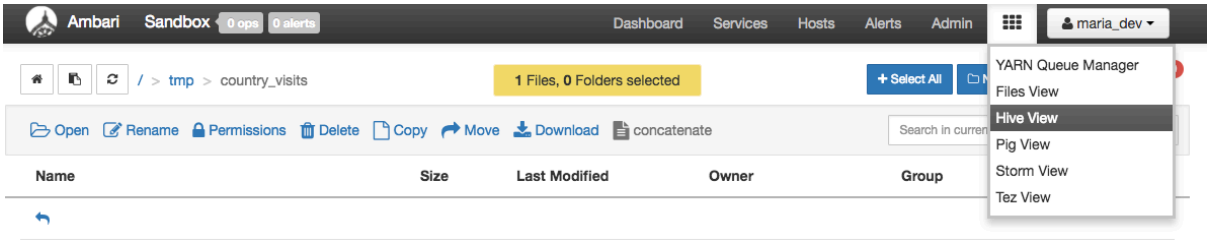


Click on '**tmp**' folder, then '**country_visits**' folder. Select the **part-0000** file and on the menu that appears click on '**Open**':

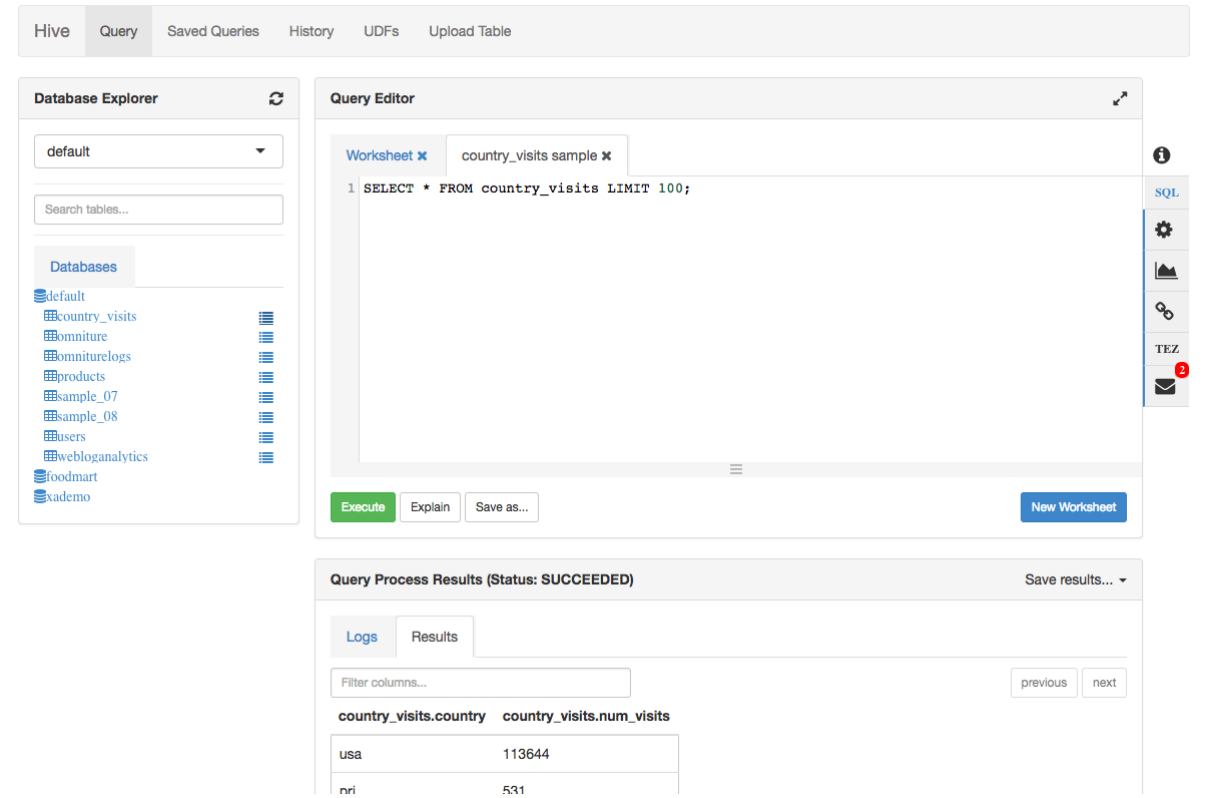


You should see the contents of the dataframe. Note that the current format is a text file. For a csv format, this requires the spark-csv library which is not loaded by default on the HDP 2.5 sandbox. This library is available by default on Spark 2.0.

