databricks home_sales_analysis_pyspark_sparksql_pandas

(https://databricks.com)

Let's look at the home sales data analysis in 3 differ

```
from pyspark import SparkFiles
import pandas as pd
# url = "https://2u-data-curriculum-team.s3.amazonaws.com/dataviz-
classroom/v1.2/22-big-data/home_sales_revised.csv"
# spark.sparkContext.addFile(url)
data_source = 'dbfs:/FileStore/homesalesdata/home_sales_revised__1_.csv'
homes_df = spark.read.csv(data_source, sep=",", header=True)
homes_df.show()
```

```
+--------------------+----------+----------+------+--------+---------+--------
---+--------+------+----------+----+
|                  id|      date|date_built| price|bedrooms|bathrooms|sqft_liv
ing|sqft_lot|floors|waterfront|view|
+--------------------+----------+----------+------+--------+---------+--------
---+--------+------+----------+----+
|f8a53099-ba1c-47d...|2022-04-08|      2016|936923|       4|        3|       3
167|   11733|     2|         1|  76|
|7530a2d8-1ae3-451...|2021-06-13|      2013|379628|       2|        2|       2
235|   14384|     1|         0|  23|
|43de979c-0bf0-4c9...|2019-04-12|      2014|417866|       2|        2|       2
127|   10575|     2|         0|   0|
|b672c137-b88c-48b...|2019-10-16|      2016|239895|       2|        2|       1
631|   11149|     2|         0|   0|
|e0726d4d-d595-407...|2022-01-08|      2017|424418|       3|        2|       2
249|   13878|     2|         0|   4|
|5aa00529-0533-46b...|2019-01-30|      2017|218712|       2|        3|       1
965|   14375|     2|         0|   7|
|131492a1-72e2-4a8...|2020-02-08|      2017|419199|       2|        3|       2
062|    8876|     2|         0|   6|
|8d54a71b-c520-44e...|2019-07-21|      2010|323956|       2|        3|       1
```

Create a Spark Dataframe

```
data_source = 'dbfs:/FileStore/homesalesdata/home_sales_revised__1_.csv'
homes_df = spark.read.csv(data_source, sep=",", header=True)
homes_df.show()
```

```
+--------------------+----------+----------+------+--------+---------+--------
---+--------+------+----------+----+
|                  id|      date|date_built| price|bedrooms|bathrooms|sqft_liv
ing|sqft_lot|floors|waterfront|view|
```

```
+--------------------+----------+----------+------+--------+---------+--------
---+--------+------+----------+----+
|f8a53099-ba1c-47d...|2022-04-08|      2016|936923|       4|        3|       3
167|   11733|     2|         1|  76|
|7530a2d8-1ae3-451...|2021-06-13|      2013|379628|       2|        2|       2
235|   14384|     1|         0|  23|
|43de979c-0bf0-4c9...|2019-04-12|      2014|417866|       2|        2|       2
127|   10575|     2|         0|   0|
|b672c137-b88c-48b...|2019-10-16|      2016|239895|       2|        2|       1
631|   11149|     2|         0|   0|
|e0726d4d-d595-407...|2022-01-08|      2017|424418|       3|        2|       2
249|   13878|     2|         0|   4|
|5aa00529-0533-46b...|2019-01-30|      2017|218712|       2|        3|       1
965|   14375|     2|         0|   7|
|131492a1-72e2-4a8...|2020-02-08|      2017|419199|       2|        3|       2
062|    8876|     2|         0|   6|
|8d54e71b-e520-44e...|2019-07-21|      2019|233056|       2|        2|       1
```

```
homes_df.printSchema()
```

```
root
 |-- id: string (nullable = true)
 |-- date: string (nullable = true)
 |-- date_built: string (nullable = true)
 |-- price: string (nullable = true)
 |-- bedrooms: string (nullable = true)
 |-- bathrooms: string (nullable = true)
 |-- sqft_living: string (nullable = true)
 |-- sqft_lot: string (nullable = true)
 |-- floors: string (nullable = true)
 |-- waterfront: string (nullable = true)
 |-- view: string (nullable = true)
```

Create a temporary view for Spark SQL

```
# why? :  A Temporary view in Spark is similar to a real SQL table that
contains rows and columns. If you're more comfortable with SQL then you must
have to create temporary view with following command and then run SQL query on
view.
homes_df.createOrReplaceTempView('home_sales')
```

