University of Salford, MSc Data Science

Module: Big Data Tools and Techniques

Date: Trimester 2, 2024-2025

Session: Workshop Week 5

Topic: Spark SQL in Databricks

Tools: Databricks Community Edition

Instructors: Nathan Topping, Dr Taha Mansouri, and Dr Kaveh Kiani.

Objectives:

After completing this workshop, you will be able to:

- Creating a database in Spark SQL
- Creating tables and views in Spark SQL
- Querying tables and views
- Visualising a query result

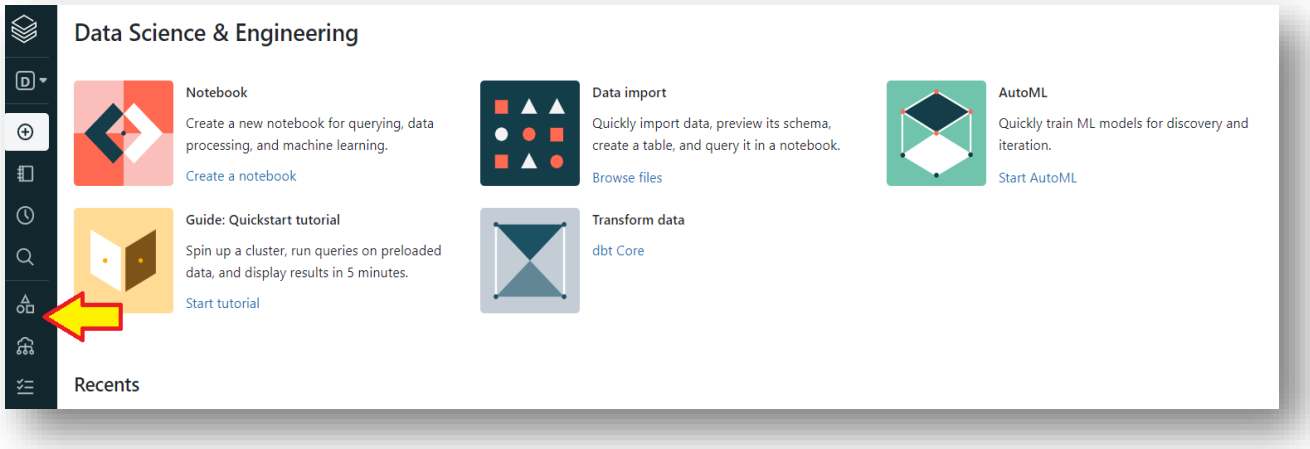
Table of Contents

Part 1: Fire up the Databricks workspace

1. Log in to your Databricks Community Edition account. Here is the link:

   https://community.cloud.databricks.com/login.html?nextUrl=%2F

2. You will need to create a cluster to start your analysis – this gives you access to a machine to use. Click on “Compute” on the main page.



3. Click on “Create Compute” and type in a new name for the cluster, any name that you like.
4. Select the most recent runtime.

   Note that it will take a few minutes to create the cluster. After some time, the green circle next to the cluster name will gain a green tick meaning the cluster has successfully started up.

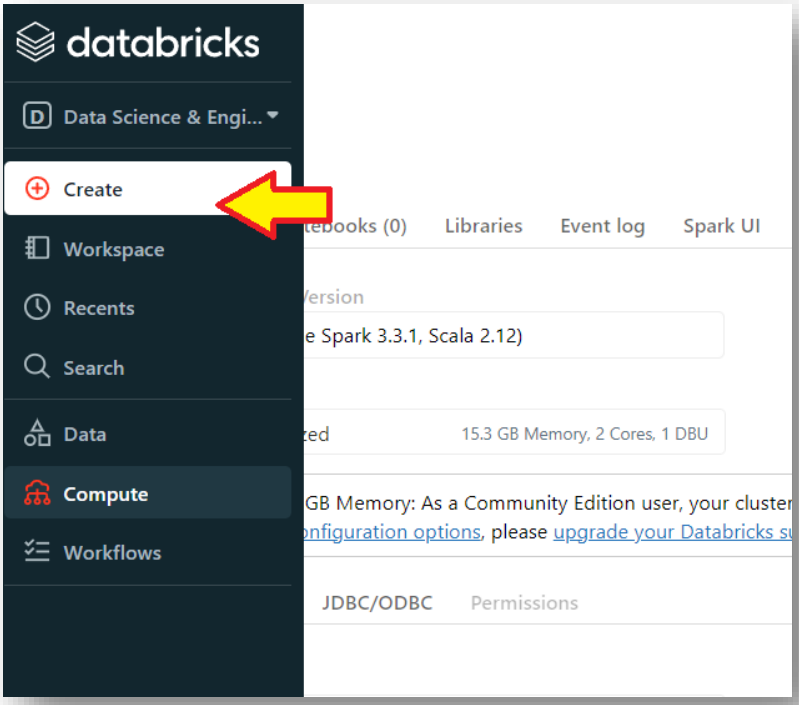
Part 2: Creating a new SQL notebook

Databricks programs are written in Notebooks. A notebook is a collection of runnable cells which contain your commands. Look at:

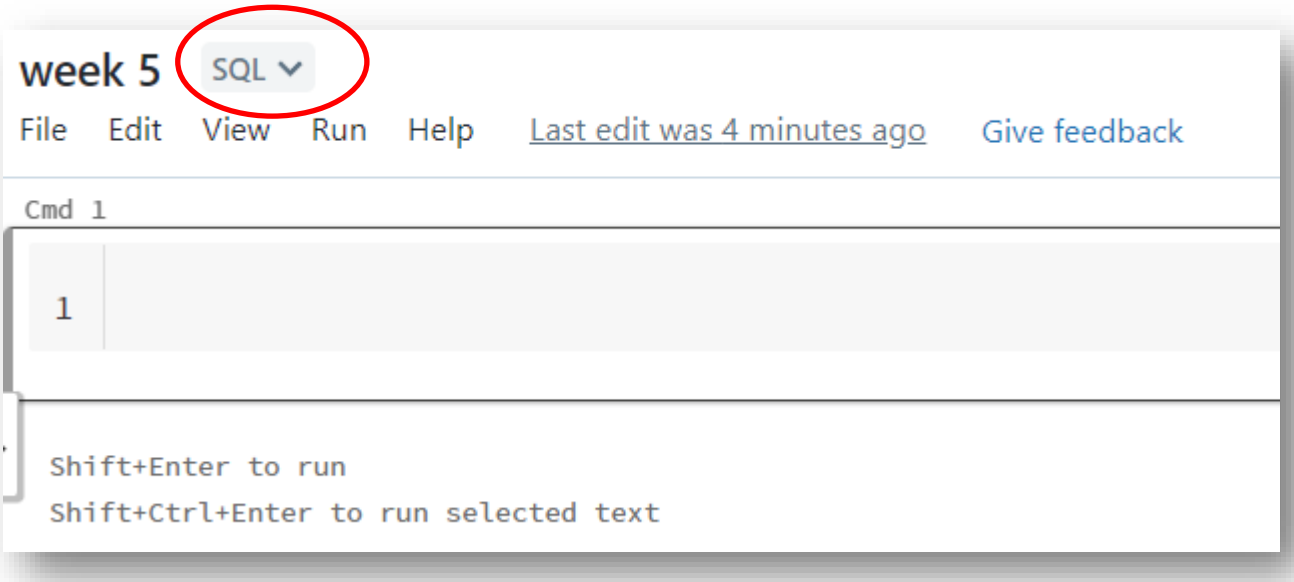
https://docs.databricks.com/notebooks/notebooks-use.html

for an overview of how to use a Databricks notebook.

1. Click on the button “Create” and select “Notebook”.



2. Name your notebook, e.g. “SQL week 5”, leave (or select) the language to be “**SQL**” and the cluster should be your currently created cluster. This will give you access to a notebook which looks something like this:

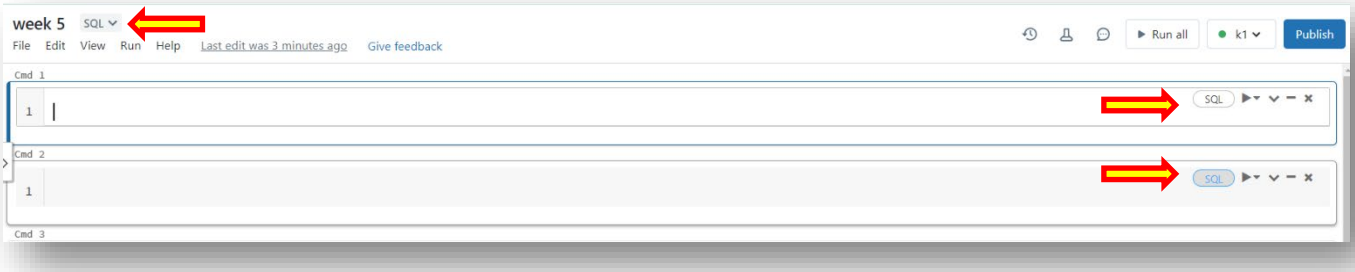


Part 3: Creating SQL tables and views from existing DataFrame

In this part, you will use `Spark SQL` to convert "webpage" `DataFrame` that you created in week 4 to `SQL` tables and views and will learn how to run `SQL` scripts on the tables. During the lab if you feel you need to retrieve what you have learnt during the `SQL` lecture go back and review the lecture slides.

1) SQL notebook and shifting cells language.

