



Hortonworks Data Platform

Hands On Labs

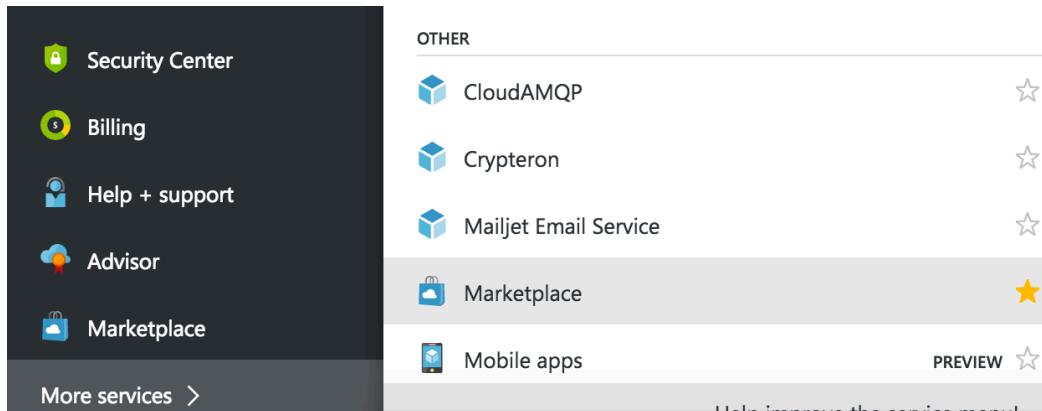
Hive , Zeppelin, Spark SQL

Table of Contents

Lab Prerequisite – Setting up the Azure Environment	3
Hive Lab – Loading and Querying Data with Hadoop.....	10
1. INTRODUCTION.....	11
2. DOWNLOAD SAMPLE DATA.....	11
3. UPLOAD DATA INTO HDFS.....	12
4. CREATE HIVE TABLES.....	15
5. LOAD DATA INTO TABLES	17
6. SAVE AND EXECUTE A QUERY.....	18
7. JOIN DATA FROM MULTIPLE TABLES	21
8. SUMMARY	22
Spark Lab – Introduction to Zeppelin	23
1. INTRODUCTION.....	24
2. ACCESS AND BROWSE ZEPPELIN	25
4. WORKING WITH ZEPPELIN INTERPRETERS	26
5. USE A PREBUILT NOTEBOOK TO EXPLORE ZEPPELIN CAPABILITIES	30
6. SUMMARY	36
Spark Lab – Data Visualization, Reporting and using Zeppelin (Scala)	37
1. INTRODUCTION.....	38
2. CREATE DATA VISUALIZATIONS NOTEBOOK	39
4. CREATE A DATAFRAME FROM HDFS FILE AND SAVE IT AS AN HIVE TABLE	41
5. ANALYSE YOUR DATA.....	42
6. USE DYNAMIC VARIABLES IN YOUR ANALYSIS	47
7. SHARE YOUR FINDINGS WITH OTHERS	49
8. SUMMARY	57
SparkSQL Lab – Using SparkSQL on Hive Tables	58
1. INTRODUCTION.....	59
2. CREATE A NEW ZEPPELIN NOTEBOOK.....	59
3. READING A HIVE TABLE IN A SPARK DATAFRAME.....	59
4. EXPLORE AND VISUALIZE HIVE TABLES USING SPARKSQL	63
5. SUMMARY	65

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.

Results			
NAME	PUBLISHER	CATEGORY	
Hortonworks Sandbox with HDP 2.4 (Staged)	Hortonworks	Compute	
Hortonworks Sandbox with HDP 2.5 (Staged)	Hortonworks	Compute	
Hortonworks Data Platform Standard (Staged)	Hortonworks	Compute	
Hortonworks Sandbox with HDP 2.5	Hortonworks	Compute	
Cloudbreak for Hortonworks Data Platform	Hortonworks	Compute	
Hortonworks Sandbox with HDP 2.4	Hortonworks	Compute	

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: *new or existing*
- Location

Basics

* Name
hdf-workshop ✓

VM disk type ⓘ
SSD

* User name
ppruski ✓

* Authentication type
SSH public key **Password**

* Password
***** ✓

* Confirm password
***** ✓

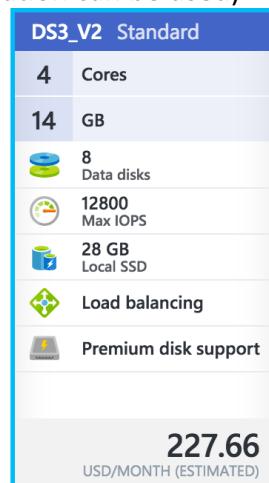
Subscription
SE

* Resource group ⓘ
 Create new Use existing
hdf-workshop ✓

Location
Canada Central

OK

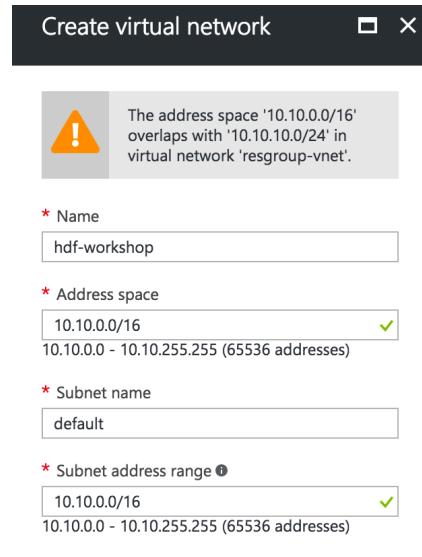
- b. For an appropriate sized environment specify DS3_V2. (Note: if the size is unavailable an alternate location can be used)



c. Settings

Use defaults for all unless otherwise specified below:

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



d. Summary

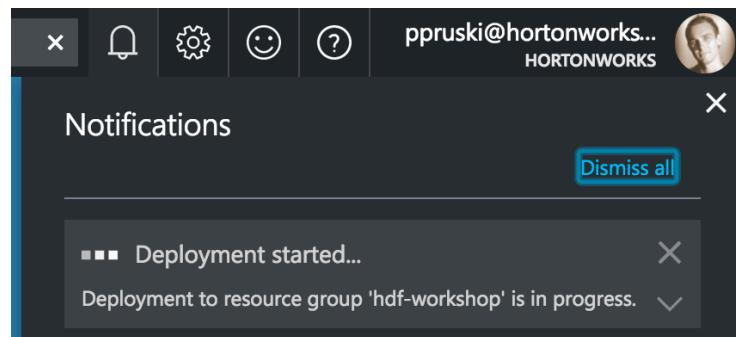
After reviewing, click **OK**.

S

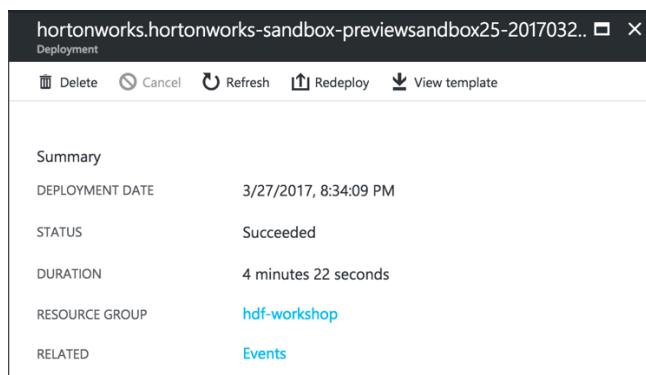
e. Buy

After reviewing, click **Purchase**.

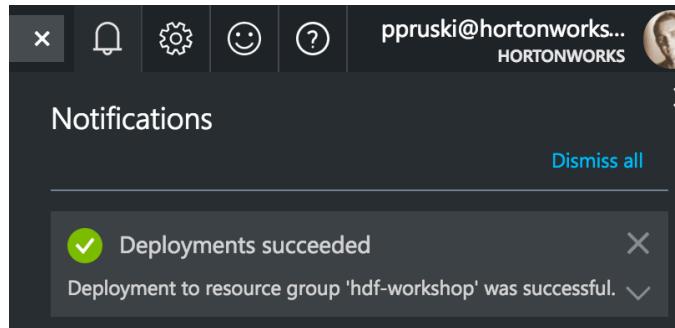
5. You can view the progress of the deployment by clicking the notification icon on the top right of the portal and selecting the deployment.



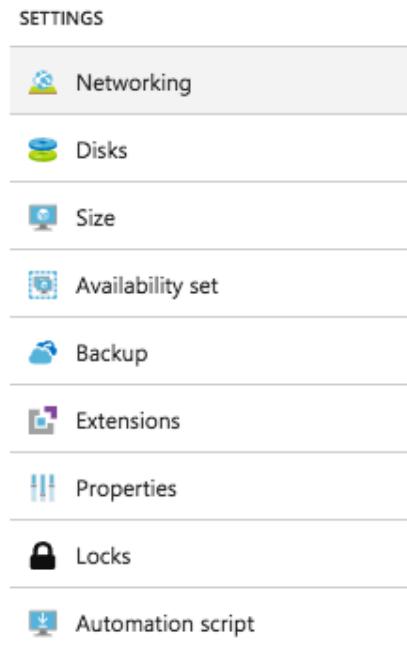
6. The deployment can take a few minutes. Once it is complete, you should see a screen similar to the following:



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**.

Add inbound security rule

hdf-workshop-nsg

Advanced

* Name
allow-all-ports

* Priority
1010

* Source
 Any CIDR block Tag

Service
Custom

* Protocol
 Any TCP UDP

* Port range
1-65535

* Action
 Deny Allow

- 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.

Actions	
	Connect
	Start
	Restart
	Stop
	Capture
	Move
	Delete
	Refresh

Resource group ([change](#))
zoharn-rg

Status
Running

Location
Canada East

Subscription ([change](#))
SE

Subscription ID
7d204bd6-841e-43fb-8638-c5eedf2ea797

Computer name
zoharn-lab

Operating system
Linux

Size
Standard DS3 v2 (4 vcpus, 14 GB memory)

Public IP address
52.235.42.127

Virtual network/subnet
zoharn-rg-vnet/default

DNS name
[Configure](#)

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

```
sudo su -
```

- 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)

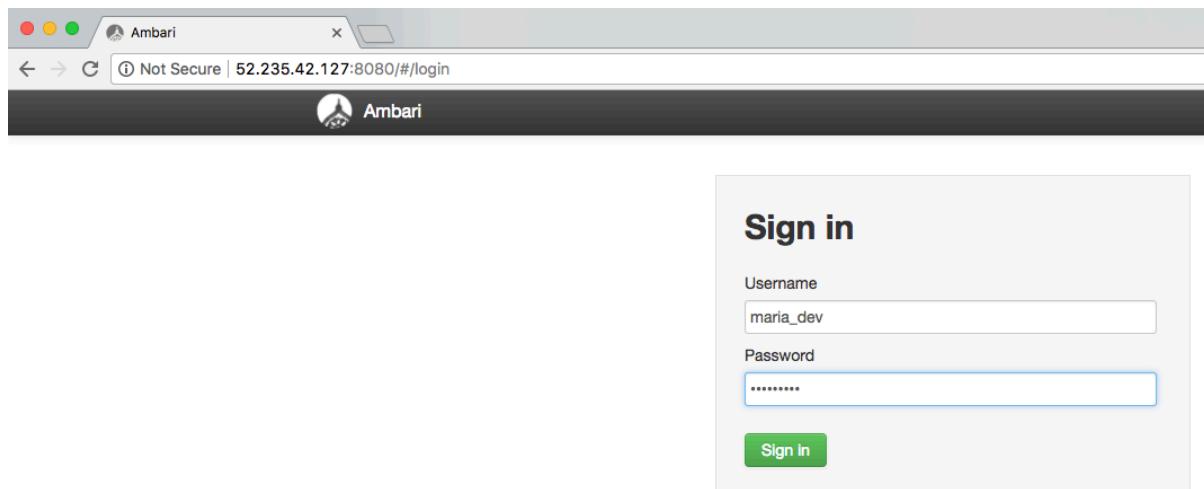
```
ambari-admin-password-reset
```

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:

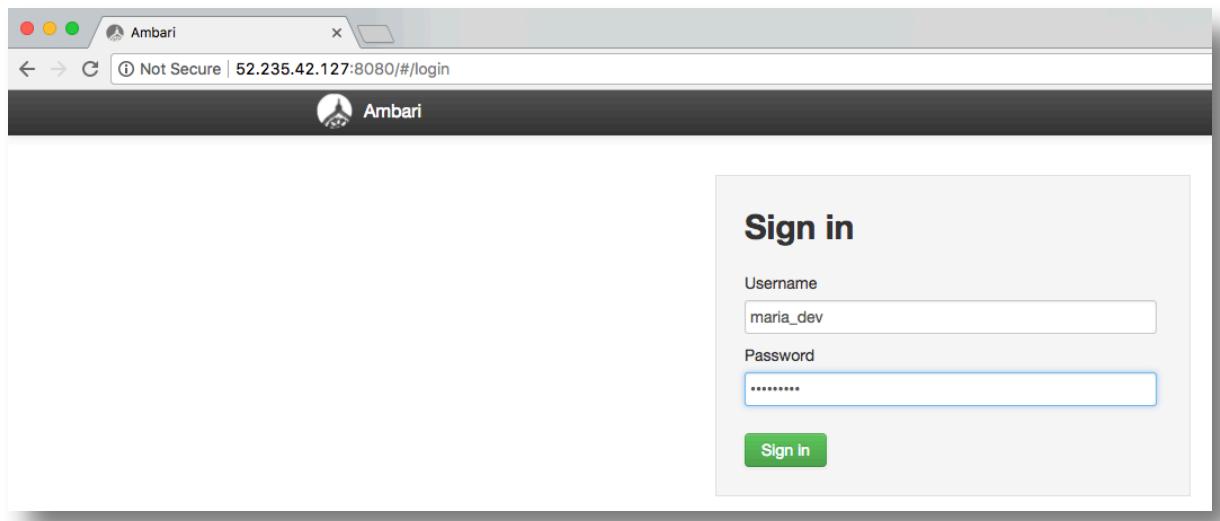
```
omniture-logs.tsv  
products.tsv  
users.tsv
```

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.

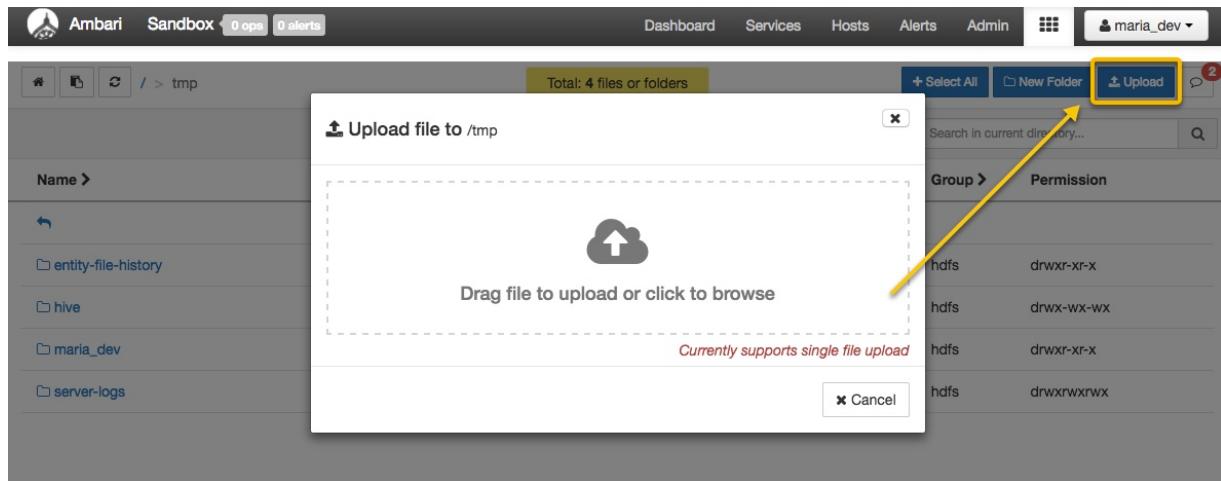
The screenshot shows the Ambari Dashboard with the Views menu open. The 'Files View' option is highlighted with a yellow box and a mouse cursor. Other options like 'Hive View', 'Hive View 2.0', etc., are also listed.

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

The screenshot shows the HDFS Files View. The current path is `/ > tmp`. A yellow arrow points to the `New Folder` button at the top right. The table below lists files and folders in the `tmp` directory.

Name >	Size >	Last Modified >	Owner >	Group >	Permission
entity-file-history	--	2017-04-19 11:48	hdfs	hdfs	drwxr-xr-x
hive	--	2017-04-26 11:21	ambari-qa	hdfs	drwx-wx-wx
server-logs	--	2017-04-26 13:51	hive	hdfs	drwxrwxrwx

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”.

