```
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-3.4.3'
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://cloud.r-project.org/bin/linux/ubuntu jammy-cran40/ InRelease
Hit:2 https://developer.download.nvidia.com/compute/cuda/repos/ubuntu2204/x86_64  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
```

```
# 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)
df = spark.read.csv(SparkFiles.get("home_sales_revised.csv"), sep=",", header=True, ignoreLeadingWhiteSpace=True)
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
```

```
# 2. Create a temporary view of the DataFrame.
df.createOrReplaceTempView('home_sales')
```

```
# 3. What is the average price for a four bedroom house sold per year, rounded to two decimal places?
avg_price_4_bedroom = spark.sql("SELECT ROUND(AVG(price),2), YEAR(date) from home_sales where bedrooms == 4 group by YEAR(date) ORDER BY YEA
avg_price_4_bedroom.show()
```

```
+-------------------+----------+
|round(avg(price), 2)|year(date)|
+-------------------+----------+
|           296363.88|      2022|
|           301819.44|      2021|
|           298353.78|      2020|
|            300263.7|      2019|
+-------------------+----------+
```

```
# 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?
avg_price_3_bed_3_bath = spark.sql("SELECT ROUND(AVG(price),2), date_built from home_sales where bedrooms == 3 AND bathrooms == 3 group by d
avg_price_3_bed_3_bath.show()
```

```
+-------------------+----------+
|round(avg(price), 2)|date_built|
+-------------------+----------+
|           292676.79|      2017|
|           290555.07|      2016|
|            288770.3|      2015|
|           290852.27|      2014|
|           295962.27|      2013|
|           293683.19|      2012|
|           291117.47|      2011|
|           292859.62|      2010|
+-------------------+----------+
```

```
# 5. What is the average price of a home for each year the home was 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?
avg_price_3_bed_3_bath_2_floors = spark.sql("SELECT ROUND(AVG(price),2), date_built from home_sales where bedrooms == 3 AND bathrooms == 3 ANI
avg_price_3_bed_3_bath_2_floors.show()
```

```
+-------------------+----------+
|round(avg(price), 2)|date_built|
+-------------------+----------+
|           280317.58|      2017|
|            293965.1|      2016|
|           297609.97|      2015|
|           298264.72|      2014|
|           303676.79|      2013|
|           307539.97|      2012|
|           276553.81|      2011|
|           285010.22|      2010|
+-------------------+----------+
```

```
# 6. What is the average price of a home per "view" rating, rounded to two decimal places,
# having an average home price greater than or equal to $350,000? Order by descending view rating.
# Although this is a small dataset, determine the run time for this query.

start_time = time.time()

spark.sql("SELECT view, ROUND(AVG(price),2) from home_sales group by view having ROUND(AVG(price),2) >= 350000").show()

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

```
+----+-------------------+
|view|round(avg(price), 2)|
+----+-------------------+
|  51|          788128.21|
|  54|          798684.82|
|  69|          750537.94|
|  87|          1072285.2|
|  73|          752861.18|
```

```
|  64|           767036.67|
|  59|            791453.0|
|  85|          1056336.74|
|  52|           733780.26|
|  71|            775651.1|
|  98|          1053739.33|
|  99|          1061201.42|
|  96|          1017815.92|
| 100|           1026669.5|
|  70|           695865.58|
|  61|           746877.59|
|  75|          1114042.94|
|  78|          1080649.37|
|  89|          1107839.15|
|  77|          1076205.56|
+----+--------------------+
only showing top 20 rows

--- 1.6073582172393799 seconds ---
```

```
# 7. Cache the the temporary table home_sales.
spark.sql("cache table home_sales")
```

```
DataFrame[]
```

```
# 8. Check if the table is cached.
spark.sql("CACHE TABLE home_sales OPTIONS('LAZY'='true')")

# Show cached tables
spark.sql("SHOW TABLES").show()
```

```
+---------+----------+-----------+
|namespace| tableName|isTemporary|
+---------+----------+-----------+
|         |home_sales|       true|
+---------+----------+-----------+
```

```
# 9. Using the cached data, run the last query above, that calculates
# the average price of a home per "view" rating, rounded to two decimal places,
# having an average home price greater than or equal to $350,000.
# Determine the runtime and compare it to the uncached runtime.

start_time = time.time()

spark.sql("SELECT view, ROUND(AVG(price),2) from home_sales group by view having ROUND(AVG(price),2) >= 350000").show()

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

```
+----+--------------------+
|view|round(avg(price), 2)|
+----+--------------------+
|  51|           788128.21|
|  54|           798684.82|
|  69|           750537.94|
|  87|          1072285.2|
|  73|           752861.18|
|  64|           767036.67|
|  59|            791453.0|
|  85|          1056336.74|
|  52|           733780.26|
|  71|            775651.1|
|  98|          1053739.33|
|  99|          1061201.42|
|  96|          1017815.92|
| 100|           1026669.5|
|  70|           695865.58|
|  61|           746877.59|
|  75|          1114042.94|
|  78|          1080649.37|
|  89|          1107839.15|
|  77|          1076205.56|
+----+--------------------+
only showing top 20 rows

--- 1.1569628715515137 seconds ---
```

```
# 10. Partition by the "date_built" field on the formatted parquet home sales data
df.write.parquet('home_parquet', mode='overwrite')


# 11. Read the parquet formatted data.
parquet_df = spark.read.parquet('home_parquet')


# 12. Create a temporary table for the parquet data.
parquet_df.createOrReplaceTempView('temp_parquet')


# 13. Using the parquet DataFrame, run the last query above, that calculates
# the average price of a home per "view" rating, rounded to two decimal places,
# having an average home price greater than or equal to $350,000.
# Determine the runtime and compare it to the cached runtime.

start_time = time.time()

spark.sql("SELECT view, ROUND(AVG(price),2) from temp_parquet group by view having ROUND(AVG(price),2) >= 350000").show()

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

```
+----+--------------------+
|view|round(avg(price), 2)|
+----+--------------------+
|  51|           788128.21|
|  54|           798684.82|
|  69|           750537.94|
|  87|           1072285.2|
|  73|           752861.18|
|  64|           767036.67|
|  59|            791453.0|
|  85|          1056336.74|
|  52|           733780.26|
|  71|            775651.1|
|  98|          1053739.33|
|  99|          1061201.42|
|  96|          1017815.92|
| 100|           1026669.5|
|  70|           695865.58|
|  61|           746877.59|
|  75|          1114042.94|
|  78|          1080649.37|
|  89|          1107839.15|
|  77|          1076205.56|
+----+--------------------+
only showing top 20 rows

--- 0.9434247016906738 seconds ---
```

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

```
DataFrame[]
```

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

```
False
```

Start coding or generate with AI.