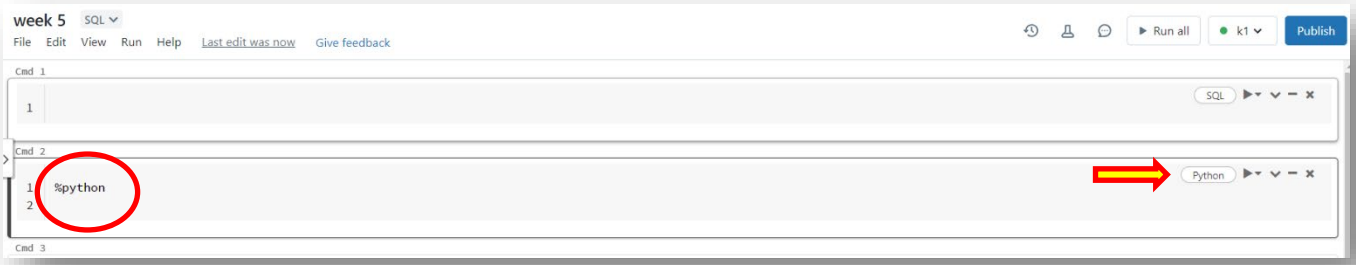
Your new notebook is a SQL notebook not Python. All the cells in this notebook will have SQL language.



But sometimes you need to run a Python cell in between of the SQL cells. Here you must change only that cell's language to Python. Click on the cell's SQL tab and change it to Python.

Then automatically you will see "%python" command on top of the cell.

"%python","%sql", "%r", … are called "`line magic function`".



2) Check (or create) the webpage directory in your DBFS.

You created webpage DataFrames in week 4 lab. Check your `DBFS` to confirm you have the webpage directory that you created DataFrames based on the data files in the webpage directory. You should have the following directory in your `DBFS` that contains 4 `parquet` files:

```
FileInfo(path='dbfs:/FileStore/tables/webpage/', name='webpage/', size=0, modificationTime=0)
```

Shift the first cell from `SQL` to `Python` and run the `ls` command.

If you don't have the directory, it means you have not done the lab 4. Then go back to lab 4 workshop hands out and create and populate the "webpage" directory. However, you can find codes in the appendix of this document.

3) Create webpages1 DataFrame

Like week 4, create again `webpages1` DataFrame based on the files in the `webpage` directory. **Shift the cell from SQL to Python** and run the following piece of codes to generate `webpages1` DataFrame.

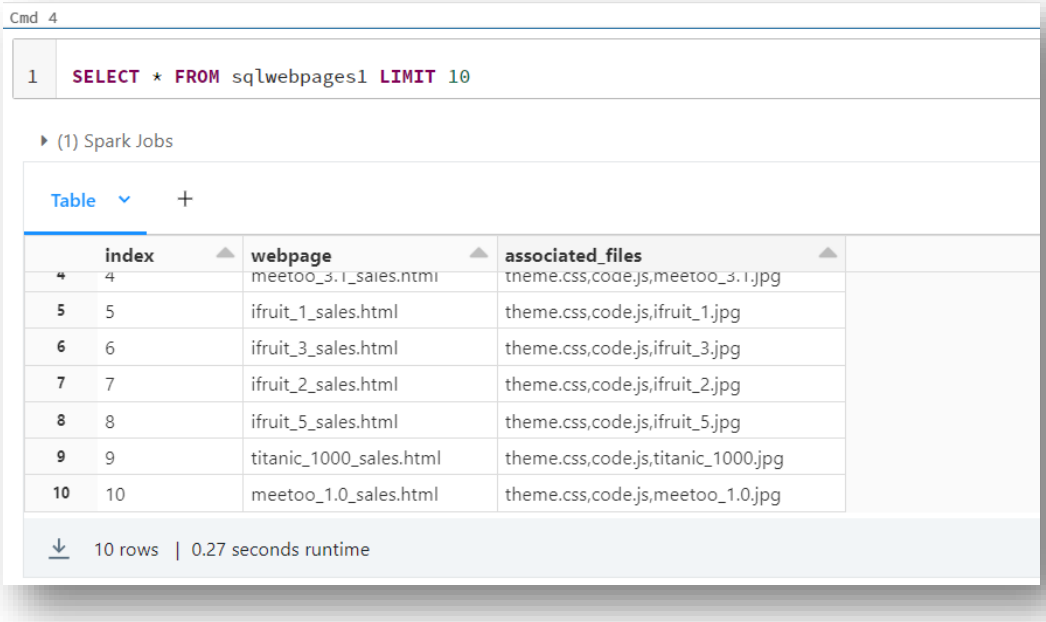
```
from pyspark.sql.types import *

mySchema = StructType ([
           StructField("index", IntegerType()) ,
           StructField("webpage", StringType()) ,
           StructField("associated_files", StringType())])

myRDD = sc.textFile("/FileStore/tables/webpage/*"). \
             map(lambda line: line.split("\t")). \
             map(lambda values: [int(values[0]) , values[1] , values[2]])


webpages1 = spark.createDataFrame(myRDD , mySchema)

webpages1.show(5)
```

4) Create SQL Databases and Tables

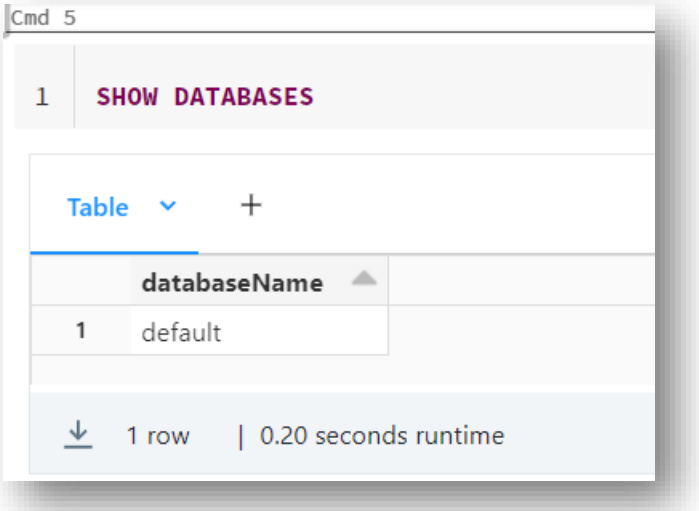
1) First, we want to create a temporary table (**view**) from the webpages1 DataFrame then will write queries on this view. Shift the cell to Python and run the following.

```
webpages1 . createOrReplaceTempView ("sqlwebpages1")
```

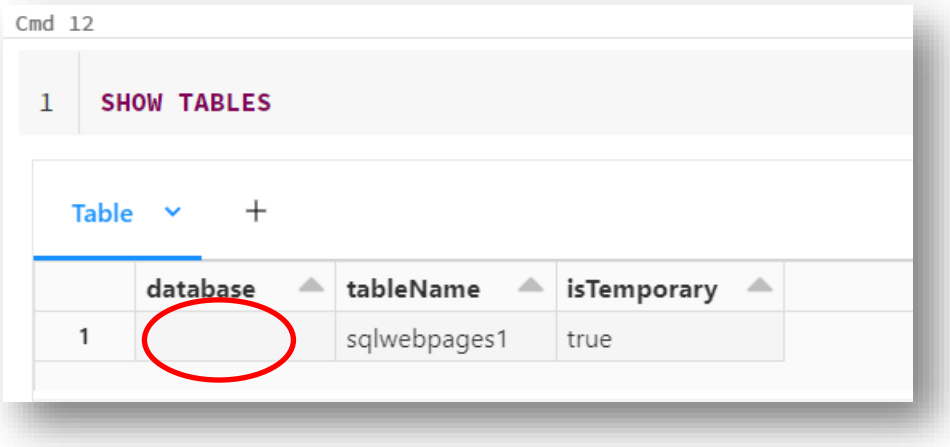
2) Check content of the view:



3) Spark SQL has a "`default`" database that stores all tables and views by default there if you don't specify a database name. To list all available databases:

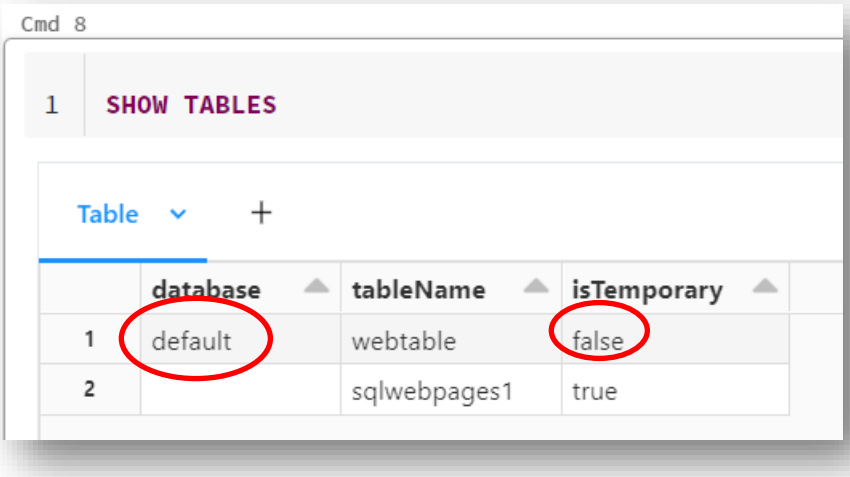


5) Let's see where the location of the `sqlwebpages1` view is:

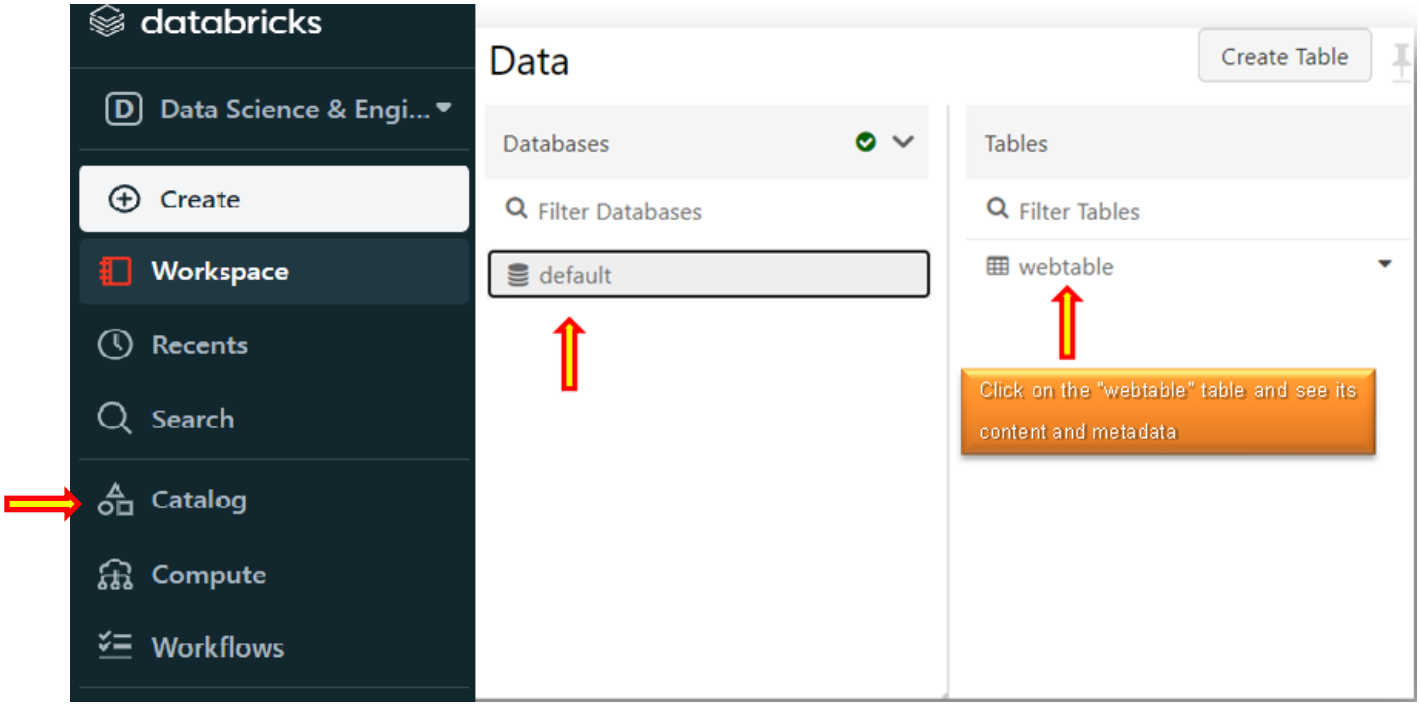


Sqlwebpages1 has not been stored in any database even "default" and the lifetime of this temporary table is tied to the SparkSession that has been used to create this DataFrame.

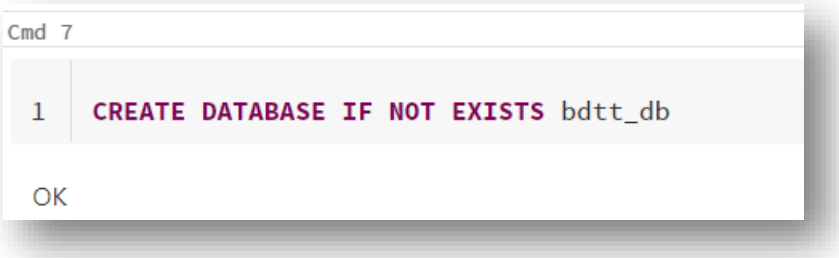
6) If you want to store this temporary view (`sqlwebpages1`) as a permanent table in the default database and name the new table as "`webtable`":



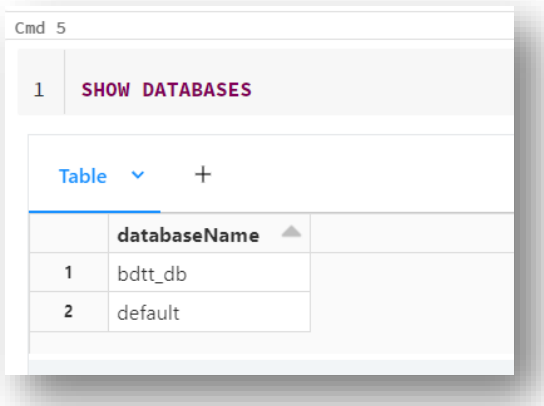
Also, you can see the "`webtable`" is in the "default" database from the Databricks dashboard.



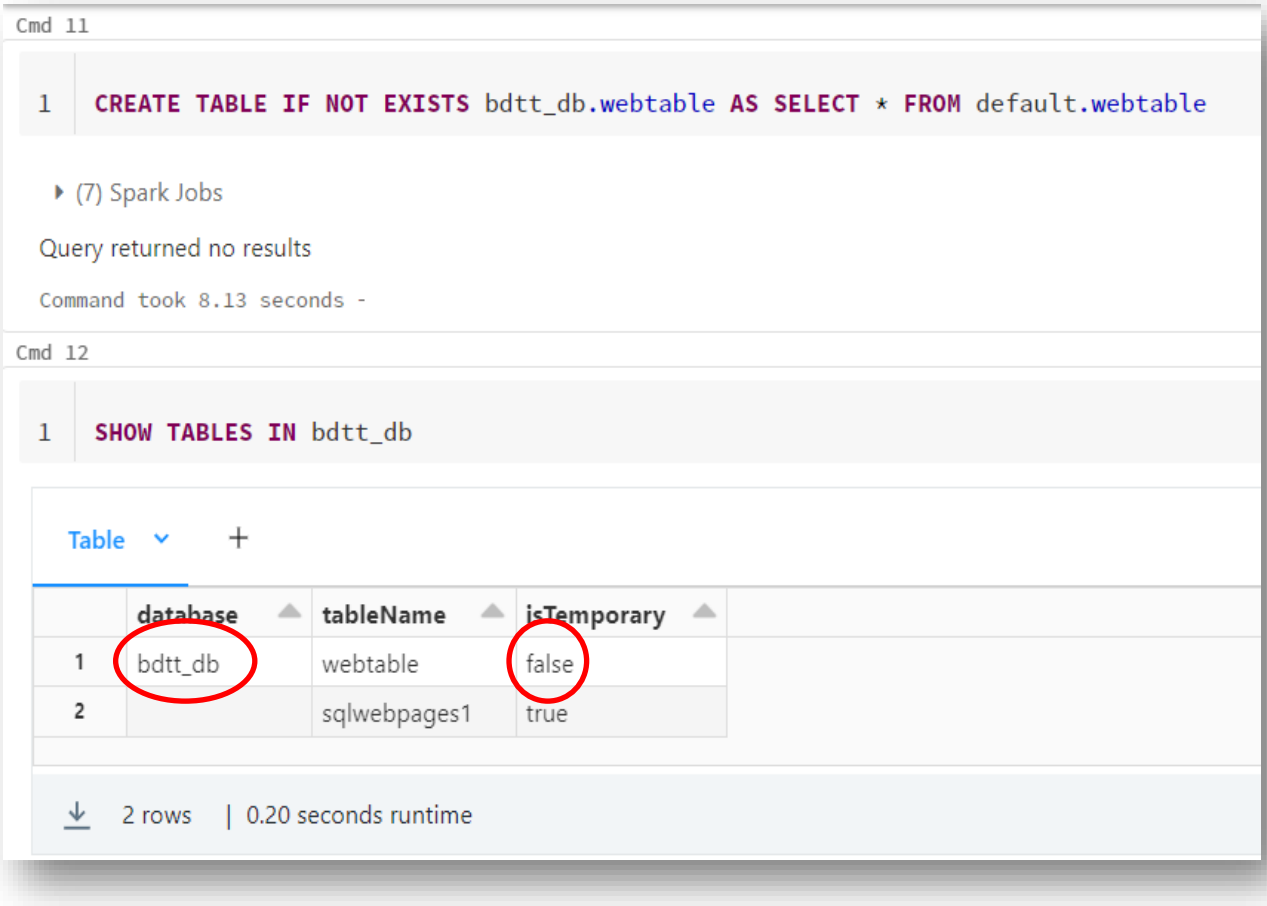
7) But we want to create a new database and store all our tables and views in it.



8) Check if the new database is on the list:



9) Copy the “webtable” to your database “bdtt_db” and check if the table has been copied to “bdtt_db”:



Temporary “sqlwebpages1” view is still in the system because you have not terminated the Spark Session yet.

Databricks Community Edition is a limited version. Therefore, it's essential to know that not only the temporary view named "sqlwebpages1" will be deleted due to its temporary nature, regardless of the Databricks version you're using, but also to note that permanent tables and any created databases will not be accessible after the session terminates.

10) The following table shows you what will happen to your tables and databases If you terminate the Spark Session in **Community Edition**.