Create a Pandas DataFrame

```
pandas_df = homes_df.toPandas()
print(pandas_df.head(5))

# Little preprocessing in pandas df
pandas_df['date'] = pd.to_datetime(pandas_df['date'], format="%Y-%m-%d")
pandas_df['date_built'] = pd.to_datetime(pandas_df['date_built'], format="%Y")
pandas_df['price'] = pd.to_numeric(pandas_df['price'], errors='coerce')
pandas_df['bedrooms'] = pd.to_numeric(pandas_df['bedrooms'], errors='coerce')
pandas_df['bathrooms'] = pd.to_numeric(pandas_df['bathrooms'], errors='coerce')
pandas_df['sqft_living'] = pd.to_numeric(pandas_df['sqft_living'],
errors='coerce')
pandas_df['sqft_lot'] = pd.to_numeric(pandas_df['sqft_lot'], errors='coerce')
pandas_df['floors'] = pd.to_numeric(pandas_df['floors'], errors='coerce')
pandas_df['waterfront'] = pd.to_numeric(pandas_df['waterfront'],
errors='coerce')
pandas_df['view'] = pd.to_numeric(pandas_df['view'], errors='coerce')
```

```
                                     id        date date_built   price  \
0  f8a53099-ba1c-47d6-9c31-7398aa8f6089  2022-04-08       2016  936923
1  7530a2d8-1ae3-4517-9f4a-befe060c4353  2021-06-13       2013  379628
2  43de979c-0bf0-4c9f-85ef-96dc27b258d5  2019-04-12       2014  417866
3  b672c137-b88c-48bf-9f18-d0a4ac62fb8b  2019-10-16       2016  239895
4  e0726d4d-d595-4074-8283-4139a54d0d63  2022-01-08       2017  424418

  bedrooms bathrooms sqft_living sqft_lot floors waterfront view
0        4         3        3167    11733      2          1   76
1        2         2        2235    14384      1          0   23
2        2         2        2127    10575      2          0    0
3        2         2        1631    11149      2          0    0
4        3         2        2249    13878      2          0    4
```

```
pandas_df.head(5)
```

	id	date	date_built	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	wa
0	f8a53099-ba1c-47d6-9c31-7398aa8f6089	2022-04-08	2016-01-01	936923	4	3	3167	11733	2	
1	7530a2d8-1ae3-4517-9f4a-befe060c4353	2021-06-13	2013-01-01	379628	2	2	2235	14384	1	

```
pandas_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 33287 entries, 0 to 33286
Data columns (total 11 columns):
 #   Column       Non-Null Count  Dtype         
---  ------       --------------  -----         
 0   id           33287 non-null  object        
 1   date         33287 non-null  datetime64[ns]
 2   date_built   33287 non-null  datetime64[ns]
 3   price        33287 non-null  int64         
 4   bedrooms     33287 non-null  int64         
 5   bathrooms    33287 non-null  int64         
 6   sqft_living  33287 non-null  int64         
 7   sqft_lot     33287 non-null  int64         
 8   floors       33287 non-null  int64         
 9   waterfront   33287 non-null  int64         
 10  view         33287 non-null  int64         
dtypes: datetime64[ns](2), int64(8), object(1)
memory usage: 2.8+ MB
```

```
spark.sql(sqlQuery="SELECT * FROM home_sales").show(2)
```

```
+--------------------+----------+----------+------+--------+---------+---------
--+--------+------+----------+----+
|                  id|      date|date_built| price|bedrooms|bathrooms|sqft_livi
ng|sqft_lot|floors|waterfront|view|
+--------------------+----------+----------+------+--------+---------+---------
--+--------+------+----------+----+
|f8a53099-ba1c-47d...|2022-04-08|      2016|936923|       4|        3|       31
67|   11733|     2|         1|  76|
|7530a2d8-1ae3-451...|2021-06-13|      2013|379628|       2|        2|       22
35|   14384|     1|         0|  23|
+--------------------+----------+----------+------+--------+---------+---------
--+--------+------+----------+----+
only showing top 2 rows
```

1. What is the average price for a four bedroom hou

```
# SOLUTION : SPARK SQL

avg_price_4BR = """
SELECT
    YEAR(DATE) AS YEAR,
    ROUND(AVG(price),2) AS AVG_PRICE
FROM
    HOME_SALES
WHERE BEDROOMS = 4
GROUP BY
    YEAR
ORDER BY
    YEAR DESC
"""

spark.sql(sqlQuery=avg_price_4BR).show()
```

