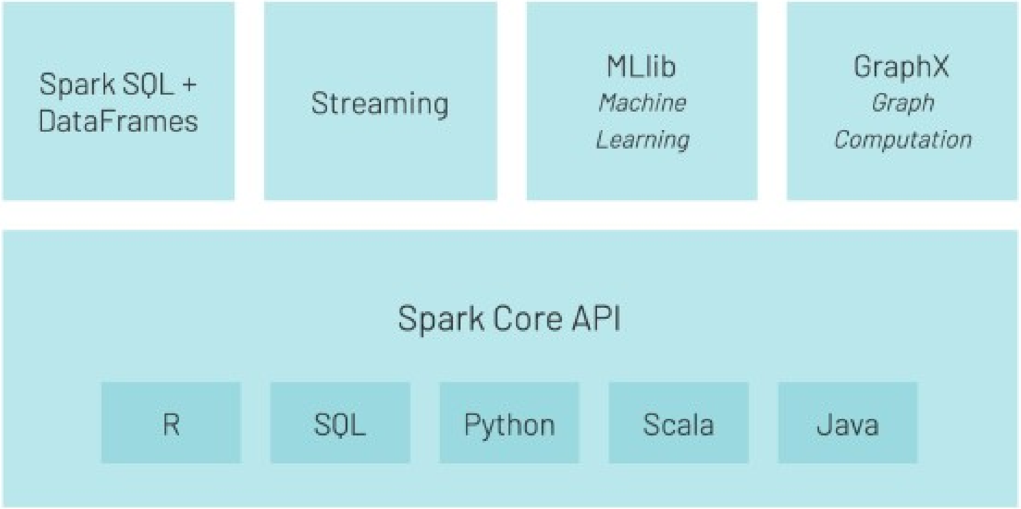
Yesterday we took a closer look into Spark Scala with notebooks in Azure Databricks and how to handle data engineering. Today we will look into the Spark SQL and DataFrames that is using Spark Core API.



"Spark SQL is a spark module for structured data processing and data querying. It provides programming abstraction called DataFrames and can also serve as distributed SQL query engine. It enables unmodified Hadoop Hive queries to run up to 100x faster on existing deployments and data. It also provides powerful integration with the rest of the Spark ecosystem (e.g.: integrating SQL query processing with machine learning)." (Apache Spark Tutorial).

Start your Azure Databricks workspace and create new Notebook. I named mine as: *Day22_SparkSQL* and set the language: *SQL*. Now let's explore the functionalities of Spark SQL.

1.Loading Data

We will load data from */databricks-datasets* using Spark SQL, R and Python languages. The CSV dataset will be **data_geo.csv** in the following folder:

```
%scala
display(dbutils.fs.ls("/databricks-datasets/samples/population-vs-price"))
```

1.1. Loading using Python

```
%python
data = spark.read.csv("/databricks-datasets/samples/population-vs-
price/data_geo.csv", header="true", inferSchema="true")
```

And materialize the data using to create a view with name *data_geo_py*:

```
%python
data.createOrReplaceTempView("data_geo_py")
```

And run the following SQL Statement:

```
SELECT * FROM data_geo_py LIMIT 10
```

1.2. Loading using SQL

```
DROP TABLE IF EXISTS data_geo;

CREATE TABLE data_geo
USING com.databricks.spark.csv
OPTIONS (path "/databricks-datasets/samples/population-vs-price/data_geo.csv",
header "true", inferSchema "true")
```

And run the following SQL Statement:

```
SELECT * FROM data_geo LIMIT 10
```

1.3. Loading using R

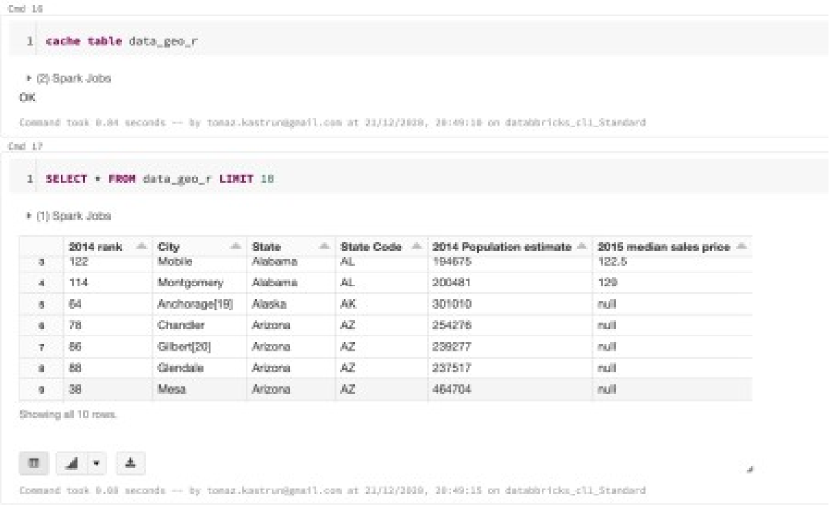
```
%r
library(SparkR)
data_geo_r <- read.df("/databricks-datasets/samples/population-vs-price/
data_geo.csv", source = "csv", header="true", inferSchema = "true")
registerTempTable(data_geo_r, "data_geo_r")
```