After Termination of the Spark Session and Starting a New Session:
Comparison between Databricks Community Edition and Paid Version

	Community Edition (free)	Main Edition (paid)
Default database	Always is in your account but is empty after each termination!	Always is in your account with all permanent tables/views in it
User created database (e.g. bdtt_db)	Is not listed as a database and you can't see it and its content after each termination!	Always is in your account with all permanent tables/views in it

11) In this case bdtt_db and its contents also contents of the default database are in DBFS but you can't retrieve and use them. After termination you can check this path and see its content:

```
dbutils.fs.ls("/user/hive/warehouse")
```

Important: So, you should drop or replace them in a new Spark Session. Because of this attribute we use SQL queries like the followings to make our notebook **rerunnable**:

```
CREATE OR REPLACE TABLE default.webtable AS SELECT * FROM sqlwebpages1
CREATE TABLE IF NOT EXISTS default.webtable AS SELECT * FROM sqlwebpages1
CREATE DATABASE IF NOT EXISTS bdtt_db
```

Activity: Search what is the difference between "`CREATE OR REPLACE TABLE`" and "`CREATE TABLE IF NOT EXISTS`"

Part 4: Executing some SQL queries and visualising the result.

1) Run some SQL queries on "webtable" and create new tables and views.

1) You know what is a view in SQL and why we create views. Now, we want to create a **view** from the "webtable" where "webpage" column should contain only elements that start with "s" and store the view in the bdtt_db database:

```
Cmd 15
1  CREATE OR REPLACE VIEW bdtt_db.s_webtable AS SELECT * FROM bdtt_db.webtable WHERE webpage LIKE 's%'

OK
Command took 0.52 seconds -- by k.kiani@salford.ac.uk at 2/11/2023, 4:10:58 PM on k1
Cmd 16
1  SELECT * FROM bdtt_db.s_webtable LIMIT 3

▸ (1) Spark Jobs
Table +
```

	index	webpage	associated_files
1	27	sorrento_f10l_sales.html	theme.css,code.js,sorrento_f10l.jpg
2	29	sorrento_f41l_sales.html	theme.css,code.js,sorrento_f41l.jpg
3	39	sorrento_f33l_sales.html	theme.css,code.js,sorrento_f33l.jpg

3 rows | 0.52 seconds runtime

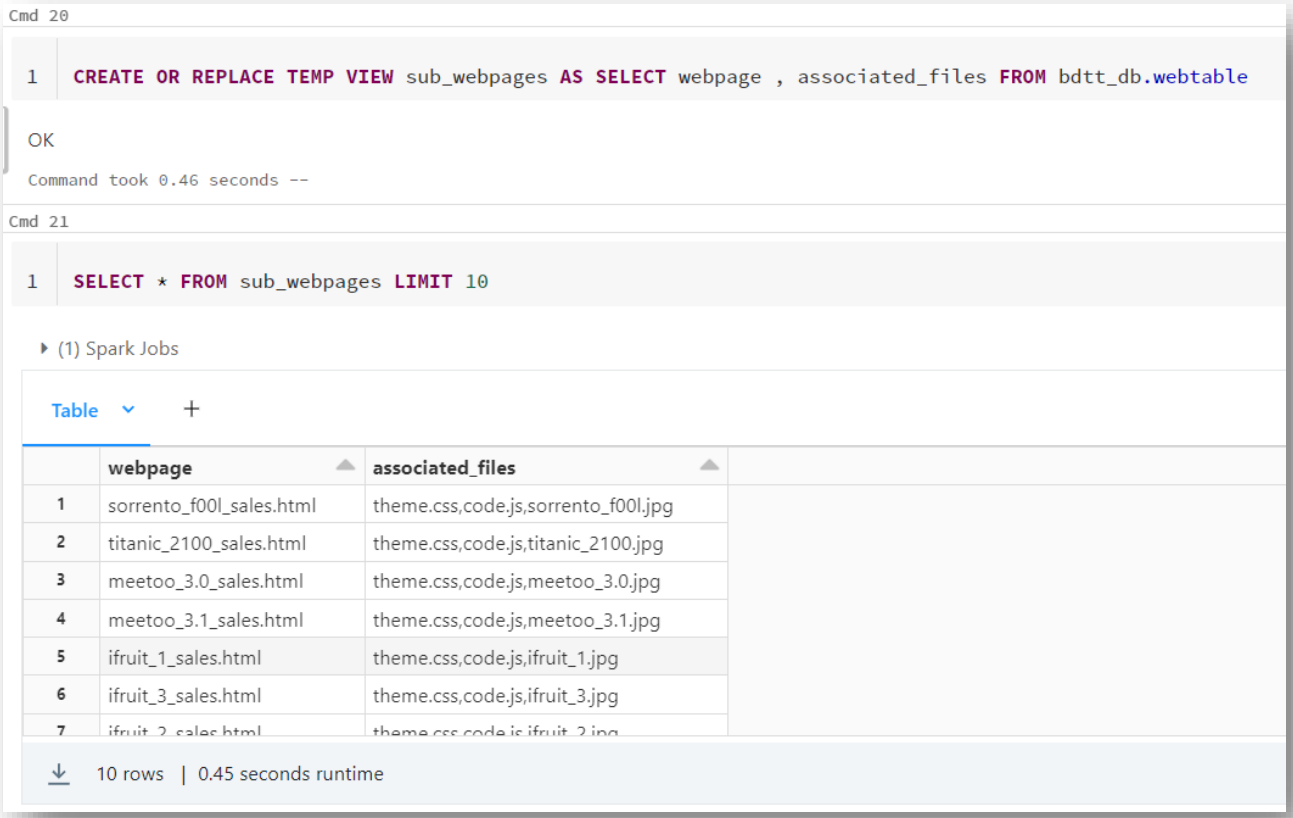
The new view is in the bdtt_db:

```
Cmd 17
1  SHOW TABLES IN bdtt_db

Table +
```

	database	tableName	isTemporary
1	bdtt_db	s_webtable	false
2	bdtt_db	webtable	false
3		sqlwebpages1	true

3 rows | 0.15 seconds runtime

2) Create a view which contains the `webpage` and `associated_files` columns from the `webtable` and check its content.



3) This time, we'll manipulate the data using SQL rather than converting to RDDs.

   We're aiming to use the SQL function `explode` which takes an array and separates out its elements into multiple rows (like flatMap). So first we need an array to be made from associated files.

   There's also a function `split` in SQL, which can be used to split a string into an array with a specific delimiter. For example, check the contents of the view created by:

Cmd 22

```
1  CREATE OR REPLACE TEMP VIEW spliteg AS SELECT split(webpage , '_') as split_wp , associated_files FROM bdtt_db.webtable
```

OK

Command took 0.34 seconds

Cmd 23

```
1  SELECT * FROM spliteg LIMIT 10
```

(1) Spark Jobs

Table +

	split_wp	associated_files
1	["sorrento", "f00l", "sales.html"]	theme.css,code.js,sorrento_f00l.jpg
2	["titanic", "2100", "sales.html"]	theme.css,code.js,titanic_2100.jpg
3	["meetoo", "3.0", "sales.html"]	theme.css,code.js,meetoo_3.0.jpg
4	["meetoo", "3.1", "sales.html"]	theme.css,code.js,meetoo_3.1.jpg
5	["ifruit", "1", "sales.html"]	theme.css,code.js,ifruit_1.jpg
6	["ifruit", "3", "sales.html"]	theme.css,code.js,ifruit_3.jpg

10 rows | 0.42 seconds runtime

4) Create a view which converts the `associated_files` column in the view from step 3 into an array, splitting on the commas. Name the view "`splitwebpages`" and new column is "`split_af`"

Cmd 24

```
1  CREATE OR REPLACE TEMP VIEW splitwebpages AS SELECT webpage , split(associated_files , ',') as split_af FROM bdtt_db.webtable
```

OK

Command took 0.22 seconds --

Cmd 25

```
1  SELECT * FROM splitwebpages LIMIT 10
```

(1) Spark Jobs

Table +

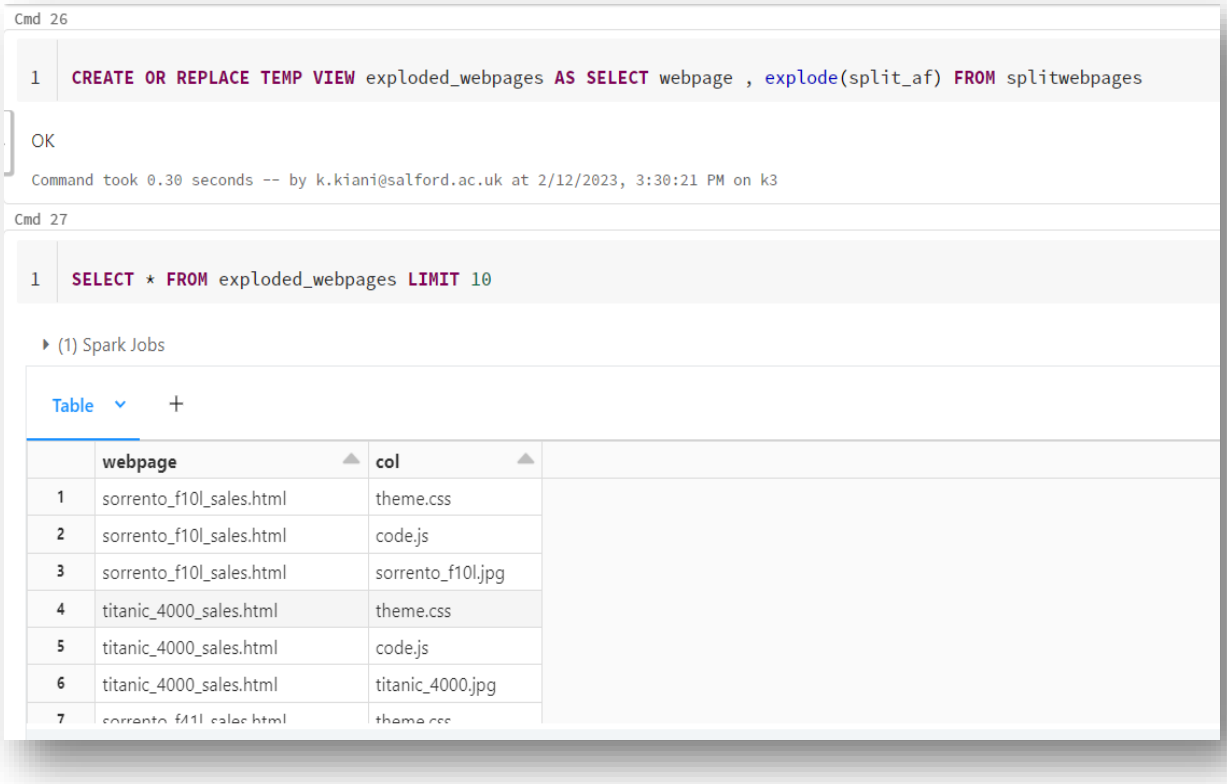
	webpage	split_af
1	sorrento_f00l_sales.html	["theme.css", "code.js", "sorrento_f00l.jpg"]
2	titanic_2100_sales.html	["theme.css", "code.js", "titanic_2100.jpg"]
3	meetoo_3.0_sales.html	["theme.css", "code.js", "meetoo_3.0.jpg"]
4	meetoo_3.1_sales.html	["theme.css", "code.js", "meetoo_3.1.jpg"]
5	ifruit_1_sales.html	["theme.css", "code.js", "ifruit_1.jpg"]
6	ifruit_3_sales.html	["theme.css", "code.js", "ifruit_3.jpg"]
7	ifruit_2_sales.html	["theme.css", "code.js", "ifruit_2.jpg"]