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

```
# SOLUTION : PYSPARK DATAFRAME
from pyspark.sql.functions import *

avg_price_4br_spark_df = homes_df.where(col('bedrooms')==4)\
                                 .withColumn('YEAR', year(col('date')))\
                                 .select('year','bedrooms','price')\
                                 .groupBy('year')\
                                 .agg(round(avg("price"),2).alias('AVG_PRICE'))\
                                 .orderBy(desc('year'))
avg_price_4br_spark_df.show()
```

```
+----+---------+
|year|AVG_PRICE|
+----+---------+
|2022|296363.88|
|2021|301819.44|
|2020|298353.78|
```

```
|2019| 300263.7|
+----+---------+


# # SOLUTION : PANDAS DATAFRAME
import datetime as dt
import pandas as pd
import time

df_4br = pandas_df[pandas_df['bedrooms']==4][['date','price','bedrooms']]
df_4br['year'] = df_4br['date'].dt.strftime('%Y')
avg_price = df_4br\
            .groupby('year')\
            .agg(AVG_PRICE=('price', 'mean')).round(2)\
            .sort_values(by='year',ascending=False)

print(avg_price)

      AVG_PRICE
year
2022  296363.88
2021  301819.44
2020  298353.78
2019  300263.70
```

2. What is the average price of a home for each yea

```
# SOLUTION : SPARK SQL


avg_price_3bad_3bath = """
SELECT
    YEAR(DATE_BUILT) AS YEAR,
    ROUND(AVG(PRICE),2) AS AVG_PRICE
FROM
    HOME_SALES
WHERE
    BEDROOMS = 3 AND BATHROOMS = 3
GROUP BY
    YEAR
ORDER BY
    YEAR DESC
"""
spark.sql(sqlQuery=avg_price_3bad_3bath).show()

+----+---------+
|YEAR|AVG_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|
+----+---------+
```

```
#SOLUTION : PYSPARK DATAFRAME
```

```
bed3_3bath_spark_df = homes_df.where((col('bedrooms')==3) &
(col('bathrooms')==3))\
                           .withColumn('YEAR',year(col('date_built')))\
                           .selectExpr('year','bedrooms','bathrooms','price')\
                           .groupBy('year')\
                           .agg(round(avg('price'),2).alias('AVG_PRICE'))\
                           .orderBy(desc('YEAR'))

bed3_3bath_spark_df.show()
```

```
+----+---------+
|year|AVG_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|
+----+---------+
```

```
#SOLUTION : PANDAS DATAFRAME


df_3br_3bath = pandas_df[(pandas_df['bedrooms'] == 3) &
(pandas_df['bathrooms']==3)][['date_built','bedrooms','bathrooms','price']]

df_3br_3bath['year'] = df_3br_3bath['date_built'].dt.strftime('%Y')

df_avg = df_3br_3bath\
            .groupby('year')\
            .agg(AVG_PRICE = ('price','mean'))\
            .round(2).sort_values(by = 'year', ascending = False)


# print(df_4br_3bath.head)
print(df_avg.head(10))
```

```
      AVG_PRICE
year
2017  292676.79
2016  290555.07
2015  288770.30
2014  290852.27
2013  295962.27
2012  293683.19
2011  291117.47
2010  292859.62
```

3.What is the average price of a home for each year

```
# SOLUTION : SPARK SQL

query = """
SELECT
    YEAR(date_built) as YEAR,
    ROUND(AVG(price),2) as AVG_PRICE
FROM
    HOME_SALES
WHERE
    BEDROOMS = 3 AND
    BATHROOMS = 3 AND
    FLOORS = 2 AND
    sqft_living >= 2000
GROUP BY
    YEAR
ORDER BY
    YEAR desc
"""
spark.sql(sqlQuery=query).show()
```

```
+----+---------+
|YEAR|AVG_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|
+----+---------+
```

```
# SOLUTION : PYSPARK DATAFRAME

avg_price_3br_3bath_3floor_spark_df = homes_df\
                                      .where((col('bedrooms') == 3) &
(col('bathrooms') == 3) & (col('floors') == 2) & (col('sqft_living') >= 2000))\
                                      .select(('date_built'),'price')\

.groupBy('date_built').agg(round(avg('price'),2).alias('AVG_PRICE'))\
                                      .orderBy(desc('date_built'))
avg_price_3br_3bath_3floor_spark_df.show()
```

```
+----------+---------+
|date_built|AVG_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|
+----------+---------+
```

```
# SOLUTION : PANDAS DATAFRAME


pandas_df_3br_3bath_3floor = pandas_df[(pandas_df['bedrooms'] == 3) &
(pandas_df['bathrooms'] == 3) & (pandas_df['floors'] == 2) &
(pandas_df['sqft_living'] >= 2000) ]
[['date_built','bedrooms','bathrooms','floors','sqft_living','price']]

pandas_df_3br_3bath_3floor['year'] =
pandas_df_3br_3bath_3floor['date_built'].dt.strftime('%Y')

avg_df = pandas_df_3br_3bath_3floor\
        .groupby('year')\
        .agg(AVG_PRICE = ('price','mean')).round(2)\
        .sort_values(by='year', ascending = False)

# print(pandas_df_3br_3bath_3floor.head)
print(avg_df.head(10))
```

```
      AVG_PRICE
year
2017  280317.58
2016  293965.10
2015  297609.97
2014  298264.72
2013  303676.79
2012  307539.97
2011  276553.81
2010  285010.22
```