Name	Size	Last Modified	Owner	Group	Permission
omniture-logs.tsv	63.6 MB	2017-09-19 13:49	maria_dev	hdfs	-rw-r--r--
products.tsv	1.5 kB	2017-09-19 13:53	maria_dev	hdfs	-rw-r--r--
users.tsv	1.8 MB	2017-09-19 13:54	maria_dev	hdfs	-rw-r--r--

This will open a “File Preview” window:

SWID	BIRTH_DT	GENDER_CD	
0001BD09-EABF-400D-81BD-09EABFC00D70	8-Apr-84	F	
00071AA7-86D2-4EB9-871A-A786D27EB9BA	7-Feb-88	F	
00071B7D-31AF-4085-871B-7D31AFFD852E	22-Oct-64	F	
0007967E-F18B-4598-9C7C-E64398482CFB	1-Jun-66	M	
000B90B2-92DC-4A7A-8B99-B292DC9A7A71	13-Jun-84	M	
000C1856-994E-476B-8C18-56994E676B29	29-Dec-80	U	
000F36E5-9891-409A-9B69-CEE78483B653	24-Mar-85	F	
00102F3F-061C-4212-9F91-1254F9D6E39F	1-Nov-91	F	
0010C6F2-B8C4-450E-90C6-F28C04B50E97	20-Jun-02	U	
0011C945-28C4-406F-B1E6-6CA7EFC14548	13-Nov-87	F	
001704E0-6CD8-429A-8E0A-89024019CA6A	10-Jul-91	M	
001720A4-44E3-43F0-BA8F-2F2E0D7B6275	8-Nov-90	M	
001834AA-7451-49EA-BE44-ED4A71C97AD1	21-Feb-88	M	
001883DA-4763-460A-B67E-1D5168E4DEB6	21-Apr-02	U	
00193DB8-1FEB-440B-B217-BA2AA5FD64	31-Jul-70	M	
001AFDA9-18D4-4FE8-B4F1-ADD932E0ACB2	16-Jan-81	U	
001BFE35-555B-48E1-9ED3-A4BE7677C36C	16-Feb-82	M	
001F01F3-11D8-470D-BF34-EA95941E525F	11-Dec-00	F	

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.

The screenshot shows the Ambari Metrics Dashboard. The left sidebar lists services: HDFS, YARN, MapReduce2, Tez, Hive, HBase, Pig, Sqoop, Oozie, ZooKeeper, Falcon, Storm, Flume, Ambari Infra, Ambari Metrics, Atlas, Kafka, Knox, Ranger, Spark, Spark2, Zeppelin, Notebook, and Slider. The main area displays metrics for HDFS Disk Usage (48%), DataNodes Live (1/1), HDFS Links (NameNode, Secondary NameNode, 1 DataNodes), Memory Usage (No Data Available), CPU Usage (No Data Available), Cluster Load (No Data Available), NameNode Heap (38%), NameNode RPC (0 ms), NameNode CPU WIO (n/a), NameNode Uptime (3.0 hr), HBase Master Heap (n/a), HBase Links (No Active Master, 1 RegionServers, n/a), HBase Ave Load (n/a), HBase Master Uptime (n/a), ResourceManager Heap (33%), ResourceManager Uptime (3.0 hr), NodeManagers Live (1/1), YARN Memory (0%), and YARN Links (ResourceManager, 1 NodeManagers). A dropdown menu on the right shows options: YARN Queue Manager, Files View, **Hive View** (selected), Pig View, Storm View, and Tez View.

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”.

The screenshot shows the Ambari Sandbox interface. The top navigation bar includes links for Dashboard, Services, Hosts, Alerts, Admin, and a user dropdown for maria_dev. Below the navigation is a menu bar with Hive, Query, Saved Queries, History, UDFs, and Upload Table. The main area is divided into two panes: Database Explorer on the left and Query Editor on the right. The Database Explorer pane shows the 'default' database selected, with a search bar and a list of databases: default, sample_07, sample_08, foodmart, and xademo. The Query Editor pane has a 'Worksheet' tab open, displaying three CREATE TABLE statements:

```

1 CREATE TABLE users (swid STRING, birth_dt STRING, gender_cd CHAR(1))
2 ROW FORMAT DELIMITED
3 FIELDS TERMINATED BY '\t'
4 STORED AS TEXTFILE
5 TBLPROPERTIES ("skip.header.line.count"="1");
6
7 CREATE TABLE products (url STRING, category STRING)
8 ROW FORMAT DELIMITED
9 FIELDS TERMINATED BY '\t'
10 STORED AS TEXTFILE
11 TBLPROPERTIES ("skip.header.line.count"="1");
12
13 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)
14 ROW FORMAT DELIMITED
15 FIELDS TERMINATED by '\t'
16 STORED AS TEXTFILE
17 TBLPROPERTIES ("skip.header.line.count"="1");

```

At the bottom of the Query Editor are buttons for Execute, Explain, Save as..., and New Worksheet.

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");
```

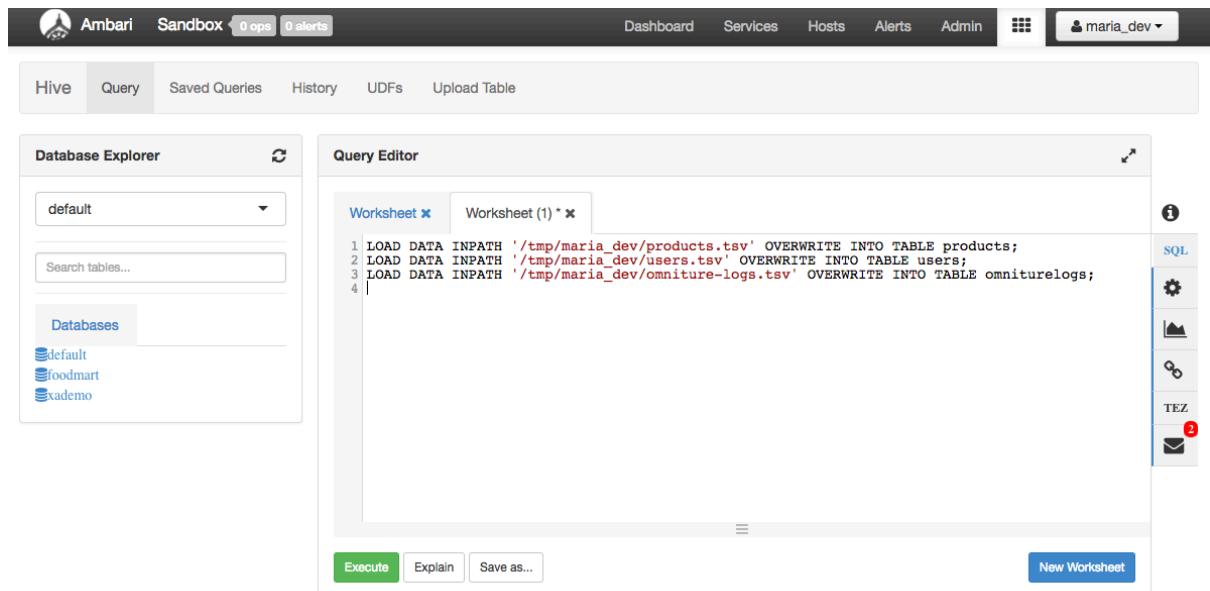
```
CREATE TABLE products (url STRING, category STRING)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY '\t'
STORED AS TEXTFILE
TBLPROPERTIES ("skip.header.line.count"="1");
```

```
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;
```



The screenshot shows the Ambari Hive Query Editor interface. At the top, there is a navigation bar with links for Dashboard, Services, Hosts, Alerts, Admin, and a user dropdown for 'maria_dev'. Below the navigation bar, there are tabs for Hive, Query, Saved Queries, History, UDFs, and Upload Table. The main area is divided into two panes: 'Database Explorer' on the left and 'Query Editor' on the right. In the Database Explorer, the 'default' database is selected, and the 'Databases' section lists 'default', 'foodmart', and 'sademo'. In the Query Editor, a 'Worksheet' tab is active, showing the following SQL code:

```
1 LOAD DATA INPATH '/tmp/maria_dev/products.tsv' OVERWRITE INTO TABLE products;
2 LOAD DATA INPATH '/tmp/maria_dev/users.tsv' OVERWRITE INTO TABLE users;
3 LOAD DATA INPATH '/tmp/maria_dev/omniture-logs.tsv' OVERWRITE INTO TABLE
omniturelogs;
4 |
```

Below the code, there are buttons for 'Execute', 'Explain', and 'Save as...', and a 'New Worksheet' button. On the right side of the editor, there is a vertical toolbar with icons for SQL, TEZ, and other tools, with a red notification badge showing '2'.

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.

The screenshot shows the Ambari Sandbox interface. At the top, there are tabs for 'Ambari', 'Sandbox' (which is selected), '0 ops', '0 alerts', 'Dashboard', 'Services', 'Hosts', 'Alerts', 'Admin', and a user dropdown 'maria_dev'. Below the tabs, there are links for 'Hive', 'Query', 'Saved Queries', 'History', 'UDFs', and 'Upload Table'. The main area has two panels: 'Database Explorer' on the left and 'Query Editor' on the right.

Database Explorer: Shows the 'default' database selected. A search bar says 'Search tables...'. Below it is a list of databases: default, omniturelogs, products, sample_07, sample_08, users, foodmart, and xademo. Each database has a small icon and a refresh button.

Query Editor: Shows a query named 'users sample' with the SQL command: 'SELECT * FROM users LIMIT 100;'. To the right of the query are several icons: a magnifying glass (SQL), a gear (⚙️), a line graph (📈), a key (🔑), and a red circle with '10' (TEZ). Below the query are buttons for 'Execute', 'Explain', and 'Save as...', and a link 'New Worksheet'.

Query Process Results: Status: SUCCEEDED. Shows a table with three columns: 'users.swid', 'users.birth_dt', and 'users.gender_cd'. The data is as follows:

users.swid	users.birth_dt	users.gender_cd
0001BDD9-EABF-4D0D-81BD-D9EABFC0D07D	8-Apr-84	F
00071AA7-86D2-4EB9-871A-A786D27EB9BA	7-Feb-88	F
00071B7D-31AF-4D85-871B-7D31AFFD852E	22-Oct-64	F

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
```

The screenshot shows the Ambari Hive Query Editor interface. At the top, there's a navigation bar with tabs for Dashboard, Services, Hosts, Alerts, Admin, and a user dropdown for 'maria_dev'. Below the navigation bar is a header with tabs for Hive, Query, Saved Queries, History, UDFs, and Upload Table. The main area is divided into two panes: 'Database Explorer' on the left and 'Query Editor' on the right. In the Database Explorer, the 'default' database is selected, and a search bar shows 'Search tables...'. A list of databases includes 'default', 'omniturelogs', 'products', 'sample_07', 'sample_08', 'users', 'foodmart', and 'xademo'. In the Query Editor, a query named 'omniture-view' is displayed in a code editor:

```

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

```

Below the code editor are three buttons: Execute, Explain, and Save as..., and a 'New Worksheet' button. To the right of the code editor is a vertical toolbar with icons for SQL, Settings, Help, TEZ, and Mail.

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 screenshot shows the Ambari Saved Queries page. At the top, there's a navigation bar with tabs for Dashboard, Services, Hosts, Alerts, Admin, and a user dropdown for 'maria_dev'. Below the navigation bar is a header with tabs for Hive, Query, Saved Queries, History, UDFs, and Upload Table. The main area displays a single query entry:

preview	title	database	owner	Clear filters
CREATE VIEW omniture AS SELECT col_2 ts, c	omniture-view	default	maria_dev	

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.

The screenshot shows the Ambari Database Explorer interface. At the top, there is a navigation bar with links for Dashboard, Services, Hosts, Alerts, Admin, and a user dropdown for 'maria_dev'. Below the navigation bar, there are tabs for Hive, Query, Saved Queries, History, UDFs, and Upload Table. The 'Query' tab is selected.

In the main area, there are two panels: 'Database Explorer' on the left and 'Query Editor' on the right.

Database Explorer: This panel shows a list of databases. The 'default' database is selected. Other databases listed include 'omniture', 'omniturelogs', 'products', 'sample_07', 'sample_08', 'users', 'foodmart', and 'xademo'. A search bar labeled 'Search tables...' is also present.

Query Editor: This panel contains a 'Query Editor' section where the following SQL query is entered:

```
1 SELECT * FROM omniture LIMIT 100;
```

Below the query editor, there are buttons for 'Execute', 'Explain', and 'Save as...'. To the right of the query editor is a sidebar with icons for 'SQL', 'TEZ' (with a red notification badge showing '14'), and other tools.

Query Process Results: This section displays the results of the executed query. The status is 'SUCCEEDED'. It includes tabs for 'Logs' and 'Results'. The results table has columns: 'omniture.ts', 'omniture.ip', 'omniture.url', 'omniture.swid', 'omniture.city', and 'or'. One row of data is shown:

omniture.ts	omniture.ip	omniture.url	omniture.swid	omniture.city	or
2012-03-15 01:34:46	69.76.12.213	http://www.acme.com/SH55126545/VD55177927	{8D0E437E-9249-4DDA-BC4F-C1E5409E3A3B}	coeur d alene	us

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**”:

The screenshot shows the Ambari Hive Query Editor interface. At the top, there are tabs for 'Hive', 'Query', 'Saved Queries', 'History', 'UDFs', and 'Upload Table'. The 'Query' tab is selected. In the center, there is a 'Database Explorer' panel on the left showing databases like 'default', 'omniture', 'omniturelogs', 'products', 'sample_07', 'sample_08', 'users', 'webloganalytics', 'foodmart', and 'xademo'. On the right, the 'Query Editor' panel contains a 'Worksheet (8)' tab with the SQL query provided in the code block. Below the query are buttons for 'Execute', 'Explain', and 'Save as...'. To the right of the editor is a sidebar with icons for 'SQL', 'TEZ', and a red notification badge showing '16'.

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.

The screenshot shows the Ambari Hive View interface. At the top, there are tabs for Hive, Query, Saved Queries, History, UDFs, and Upload Table. The Database Explorer on the left lists databases: default, omniture, omniturelogs, products, sample_07, sample_08, users, webloganalytics, foodmart, and xademo. The Query Editor in the center contains a query window with tabs for omniture-view, Worksheet (8), and webloganalytics sample. The SQL tab is selected, showing the query: `SELECT * FROM webloganalytics LIMIT 100;`. Below the query are buttons for Execute, Explain, Save as..., and New Worksheet. To the right of the editor is a sidebar with icons for Help, SQL, Configuration, Metrics, and Tez, with a red notification badge for Tez. The Query Process Results section at the bottom shows a table with columns: webloganalytics.logdate, webloganalytics.url, webloganalytics.ip, and webloganalytics.city. The results are as follows:

logdate	url	ip	city
2012-03-15	http://www.acme.com/SH55126545/VD55177927	69.76.12.213	coeur d alene
2012-03-15	http://www.acme.com/SH55126545/VD55166807	67.240.15.94	queensbury
2012-03-15	http://www.acme.com/SH55126545/VD55149415	67.240.15.94	queensbury
2012-03-15	http://www.acme.com/SH55126545/VD55179433	98.234.107.75	sunnyvale
2012-03-15	http://www.acme.com/SH55126545/VD55170123	75.85.185.38	san diego

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

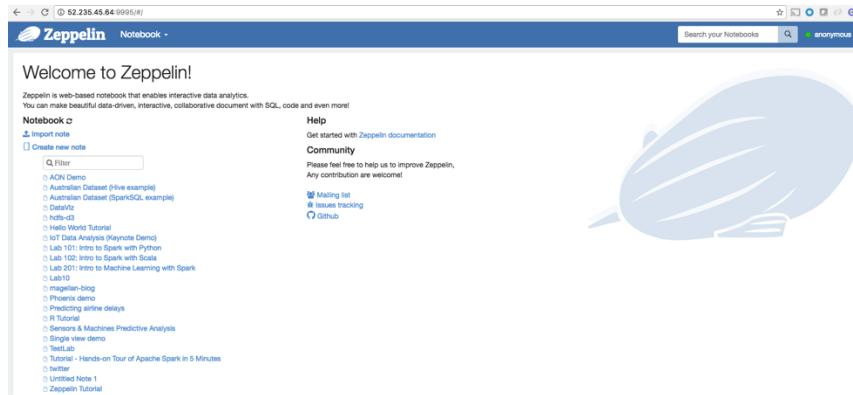
The HDP Sandbox includes the core Hadoop components, as well as all the tools needed for data ingestion and processing. You are able to access and analyze data in the sandbox using any number of Business Intelligence (BI) applications.

