Module 22 home_sales challenge - screen shots v1
Raj Agrawal / SMU DS / Sep 2023
===

Challenge – To use, SparkSQL to determine key metrics about home sales data. Then use Spark to create temporary views, partition the data, cache and un-cache a temporary table, and verify that the table has been un-cached.

Deliverable :

1. Rename the Home_Sales_starter_code.ipynb file as **Home_Sales.ipynb.**
2. Import the necessary PySpark SQL functions for this assignment.
3. Read the home_sales_revised.csv data in the starter code into a Spark DataFrame.
4. Create a temporary table called home_sales.
5. Answer the following questions using SparkSQL:
    - What is the average price for a four-bedroom house sold for each year? Round off your answer to two decimal places.

```
+----+-------------+
|YEAR|AVERAGE_PRICE|
+----+-------------+
|2022|    296363.88|
|2021|    301819.44|
|2020|    298353.78|
|2019|     300263.7|
+----+-------------+
```

    - What is the average price of a home for each year it was built that has three bedrooms and three bathrooms? Round off your answer to two decimal places.

```
+----+-------------+
|YEAR|AVERAGE_PRICE|
+----+-------------+
|2017|    292676.79|
|2016|    290555.07|
|2015|     288770.3|
|2014|    290852.27|
|2013|    295962.27|
|2012|    293683.19|
|2011|    291117.47|
|2010|    292859.62|
+----+-------------+
```

- What is the average price of a home for each year that has three bedrooms, three bathrooms, two floors, and is greater than or equal to 2,000 square feet? Round off your answer to two decimal places.

YEAR_BUILT	AVERAGE_PRICE
2017	280317.58
2016	293965.1
2015	297609.97
2014	298264.72
2013	303676.79
2012	307539.97
2011	276553.81
2010	285010.22

- What is the "view" rating for homes costing more than or equal to $350,000? Determine the run time for this query, and round off your answer to two decimal places.

view	AVERAGE_PRICE
99	1061201.42
98	1053739.33
97	1129040.15
96	1017815.92
95	1054325.6
94	1033536.2
93	1026006.06
92	970402.55
91	1137372.73
90	1062654.16
89	1107839.15
88	1031719.35
87	1072285.2
86	1070444.25
85	1056336.74
84	1117233.13
83	1033965.93
82	1063498.0
81	1053472.79
80	991767.38

==========<<pls see below screen-shot for the answer>>>==========

6. Cache your temporary table home_sales.
7. Check if your temporary table is cached.
8. Using the cached data, run the query that filters out the view ratings with an average price of greater than or equal to $350,000. Determine the runtime and compare it to uncached runtime.
9. Partition by the "date_built" field on the formatted parquet home sales data.
10. Create a temporary table for the parquet data.
11. Run the query that filters out the view ratings with an average price of greater than or equal to $350,000. Determine the runtime and compare it to uncached runtime.
12. Uncache the home_sales temporary table.
13. Verify that the home_sales temporary table is uncached using PySpark.
14. Download your Home_Sales.ipynb file and upload it into your "Home_Sales" GitHub repository.