10 rows | 0.61 seconds runtime

5) Now we move onto using explode command. The definition of SQL explode is:

> `explode(expr)` - Separates the elements of array expr into multiple rows, or the elements of map expr into multiple rows and columns. Unless specified otherwise, uses the default column name col for elements of the array or key and value for the elements of the map.

Create a new view from the view you created in step 4 which explodes the `split_af` array as was wanted.



```
Cmd 26
1  CREATE OR REPLACE TEMP VIEW exploded_webpages AS SELECT webpage , explode(split_af) FROM splitwebpages
OK
Command took 0.30 seconds -- by k.kiani@salford.ac.uk at 2/12/2023, 3:30:21 PM on k3
Cmd 27
1  SELECT * FROM exploded_webpages LIMIT 10
▸ (1) Spark Jobs
```

Table +

	webpage	col
1	sorrento_f10l_sales.html	theme.css
2	sorrento_f10l_sales.html	code.js
3	sorrento_f10l_sales.html	sorrento_f10l.jpg
4	titanic_4000_sales.html	theme.css
5	titanic_4000_sales.html	code.js
6	titanic_4000_sales.html	titanic_4000.jpg
7	sorrento_f41l_sales.html	theme.css

Now you must complete the first Challenge:

Challenge 1:

In the `explode_webpage` table count number of webpages that have a .jpg file. **Count** webpages with .js and .css files as well (you should have 3 counts).

Then we need a **view** with only webpage and its associate .js file and remove any other records (name this view `web_js`). Outputs should look like:

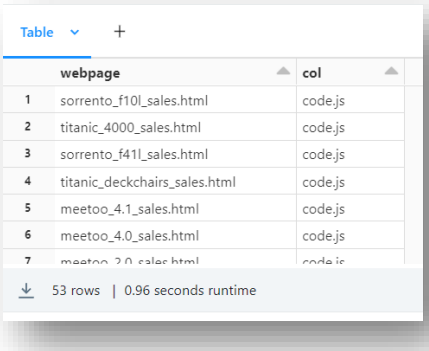


Table +

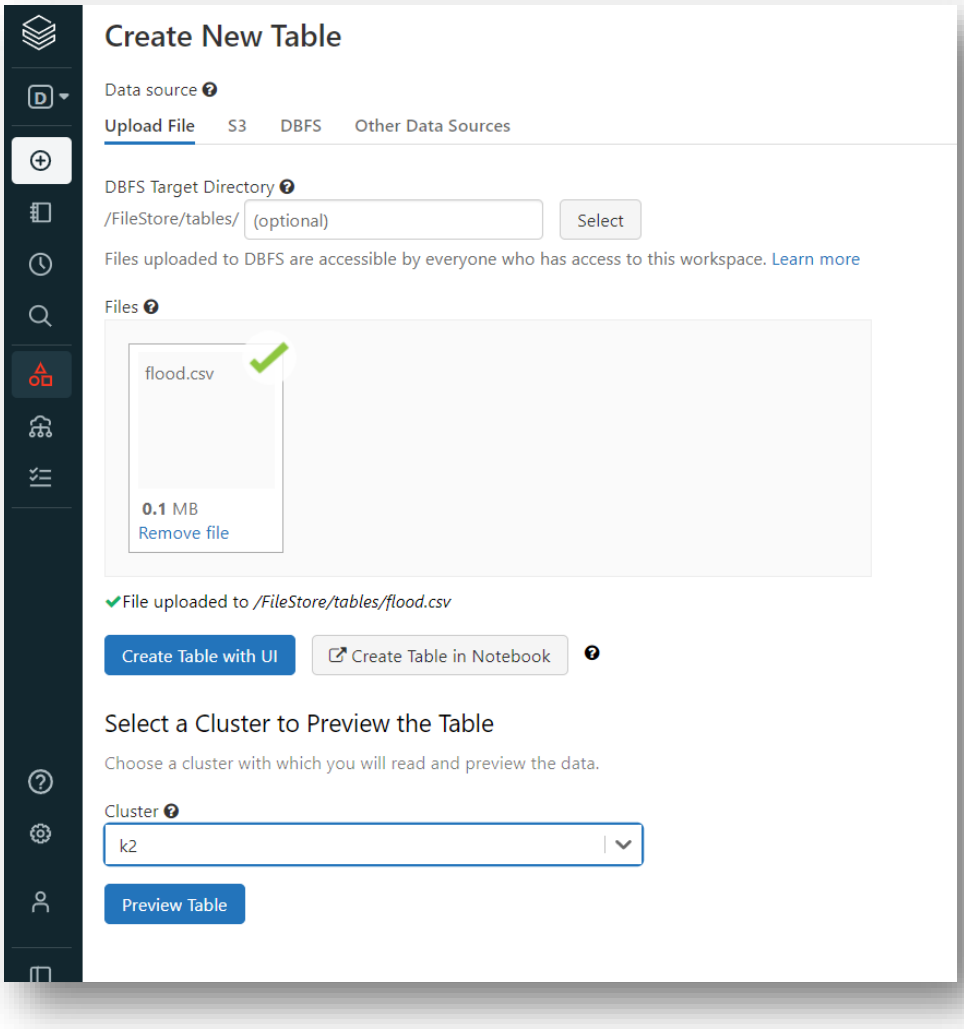
	webpage	col
1	sorrento_f10l_sales.html	code.js
2	titanic_4000_sales.html	code.js
3	sorrento_f41l_sales.html	code.js
4	titanic_deckchairs_sales.html	code.js
5	meetoo_4.1_sales.html	code.js
6	meetoo_4.0_sales.html	code.js
7	meetoo_2.0_sales.html	code.js

53 rows | 0.96 seconds runtime

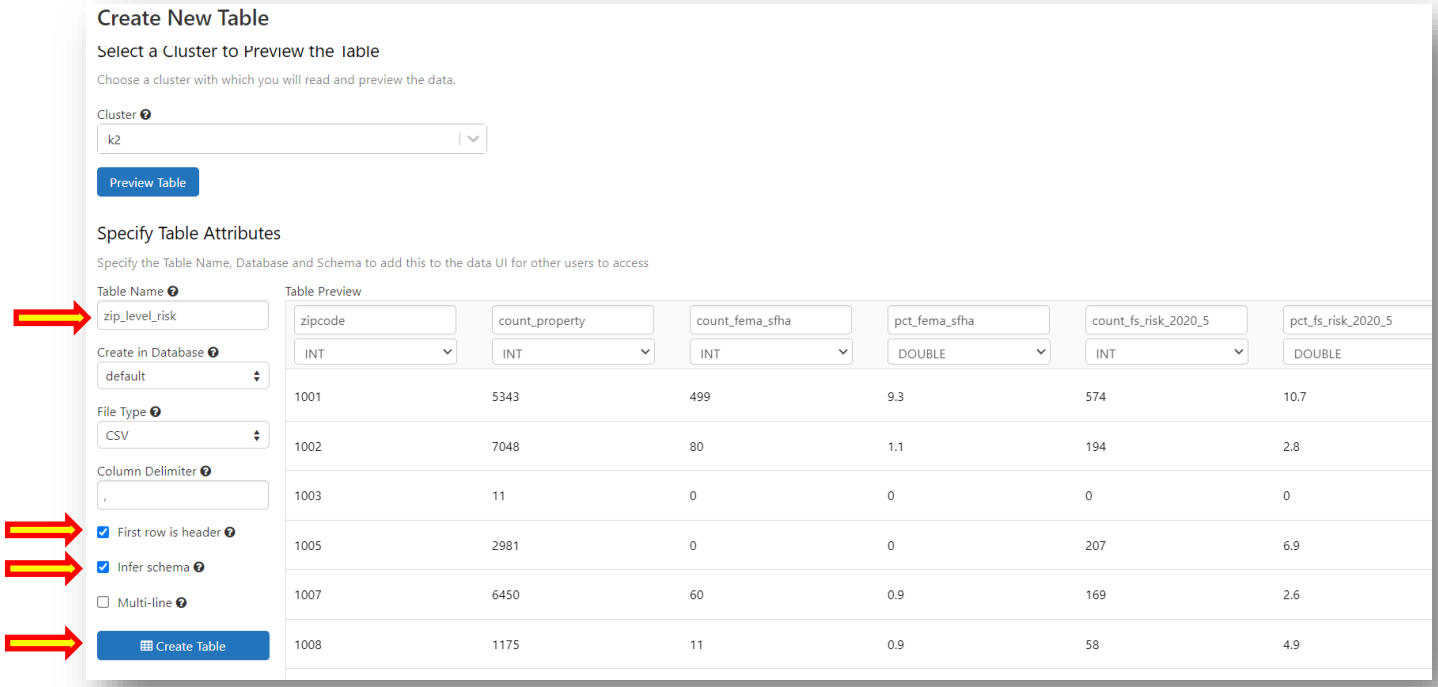
2) Run some SQL queries on "flood" dataset and create some graphs.