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/>



Welcome to Zeppelin!

Zeppelin is web-based notebook that enables interactive data analytics. You can make beautiful data-driven, interactive, collaborative document with SQL, code and even more!

Notebook   

Filter

- AON Demo
- Australian Dataset (Hive example)
- Australian Dataset (SparkSQL example)
- DataViz
- HDFS-d3
- Hello World Tutorial
- Machine Analysis (Keynote Demo)
- Lab 101: Intro to Spark with Python
- Lab 102: Intro to Spark with Scala
- Lab 201: Intro to Machine Learning with Spark
- Lab 202: MLlib API
- magellan-bug
- Phoenix demo
- Sensors & Machines
- Sensors & Machines Predictive Analysis
- Single view demo
- Twitter Stream Processing
- Tutorial - Hands-on Tour of Apache Spark in 5 Minutes
- Twitter
- Untitled Note 1
- Zeppelin Tutorial

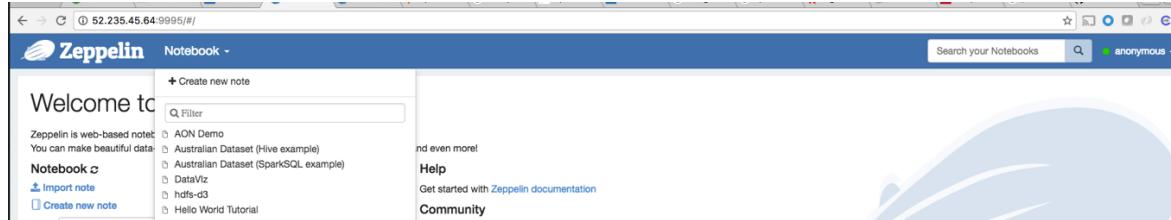
Help  Get started with [Zeppelin documentation](#)

Community  Please feel free to help us to improve Zeppelin. Any contribution are welcome!

[Mailing list](#) [Issues tracking](#) [GitHub](#)

Search your Notebooks anonymous

Create a new note named Introduction to Zeppelin



Welcome to Zeppelin!

Zeppelin is web-based notebook that enables interactive data analytics. You can make beautiful data-driven, interactive, collaborative document with SQL, code and even more!

Notebook   

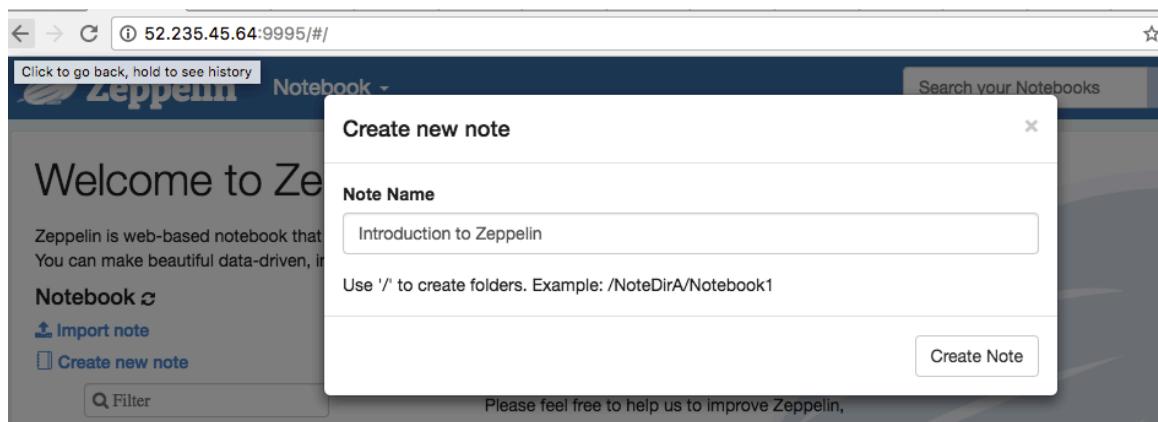
Filter

- AON Demo
- Australian Dataset (Hive example)
- Australian Dataset (SparkSQL example)
- DataViz
- HDFS-d3
- Hello World Tutorial

Help  Get started with [Zeppelin documentation](#)

Community  Please feel free to help us to improve Zeppelin. Any contribution are welcome!

Search your Notebooks anonymous



Welcome to Zeppelin!

Zeppelin is web-based notebook that enables interactive data analytics. You can make beautiful data-driven, interactive, collaborative document with SQL, code and even more!

Notebook   

Filter

Create new note

Note Name

Introduction to Zeppelin

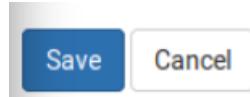
Use '/' to create folders. Example: /NoteDirA/Notebook1

Please feel free to help us to improve 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.

The screenshot shows the Zeppelin Interpreter Binding dialog box. At the top, there are three icons: a question mark, a gear (highlighted with a red circle), and a lock. To the right of the gear icon is a dropdown menu labeled "default". Below the dialog is the Zeppelin interface. The title bar says "Zeppelin Notebook". The main area shows a list of interpreters: spark %spark (default), %pyspark, %sql, %dep; md %md; angular %angular; sh %sh; jdbc %jdbc; and ivy %ivy, %ivy.pyspark, %ivy.sparkr, %ivy.sql. The "spark" entry is highlighted with a blue background. At the bottom of the dialog are "Save" and "Cancel" buttons.



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")
```



```
sc.version  
sc.getConf.get("spark.home")
```

The screenshot shows a Zeppelin notebook cell containing the same code as above. To the right of the cell, there is a status bar with the word "READY" followed by a blue play button icon. A red circle highlights the play button icon.

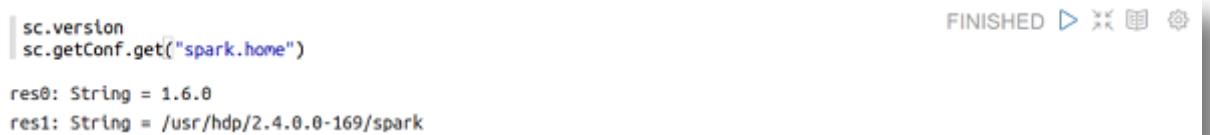
While processing, Zeppelin will display a status of RUNNING. It will also display a Pause icon should it become necessary.



```
sc.version  
sc.getConf.get("spark.home")
```

The screenshot shows the same notebook cell and status bar as the previous one, but the status bar now says "RUNNING 0%" instead of "READY". The play button icon has changed to a pause button icon.

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



```
sc.version  
sc.getConf.get("spark.home")
```

The screenshot shows the same notebook cell and status bar as the previous ones, but the status bar now says "FINISHED 0%" instead of "RUNNING". The play/pause button icon has changed to a refresh/circular arrow icon.

```
res0: String = 1.6.0  
res1: String = /usr/hdp/2.4.0.0-169/spark
```

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
```

%sql
show tables



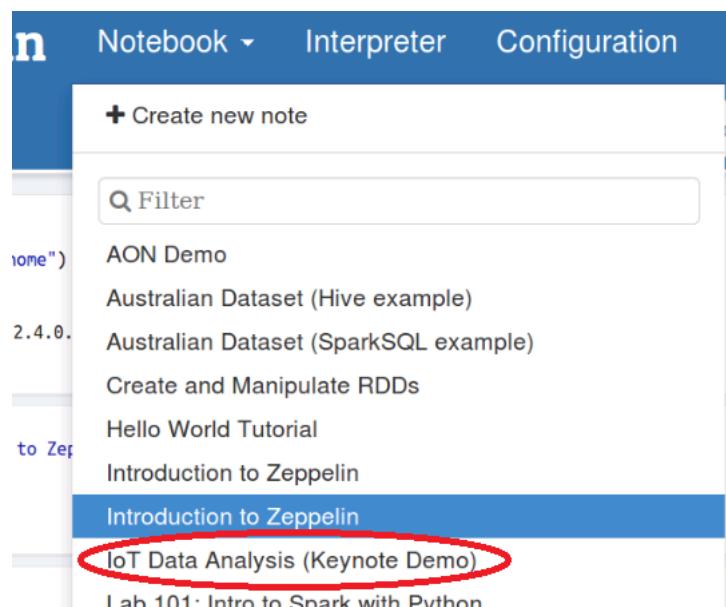
tableName	isTemporary
-----------	-------------

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 screenshot shows the Zeppelin interface with a notebook titled "IoT Data Analysis (Keynote Demo)". The first cell contains the following text:

Using Zeppelin for Data Science Tasks: Data Ingestion, Data Formatting, Exploratory Analysis and Model Building.

Data Science involves a typical sequence of tasks: acquiring data, cleaning it, analyzing it for relationships, and then building a model. Zeppelin allows you to do all these from one unified interface.

It took 2 seconds.

The second cell contains the following text:

First, let's load the data into HDFS and make sure we can access it.

It took 0 seconds.

```
%sh
whoami
curl -sSL -O "https://www.dropbox.com/s/ggj1robwxpl9vrt/iotdemo-notebook-data.zip"
unzip iotdemo-notebook-data.zip
hadoop fs -mkdir -p /user/zeppelin/iotdemo
hadoop fs -copyFromLocal -f trainingData /user/zeppelin/iotdemo/
hadoop fs -copyFromLocal -f enrichedEvents /user/zeppelin/iotdemo/
hadoop fs -ls /user/zeppelin/iotdemo/
zeppelin
Archive: iotdemo-notebook-data.zip
```

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.

```
%sh
whoami
curl -sSL -O "https://www.dropbox.com/s/ggj1robwxpl9vrt/iotdemo-notebook-data.zip"
unzip iotdemo-notebook-data.zip
hadoop fs -mkdir -p /user/zeppelin/iotdemo
hadoop fs -copyFromLocal -f trainingData /user/zeppelin/iotdemo/
hadoop fs -copyFromLocal -f enrichedEvents /user/zeppelin/iotdemo/
hadoop fs -ls /user/zeppelin/iotdemo/
```

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

```
hadoop fs -ls /user/zeppelin/iotdemo/
zeppelin
Archive: iotdemo-notebook-data.zip
Found 2 items
-rw-r--r-- 3 zeppelin zeppelin 63570 2016-05-27 16:50 /user/zeppelin/iotdemo/enrichedEvents
-rw-r--r-- 3 zeppelin zeppelin 33084 2016-05-27 16:50 /user/zeppelin/iotdemo/trainingData
```