<<pls see below screen-shot for the answer>>>

```
import os
# Find the latest version of spark 3.x  from http://www.apache.org/dist/spark/ and enter as the spark version
# For example:
# spark_version = 'spark-3.4.0'
spark_version = 'spark-3.4.0'
os.environ['SPARK_VERSION']=spark_version

# Install Spark and Java
!apt-get update
!apt-get install openjdk-11-jdk-headless -qq > /dev/null
!wget -q http://www.apache.org/dist/spark/$SPARK_VERSION/$SPARK_VERSION-bin-hadoop3.tgz
!tar xf $SPARK_VERSION-bin-hadoop3.tgz
!pip install -q findspark

# Set Environment Variables
os.environ["JAVA_HOME"] = "/usr/lib/jvm/java-11-openjdk-amd64"
os.environ["SPARK_HOME"] = f"/content/{spark_version}-bin-hadoop3"

# Start a SparkSession
import findspark
findspark.init()
```

```
Hit:1 https://developer.download.nvidia.com/compute/cuda/repos/ubuntu2204/x86_64  InRelease
Hit:2 https://cloud.r-project.org/bin/linux/ubuntu jammy-cran40/ InRelease
Hit:3 http://security.ubuntu.com/ubuntu jammy-security InRelease
Hit:4 http://archive.ubuntu.com/ubuntu jammy InRelease
Hit:5 http://archive.ubuntu.com/ubuntu jammy-updates InRelease
Hit:6 http://archive.ubuntu.com/ubuntu jammy-backports InRelease
Hit:7 https://ppa.launchpadcontent.net/c2d4u.team/c2d4u4.0+/ubuntu jammy InRelease
Hit:8 https://ppa.launchpadcontent.net/deadsnakes/ppa/ubuntu jammy InRelease
Hit:9 https://ppa.launchpadcontent.net/graphics-drivers/ppa/ubuntu jammy InRelease
Hit:10 https://ppa.launchpadcontent.net/ubuntugis/ppa/ubuntu jammy InRelease
Reading package lists... Done
```

```
[12] # Import packages
     from pyspark.sql import SparkSession
     import time

     # Create a SparkSession
     spark = SparkSession.builder.appName("SparkSQL").getOrCreate()
```

```
# 1. Read in the AWS S3 bucket into a DataFrame.
from pyspark import SparkFiles
url = "https://2u-data-curriculum-team.s3.amazonaws.com/dataviz-classroom/v1.2/22-big-data/home_sales_revised.csv"
spark.sparkContext.addFile(url)
home_sales_df = spark.read.csv(SparkFiles.get("home_sales_revised.csv"), sep=",", header=True)
home_sales_df.show()
```

```
+--------------------+----------+----------+------+--------+---------+-----------+--------+------+----------+----+
|                  id|      date|date_built| price|bedrooms|bathrooms|sqft_living|sqft_lot|floors|waterfront|view|
+--------------------+----------+----------+------+--------+---------+-----------+--------+------+----------+----+
|f8a53099-ba1c-47d...|2022-04-08|      2016|936923|       4|        3|       3167|   11733|     2|         1|  76|
|7530a2d8-1ae3-451...|2021-06-13|      2013|379628|       2|        2|       2235|   14384|     1|         0|  23|
|43de979c-0bf0-4c9...|2019-04-12|      2014|417866|       2|        2|       2127|   10575|     2|         0|   0|
|b672c137-b88c-48b...|2019-10-16|      2016|239895|       2|        2|       1631|   11149|     2|         0|   0|
|e0726d4d-d595-407...|2022-01-08|      2017|424418|       3|        2|       2249|   13878|     2|         0|   4|
|5aa00529-0533-46b...|2019-01-30|      2017|218712|       2|        3|       1965|   14375|     2|         0|   7|
|131492a1-72e2-4a8...|2020-02-08|      2017|419199|       2|        3|       2062|    8876|     2|         0|   6|
|8d54a71b-c520-44e...|2019-07-21|      2010|323956|       2|        3|       1506|   11816|     1|         0|  25|
|e81aacfe-17fe-46b...|2020-06-16|      2016|181925|       3|        3|       2137|   11709|     2|         0|  22|
|2ed8d509-7372-46d...|2021-08-06|      2015|258710|       3|        3|       1918|    9666|     1|         0|  25|
|f876d86f-3c9f-42b...|2019-02-27|      2011|167864|       3|        3|       2471|   13924|     2|         0|  15|
|0a2bd445-8508-4d8...|2021-12-30|      2014|337527|       2|        3|       1926|   12556|     1|         0|  23|