1) Download `flood.zip` from Blackboard and **unzip the file in your computer**. You should have `flood.csv` in your local machine ready to upload to Databricks (no need to unzip the file inside the Datarbricks and Linux).

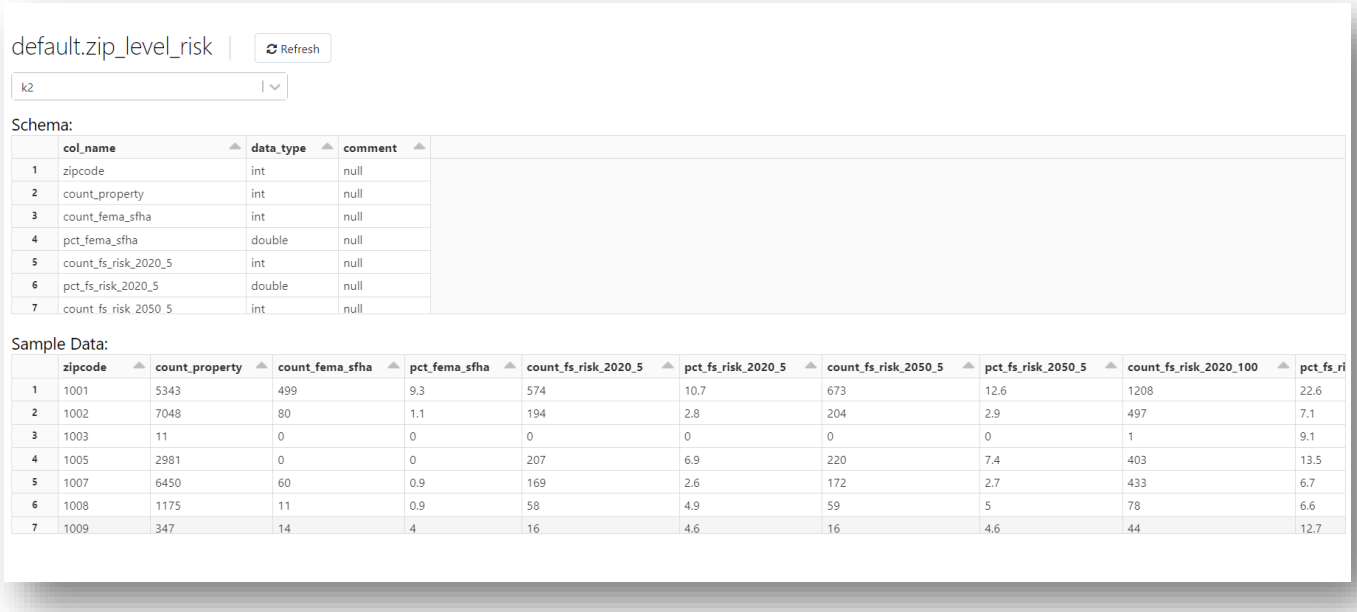
2) Upload the `flood.csv` file in the `tables directory`



3) Click on “Create Table with UI”
4) Change the value of Table Name to “zip_level_risk”
5) You can see that the first row is the header, so click on First row is header.
6) All the columns are strings by default, click on Infer schema to get the system to work out which are integers or doubles.
7) Once you have made the three modifications above, click Create table to put this table into your Hive metastore.



The page you get roughly should look like the following:



8) If you make a mistake, you'll need to delete the table and start again (to delete the table, you can go through the Data icon on the left, navigate to the table and the down arrow will give you the option to delete).

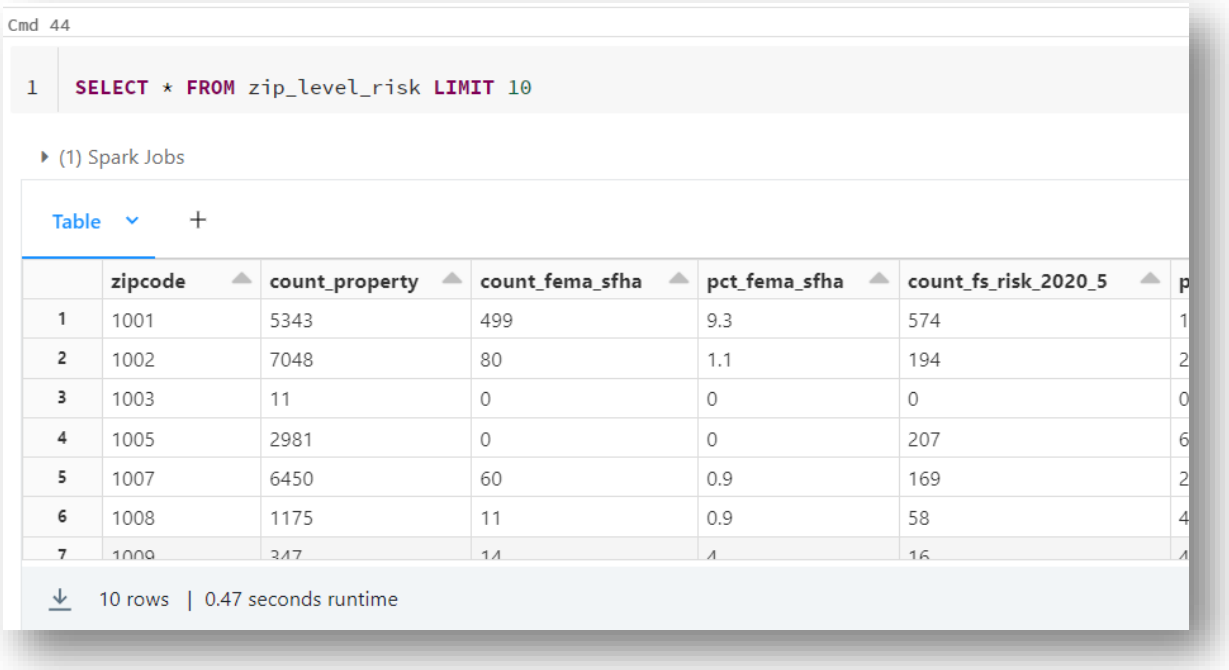
9) “flood.csv” dataset is a sample that has been taken from the First Street Foundation flood model. Check the following link to see this huge project that they gather and analyse data continuously.

https://firststreet.org/data-access/getting-started-with-first-street-data/
https://www.rebuild.nc.gov/media/2256/open

Count property	Total properties
count_fema_sfha	Number of properties in FEMA SFHA
pct_fema_sfha	Percent of properties in FEMA SFHA
count_fs_risk_2020_5	Number of properties flooded in the First Street return period 5 scenario for 2020
pct_fs_risk_2020_5	Percent of properties flooded in the First Street return period 5 scenario for 2020
count_fs_risk_2050_5	Number of properties flooded in the First Street return period 5 scenario for 2050
pct_fs_risk_2050_5	Percent of properties flooded in the First Street return period 5 scenario for 2050
count_fs_risk_2020_100	Number of properties flooded in the First Street return period 100 scenario for 2020
pct_fs_risk_2020_100	Percent of properties flooded in the First Street return period 100 scenario for 2020
count_fs_risk_2050_100	Number of properties flooded in the First Street return period 100 scenario for 2050
pct_fs_risk_2050_100	Percent of properties flooded in the First Street return period 100 scenario for 2050
count_fs_risk_2020_500	Number of properties flooded in the First Street return period 500 scenario for 2020
pct_fs_risk_2020_500	Percent of properties flooded in the First Street return period 500 scenario for 2020
count_fs_risk_2050_500	Number of properties flooded in the First Street return period 500 scenario for 2050
pct_fs_risk_2050_500	Percent of properties flooded in the First Street return period 500 scenario for 2050
count_fs_fema_difference_2020	Difference in number of properties at risk between First Street and FEMA for the 100 year flood in 2020
pct_fs_fema_difference_2020	Percent-point difference in properties at risk between First Street and FEMA for the 100 year flood in 2020
avg_risk_score_all	Average Flood Factor risk score - all properties
avg_risk_score_2_10	Average Flood Factor risk score among properties with risk - properties with risk score 2-10, excluding 1 (minimal risk)
avg_risk_fsf_2020_100	Average Flood Factor risk score - properties with flooding in the First Street return period 100 scenario for 2020
avg_risk_fsf_2020_500	Average Flood Factor risk score - properties with flooding in the First Street return period 500 scenario for 2020
avg_risk_score_sfha	Average Flood Factor risk score - properties within FEMA SFHA
avg_risk_score_no_sfha	Average Flood Factor risk score - properties outside of FEMA SFHA
count_floodfactor1	Number of properties with Flood Factor risk score = 1
count_floodfactor2	Number of properties with Flood Factor risk score = 2
count_floodfactor3	Number of properties with Flood Factor risk score = 3
count_floodfactor4	Number of properties with Flood Factor risk score = 4
count_floodfactor5	Number of properties with Flood Factor risk score = 5
count_floodfactor6	Number of properties with Flood Factor risk score = 6
count_floodfactor7	Number of properties with Flood Factor risk score = 7
count_floodfactor8	Number of properties with Flood Factor risk score = 8
count_floodfactor9	Number of properties with Flood Factor risk score = 9
count_floodfactor10	Number of properties with Flood Factor risk score = 10

The data set has 33 columns and in the above “data description table” you can see a short definition for each variable.