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  
Normal,"Y","hours",45,2773,-90.68,35.12,0,0,0
```

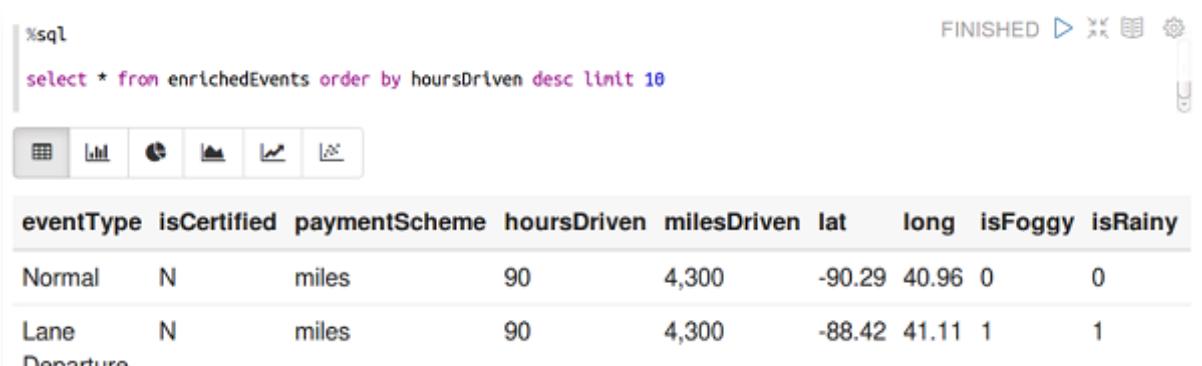
The next section of the notebook performs actions necessary to import and use this data with Spark SQL. You may note that the status to the left of the Play icon is shown as ERROR. This is due to the fact that the file being manipulated did not exist at the time the notebook was opened on this system.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

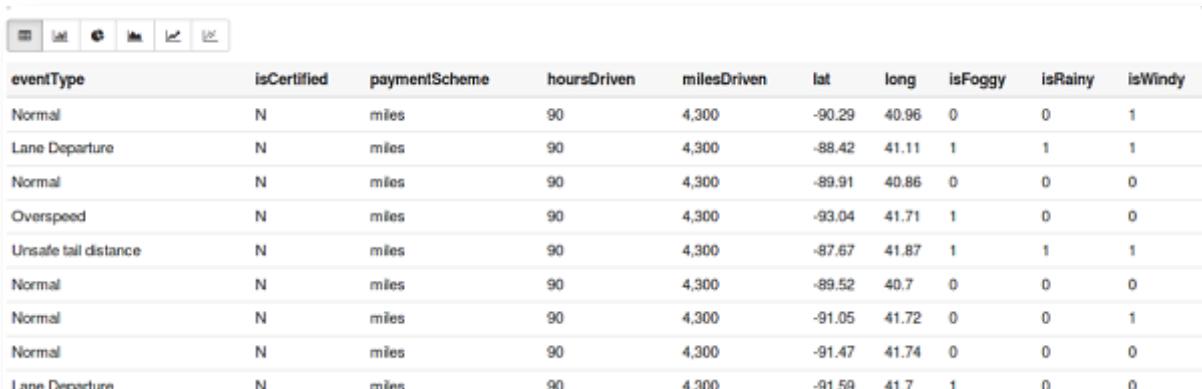
The next block of code utilizes Spark SQL to view this data. Run this code and examine the output.



A screenshot of a Spark SQL interface. At the top, it says "sql" and "FINISHED". Below that is a query: "select * from enrichedEvents order by hoursDriven desc limit 10". Underneath the query is a table with the following data:

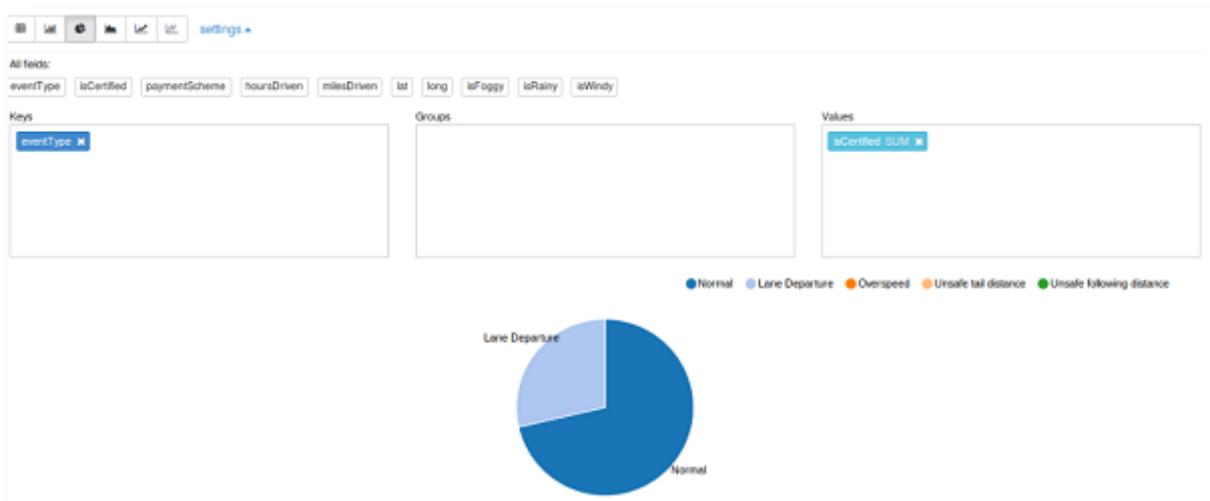
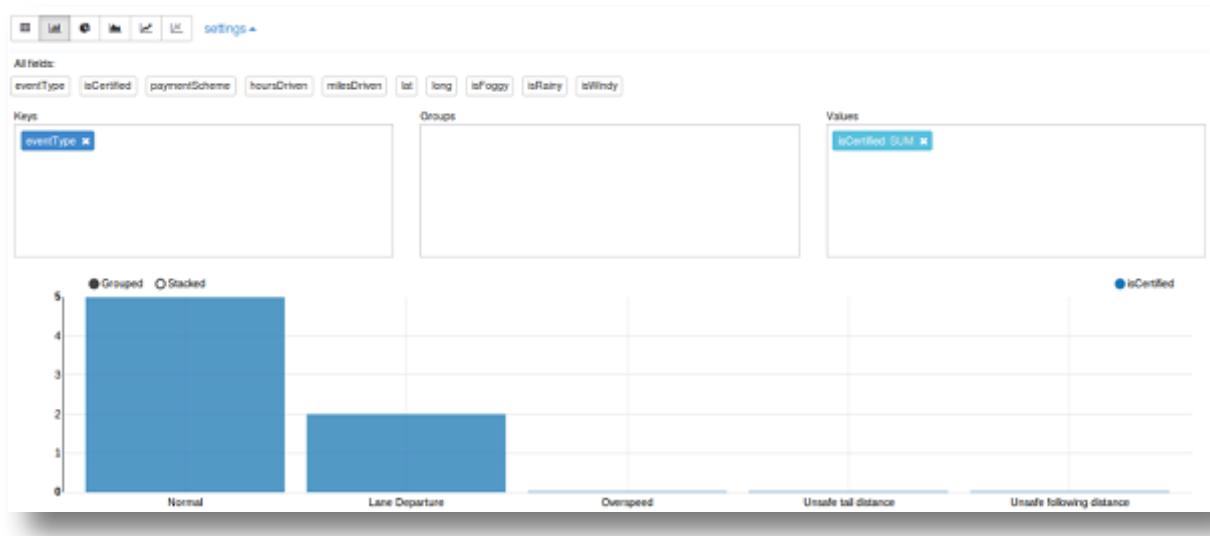
eventType	isCertified	paymentScheme	hoursDriven	milesDriven	lat	long	isFoggy	isRainy
Normal	N	miles	90	4,300	-90.29	40.96	0	0
Lane Departure	N	miles	90	4,300	-88.42	41.11	1	1

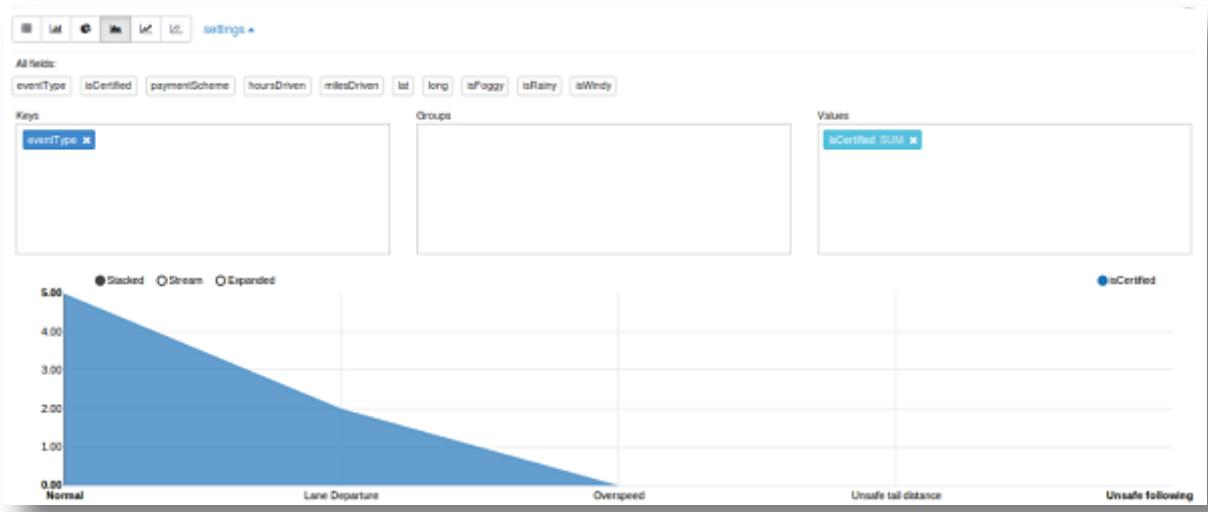
Note that at the top of the results there are six buttons that allow you to display the results using six different visualizations. Click on each one to view the differences between them.

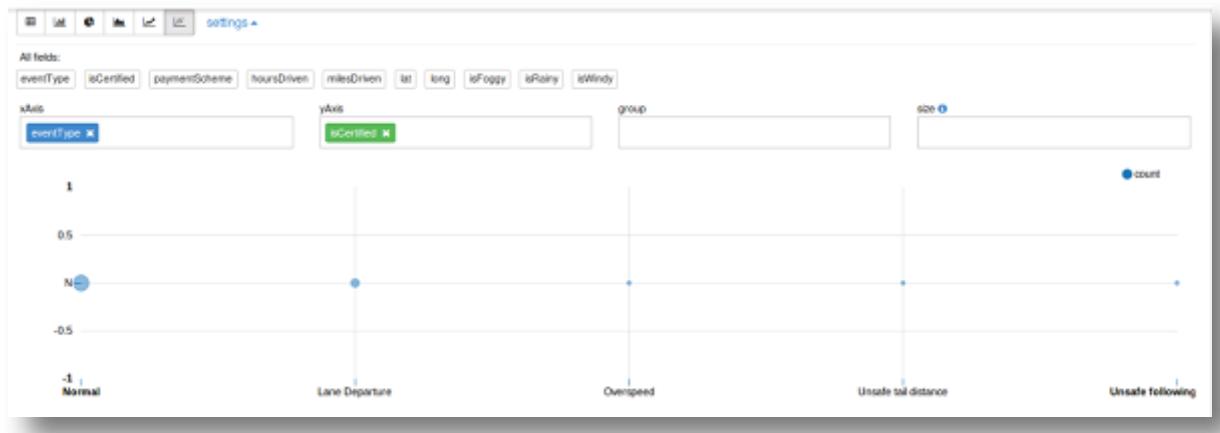


A screenshot of a Spark SQL interface showing the same table of enriched event data. At the top, there are six small icons representing different visualization options: grid, list, bar chart, line chart, scatter plot, and map. The table data is identical to the previous screenshot.

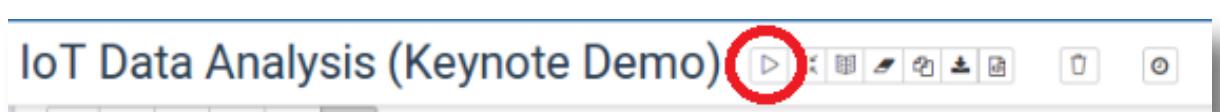
eventType	isCertified	paymentScheme	hoursDriven	milesDriven	lat	long	isFoggy	isRainy	isWindy
Normal	N	miles	90	4,300	-90.29	40.96	0	0	1
Lane Departure	N	miles	90	4,300	-88.42	41.11	1	1	1
Normal	N	miles	90	4,300	-89.91	40.86	0	0	0
Overspeed	N	miles	90	4,300	-93.04	41.71	1	0	0
Unsafe tail distance	N	miles	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







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

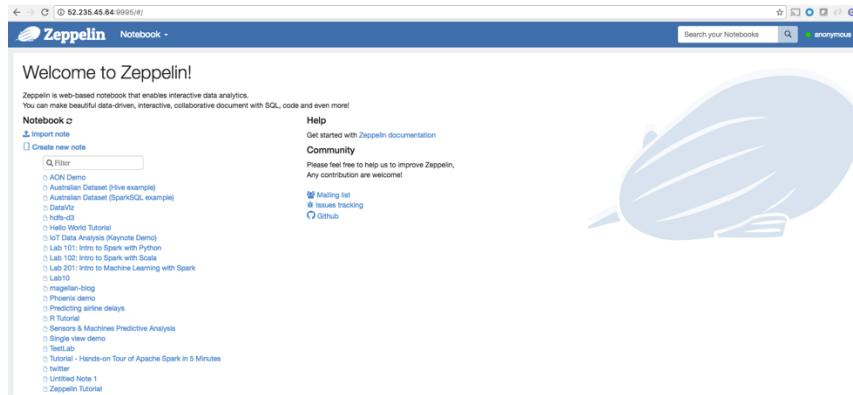
In this lab, we are going to learn to how to use Zeppelin to perform data visualizations, collaborate, and integrate visualization into reports.

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/>



Welcome to Zeppelin!

Zeppelin is web-based notebook that enables interactive data analytics. You can make beautiful data-driven, interactive, collaborative document with SQL, code and even more!

Notebook

Import note

Create new note

Q Filter

- AON Demo
- Australian Dataset (Hive example)
- Australian Dataset (SparkSQL example)
- DataViz
- hdf5-d3
- Hello World Tutorial
- Home Analytics (Keynote Demo)
- Lab 101: Intro to Spark with Python
- Lab 102: Intro to Spark with Scala
- Lab 201: Intro to Machine Learning with Spark
- Lab 202: MLlib API
- magellan-bug
- Phoenix demo
- R tutorial
- Sensors & Machines Predictive Analysis
- Single view demo
- Twitter Stream
- Tutorial - Hands-on Tour of Apache Spark in 5 Minutes
- twitter
- Untitled Note 1
- Zeppelin Tutorial

Help

Get started with [Zeppelin documentation](#)

Community

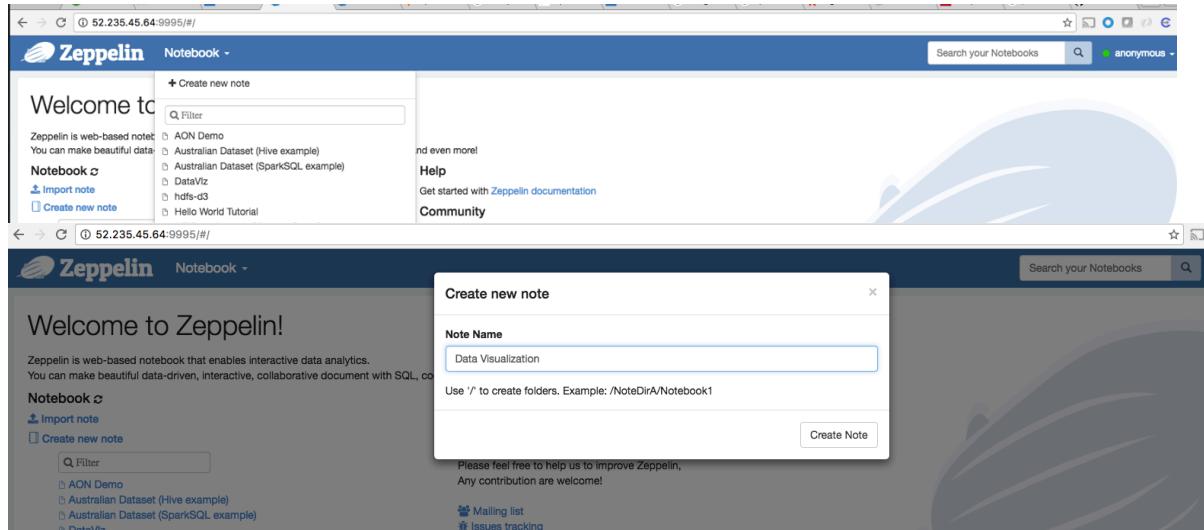
Please feel free to help us to improve Zeppelin. Any contribution are welcome!

[Mailing list](#)

[Issues tracking](#)

[Github](#)

Create a new note named Data Visualization



Welcome to Zeppelin!

Zeppelin is web-based notebook that enables interactive data analytics. You can make beautiful data-driven, interactive, collaborative document with SQL, code and even more!

Notebook

Import note

Create new note

Q Filter

- AON Demo
- Australian Dataset (Hive example)
- Australian Dataset (SparkSQL example)
- DataViz
- hdf5-d3
- Hello World Tutorial

Help

Get started with [Zeppelin documentation](#)

Community

Please feel free to help us to improve Zeppelin. Any contribution are welcome!

Use '/' to create folders. Example: /NoteDirA/Notebook1

Create new note

Note Name

Data Visualization

Create Note

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
```

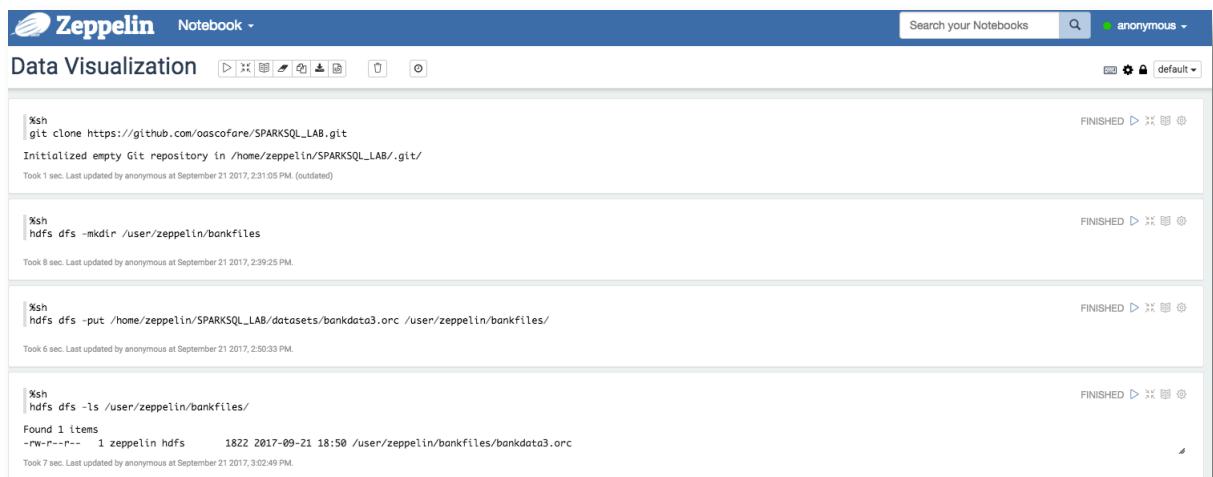
Upload the sample data into HDFS :

```
%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:



The screenshot shows a Zeppelin Notebook interface with a blue header bar. The main area contains four completed shell command cells:

- Cell 1:** %sh
git clone https://github.com/oascofare/SPARKSQL_LAB.git
Initialized empty Git repository in /home/zeppelin/SPARKSQL_LAB/.git/
Took 1 sec. Last updated by anonymous at September 21 2017, 2:31:05 PM (outdated)
- Cell 2:** %sh
hdfs dfs -mkdir /user/zeppelin/bankfiles
Took 8 sec. Last updated by anonymous at September 21 2017, 2:39:25 PM.
- Cell 3:** %sh
hdfs dfs -put /home/zeppelin/SPARKSQL_LAB/datasets/bankdata3.orc /user/zeppelin/bankfiles/
Took 6 sec. Last updated by anonymous at September 21 2017, 2:50:33 PM.
- Cell 4:** %sh
hdfs dfs -ls /user/zeppelin/bankfiles/
Found 1 items
-rw-r--r-- 1 zeppelin hdfs 1622 2017-09-21 18:50 /user/zeppelin/bankfiles/bankdata3.orc
Took 7 sec. Last updated by anonymous at September 21 2017, 3:02:49 PM.

The status of each cell is "FINISHED".

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")
```

```
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
```

```
%sql  
show tables|
```