Cache the results:

```
CACHE TABLE data_geo_r
```

And run the following SQL Statement:

```
SELECT * FROM data_geo_r LIMIT 10
```



Cmd 16

```
cache table data_geo_r
```

▸ (2) Spark Jobs

OK

Command took 0.04 seconds -- by tomaz.kastrun@gmail.com at 21/12/2020, 20:49:10 on databbricks_cl1_Standard

Cmd 17

```
SELECT * FROM data_geo_r LIMIT 10
```

▸ (1) Spark Jobs

	2014 rank	City	State	State Code	2014 Population estimate	2015 median sales price
3	122	Mobile	Alabama	AL	194675	122.5
4	114	Montgomery	Alabama	AL	200481	129
5	64	Anchorage[19]	Alaska	AK	301010	null
6	78	Chandler	Arizona	AZ	254276	null
7	86	Gilbert[20]	Arizona	AZ	239277	null
8	88	Glendale	Arizona	AZ	237517	null
9	38	Mesa	Arizona	AZ	464704	null

Showing all 10 rows.

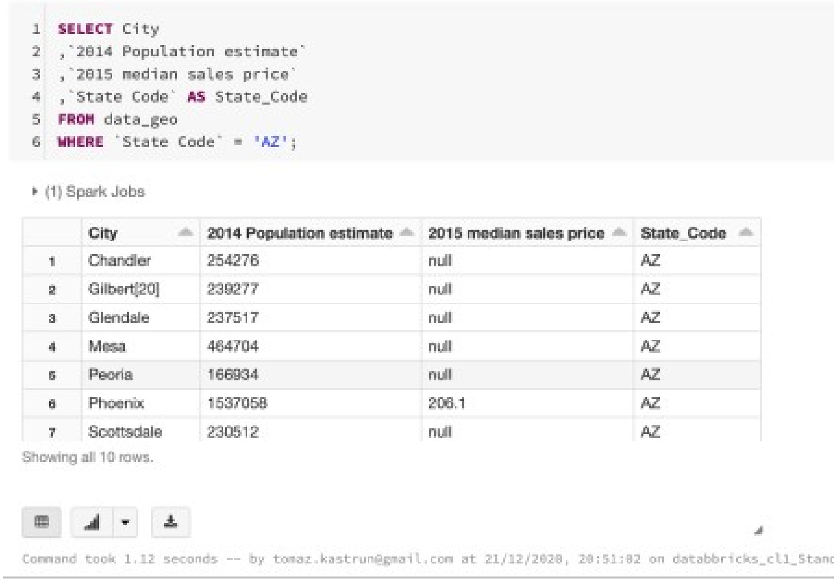
Command took 0.08 seconds -- by tomaz.kastrun@gmail.com at 21/12/2020, 20:49:15 on databbricks_cl1_Standard

All three DataFrames are the same (unless additional modification are done; like: dropping rows with null values, etc).

2.Viewing DataFrame

Viewing DataFrame is done by simple SELECT statement, the ANSI SQL Standard. E.g.:

```
SELECT City
,`2014 Population estimate`
,`2015 median sales price`
,`State Code` AS State_Code
FROM data_geo
WHERE `State Code` = 'AZ';
```



```
SELECT City
,`2014 Population estimate`
,`2015 median sales price`
,`State Code` AS State_Code
FROM data_geo
WHERE `State Code` = 'AZ';
```

▸ (1) Spark Jobs

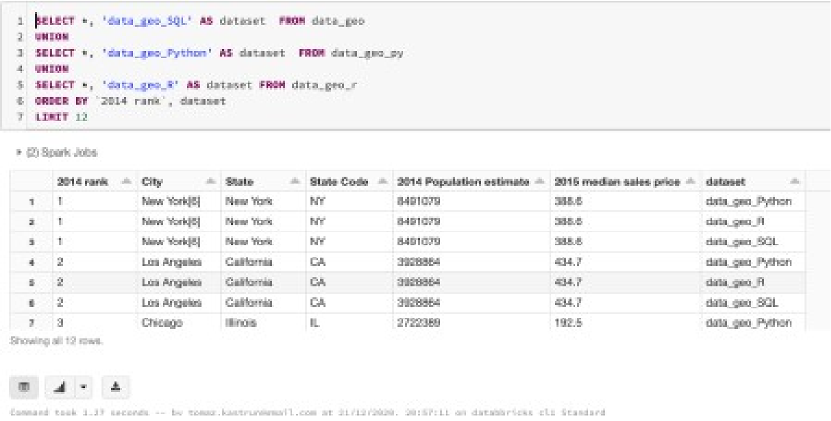
	City	2014 Population estimate	2015 median sales price	State_Code
1	Chandler	254276	null	AZ
2	Gilbert[20]	239277	null	AZ
3	Glendale	237517	null	AZ
4	Mesa	464704	null	AZ
5	Peoria	166934	null	AZ
6	Phoenix	1537058	206.1	AZ
7	Scottsdale	230512	null	AZ

Showing all 10 rows.

