**Methodology**

In this project, I utilized **SparkSQL** to perform various data analysis tasks on a home sales dataset. The main objective was to answer several questions while optimizing query performance through caching, uncaching, and partitioning strategies.

**1. Importing Necessary PySpark SQL Functions**

The first step involved importing the essential PySpark SQL functions required for data processing and querying. This included functions for handling columns, rounding off numerical results, and managing DataFrames within Spark.

**2. Reading Data into PySpark DataFrame**

Once the necessary functions were imported, I proceeded to read the home\_sales\_revised.csv file from the provided AWS S3 bucket into a PySpark DataFrame.

**3. Creating a Temporary Table**

After loading the data into a DataFrame, I created a temporary table called home\_sales. This allowed me to run SQL queries directly on the DataFrame using SparkSQL, enabling more flexibility in data manipulation and analysis.

**4. Answering Queries Using SparkSQL**

Using **SparkSQL**, I addressed several questions related to home sales:

* What is the average price for a four-bedroom house sold for each year, rounded to two decimal places?
* What is the average price of a home for each year the home was built, with three bedrooms and three bathrooms?
* What is the average price of a home for each year the home was built, with three bedrooms, three bathrooms, two floors, and >= 2,000 square feet?
* What is the average price of a home per "view" rating, for homes with an average price >= $350,000, and what is the runtime of this query?

These queries helped extract valuable insights regarding the trends in home prices over time based on different attributes such as the number of bedrooms, bathrooms, floors, and view ratings.

**5. Caching the Temporary Table**

To optimize the performance of repeated queries, I cached the home\_sales temporary table. This step stored the data in memory, significantly improving the speed of subsequent queries that accessed the same data.

**6. Running Queries Using Cached Data**

Once the table was cached, I re-ran the query regarding the **average price of homes per "view" rating** with a home price greater than or equal to $350,000.

**7. Partitioning Data and Storing in Parquet Format**

To further optimize data storage and query performance, I partitioned the data by the date\_built field and saved the data as a **Parquet** file.

**8. Creating a Temporary Table for Parquet Data**

After partitioning the data and saving it in Parquet format, I created a new temporary table from the Parquet file. This allowed me to continue querying the partitioned data using SparkSQL.

**9. Running the Query on Partitioned Data**

I then executed the same query regarding the average price of homes per "view" rating on the partitioned Parquet data.

**10. Uncaching the Table**

Finally, after completing the analysis, I uncached the home\_sales temporary table to free up memory. I also verified that the table had been uncached successfully.

**Conclusion**

Through this project, I utilized PySpark's powerful SQL capabilities to perform detailed data analysis on home sales. By applying techniques like caching, partitioning, and storing the data in efficient formats such as Parquet, I was able to optimize the performance of my queries. reference.