tableName	isTemporary
health_table	true
bankdataperm	false
table1hive	false

5. ANALYSE YOUR DATA

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

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

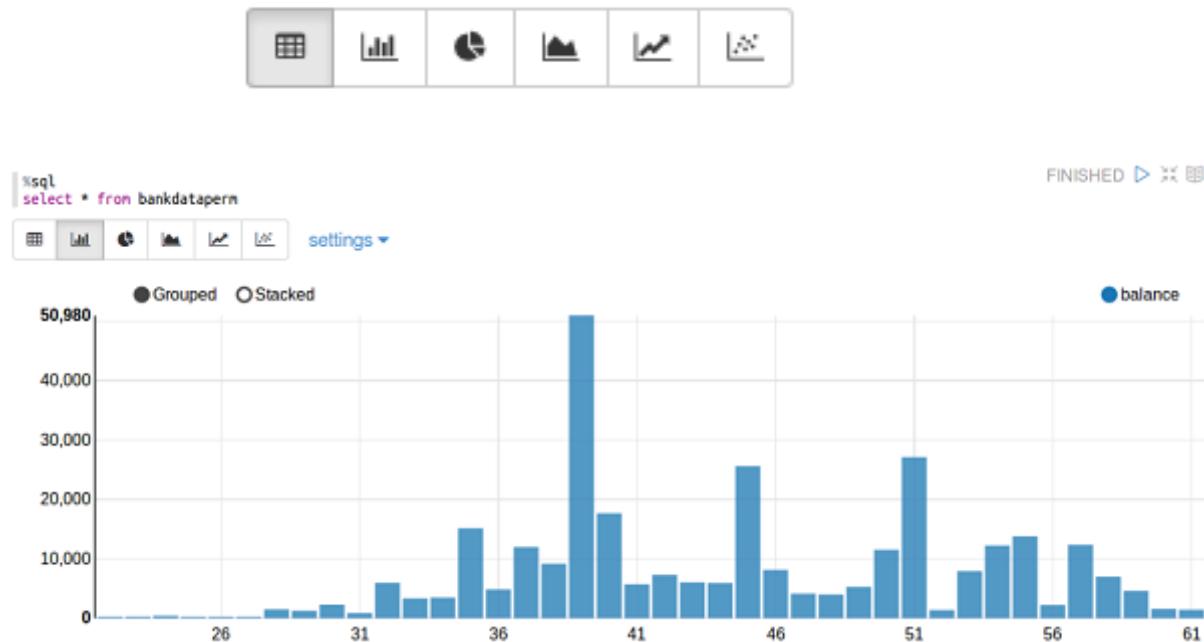
SQL

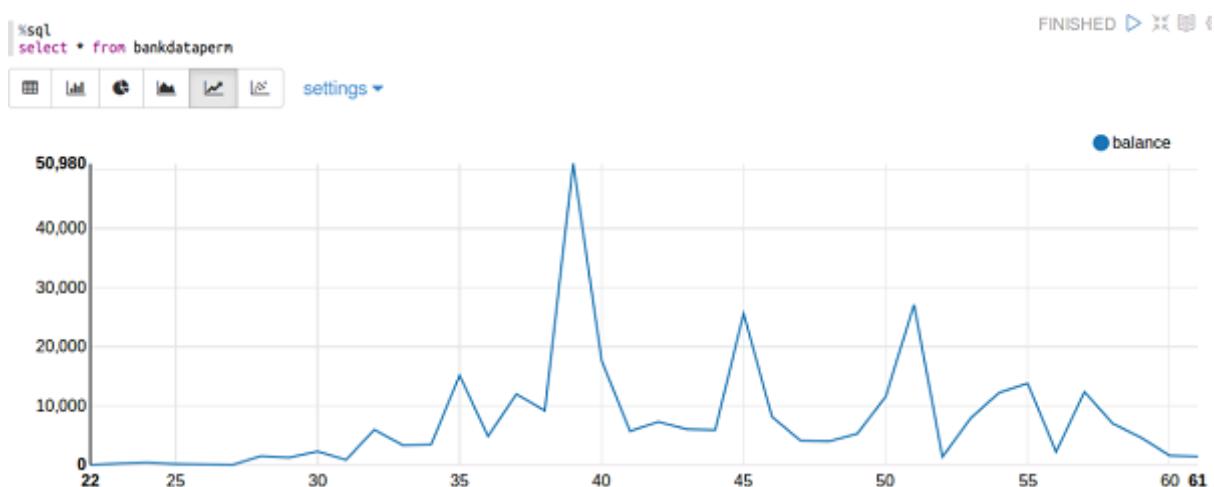
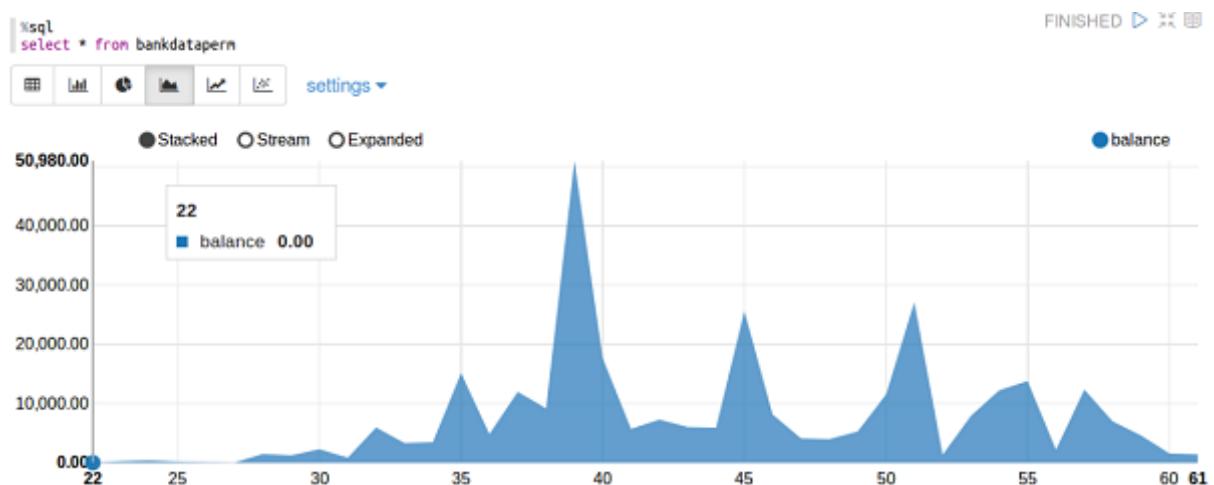
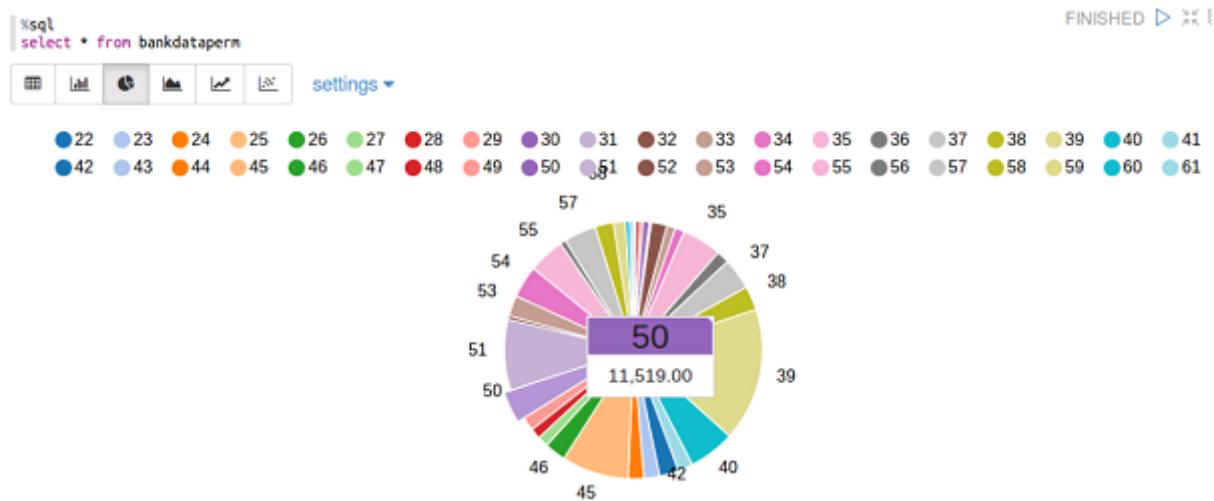
```
%sql  
select * from bankdataperm
```

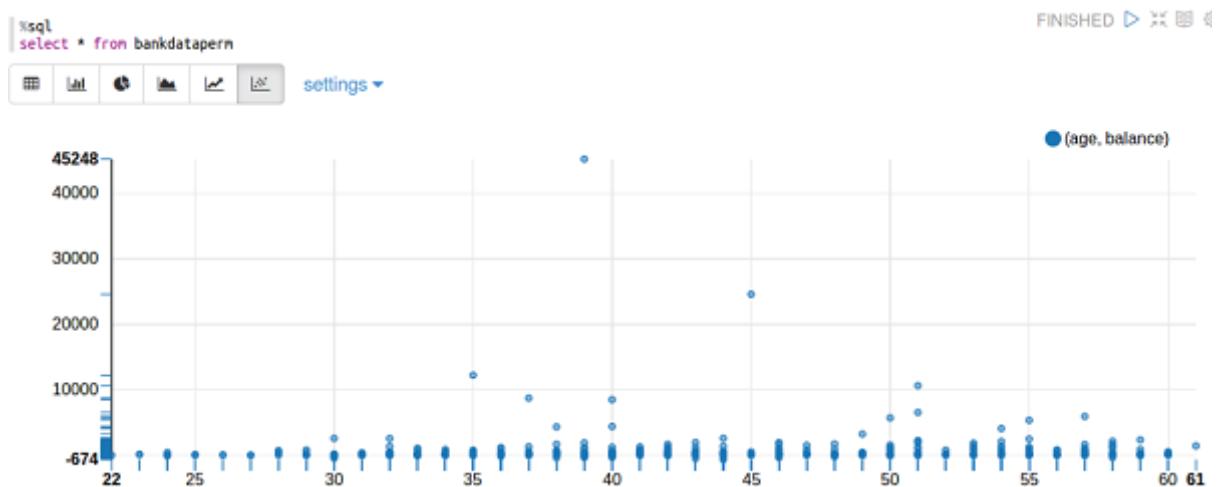
grid chart pie line scatter

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.

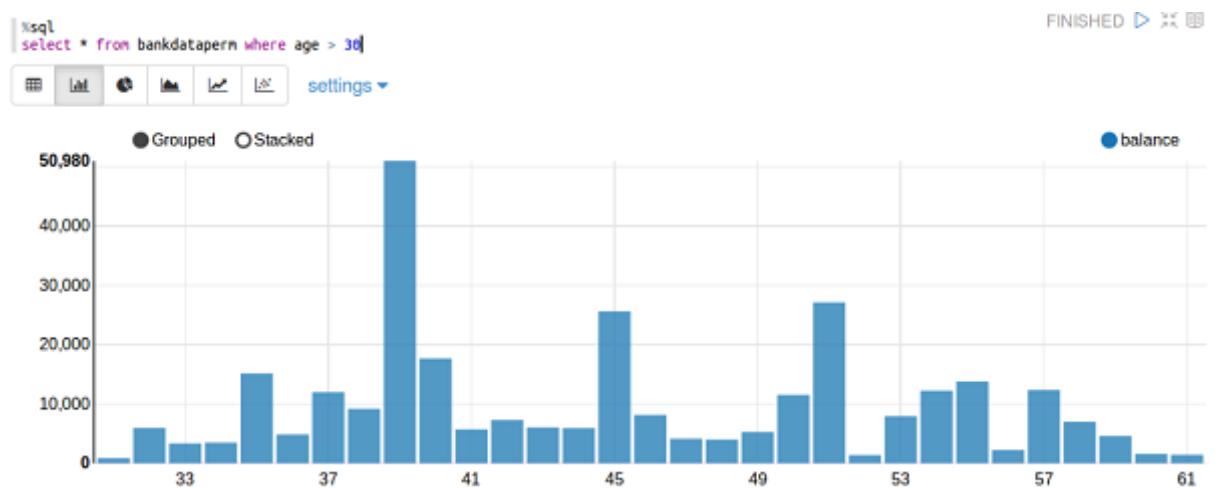




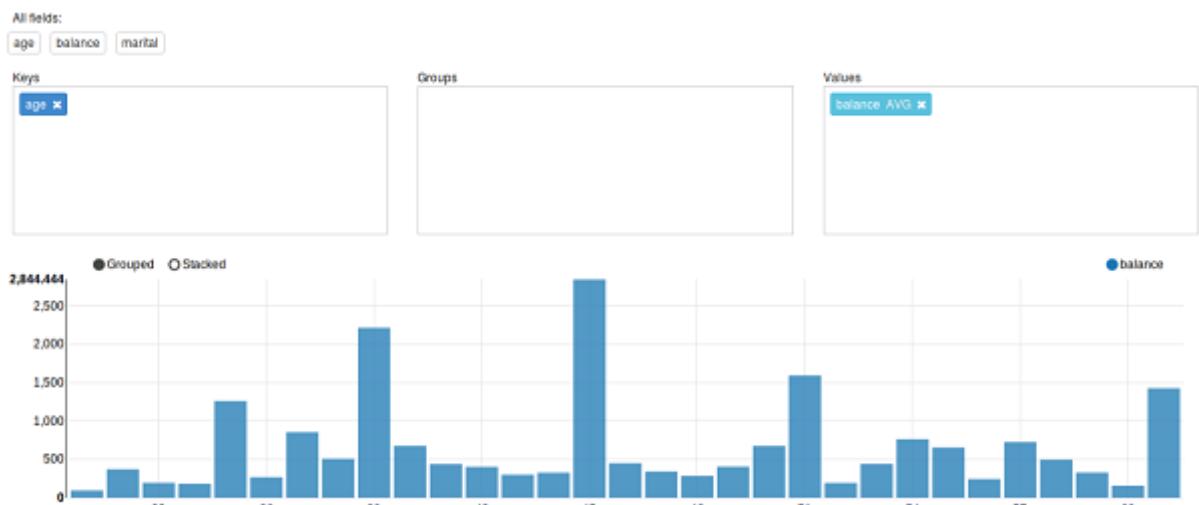
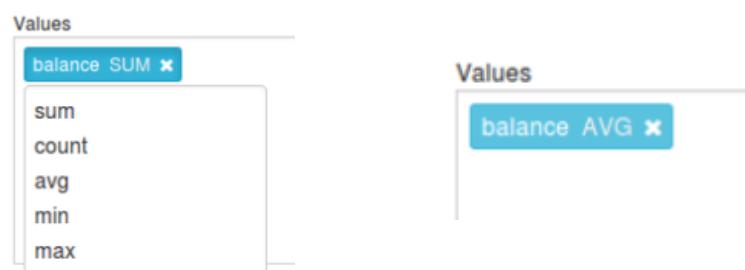
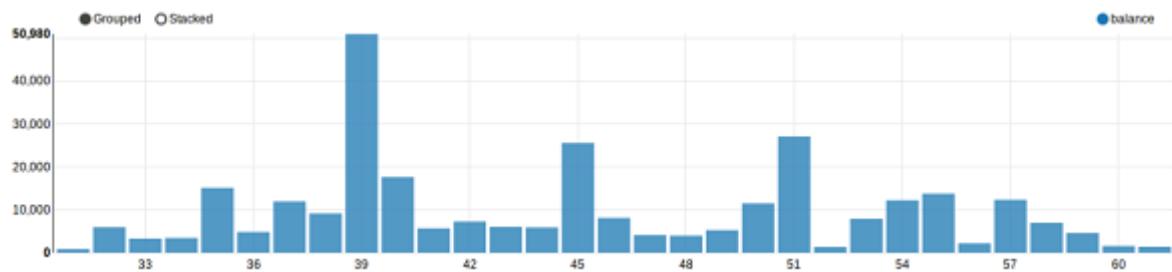
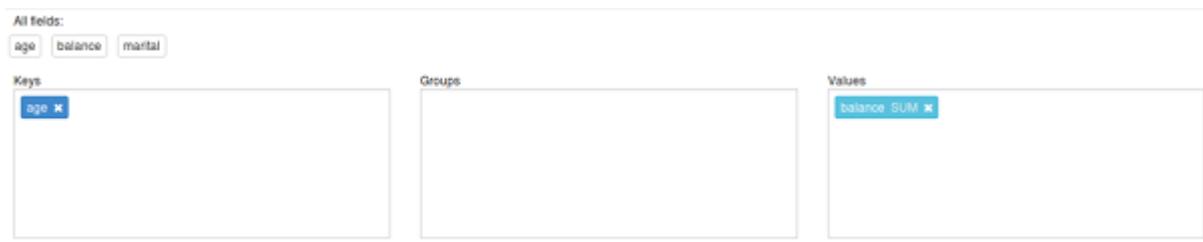


Now, go back to the bar chart view. Then, edit your SQL query so that it only shows data for individuals over the age of 30. Run the query and note the change in the chart.