We can now check the existence of the Hive table. Navigate to the Hive view:



Once the Hive view, click on the **Database Explorer**, and expand the **default** database. You should see the **country_visits** table listed there. Click on the icon on the right hand side to get a sample of the data. In the **Results** tab at the bottom, you should see the contents of the table corresponding to the original dataframe.

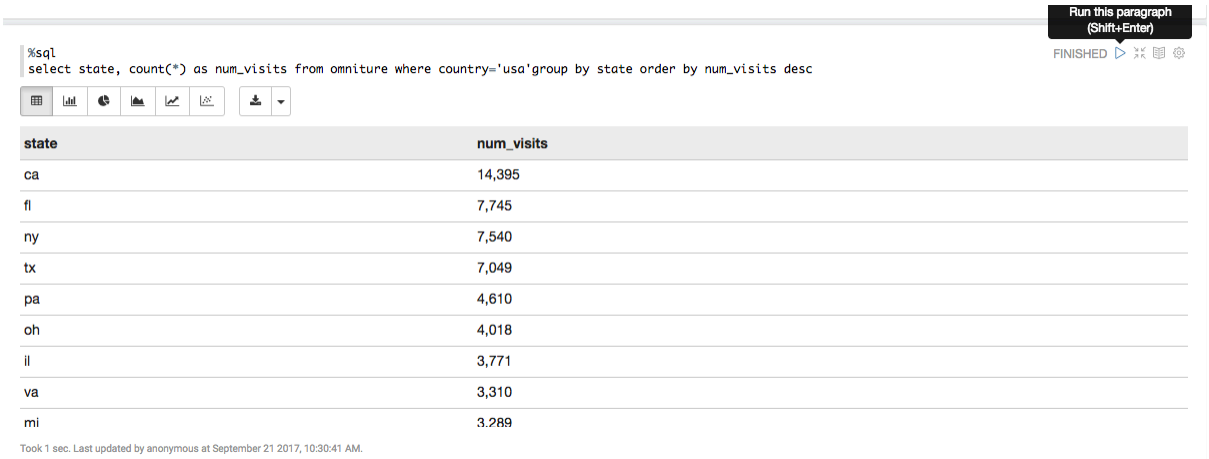


4. EXPLORE AND VISUALIZE HIVE TABLES USING SPARKSQL

Start a new paragraph and invoke the SparkSQL interpreter %sql. Type the following query to get a drill down of the number of visits in the US per state, as this seems to be the main geography visiting the web site.