Command took 1.12 seconds -- by tomaz.kastrun@gmail.com at 21/12/2020, 20:51:02 on databbricks_cl1_Stanc

You can also combine all three DataFrames that were imported using three different languages (SQL, R, Python).

```
SELECT *, 'data_geo_SQL' AS dataset  FROM data_geo
UNION
SELECT *, 'data_geo_Python' AS dataset  FROM data_geo_py
UNION
SELECT *, 'data_geo_R' AS dataset FROM data_geo_r
ORDER BY `2014 rank`, dataset
LIMIT 12
```



```
SELECT *, 'data_geo_SQL' AS dataset  FROM data_geo
UNION
SELECT *, 'data_geo_Python' AS dataset  FROM data_geo_py
UNION
SELECT *, 'data_geo_R' AS dataset FROM data_geo_r
ORDER BY `2014 rank`, dataset
LIMIT 12
```

▸ (2) Spark Jobs

	2014 rank	City	State	State Code	2014 Population estimate	2015 median sales price	dataset
1	1	New York[6]	New York	NY	8491079	388.6	data_geo_Python
2	1	New York[6]	New York	NY	8491079	388.6	data_geo_R
3	1	New York[6]	New York	NY	8491079	388.6	data_geo_SQL
4	2	Los Angeles	California	CA	3928864	434.7	data_geo_Python
5	2	Los Angeles	California	CA	3928864	434.7	data_geo_R
6	2	Los Angeles	California	CA	3928864	434.7	data_geo_SQL
7	3	Chicago	Illinois	IL	2722389	192.5	data_geo_Python

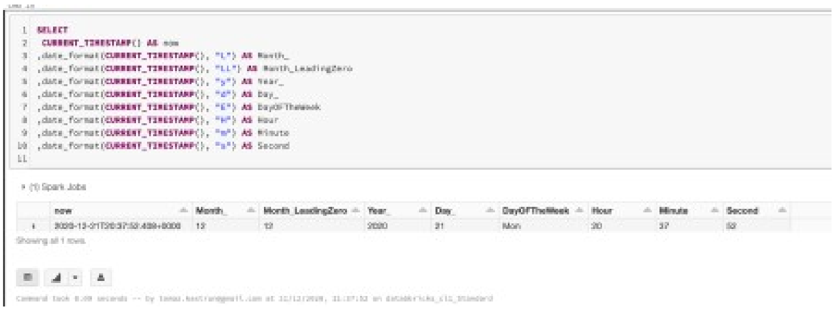
Showing all 12 rows.

Command took 1.37 seconds -- by tomaz.kastrun@gmail.com at 21/12/2020, 20:57:11 on databbricks cl1 Standard

3.Running SQL

3.1. Date and Time functions

```
SELECT
 CURRENT_TIMESTAMP() AS now
,date_format(CURRENT_TIMESTAMP(), "L") AS Month_
,date_format(CURRENT_TIMESTAMP(), "LL") AS Month_LeadingZero
,date_format(CURRENT_TIMESTAMP(), "y") AS Year_
,date_format(CURRENT_TIMESTAMP(), "d") AS Day_
,date_format(CURRENT_TIMESTAMP(), "E") AS DayOFTheWeek
,date_format(CURRENT_TIMESTAMP(), "H") AS Hour
,date_format(CURRENT_TIMESTAMP(), "m") AS Minute
,date_format(CURRENT_TIMESTAMP(), "s") AS Second
```



```
SELECT
 CURRENT_TIMESTAMP() AS now
,date_format(CURRENT_TIMESTAMP(), "L") AS Month_
,date_format(CURRENT_TIMESTAMP(), "LL") AS Month_LeadingZero
,date_format(CURRENT_TIMESTAMP(), "y") AS Year_
,date_format(CURRENT_TIMESTAMP(), "d") AS Day_
,date_format(CURRENT_TIMESTAMP(), "E") AS DayOfTheWeek
,date_format(CURRENT_TIMESTAMP(), "H") AS Hour
,date_format(CURRENT_TIMESTAMP(), "m") AS Minute
,date_format(CURRENT_TIMESTAMP(), "s") AS Second
```

▸ (1) Spark Jobs

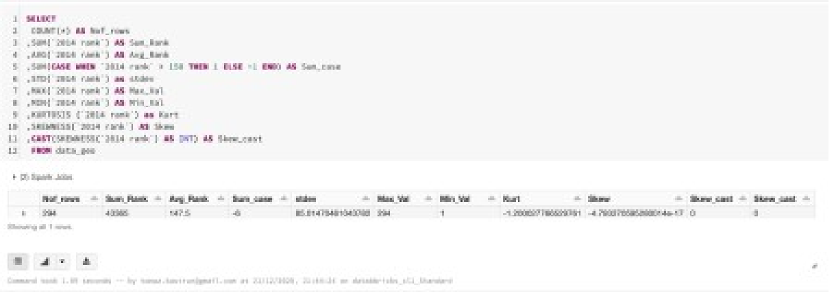
	now	Month_	Month_LeadingZero	Year_	Day_	DayOfTheWeek	Hour	Minute	Second
1	2020-12-21T20:37:52.489+0000	12	12	2020	21	Mon	20	37	52

Showing all 1 rows.