10) Go back to your notebook of this week and check the first 10 rows of the table.



Cmd 44

```
1  SELECT * FROM zip_level_risk LIMIT 10
```

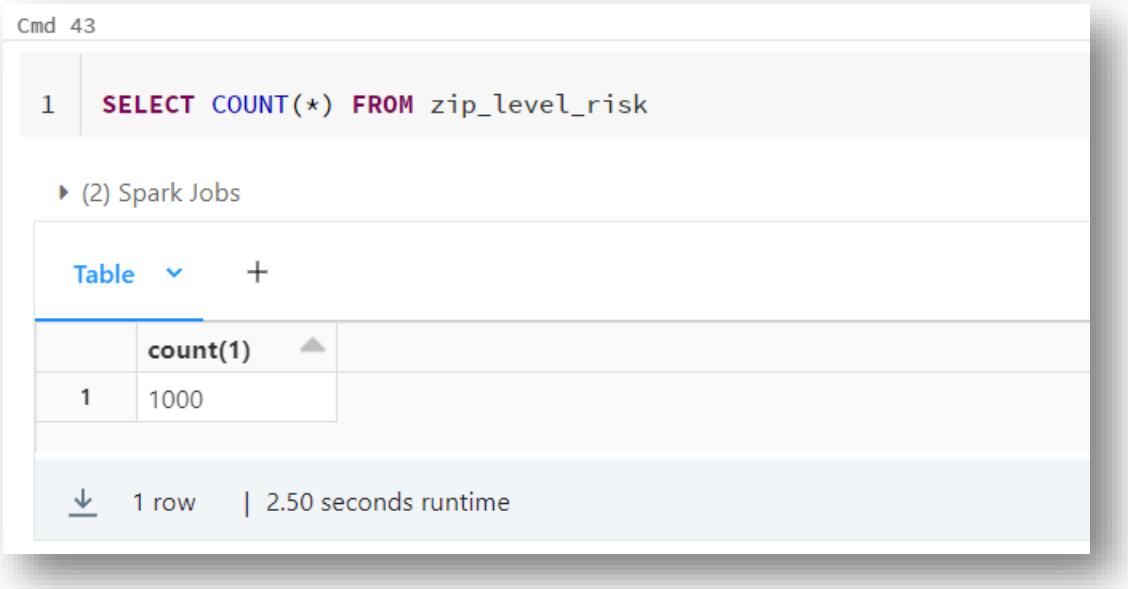
(1) Spark Jobs

Table +

	zipcode	count_property	count_fema_sfha	pct_fema_sfha	count_fs_risk_2020_5	p
1	1001	5343	499	9.3	574	1
2	1002	7048	80	1.1	194	2
3	1003	11	0	0	0	0
4	1005	2981	0	0	207	6
5	1007	6450	60	0.9	169	2
6	1008	1175	11	0.9	58	4
7	1009	347	14	4	16	4

10 rows | 0.47 seconds runtime

11) Use `SELECT` with a `COUNT` to find out the number of rows in the `zip_level_risk` table.



Cmd 43

```
1  SELECT COUNT(*) FROM zip_level_risk
```

(2) Spark Jobs

Table +

	count(1)
1	1000

1 row | 2.50 seconds runtime

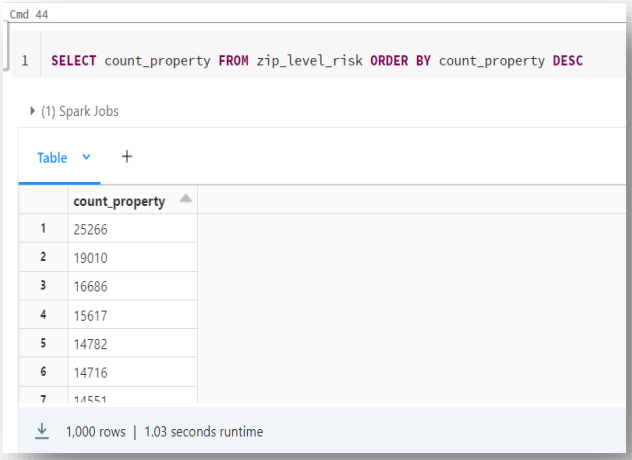
There are 1000 records in this table which is a short version of the main table that has 32000 records.

12) You can use `ORDER BY` to order results of a query by the values in a specific column. For example:

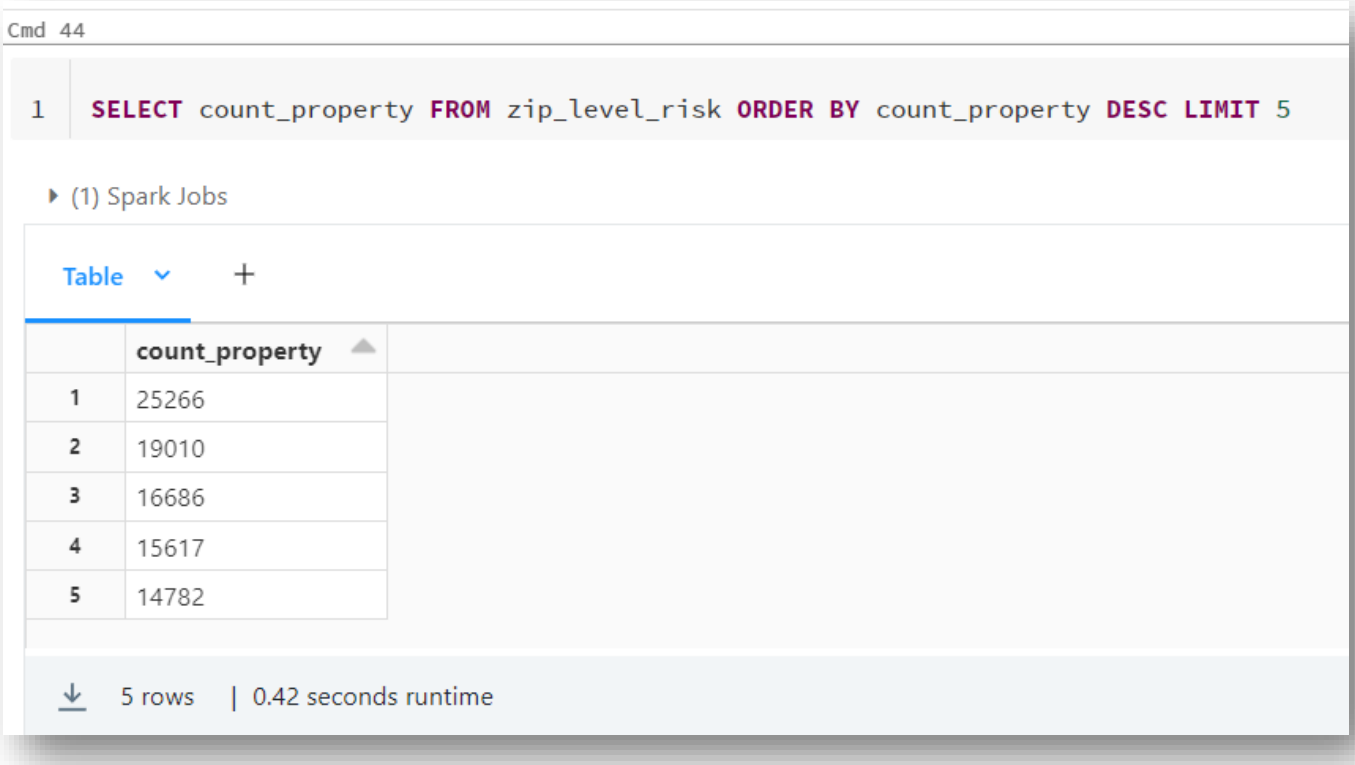
```
SELECT column_x FROM table_y ORDER BY column_x ASC
```

will give us all the values of column_x in table_y with the values presented in an ascending order.

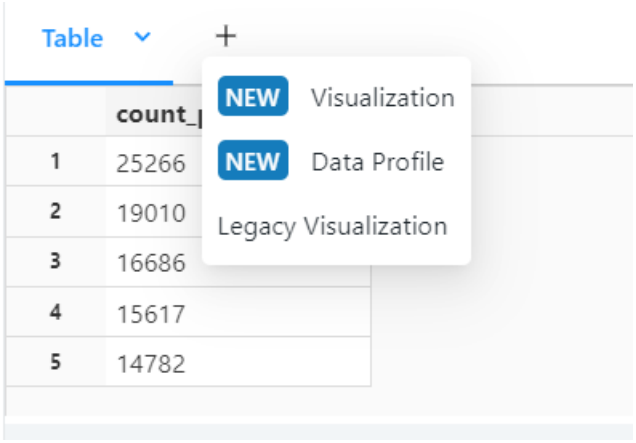
Now, Select the values of count property from the `zip_level_risk` table and order the output in a descending manner.



13) Find an SQL clause which selects a limited number of records. Restrict the number of records returned in the query in the previous point to 5.