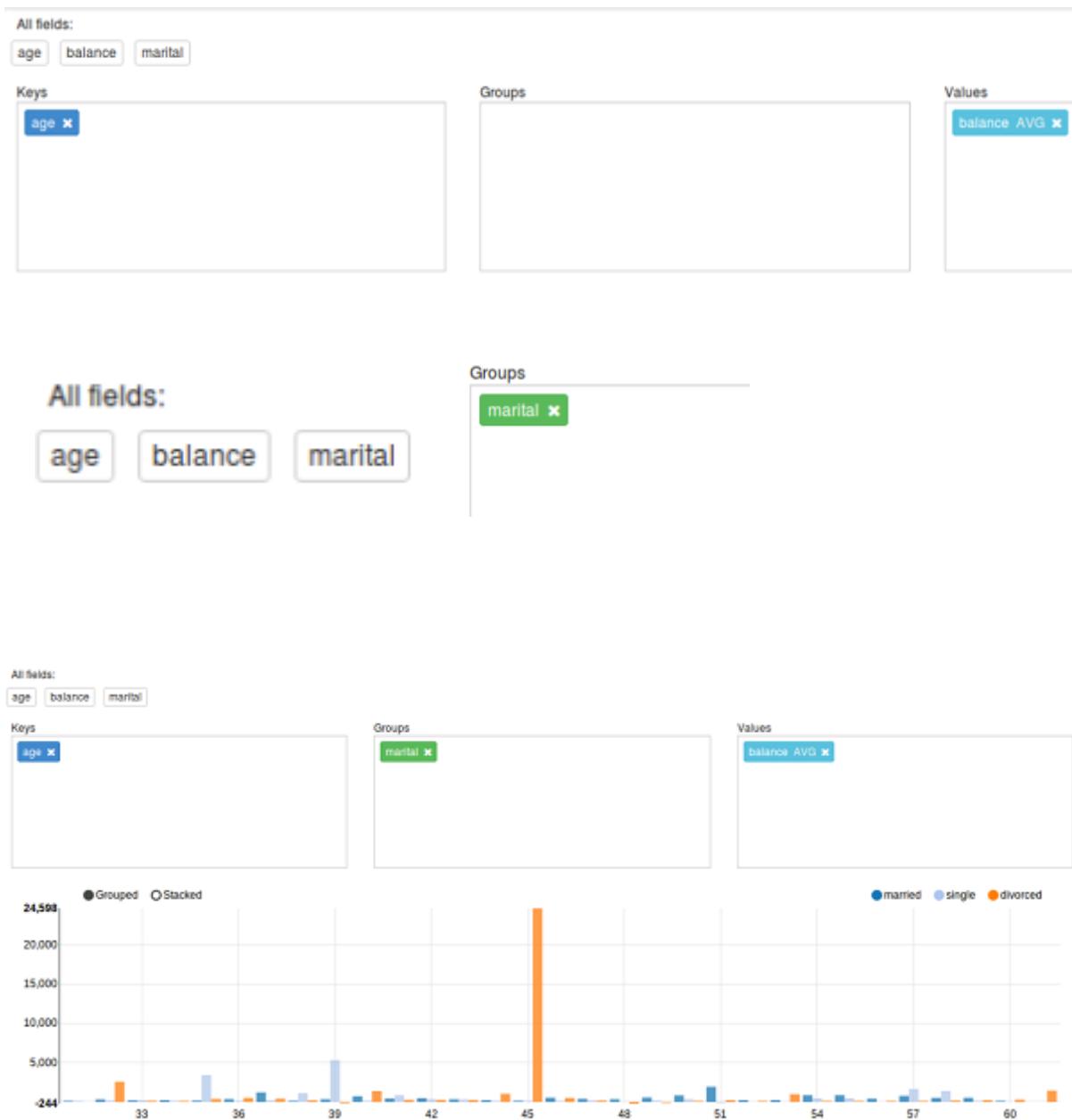
```
%sql
select * from bankdataperm where age > 30
```



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.



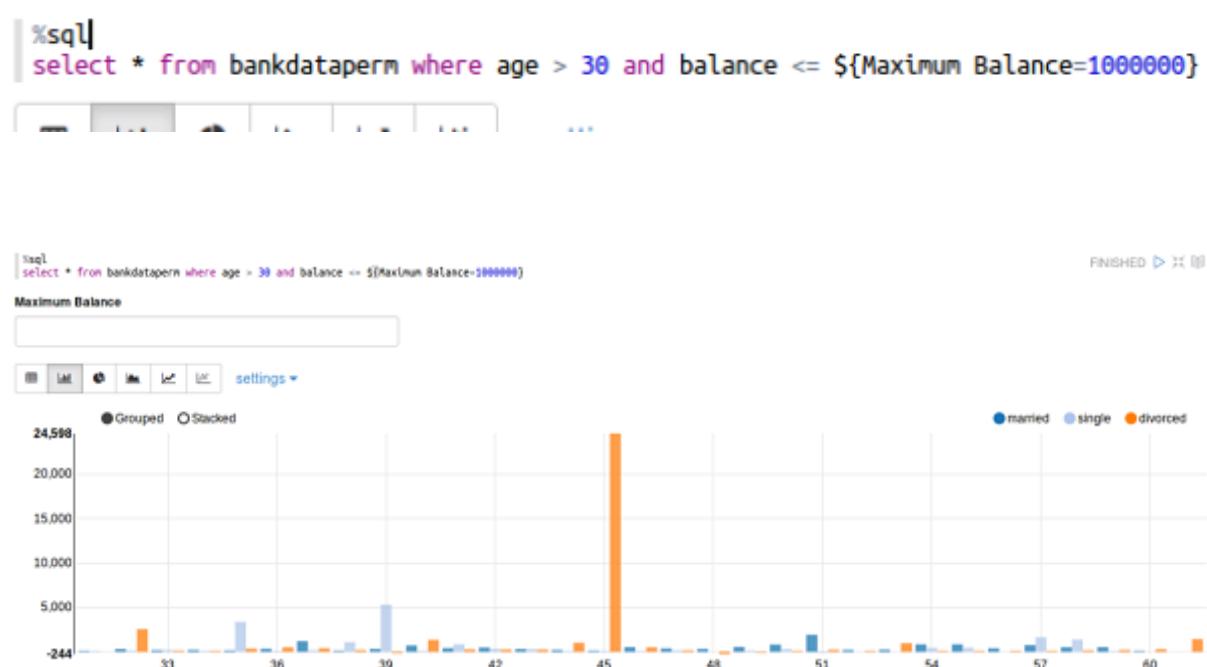
Click and drag the available **marital** field into the **Groups** category to modify the visualization so that data is shown not only by age, but also grouped by **marital status**. When you are finished, click the settings link again to close the pivot chart options.

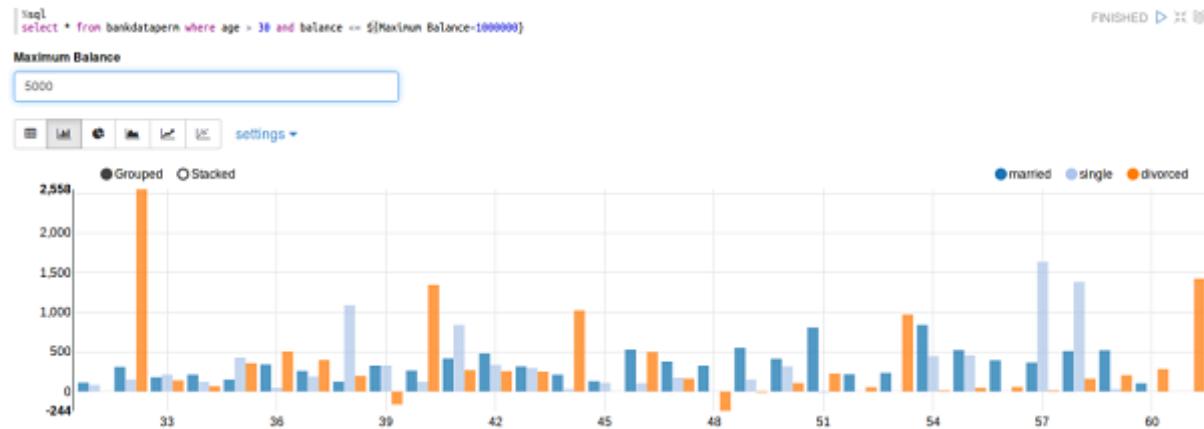
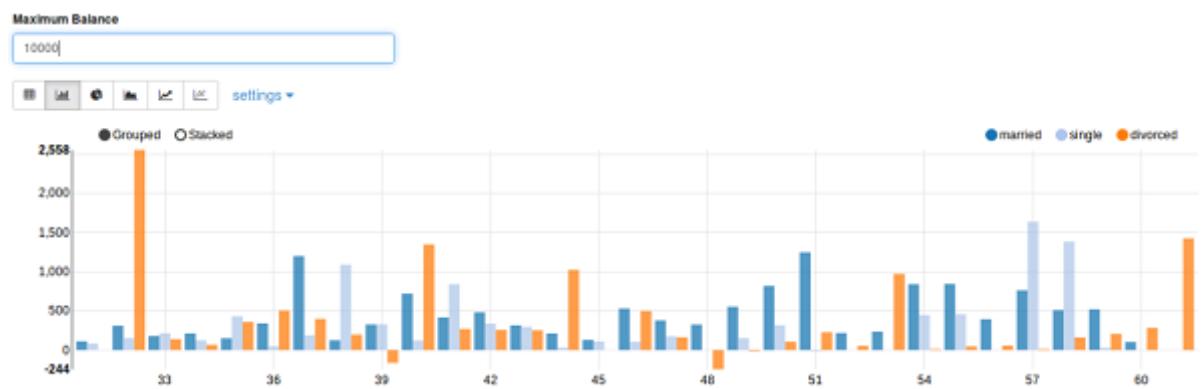