Command took 0.09 seconds -- by tomaz.kastrun@gmail.com at 21/12/2020, 21:37:52 on databricks_c11_Standard

3.2. Built-in functions

```
SELECT
 COUNT(*) AS Nof_rows
,SUM(`2014 rank`) AS Sum_Rank
,AVG(`2014 rank`) AS Avg_Rank
,SUM(CASE WHEN `2014 rank` > 150 THEN 1 ELSE -1 END) AS Sum_case
,STD(`2014 rank`) as stdev
,MAX(`2014 rank`) AS Max_Val
,MIN(`2014 rank`) AS Min_Val
,KURTOSIS (`2014 rank`) as Kurt
,SKEWNESS(`2014 rank`) AS Skew
,CAST(SKEWNESS(`2014 rank`) AS INT) AS Skew_cast
 FROM data_geo
```



```
SELECT
 COUNT(*) AS Nof_rows
,SUM(`2014 rank`) AS Sum_Rank
,AVG(`2014 rank`) AS Avg_Rank
,SUM(CASE WHEN `2014 rank` > 150 THEN 1 ELSE -1 END) AS Sum_case
,STD(`2014 rank`) as stdev
,MAX(`2014 rank`) AS Max_Val
,MIN(`2014 rank`) AS Min_Val
,KURTOSIS (`2014 rank`) as Kurt
,SKEWNESS(`2014 rank`) AS Skew
,CAST(SKEWNESS(`2014 rank`) AS INT) AS Skew_cast
 FROM data_geo
```

▸ (2) Spark Jobs

	Nof_rows	Sum_Rank	Avg_Rank	Sum_case	stdev	Max_Val	Min_Val	Kurt	Skew	Skew_cast	Skew_cast
1	294	43365	147.5	-6	85.01470481043783	294	1	-1.2000277665529781	-4.793370595280014e-17	0	0

Showing all 1 rows.

Command took 1.05 seconds -- by tomaz.kastrun@gmail.com at 21/12/2020, 21:44:24 on databricks_c11_Standard

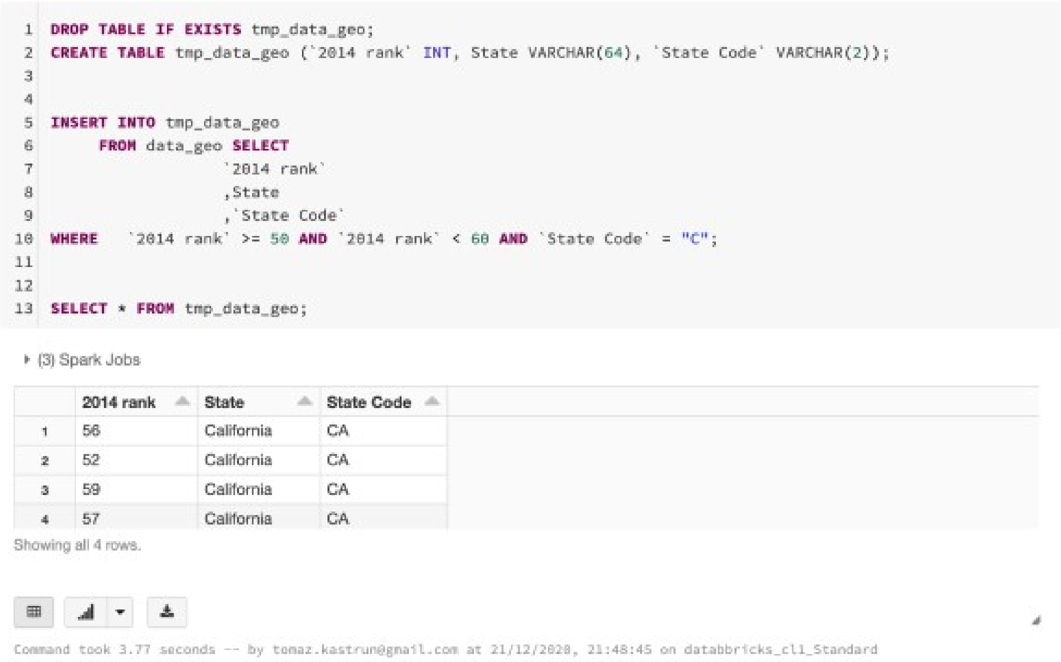
3.3. SELECT INTO

You can also store results using SELECT INTO statement, with table being predifined:

```
DROP TABLE IF EXISTS tmp_data_geo;
CREATE TABLE tmp_data_geo (`2014 rank` INT, State VARCHAR(64), `State Code`
VARCHAR(2));


INSERT INTO tmp_data_geo
     FROM data_geo SELECT
                  `2014 rank`
                  ,State
                  ,`State Code`
WHERE   `2014 rank` >= 50 AND `2014 rank` < 60 AND `State Code` = "C";


SELECT * FROM tmp_data_geo;
```



```
DROP TABLE IF EXISTS tmp_data_geo;
CREATE TABLE tmp_data_geo (`2014 rank` INT, State VARCHAR(64), `State Code` VARCHAR(2));


INSERT INTO tmp_data_geo
     FROM data_geo SELECT
                  `2014 rank`
                  ,State
                  ,`State Code`
WHERE   `2014 rank` >= 50 AND `2014 rank` < 60 AND `State Code` = "C";


SELECT * FROM tmp_data_geo;
```