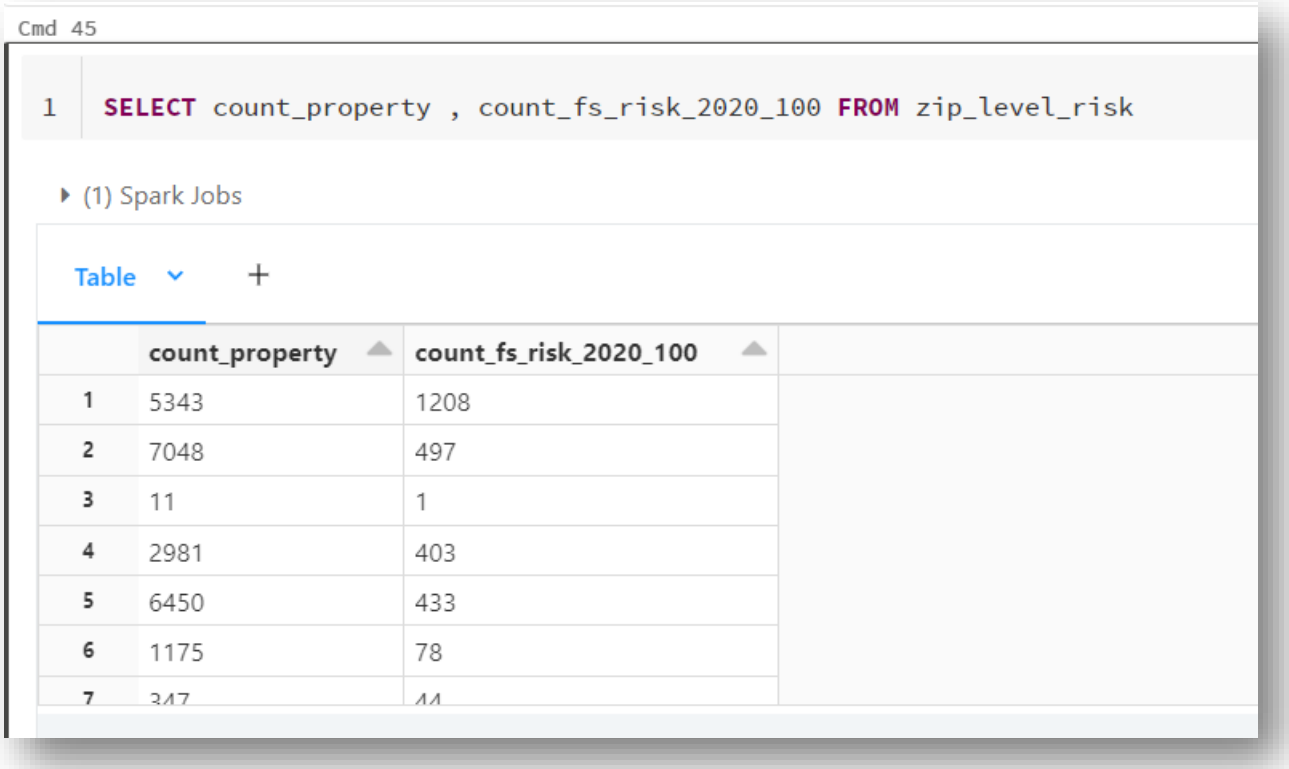


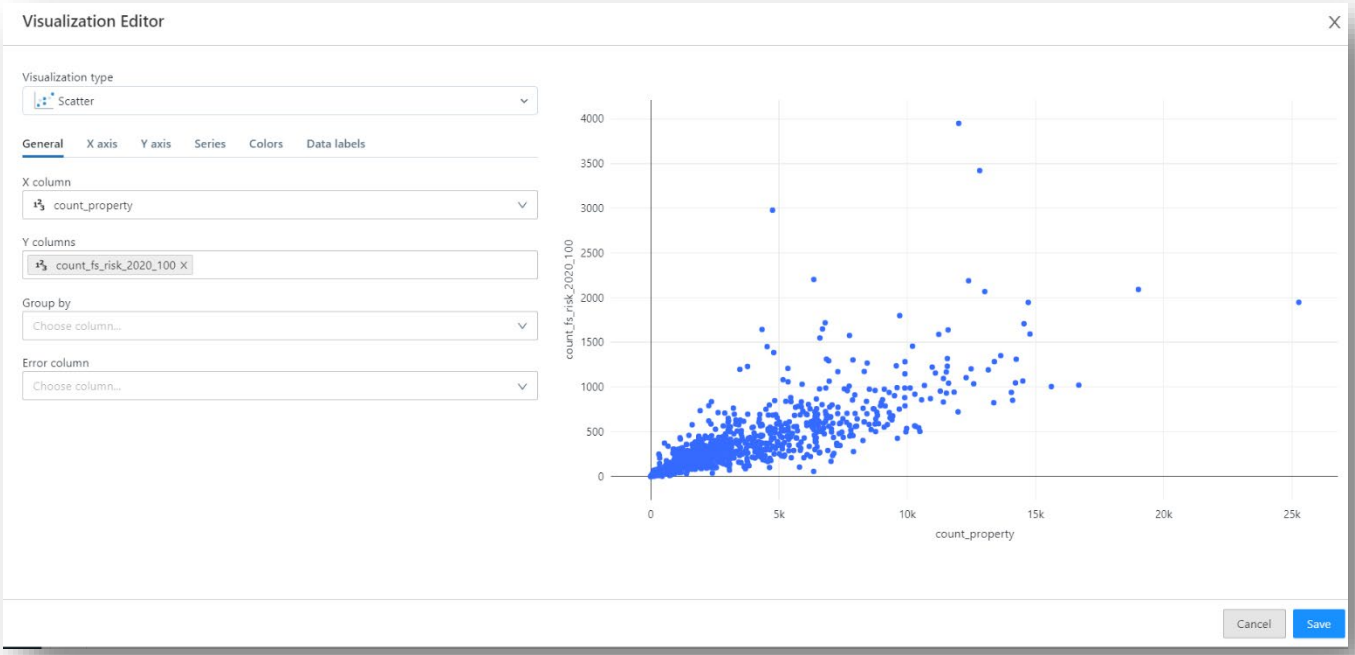
14) The “+” sign on your Databricks results will give more options to generate results other than the tabular format:



Explore Visualization and Data Profile options.

15) If you look at the values of two of the columns in your table (i.e. you SELECT those), you can use the plots (such as a scatter plot) to explore the relationship between the two features. Select the values of `count_property` and `count_fs_risk_2020_100` from your table. Look at the scatter plot these yields.





Now, challenge yourself with a query based on what you've learned today.

Challenge 2:

Compare average flood risk scores between two regions "`within FEMA SFHA`" and "`outside of FEMA SFHA`". To do this you should find relevant variables in the data description table and then write a query in SPARK SQL and visualise the result. You should compute a metric and then use a proper chart to compare two regions.

Challenge 3:

Each property in each zip code has got a **Flood Risk Factor Scores** from 1 of 10. For example, in the zip code 1001 there are 5343 properties and breakdown of the Flood Risk Factor Scores is:

	A	B	X	Y	Z	AA	AB	AC	AD	AE	AF	AG
1	zipcode	count_property	count_floodfactor1	count_floodfactor2	count_floodfactor3	count_floodfactor4	count_floodfactor5	count_floodfactor6	count_floodfactor7	count_floodfactor8	count_floodfactor9	count_floodfactor10
2	1001	5343	3813	52	104	122	40	279	234	15	144	540
3	1002	7048	6378	15	53	87	34	172	94	14	98	103
4	1003	11	9	0	1	0	0	1	0	0	0	0
5	1005	2981	2498	7	20	40	20	100	74	18	104	100

Total	count_floodfactor1	count_floodfactor2	count_floodfactor3	count_floodfactor4	count_floodfactor5	count_floodfactor6	count_floodfactor7	count_floodfactor8	count_floodfactor9	count_floodfactor10
5343	3813	52	104	122	40	279	234	15	144	540

3813 properties out of the **5343** properties have got a low risk factor (flood factor 1).

No, you should write an SQL query to calculate which `zipcode` relatively has the highest percentage of the risk factor 10.

Clue: You should compute ratio of the score 10 to total in each `zipcode` and then sort `zipcodes` by this new measure.

Result should look like the following table. `zipcode` 4344 has the highest percentage of the worst score. However this `zipcode` has only 2 properties. It seems people don't like to live in this `zipcode` 😊

Table +

	zipcode	count_property	Risk_10_percentage
1	4344	2	50
2	4063	534	36
3	1840	675	33.2
4	1342	1156	21.5
5	3812	1140	18.3
6	3581	1923	17.6
7	1952	4335	17.3

References and Resources:

https://docs.databricks.com/sql/index.html

https://sparkbyexamples.com/pyspark/pyspark-sql-with-examples/#

https://docs.databricks.com/sql/language-manual/index.html

Appendix:

If you don’t have webpage.zip in the DBFS upload the zipped file to Databricks and follow the steps:

```
fileroot = "webpage"

import os
os.environ ['fileroot'] = fileroot
```

```
## Clean the local file system from the contents that the notebook needs to create again

%sh

rm -r   /tmp/$fileroot
rm     /tmp/$fileroot.zip
```

```
## Clean the DBFS from the contents that the notebook needs to create again

dbutils.fs.rm("/FileStore/tables/" + fileroot , True)
```

```
dbutils.fs.cp("/FileStore/tables/" + fileroot + ".zip", "file:/tmp/")
```

```
%sh

unzip -d   /tmp  /tmp/$fileroot.zip
```

```
dbutils.fs.mkdirs("/FileStore/tables/" + fileroot )

dbutils.fs.mv("file:/tmp/" + fileroot , "/FileStore/tables/" + fileroot , True )
```