|941bad30-eb49-4a7...|2020-05-09|      2015|229896|       3|        3|       2197|    8641|     1|         0|   3|
|dd61eb34-6589-4c0...|2021-07-25|      2016|210247|       3|        2|       1672|   11986|     2|         0|  28|
|f1e4cef7-d151-439...|2019-02-01|      2011|398667|       2|        3|       2331|   11356|     1|         0|   7|
|ea620c7b-c2f7-4c6...|2021-05-31|      2011|437958|       3|        3|       2356|   11052|     1|         0|  26|
|f233cb41-6f33-4b0...|2021-07-18|      2016|437375|       4|        3|       1704|   11721|     2|         0|  34|
|c797ca12-52cd-4b1...|2019-06-08|      2015|288650|       2|        3|       2100|   10419|     2|         0|   7|
|0cfe57f3-28c2-472...|2019-10-04|      2015|308313|       3|        3|       1960|    9453|     2|         0|   2|
|4566cd2a-ac6e-435...|2019-07-15|      2016|177541|       3|        3|       2130|   10517|     2|         0|  25|
+--------------------+----------+----------+------+--------+---------+-----------+--------+------+----------+----+
only showing top 20 rows
```

```
[17] # 2. Create a temporary view of the DataFrame.

     home_sales_df.createOrReplaceTempView('home_sales')
```

```
# 3. What is the average price for a four bedroom house sold in each year rounded to two decimal places?

query = """
SELECT
  YEAR(date) AS YEAR,
  ROUND(AVG(price), 2) AS AVERAGE_PRICE
FROM home_sales
WHERE bedrooms = 4
GROUP BY YEAR
ORDER BY YEAR DESC
"""

spark.sql(query).show()
```

```
+----+-------------+
|YEAR|AVERAGE_PRICE|
+----+-------------+
|2022|    296363.88|
|2021|    301819.44|
|2020|    298353.78|
|2019|     300263.7|
+----+-------------+
```

```python
# 4. What is the average price of a home for each year the home was built that have 3 bedrooms and 3 bathrooms rounded to two decimal places?
query = """
SELECT
  YEAR(date_built) AS YEAR,
  ROUND(AVG(price), 2) AS AVERAGE_PRICE
FROM home_sales
WHERE bedrooms = 3
and bathrooms = 3
GROUP BY YEAR
ORDER BY YEAR DESC
"""

spark.sql(query).show()
```

```
+----+-------------+
|YEAR|AVERAGE_PRICE|
+----+-------------+
|2017|    292676.79|
|2016|    290555.07|
|2015|     288770.3|
|2014|    290852.27|
```

```
+----+-------------+
|YEAR|AVERAGE_PRICE|
+----+-------------+
|2017|    292676.79|
|2016|    290555.07|
|2015|     288770.3|
|2014|    290852.27|
|2013|    295962.27|
|2012|    293683.19|
|2011|    291117.47|
|2010|    292859.62|
+----+-------------+
```

```python
# 5. What is the average price of a home for each year built that have 3 bedrooms, 3 bathrooms, with two floors,
# and are greater than or equal to 2,000 square feet rounded to two decimal places?

query = """
SELECT
  YEAR(date_built) AS YEAR_BUILT,
  ROUND(AVG(price), 2) AS AVERAGE_PRICE
FROM home_sales
WHERE bedrooms = 3
and bathrooms = 3
and floors = 2
and sqft_living >= 2000
GROUP BY YEAR_BUILT
ORDER BY YEAR_BUILT DESC
"""

spark.sql(query).show()
```

```
+----------+-------------+
|YEAR_BUILT|AVERAGE_PRICE|
+----------+-------------+
|      2017|    280317.58|
```

```
+----------+-------------+
|YEAR_BUILT|AVERAGE_PRICE|
+----------+-------------+
|      2017|    280317.58|
|      2016|     293965.1|
|      2015|    297609.97|
|      2014|    298264.72|
|      2013|    303676.79|
|      2012|    307539.97|
|      2011|    276553.81|
|      2010|    285010.22|
+----------+-------------+
```

```python
# 6. What is the "view" rating for the average price of a home, rounded to two decimal places, where the homes are greater than
# or equal to $350,000? Although this is a small dataset, determine the run time for this query.

start_time = time.time()

query = """
SELECT
  view,
  ROUND(AVG(price), 2) AS AVERAGE_PRICE
FROM home_sales
GROUP BY view
HAVING AVG(price) >= 350000
ORDER BY view DESC
"""