▸ (3) Spark Jobs

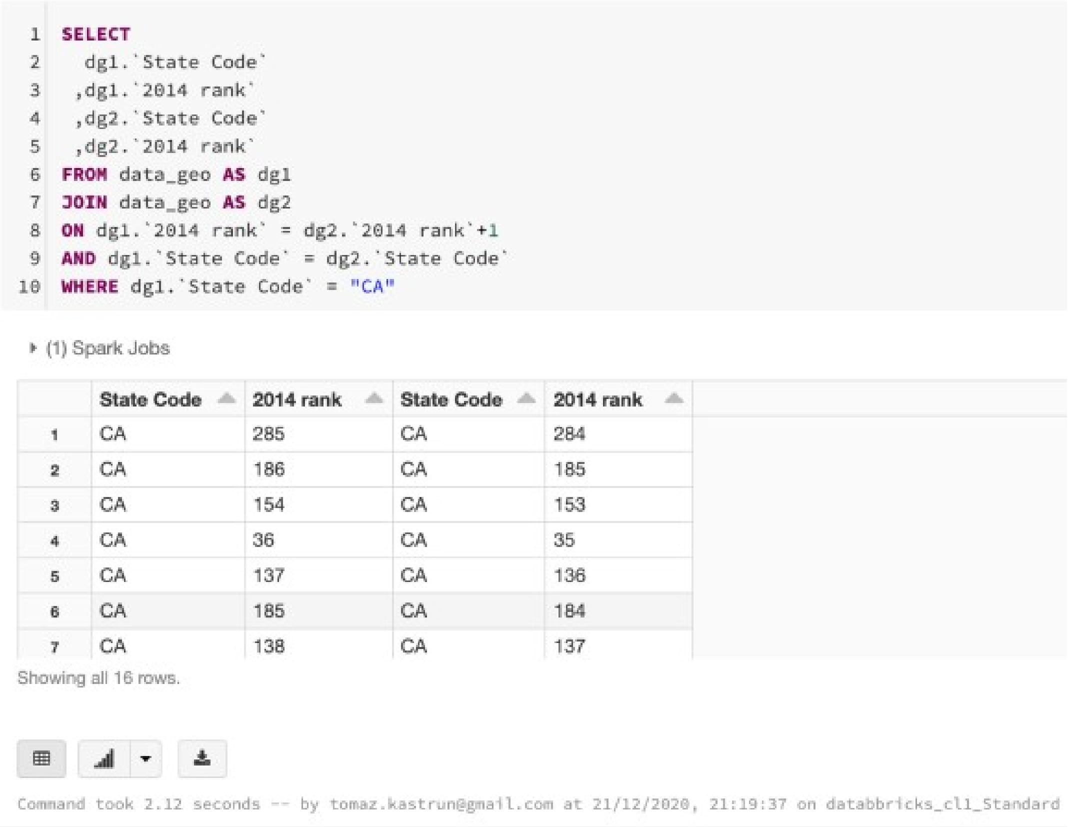
	2014 rank	State	State Code
1	56	California	CA
2	52	California	CA
3	59	California	CA
4	57	California	CA

Showing all 4 rows.

```
Command took 3.77 seconds -- by tomaz.kastrun@gmail.com at 21/12/2020, 21:48:45 on databbricks_cl1_Standard
```

3.4. JOIN

```
SELECT
  dg1.`State Code`
 ,dg1.`2014 rank`
 ,dg2.`State Code`
 ,dg2.`2014 rank`
FROM data_geo AS dg1
JOIN data_geo AS dg2
ON dg1.`2014 rank` = dg2.`2014 rank`+1
AND dg1.`State Code` = dg2.`State Code`
WHERE dg1.`State Code` = "CA"
```



```
SELECT
  dg1.`State Code`
 ,dg1.`2014 rank`
 ,dg2.`State Code`
 ,dg2.`2014 rank`
FROM data_geo AS dg1
JOIN data_geo AS dg2
ON dg1.`2014 rank` = dg2.`2014 rank`+1
AND dg1.`State Code` = dg2.`State Code`
WHERE dg1.`State Code` = "CA"
```

▸ (1) Spark Jobs

	State Code	2014 rank	State Code	2014 rank
1	CA	285	CA	284
2	CA	186	CA	185
3	CA	154	CA	153
4	CA	36	CA	35
5	CA	137	CA	136
6	CA	185	CA	184
7	CA	138	CA	137

Showing all 16 rows.

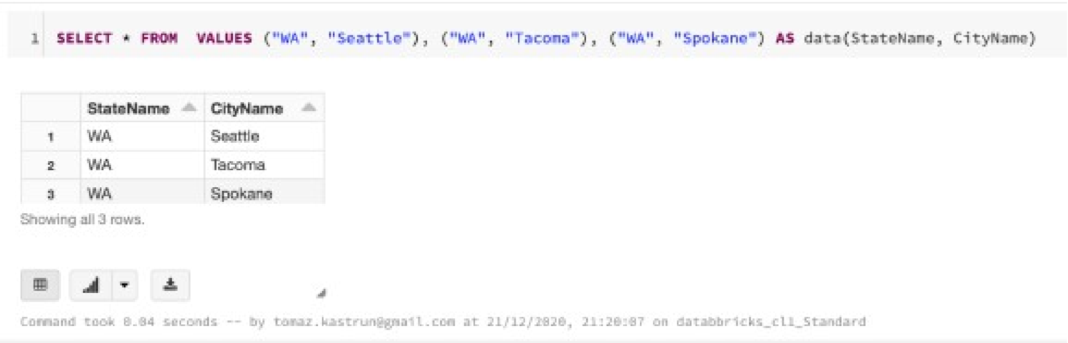
```
Command took 2.12 seconds -- by tomaz.kastrun@gmail.com at 21/12/2020, 21:19:37 on databbricks_cl1_Standard
```

