```
import os
# Find the latest version of spark 3.x  from http://www.apache.org/dist/spark/ and enter as the spark version
spark_version = 'spark-3.5.0'
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 http://archive.ubuntu.com/ubuntu jammy InRelease
Hit:2 http://archive.ubuntu.com/ubuntu jammy-updates InRelease
Hit:3 https://cloud.r-project.org/bin/linux/ubuntu jammy-cran40/ InRelease
Hit:4 https://developer.download.nvidia.com/compute/cuda/repos/ubuntu2204/x86_64  InRelease
Hit:5 http://archive.ubuntu.com/ubuntu jammy-backports InRelease
Hit:6 http://security.ubuntu.com/ubuntu jammy-security 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.
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("file://" + SparkFiles.get("home_sales_revised.csv"), header=True, inferSchema=True)

# 2. Create a temporary view of the DataFrame.
df.createOrReplaceTempView("home_sales_view")



# 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 SaleYear,
    ROUND(AVG(price), 2) AS AveragePrice
FROM
    home_sales_view
WHERE
    bedrooms = 4
GROUP BY
    SaleYear
ORDER BY
    SaleYear
"""

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

```
+--------+------------+
|SaleYear|AveragePrice|
+--------+------------+
|    2019|    300263.7|
```

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

```
# 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
    date_built AS YearBuilt,
    ROUND(AVG(price), 2) AS AveragePrice
FROM
    home_sales_view
WHERE
    bedrooms = 3
    AND bathrooms = 3
GROUP BY
    YearBuilt
ORDER BY
    YearBuilt
"""

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

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

```
# 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
    date_built AS YearBuilt,
    ROUND(AVG(price), 2) AS AveragePrice
FROM
    home_sales_view
WHERE
    bedrooms = 3
    AND bathrooms = 3
    AND floors = 2
    AND sqft_living >= 2000
GROUP BY
    YearBuilt
ORDER BY
    YearBuilt
"""

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

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

```
# 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?
query = """
SELECT
    view,
    ROUND(AVG(price), 2) AS AveragePrice
FROM
    home_sales_view
WHERE
    price >= 350000
GROUP BY
    view
ORDER BY
    view
"""

start_time = time.time()

result = spark.sql(query)
result.show()

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

```
+----+------------+
|view|AveragePrice|
+----+------------+
|   0|   403848.51|
|   1|   401044.25|
|   2|   397389.25|
|   3|    398867.6|
|   4|   399631.89|
|   5|   401471.82|
|   6|   395655.38|
|   7|   403005.77|
|   8|   398592.71|
|   9|   401393.34|
|  10|   401868.43|
|  11|   399548.12|
|  12|   401501.32|
|  13|   398917.98|
|  14|   398570.03|
|  15|    404673.3|
|  16|   399586.53|
|  17|   398474.49|
|  18|   399332.91|
|  19|   398953.17|
+----+------------+
only showing top 20 rows

--- 0.8750758171081543 seconds ---
```

```
# 7. Cache the the temporary table home_sales.
# Cache the temporary table home_sales_view
spark.sql("CACHE TABLE home_sales_view")

# Now, you can run your queries on the cached table
# For example, let's rerun the first query to find the average price for a four-bedroom house sold in each year rounded to two decimal places

# 3. What is the average price for a four-bedroom house sold in each year rounded to two decimal places?
query_3 = """
SELECT
    YEAR(date) AS SaleYear,
    ROUND(AVG(price), 2) AS AveragePrice
FROM
    home_sales_view
WHERE
    bedrooms = 4
GROUP BY
    SaleYear
ORDER BY
    SaleYear
"""

result_3 = spark.sql(query_3)
result_3.show()
```

```
+--------+------------+
|SaleYear|AveragePrice|
+--------+------------+
|    2019|    300263.7|
|    2020|   298353.78|
|    2021|   301819.44|
|    2022|   296363.88|
+--------+------------+
```

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

```
True
```

```
# Using the cached data, run the query that filters out the view ratings with average price greater than or equal to $350,000
query_cached = """
SELECT
    view,
    ROUND(AVG(price), 2) AS AveragePrice
FROM
    home_sales_view
WHERE
    price >= 350000
GROUP BY
    view
ORDER BY
    view
"""

start_time_cached = time.time()

result_cached = spark.sql(query_cached)
result_cached.show()

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

```
+----+------------+
|view|AveragePrice|
+----+------------+
|   0|   403848.51|
|   1|   401044.25|
|   2|   397389.25|
|   3|    398867.6|
|   4|   399631.89|
|   5|   401471.82|
|   6|   395655.38|
|   7|   403005.77|
|   8|   398592.71|
|   9|   401393.34|
|  10|   401868.43|
|  11|   399548.12|
|  12|   401501.32|
|  13|   398917.98|
|  14|   398570.03|
|  15|    404673.3|
|  16|   399586.53|
|  17|   398474.49|
|  18|   399332.91|
|  19|   398953.17|
+----+------------+
only showing top 20 rows