4. What is the "view" rating for the average price of

```
# Although this is a small dataset, determine the run time for this query.

# SOLUTION : SPARK SQL

view_ratings = """
SELECT
    VIEW,
    ROUND(AVG(PRICE),2) AS AVG_PRICE
FROM
    HOME_SALES
GROUP BY
    VIEW
HAVING
    AVG_PRICE > 350000
ORDER BY
    VIEW DESC
"""

spark.sql(sqlQuery=view_ratings).show()

# FIND RUN TIME OF THE QUERY
start_time = time.time()
print("--- %s seconds ---" % (time.time() - start_time))
```

```
+----+----------+
|VIEW| AVG_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|
```

```
# SOLUTION : SPARK DATAFRAME

# NOTE : HAVING doesn't exist in spark dataframe. You express the same logic
with agg followed by WHERE

view_ratings_spark_df = homes_df.select('view','price')\
                        .groupBy('view')\
                        .agg(round(avg('price'),2).alias('AVG_PRICE'))\
                        .where(col('AVG_PRICE') >= 350000)\
                        .orderBy(desc('view'))

view_ratings_spark_df.show(100)

# FIND RUN TIME OF THE QUERY
start_time = time.time()
print("--- %s seconds ---" % (time.time() - start_time))
```

```
+----+----------+
|view| AVG_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|
```

```
# SOLUTION : PANDAS DATAFRAME

view_ratings_pandas_df = pandas_df[['view', 'price']]\
                         .groupby('view')\
                         .agg(AVERAGE_PRICE=('price','mean')).round(2)

filter_records = view_ratings_pandas_df[view_ratings_pandas_df['AVERAGE_PRICE']
>= 350000]

print(filter_records)

# FIND RUN TIME OF THE QUERY
start_time = time.time()
print("--- %s seconds ---" % (time.time() - start_time))
```

```
      AVERAGE_PRICE
view
51        788128.21
52        733780.26
53        755214.80
54        798684.82
55        771153.32
56        718176.40
57        734340.50
58        759764.65
59        791453.00
60        754939.65
61        746877.59
62        759150.14
63        711614.55
64        767036.67
65        736679.93
66        712475.00
67        737970.96
68        716785.44
69        750537.94
```

Cache Table

```
# Cache the temporary table home_sales
spark.sql(sqlQuery='cache table HOME_SALES')

Out[59]: DataFrame[]
```

```
# check if the table is cached
spark.catalog.isCached('HOME_SALES')
```

```
Out[60]: True
```

```
# 5. 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.
```

```
# SOLUTION : SPARK SQL

view_ratings = """
SELECT
    VIEW,
    ROUND(AVG(PRICE),2) AS AVG_PRICE
FROM
    HOME_SALES
GROUP BY
    VIEW
HAVING
    AVG_PRICE > 350000
ORDER BY
    VIEW DESC
"""

spark.sql(sqlQuery=view_ratings).show()

# FIND RUN TIME OF THE QUERY
start_time = time.time()
print("--- %s seconds ---" % (time.time() - start_time))
```

```
+----+----------+
|VIEW| AVG_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|
```

Partition by the "date_built" field on the formatted p

```
homes_df.write.partitionBy('date_built').parquet('parquet_home_sales',
mode='overwrite')
```

Read the formated Parquet Data

```
parquet_df = spark.read.parquet('dbfs:/parquet_home_sales')
print(parquet_df.head)
```

```
<bound method DataFrame.head of DataFrame[id: string, date: string, price: stri
ng, bedrooms: string, bathrooms: string, sqft_living: string, sqft_lot: string,
floors: string, waterfront: string, view: string, date_built: int]>
```

Create a temporary table for the parquet data.

```
spark.sql(sqlQuery='uncache table HOME_SALES')
```

```
Out[64]: DataFrame[]
```

```
# Run the query that filters out the view ratings with average price of greater than or eqaul to $350,000 with the parquet DataFrame. Round your average to two decimal places.

# Determine the runtime and compare it to the cached version.

view_ratings_spark_df = parquet_df.select('view','price')\
                        .groupBy('view')\
                        .agg(round(avg('price'),2).alias('AVG_PRICE'))\
                        .where(col('AVG_PRICE') >= 350000)\
                        .orderBy(desc('view'))

view_ratings_spark_df.show(100)

# FIND RUN TIME OF THE QUERY
start_time = time.time()
print("--- %s seconds ---" % (time.time() - start_time))
```

```
+----+----------+
|view| AVG_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|
```

```
not cached
```