3.5. Common Table Expressions

```
WITH cte AS (
    SELECT * FROM data_geo
    WHERE `2014 rank` >= 50 AND `2014 rank` < 60
)
SELECT * FROM cte;
```

3.6. Inline tables

```
SELECT * FROM  VALUES
("WA", "Seattle"),
("WA", "Tacoma"),
("WA", "Spokane") AS data(StateName, CityName)
```



```
SELECT * FROM  VALUES ("WA", "Seattle"), ("WA", "Tacoma"), ("WA", "Spokane") AS data(StateName, CityName)
```

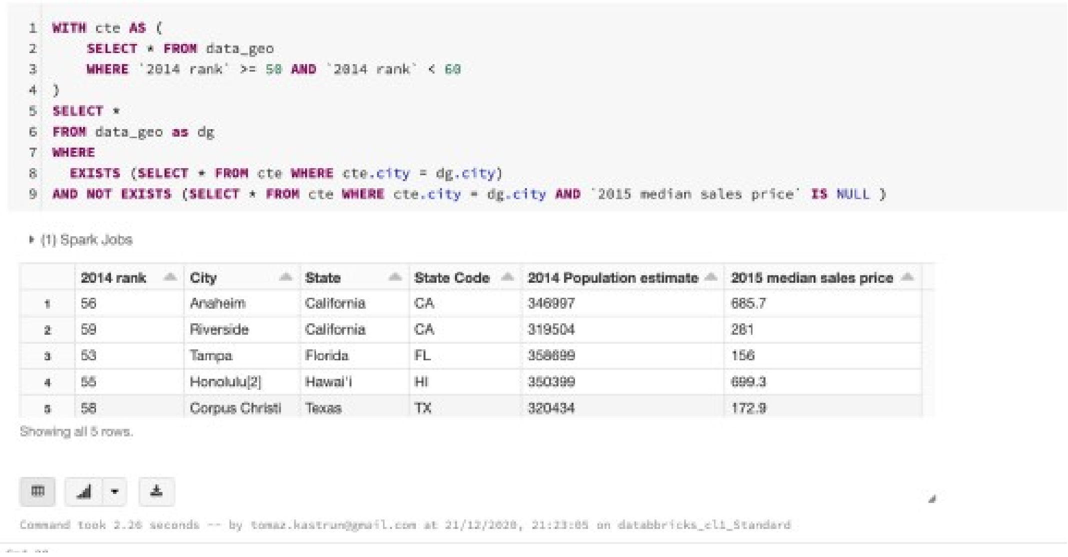
	StateName	CityName
1	WA	Seattle
2	WA	Tacoma
3	WA	Spokane

Showing all 3 rows.

```
Command took 0.04 seconds -- by tomaz.kastrun@gmail.com at 21/12/2020, 21:20:07 on databbricks_cl1_Standard
```

3.7. EXISTS

```
WITH cte AS (
    SELECT * FROM data_geo
    WHERE `2014 rank` >= 50 AND `2014 rank` < 60
)
SELECT *
FROM data_geo as dg
WHERE
  EXISTS (SELECT * FROM cte WHERE cte.city = dg.city)
AND NOT EXISTS (SELECT * FROM cte WHERE cte.city = dg.city AND `2015 median
sales price` IS NULL )
```



```
WITH cte AS (
    SELECT * FROM data_geo
    WHERE `2014 rank` >= 50 AND `2014 rank` < 60
)
SELECT *
FROM data_geo as dg
WHERE
  EXISTS (SELECT * FROM cte WHERE cte.city = dg.city)
AND NOT EXISTS (SELECT * FROM cte WHERE cte.city = dg.city AND `2015 median sales price` IS NULL )
```

▸ (1) Spark Jobs

	2014 rank	City	State	State Code	2014 Population estimate	2015 median sales price
1	56	Anaheim	California	CA	346997	685.7
2	59	Riverside	California	CA	319504	281
3	53	Tampa	Florida	FL	358699	156
4	55	Honolulu[2]	Hawai'i	HI	350399	699.3
5	58	Corpus Christi	Texas	TX	320434	172.9