spark.sql(query).show()


print("--- %s seconds ---" % (time.time() - start_time))
```

```
+----+-------------+
|view|AVERAGE_PRICE|
+----+-------------+
|  99|   1061201.42|
|  98|   1053739.33|
|  97|   1129040.15|
|  96|   1017815.92|
|  95|    1054325.6|
|  94|    1033536.2|
|  93|   1026006.06|
|  92|    970402.55|
|  91|   1137372.73|
|  90|   1062654.16|
|  89|   1107839.15|
|  88|   1031719.35|
|  87|    1072285.2|
|  86|   1070444.25|
```

```
|  86|   1070444.25|
|  85|   1056336.74|
|  84|   1117233.13|
|  83|   1033965.93|
|  82|    1063498.0|
|  81|   1053472.79|
|  80|    991767.38|
+----+-------------+
only showing top 20 rows

--- 1.6661772727966309 seconds ---
```

```python
[24] # 7. Cache the the temporary table home_sales.

     spark.sql('cache table home_sales')

     DataFrame[]

     # 8. Check if the table is cached.
     spark.catalog.isCached('home_sales')

     True
```

```python
# 9. Using the cached data, run the query that filters out the view ratings with average price
#  greater than or equal to $350,000. Determine the runtime and compare it to uncached runtime.

start_time = time.time()

query = """
SELECT
  view,
  ROUND(AVG(price), 2) AS AVERAGE_PRICE
FROM home_sales
GROUP BY view
HAVING AVG(price) >= 350000
ORDER BY view DESC
"""

spark.sql(query).show()

print("--- %s seconds ---" % (time.time() - start_time))
```

```
+----+-------------+
|view|AVERAGE_PRICE|
+----+-------------+
|  99|   1061201.42|
|  98|   1053739.33|
|  97|   1129040.15|
|  96|   1017815.92|
|  95|    1054325.6|
|  94|    1033536.2|
|  93|   1026006.06|
|  92|    970402.55|
|  91|   1137372.73|
|  90|   1062654.16|
|  89|   1107839.15|
|  88|   1031719.35|
```

```
|  88|   1031719.35|
|  87|    1072285.2|
|  86|   1070444.25|
|  85|   1056336.74|
|  84|   1117233.13|
|  83|   1033965.93|
|  82|    1063498.0|
|  81|   1053472.79|
|  80|    991767.38|
+----+-------------+
only showing top 20 rows

--- 0.6723017692565918 seconds ---
```

```python
[27] # 10. Partition by the "date_built" field on the formatted parquet home sales data

     home_sales_df.write.partitionBy('date_built').parquet('new_home_sales', mode='overwrite')

[28] # 11. Read the parquet formatted data.

     new_home_sales_df = spark.read.parquet('new_home_sales')

# 12. Create a temporary table for the parquet data.

new_home_sales_df.createOrReplaceTempView('new_home_sales')
```

```python
# 13. Run the query that filters out the view ratings with average price of greater than or equal to $350,000
# with the parquet DataFrame. Round your average to two decimal places.
# Determine the runtime and compare it to the cached version.

start_time = time.time()

query = """
SELECT
  view,
  ROUND(AVG(price), 2) AS AVERAGE_PRICE
FROM new_home_sales
GROUP BY view
HAVING AVG(price) >= 350000
ORDER BY view DESC
"""

spark.sql(query).show()


print("--- %s seconds ---" % (time.time() - start_time))
```

```
+----+-------------+
|view|AVERAGE_PRICE|
+----+-------------+
|  99|   1061201.42|
|  98|   1053739.33|
|  97|   1129040.15|
|  96|   1017815.92|
|  95|    1054325.6|
|  94|    1033536.2|
|  93|   1026006.06|
|  92|    970402.55|
|  91|   1137372.73|
|  90|   1062654.16|
|  89|   1107839.15|
|  88|   1031719.35|
|  87|    1072285.2|
|  86|   1070444.25|
|  85|   1056336.74|
|  84|   1117233.13|
|  83|   1033965.93|
```

```
|  83|   1033965.93|
|  82|    1063498.0|
|  81|   1053472.79|
|  80|    991767.38|
+----+-------------+
only showing top 20 rows

--- 0.9201929569244385 seconds ---
```

```python
# 14. Uncache the home_sales temporary table.
spark.sql('uncache table home_sales')
```

```
DataFrame[]
```

```python
[32] # 15. Check if the home_sales is no longer cached
spark.catalog.isCached('home_sales')
```

```
False
```

```python
#END
```

#END