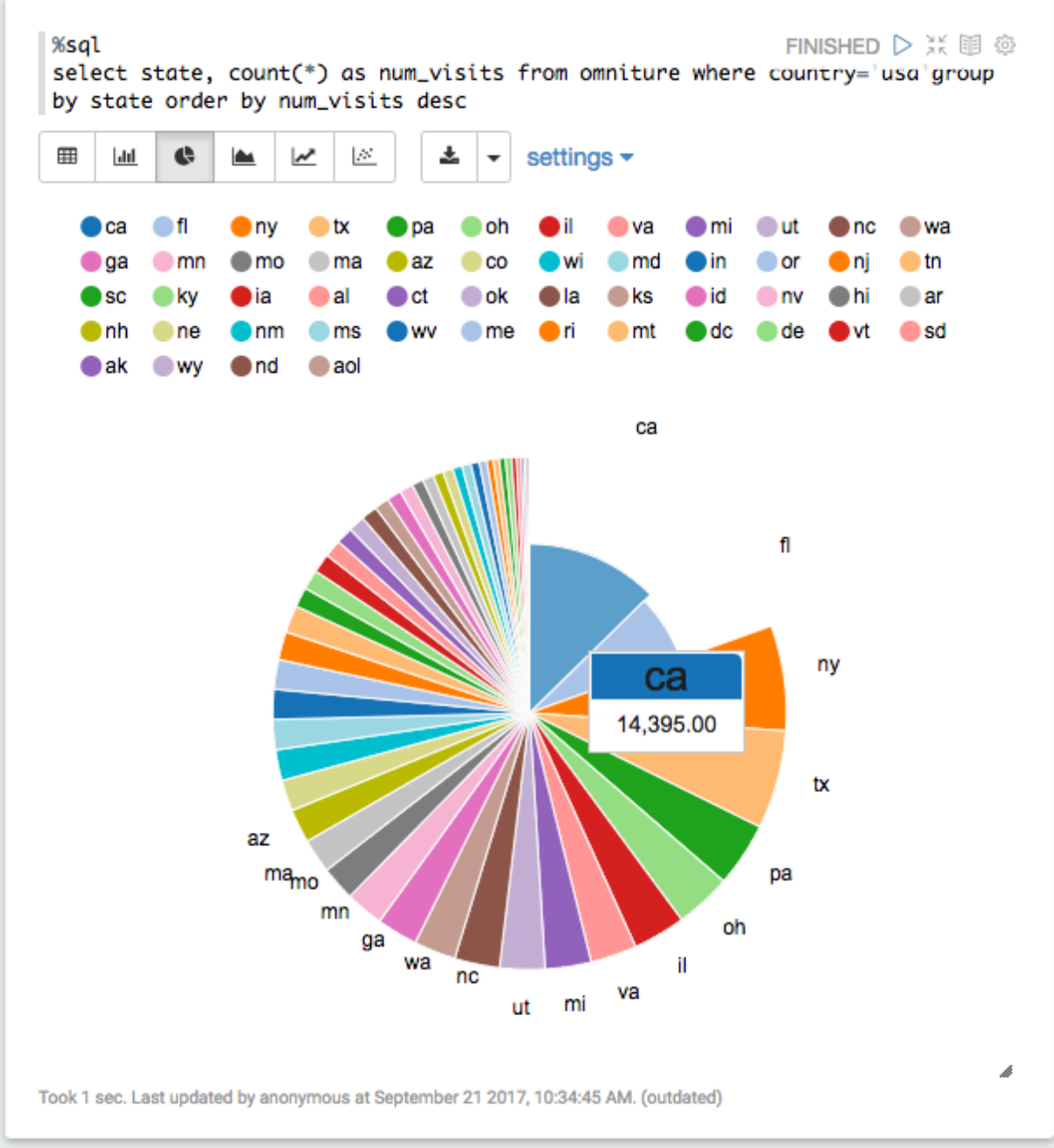
```
%sql
select state, count(*) as num_visits from omniture where country='usa'group
by state order by num_visits desc
```

Click on the Run icon.



Let's visualize this data in a pie chart. Click on the pie chart icon, and expand the settings drop down. Drag and drop "**state**" field in the Keys section, and **num_visits** in the value section as follows:

Click on settings again to hide the setting section. Resize the paragraph window appropriately and hover over the pie chart, you should see the different values corresponding to the various states:

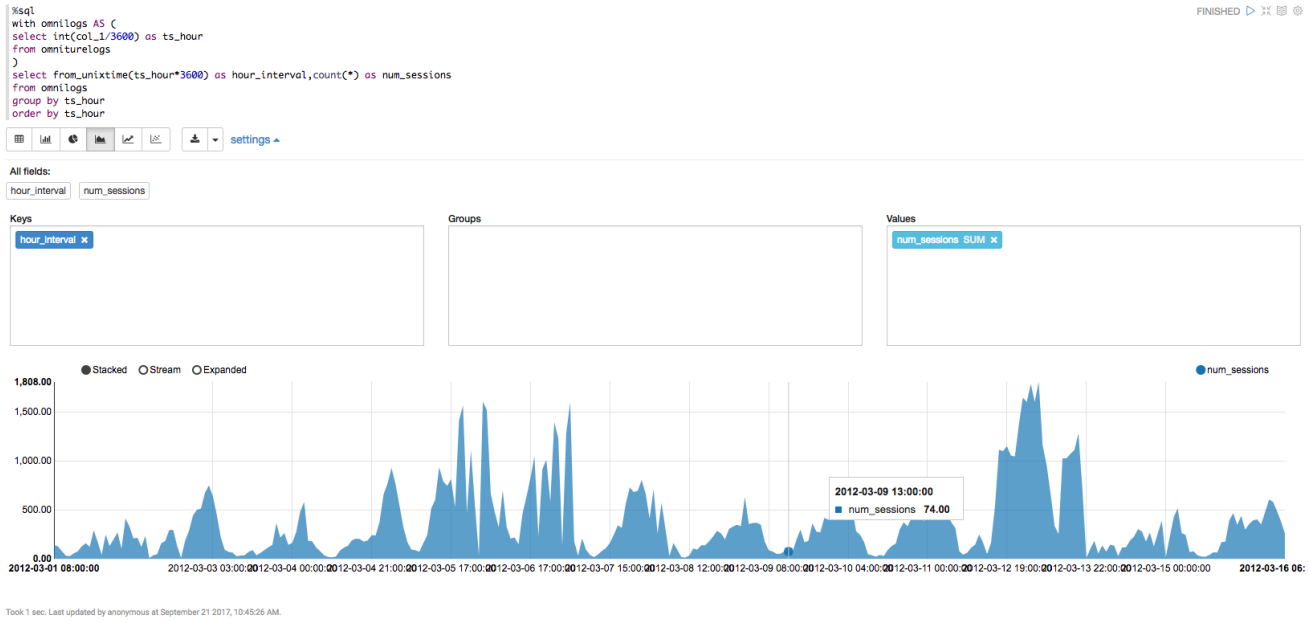


We can also try an interesting visualization which gives us the number of visits per hour from the omniture logs. Start a new paragraph, and type the following query:

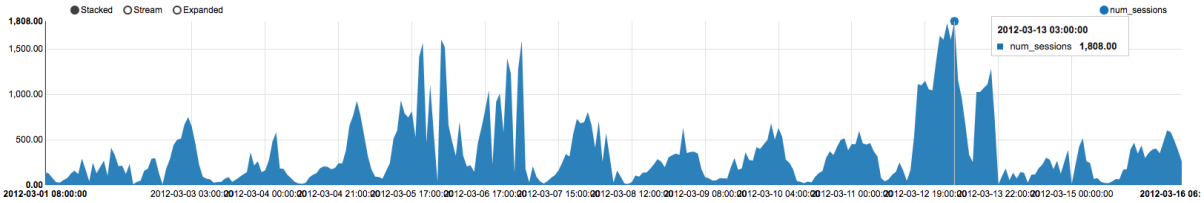
```
%sql
with omnilogs AS (
select int(col_1/3600) as ts_hour
from omniturelogs)
select from_unixtime(ts_hour*3600) as hour_interval,count(*) as num_sessions
from omnilogs
group by ts_hour
order by ts_hour
```

Click on the run icon. We can visualize the data as a graph as follows. Select the graph icon (in grey below). Use **hour_interval** as Keys, and **num_sessions** as Values. You

should see the following visualization which shows peak times when the web site was visited.



Hover over the graph to find out the peak from the period captured from the time interval of the web log:



5. SUMMARY

Excellent! We've learned how to access data in Hive tables using the SparkSQL API, as well as how to use the SparkSQL interpreter to seamlessly query and visualize Hive tables using Zeppelin.