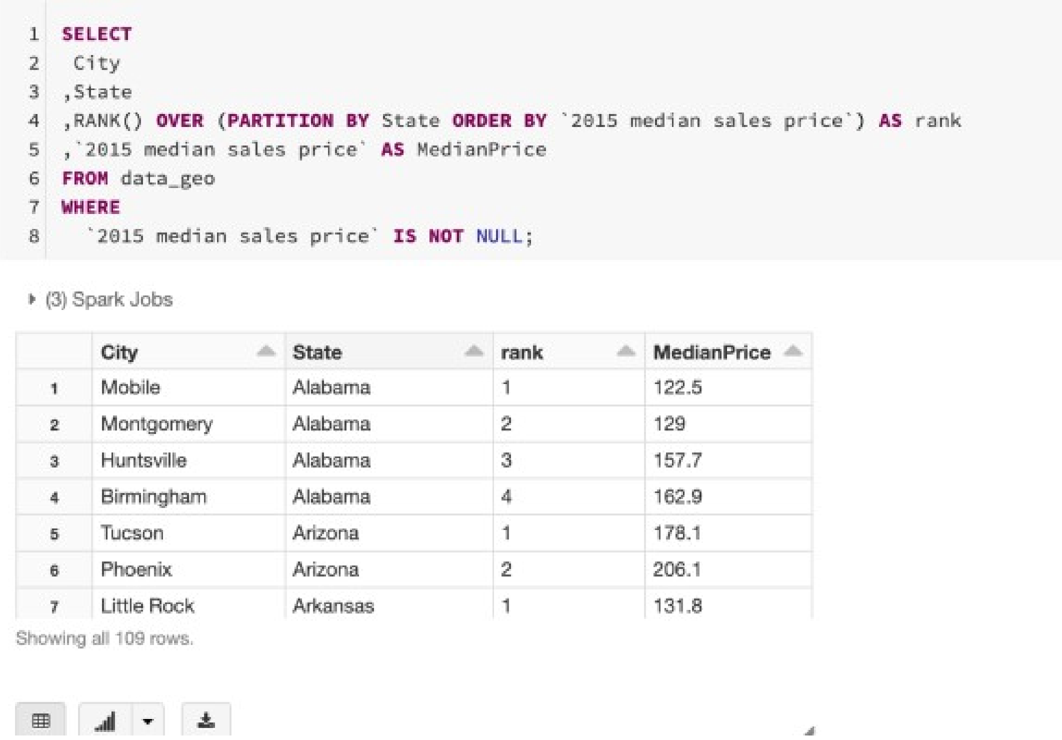
Showing all 5 rows.

```
Command took 2.26 seconds -- by tomaz.kastrun@gmail.com at 21/12/2020, 21:23:05 on databbricks_cl1_Standard
```

3.8.Window functions

```
SELECT
```

```
 City
,State
,RANK() OVER (PARTITION BY State ORDER BY `2015 median sales price`) AS rank
,`2015 median sales price` AS MedianPrice
FROM data_geo
WHERE
  `2015 median sales price` IS NOT NULL;
```



```
SELECT
 City
,State
,RANK() OVER (PARTITION BY State ORDER BY `2015 median sales price`) AS rank
,`2015 median sales price` AS MedianPrice
FROM data_geo
WHERE
  `2015 median sales price` IS NOT NULL;
```

(3) Spark Jobs

	City	State	rank	MedianPrice
1	Mobile	Alabama	1	122.5
2	Montgomery	Alabama	2	129
3	Huntsville	Alabama	3	157.7
4	Birmingham	Alabama	4	162.9
5	Tucson	Arizona	1	178.1
6	Phoenix	Arizona	2	206.1
7	Little Rock	Arkansas	1	131.8

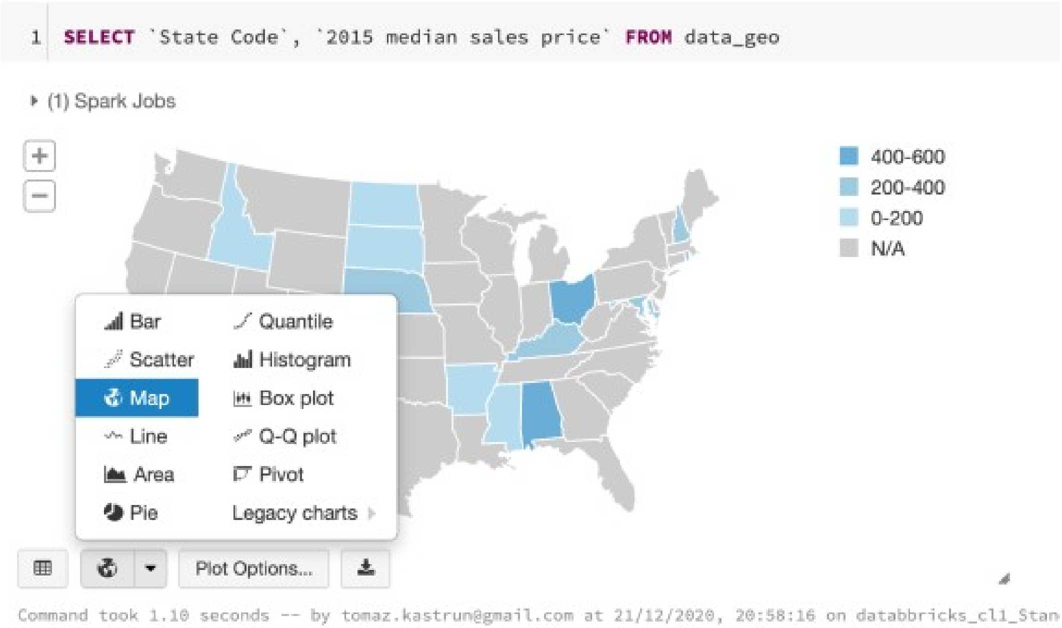
Showing all 109 rows.