--- 0.47432422637939453 seconds ---
```

```
# 10. Partition by the "date_built" field on the formatted parquet home sales data
df.write.partitionBy("date_built").parquet("/path/to/save/parquet")
```

```
# 11. Read the formatted parquet data.
parquet_data = spark.read.parquet("/path/to/save/parquet")
```

```
# Show the first few rows of the DataFrame
parquet_data.show()
```

```
+--------------------+----------+------+--------+---------+-----------+--------+------+----------+----+----------+
|                  id|      date| price|bedrooms|bathrooms|sqft_living|sqft_lot|floors|waterfront|view|date_built|
+--------------------+----------+------+--------+---------+-----------+--------+------+----------+----+----------+
|2ed8d509-7372-46d...|2021-08-06|258710|       3|        3|       1918|    9666|     1|         0|  25|      2015|
|941bad30-eb49-4a7...|2020-05-09|229896|       3|        3|       2197|    8641|     1|         0|   3|      2015|
|c797ca12-52cd-4b1...|2019-06-08|288650|       2|        3|       2100|   10419|     2|         0|   7|      2015|
|0cfe57f3-28c2-472...|2019-10-04|308313|       3|        3|       1960|    9453|     2|         0|   2|      2015|
|d715f295-2fbf-4e9...|2021-05-17|391574|       3|        2|       1635|    8040|     2|         0|  10|      2015|
|a18515a2-86f3-46b...|2022-02-18|419543|       3|        2|       1642|   12826|     2|         0|  24|      2015|
|98f6a9ad-8870-474...|2022-05-07|136752|       2|        3|       1701|   10771|     2|         0|   5|      2015|
|7ac67498-b6f3-403...|2021-05-12|349318|       4|        3|       2417|   11304|     2|         0|  37|      2015|
|c9bfdb1c-2499-4e3...|2021-12-07|268874|       2|        2|       1537|   12177|     1|         0|  10|      2015|
|34c31a34-220d-469...|2019-02-06|409011|       3|        3|       2356|   10507|     1|         0|   1|      2015|
|be0ccb95-415d-411...|2020-05-15|425154|       4|        3|       2120|   14229|     2|         0|   4|      2015|
|e9031a86-1294-444...|2021-10-09|222322|       4|        3|       1928|   10510|     1|         0|  38|      2015|
|e6d7c2a7-596e-4ec...|2019-03-15|131201|       4|        3|       1633|   14655|     1|         0|  22|      2015|
|6683714b-3df7-454...|2022-02-01|333403|       4|        2|       2059|    9793|     2|         0|   4|      2015|
|00fc996f-508c-430...|2021-07-15|373139|       3|        3|       1763|   11363|     1|         0|  39|      2015|
|3d5545f8-bd3b-476...|2020-09-19|797862|       4|        6|       3494|   10385|     2|         0|  90|      2015|
|ec6d357c-2435-43e...|2019-05-28|401792|       3|        2|       1627|   10765|     1|         0|  50|      2015|
|c2be38fb-814a-403...|2020-03-20|352237|       3|        3|       2485|   10954|     2|         0|   6|      2015|
|9570de1f-5a74-45b...|2021-11-29|298453|       3|        2|       2222|   10634|     1|         0|   6|      2015|
|1baeff4f-fc00-489...|2020-12-17|152775|       3|        2|       1623|   13851|     1|         0|  41|      2015|
+--------------------+----------+------+--------+---------+-----------+--------+------+----------+----+----------+
only showing top 20 rows
```

```
# 12. Create a temporary table for the parquet data.
parquet_data.createOrReplaceTempView("parquet_table")
```

```
# 13. Run the query with the parquet DataFrame
query_parquet = """
SELECT
    view,
    ROUND(AVG(price), 2) AS AveragePrice
FROM
    parquet_table
WHERE
    price >= 350000
GROUP BY
    view
ORDER BY
    view
"""

start_time_parquet = time.time()

result_parquet = spark.sql(query_parquet)
result_parquet.show()

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

```
+----+------------+
|view|AveragePrice|
+----+------------+
|   0|   403848.51|
|   1|   401044.25|
|   2|   397389.25|
|   3|    398867.6|
|   4|   399631.89|
|   5|   401471.82|
|   6|   395655.38|
|   7|   403005.77|
|   8|   398592.71|
|   9|   401393.34|
|  10|   401868.43|
|  11|   399548.12|
|  12|   401501.32|
|  13|   398917.98|
|  14|   398570.03|
|  15|    404673.3|
|  16|   399586.53|
```

```
|  17|   398474.49|
|  18|   399332.91|
|  19|   398953.17|
+----+------------+
only showing top 20 rows

--- 0.8140139579772949 seconds ---
```

```
# 14. Uncache the home_sales temporary table.
spark.catalog.uncacheTable("home_sales_view")
```

```
# 15. Check if the home_sales is no longer cached
is_cached = spark.catalog.isCached('home_sales_view')
print("Is home_sales_view cached?", is_cached)
```

```
Is home_sales_view cached? False
```