6. USE DYNAMIC VARIABLES IN YOUR ANALYSIS

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}
```





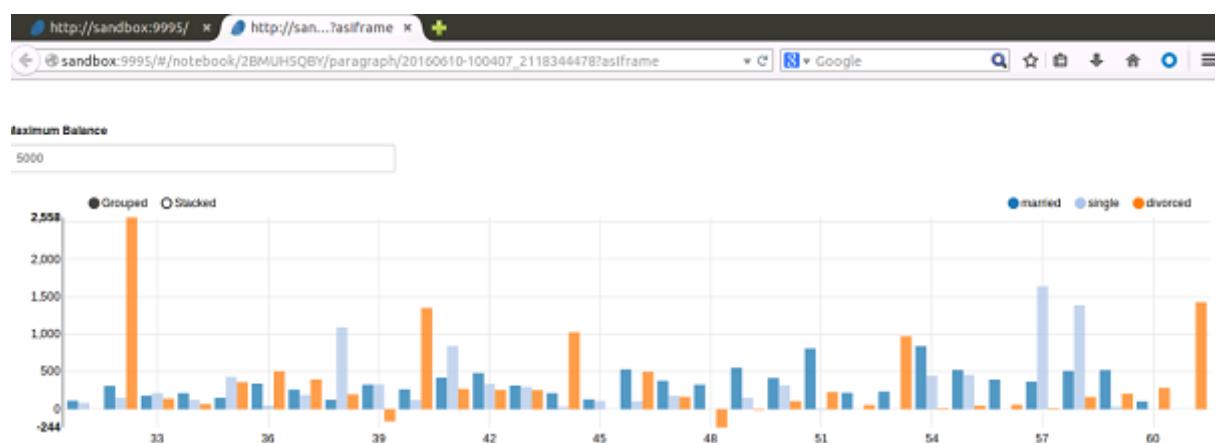
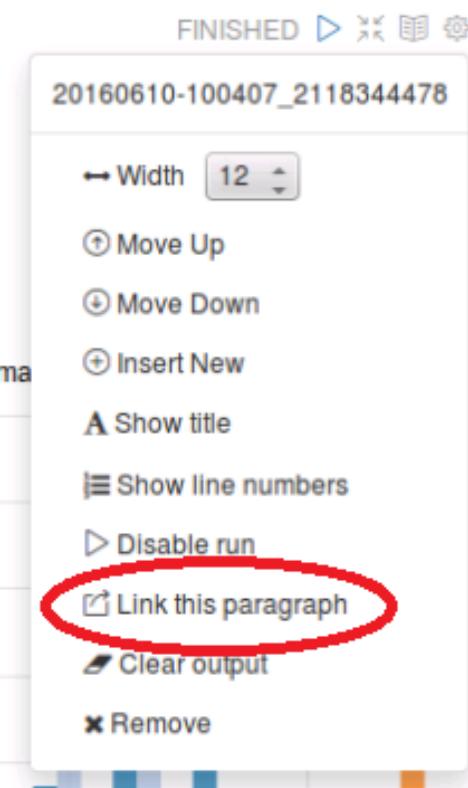
QUESTIONS:

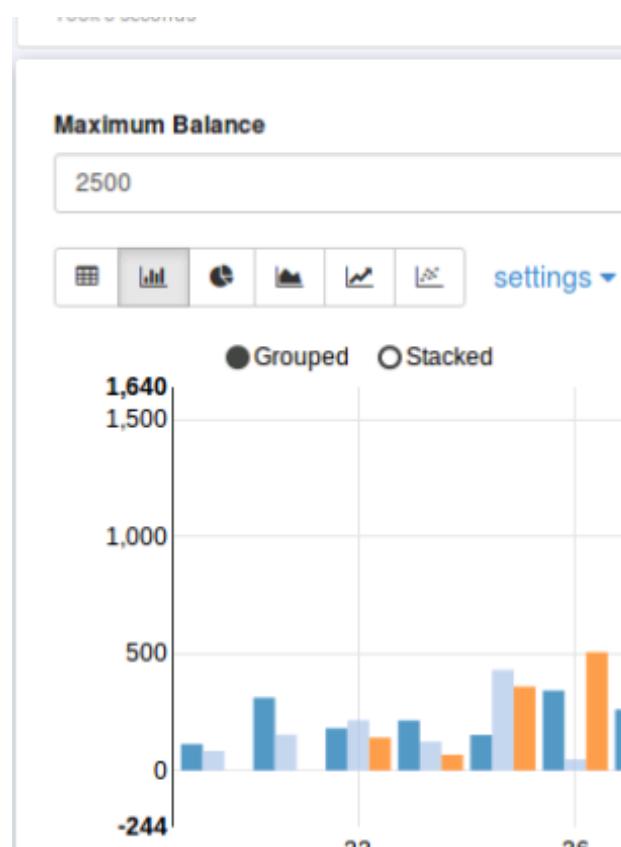
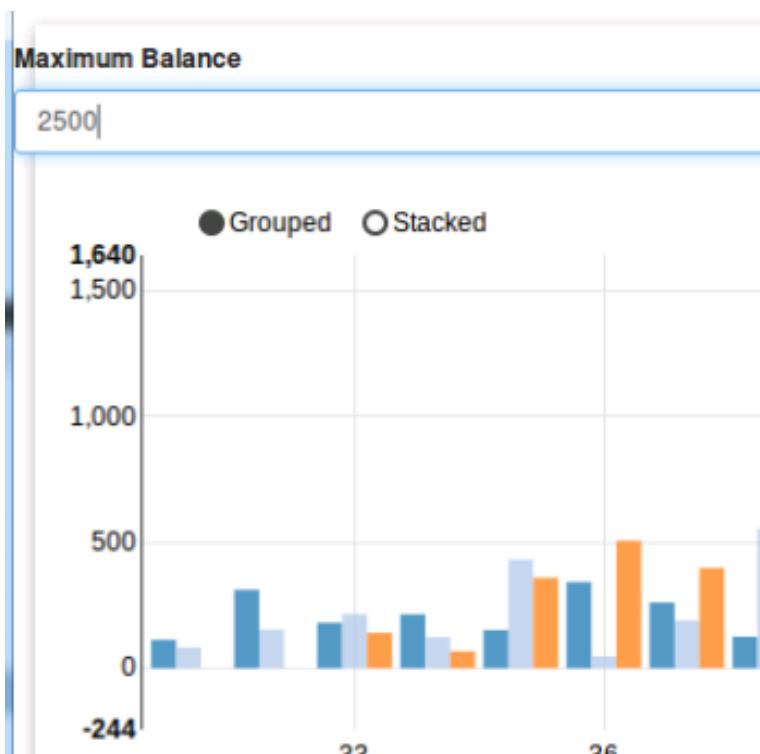
Why do you think changing the maximum balance from \$10,000 to \$5,000 had so little effect on the chart?

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.





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

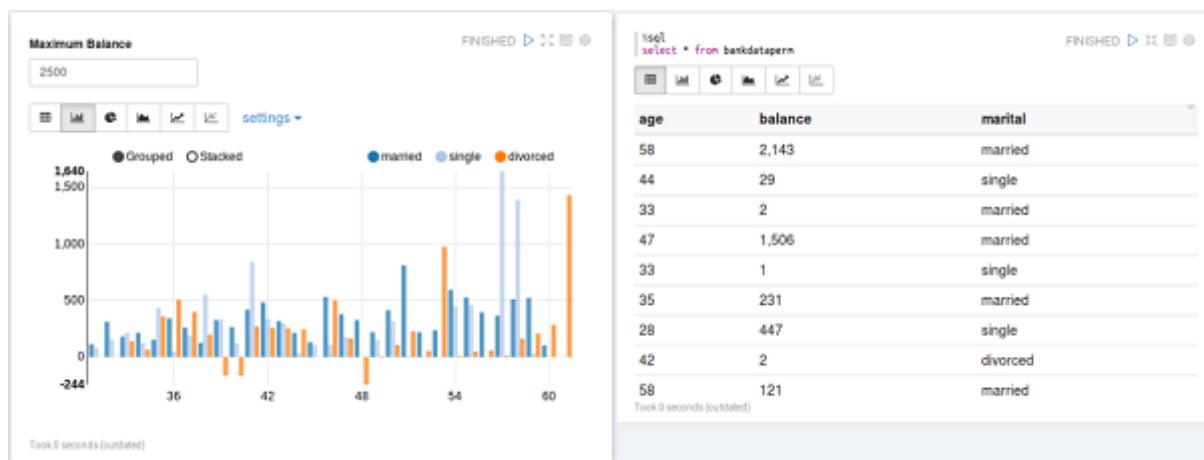
age	balance	marital
58	2,143	married
44	29	single
33	2	married
47	1,508	married

FINISHED ▶ ✎ 📄 ⚙

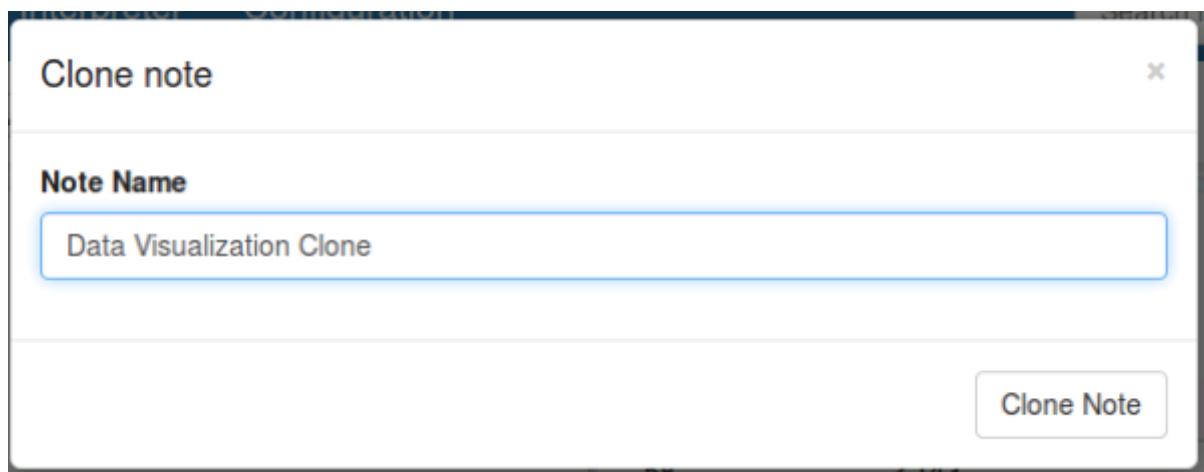
20160610-101920_468564635

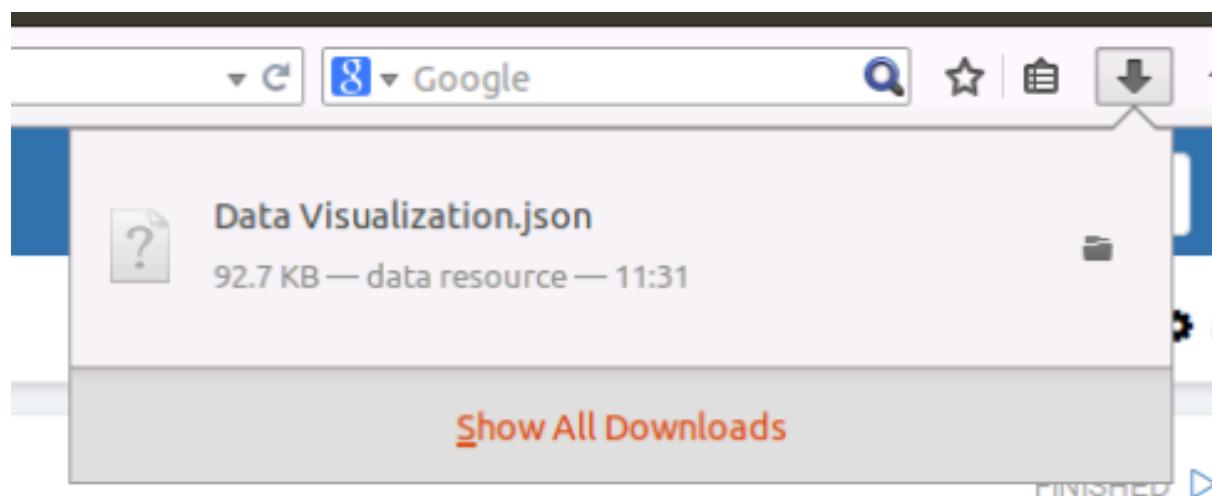
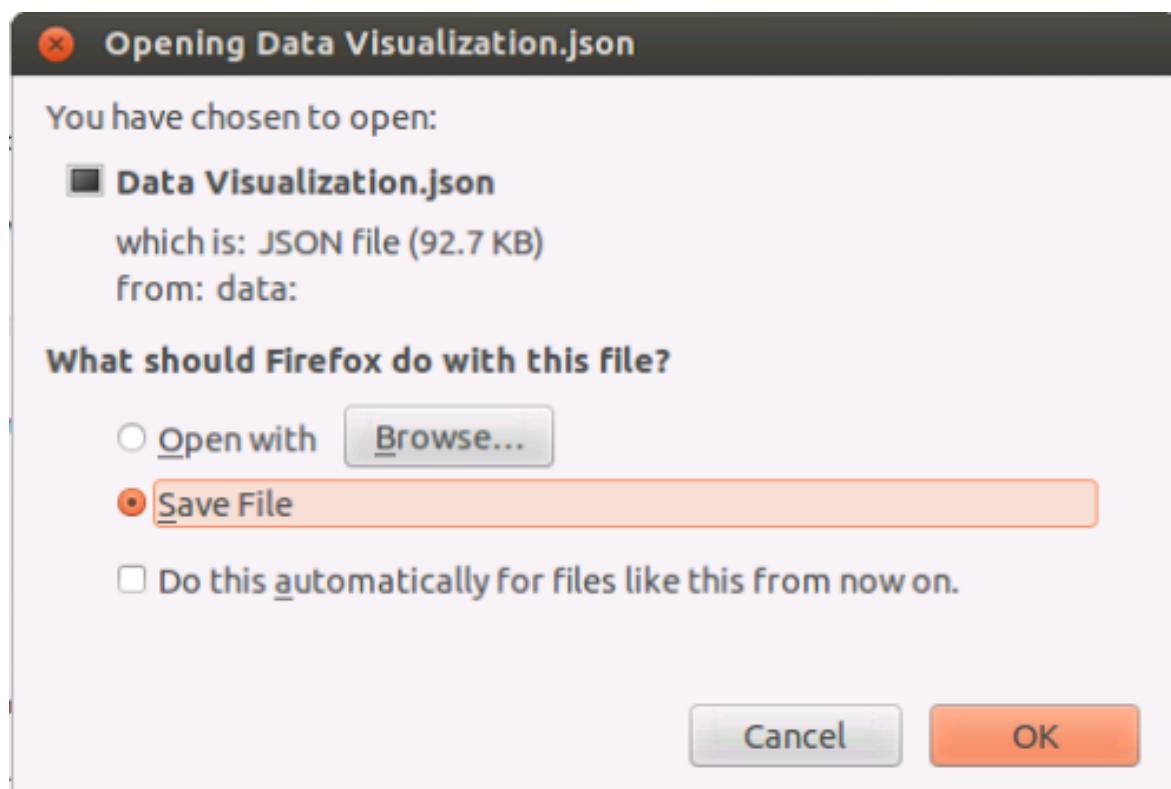
↔ Width

- Move Up
- Move Down
- Insert New
- Show tip
- Show lines
- Disable
- Link this



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.





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.

Data Visualization

The screenshot shows a Data Visualization interface with the following components:

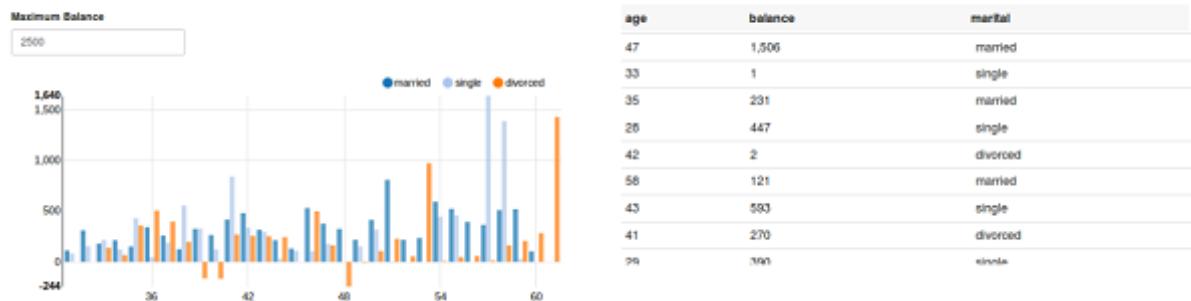
- Top Bar:** Includes icons for back, forward, search, and other navigation.
- Progress Indicators:** FINISHED (20/20), ERROR (0/20), and another FINISHED (20/20).
- Chart Area:** Titled "Maximum Balance". It has a dropdown set to 2500. The chart is grouped (not stacked) and shows balance by age (36, 42, 48, 54, 60) for three marital statuses: married (blue), single (light blue), and divorced (orange). The Y-axis ranges from -244 to 1,640.
- Table Area:** A table with columns "age", "balance", and "marital". The data is as follows:

age	balance	marital
47	1,506	married
33	1	single
35	231	married
28	447	single
42	2	divorced
58	121	married
43	593	single
41	270	divorced
29	390	single

Task 5 seconds (estimated)

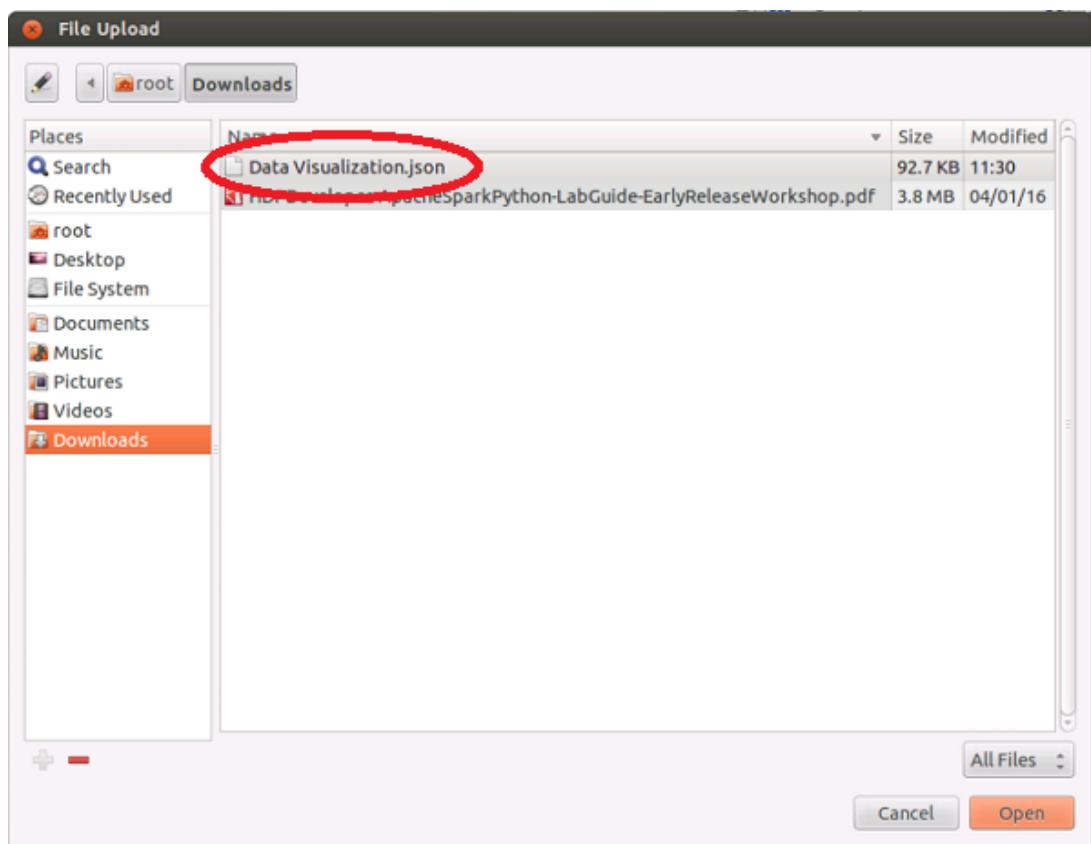
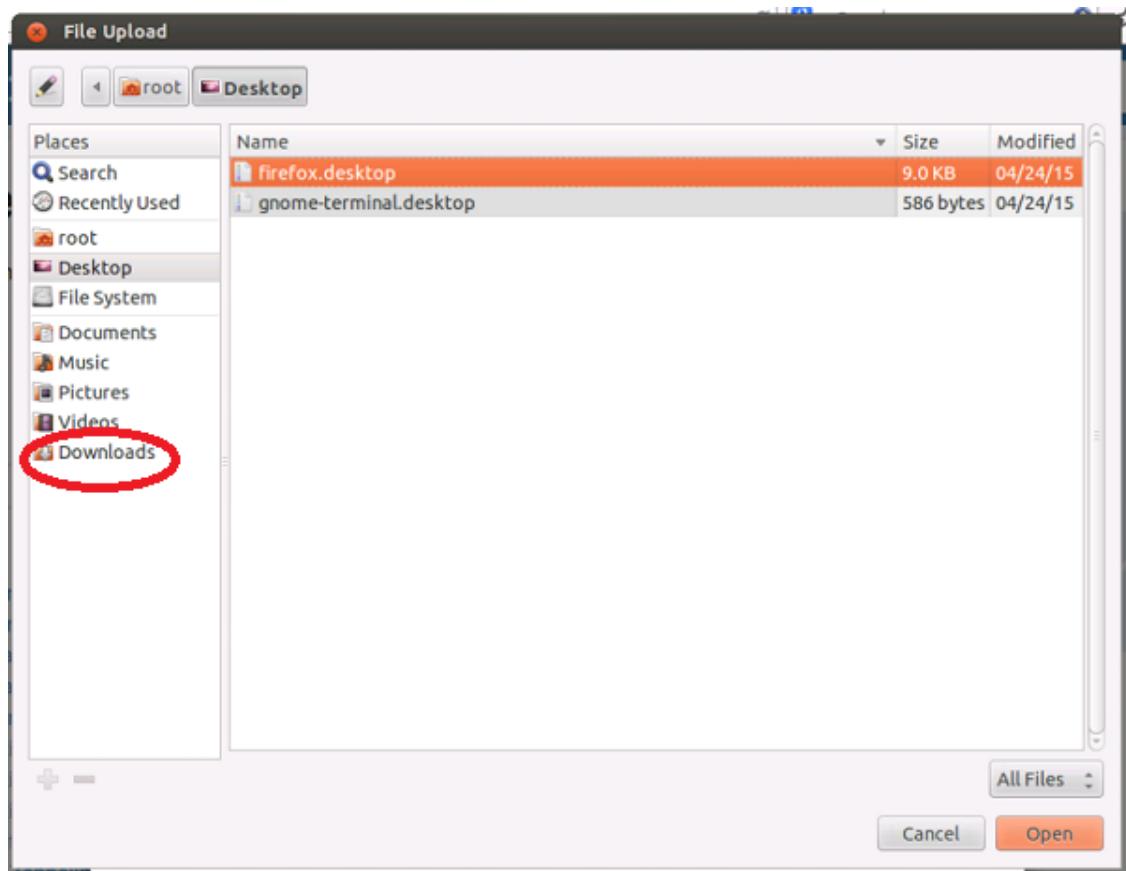
Next, convert this from the default view to report view. Now the URL to this note is ready to share with your stakeholders.

Data Visualization



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.

The screenshot shows the Zeppelin web interface. At the top, there is a blue header bar with the Zeppelin logo, a navigation menu with 'Notebook', 'Interpreter', and 'Configuration' items, and a search bar. Below the header, a large title says 'Welcome to Zeppelin! (0.6.0-incubating)'. Underneath the title, there is a brief introduction: 'Zeppelin is web-based notebook that enables interactive data analytics. You can make beautiful data-driven, interactive, collaborative document with SQL, code'. On the left side, there is a sidebar with 'Notebook' and 'Import note' (which is circled in red), 'Create new note', and a 'Filter' button. On the right side, there are links for 'Help', 'Get started with Zeppelin documentation', 'Community', and a 'Please feel free to help us to improve Zeppelin!' link. In the center, there is a modal dialog titled 'Import new note'. The dialog has a 'Import AS' field containing 'Data Visualization Imported'. It also has two options: 'Choose a JSON here' (represented by a cloud icon with an upward arrow) and 'Add from URL' (represented by a chain icon).



- [Data Visualization](#)
- [Data Visualization Clone](#)
- [Data Visualization Imported](#)
- [Hello World Tutorial](#)

Data Visualization Imported

The screenshot shows the Zeppelin interface with the title "Data Visualization Imported". At the top, there is a table with three rows:

name	value
insert_user	true
bankdataperm	false
table1hive	false

Below the table are two visualization cells. The left cell is titled "Maximum Balance" and contains a text input field with "2500" and a "settings" dropdown. The right cell is titled "SQL" and contains the query "select * from bankdataperm" and a preview table with columns "age", "balance", and "marital".

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.

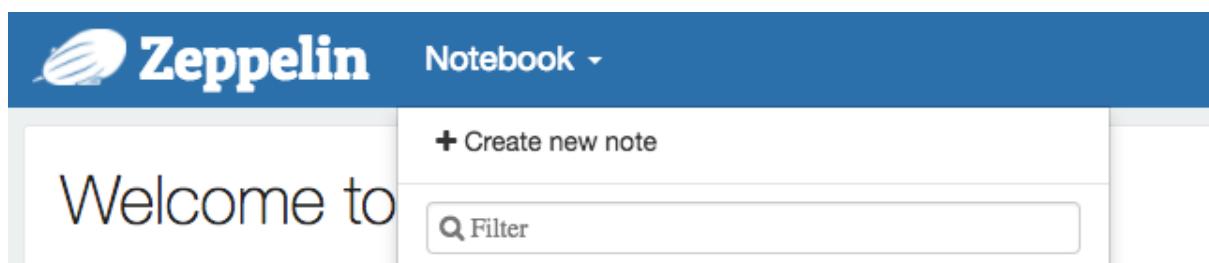
Caution! : Please don't copy and paste in the notebooks, it will generate errors in the code syntax if you do so.

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

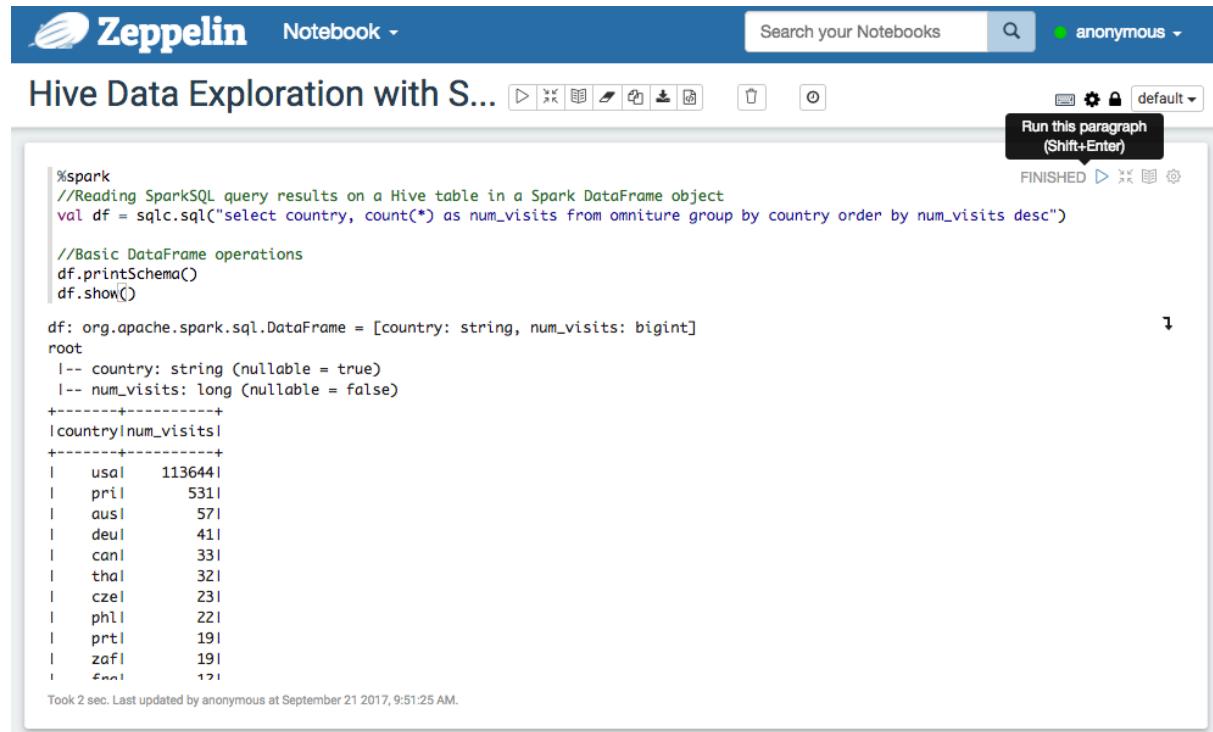
In the first paragraph, we are going to invoke the Spark Hive Context sqlc to query the omniture Hive table created in the Hive lab using the spark interpreter, and read the data into a Spark DataFrame object:

```
%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:



The screenshot shows a Zeppelin Notebook interface. The title bar says "Zeppelin Notebook". The search bar contains "Search your Notebooks". The user is "anonymous". The notebook content is titled "Hive Data Exploration with S...". The code block contains SparkSQL code to read a Hive table and print its schema and contents. The output shows the schema and a list of countries with their visit counts. A tooltip "Run this paragraph (Shift+Enter)" is visible above the run button. The status bar at the bottom says "FINISHED" and shows the execution time: "Took 2 sec. Last updated by anonymous at September 21 2017, 9:51:25 AM."

```
%spark
//Reading SparkSQL query results on a Hive table in a Spark DataFrame object
val df = sqlc.sql("select country, count(*) as num_visits from omniture group by country order by num_visits desc")

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

df: org.apache.spark.sql.DataFrame = [country: string, num_visits: bigint]
root
 |-- country: string (nullable = true)
 |-- num_visits: long (nullable = false)
+-----+-----+
|country|num_visits|
+-----+-----+
|  usal    1136441|
|  pril     531|
|  ausl      571|
|  deul      411|
|  canl      331|
|  thal      321|
|  czel      231|
|  phll      221|
|  prtl      191|
|  zafl      191|
|  enol      121|

```

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.

The screenshot shows a Zeppelin Notebook interface. At the top, there's a header with the Zeppelin logo, a search bar labeled "Search your Notebooks", and a user status "anonymous". Below the header, the title "Hive Data Exploration with SparkSQL" is displayed. The main area contains a code cell with the following content:

```
%spark
//Reading SparkSQL query results on a Hive table in a Spark DataFrame object
val df = sqlc.sql("select country, count(*) as num_visits from omniture group by country order by num_visits desc")

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

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

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

df: org.apache.spark.sql.DataFrame = [country: string, num_visits: bigint]
```

The output of the `df.show()` command is shown below the code:

country	num_visits
us	1136441
pr	5311
ausl	571
deul	411
canl	331
thal	321
czel	231
phl	221
prtl	191
zafl	191
finl	171

At the bottom of the code cell, it says "Took 5 sec. Last updated by anonymous at September 21 2017, 10:01:07 AM."

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:

The screenshot shows the Ambari Metrics dashboard. On the left, there's a sidebar with links to HDFS, YARN, MapReduce2, Tez, Hive, and HBase. The main area has tabs for Metrics, Heatmaps, and Config History. Below these are four cards: HDFS Disk Usage (56%), DataNodes Live (1/1), HDFS Links (NameNode, Secondary NameNode, 1 DataNodes), and Memory Usage (No Data Available). A dropdown menu is open over the YARN Queue Manager section, listing options: Files View, Hive View, Pig View, Storm View, and Tez View.

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

The screenshot shows an HDFS file browser interface. The path is /tmp/country_visits. The file list shows two files: _SUCCESS (0.1 kB, 2017-09-21 10:01, owner: zeppelin, group: hdfs, permission: -rw-r--r--) and part-00000 (0.2 kB, 2017-09-21 10:01, owner: zeppelin, group: hdfs, permission: -rw-r--r--). The part-00000 file is highlighted with a blue background. At the top, there are buttons for Open, Rename, Permissions, Delete, Copy, Move, Download, and Upload. There's also a search bar and a message "1 Files, 0 Folders selected".

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:

The screenshot shows the Ambari interface with the 'Sandbox' tab selected. In the top right, there is a user dropdown for 'maria_dev'. A context menu is open over a table listing files in '/tmp/country_visits'. The menu items include 'YARN Queue Manager', 'Files View', 'Hive View' (which is highlighted in dark grey), 'Pig View', 'Storm View', and 'Tez View'. Other menu items like 'Select All' and 'New' are also visible.

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.

The screenshot shows the Database Explorer and Query Editor in the Ambari interface. The Database Explorer on the left shows the 'default' database expanded, revealing tables like 'country_visits', 'products', 'sample_07', 'sample_08', 'users', 'webloganalytics', 'foodmart', and 'xademo'. The Query Editor on the right has a 'Worksheet' tab open with the query: 'SELECT * FROM country_visits LIMIT 100;'. Below the query, there are 'Execute', 'Explain', and 'Save as...' buttons. At the bottom, the 'Query Process Results' section shows the status 'SUCCEEDED' and a table with two rows:

country_visits.country	country_visits.num_visits
usa	113644
nri	531

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.

state	num_visits
ca	14,395
fl	7,745
ny	7,540
tx	7,049
pa	4,610
oh	4,018
il	3,771
va	3,310
mi	3,289

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:

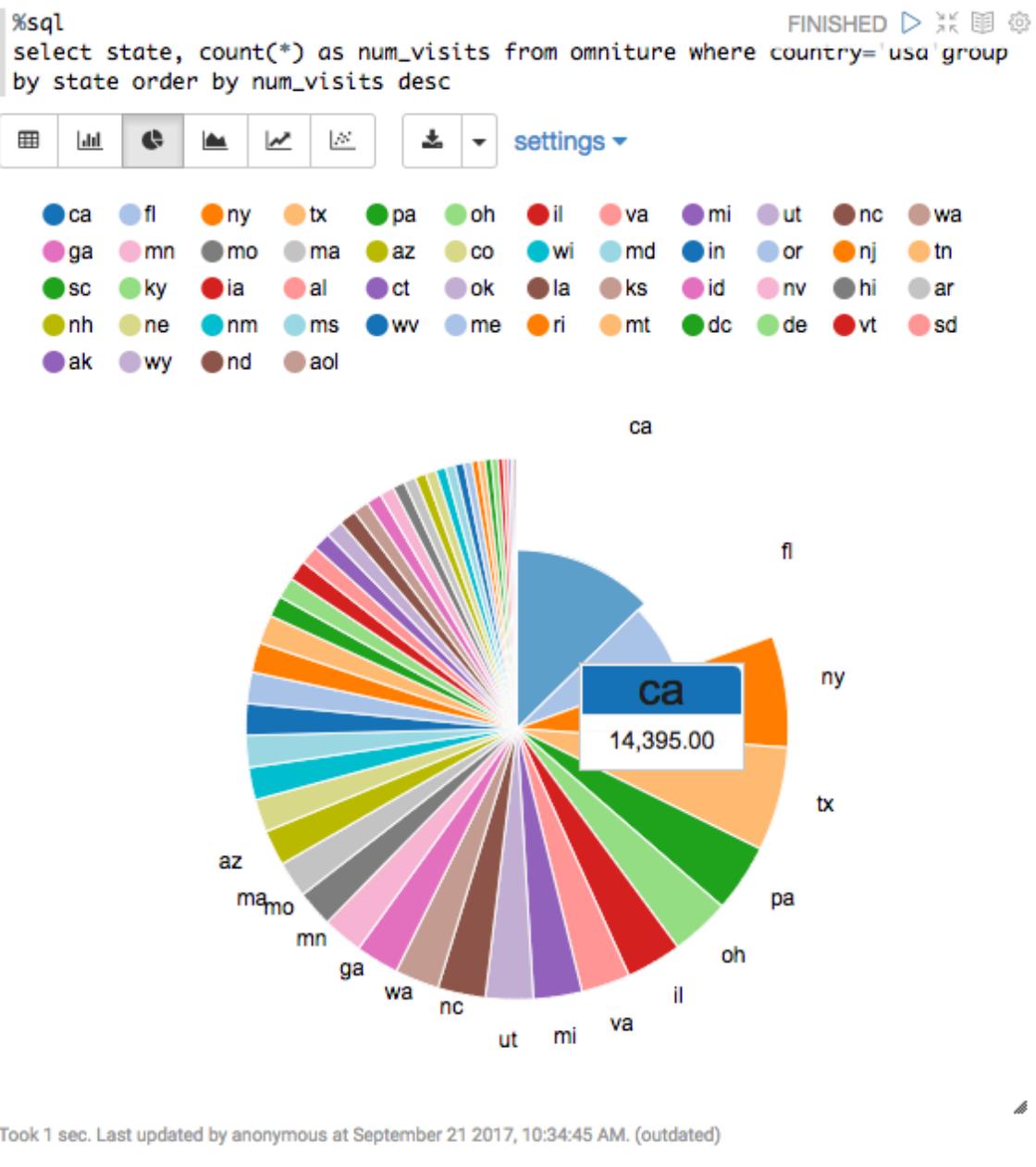
All fields: state num_visits

Keys: state

Groups:

Values: num_visits SUM

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:

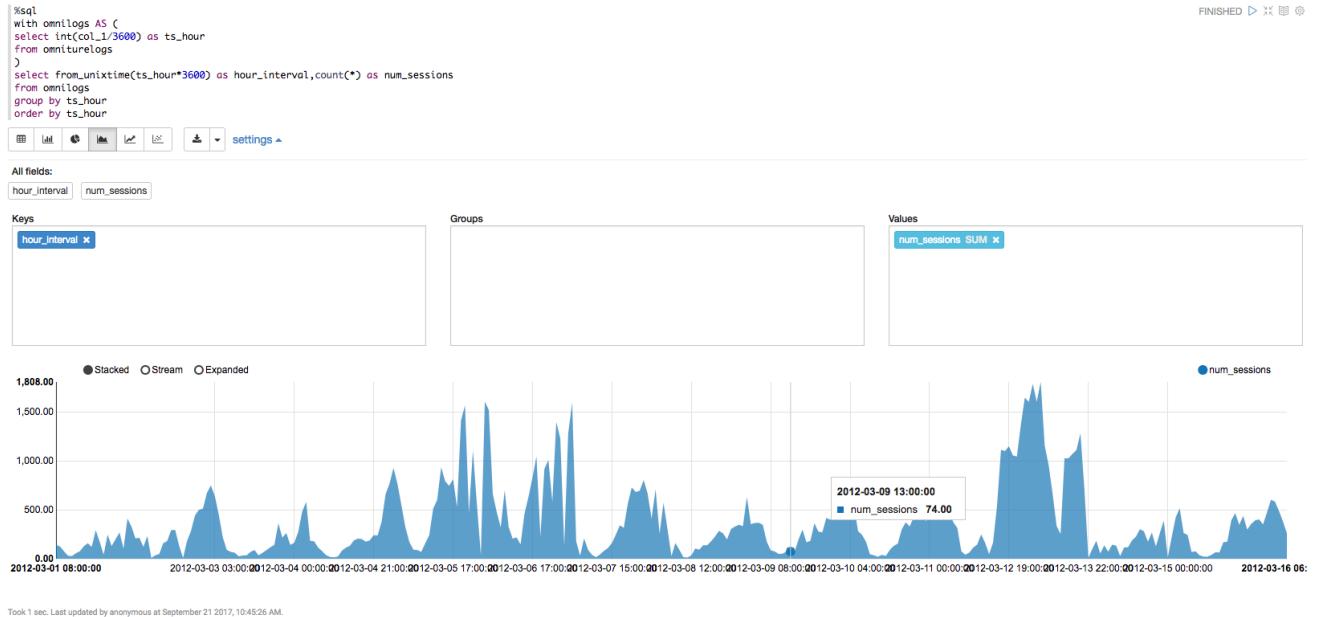


We can also try an interesting visualization which gives us the number of visits per hour from the omniture logs. Start a new paragraph, and type the following query:

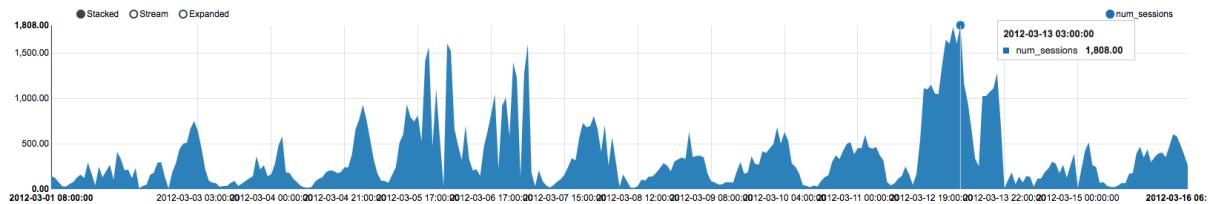
```
%sql
with omnilog AS (
  select int(col_1/3600) as ts_hour
  from omniturelogs)
  select from_unixtime(ts_hour*3600) as hour_interval, count(*) as num_sessions
  from omnilog
  group by ts_hour
  order by ts_hour
```

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.