4. Exploring the visuals

Results of a SQL SELECT statements that are returned as a table, can also be visualised. Given the following SQL Statement:

```
SELECT `State Code`, `2015 median sales price` FROM data_geo
```

in the result cell you can select the plot icon and pick Map.



```
SELECT `State Code`, `2015 median sales price` FROM data_geo
```

(1) Spark Jobs

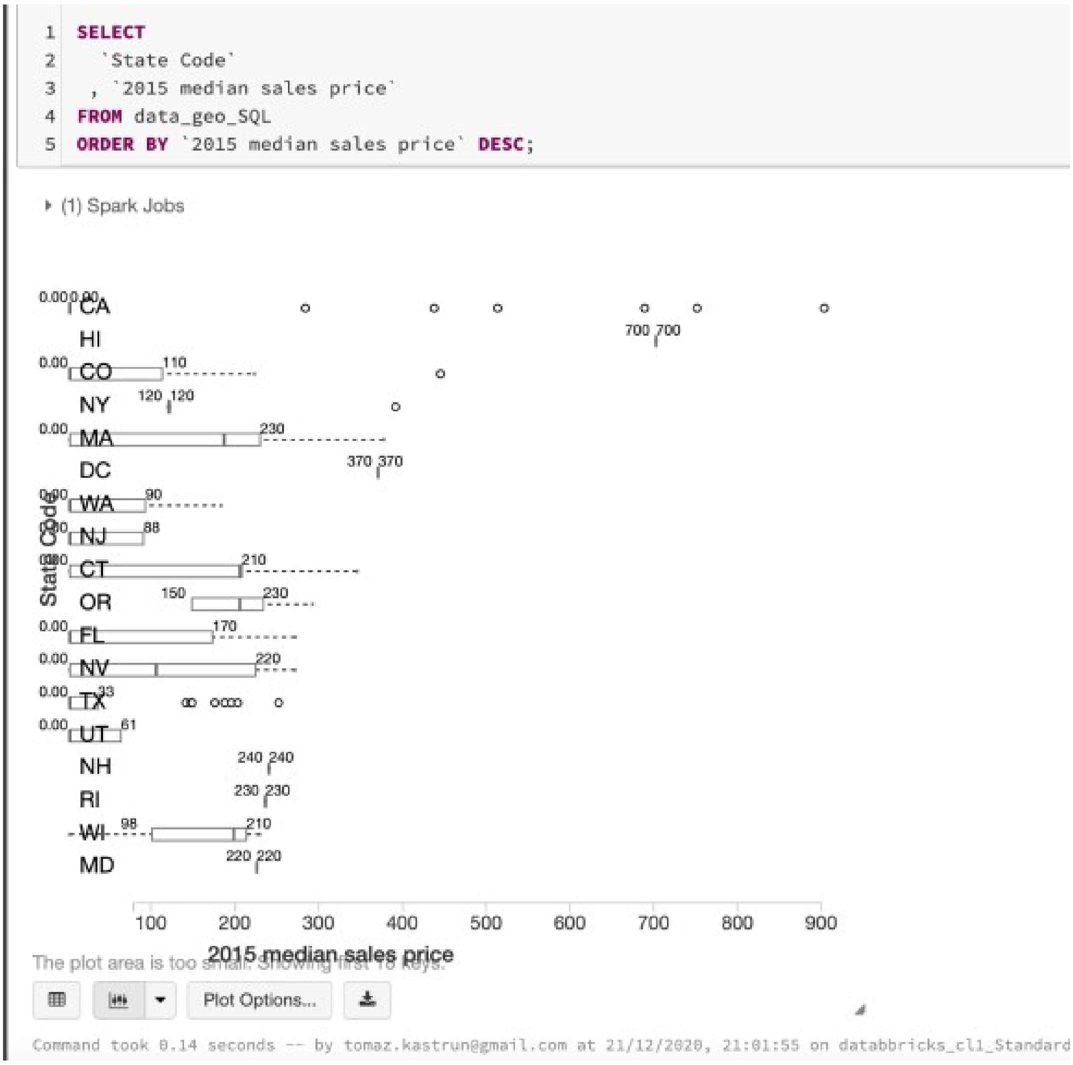
Command took 1.10 seconds -- by tomaz.kastrun@gmail.com at 21/12/2020, 20:58:16 on databbricks_cl1_Stan

Furthermore, using “Plot Options…” you can change the settings of the variables on the graph, aggregations and data series.

With additional query:

```
SELECT
   `State Code`
 , `2015 median sales price`
FROM data_geo_SQL
ORDER BY `2015 median sales price` DESC;
```

you can also create a box-plot; again selecting the desired plot type.



```
SELECT
  `State Code`
 , `2015 median sales price`
FROM data_geo_SQL
ORDER BY `2015 median sales price` DESC;
```

▸ (1) Spark Jobs

The plot area is too small. Showing first 10 keys.

Command took 0.14 seconds -- by tomaz.kastrun@gmail.com at 21/12/2020, 21:01:55 on databbricks_cl1_Standard

There are also many other visuals available and much more SQL statements to explore and feel free to go a step further and beyond this blogpost.