Ethan Handelman
Professor Pierce
CSC 369
10 February 2025

“Week 5 Analysis: Compare & Contrast PySpark”

I chose DuckDB for my analysis thus far because it was the fastest library I used in Week 2. I could write a concise SQL query to calculate color statistics directly on my Parquet files, all within an in-process connection. The entire process, from connecting to the database, executing the query, to fetching the results, felt intuitive and required minimal boilerplate code. This worked especially well on datasets that comfortably fit in memory.

Attempting to use PySpark for its distributed processing capabilities was more difficult than I expected. This move introduced a whole new set of challenges. For one, I had to set up a Spark session and work with DataFrames instead of a simple SQL connection. Unlike DuckDB's one-shot SQL query, PySpark required multiple operations: reading the Parquet file, filtering the DataFrame for the target color, and counting both the filtered and total rows. These operations, triggered lazily, sometimes made debugging difficult because errors only surfaced during execution rather than at the time of writing the code.

Another significant challenge was optimizing the performance of these distributed operations. In DuckDB, the entire aggregation was handled internally by the engine, but in PySpark, I had to be mindful of how actions were distributed across the cluster. For example, performing two separate count actions could lead to redundant computations and increased overhead. I had to refactor my logic to minimize multiple passes over the data and carefully manage resource allocation across nodes. Additionally, ensuring consistency in transformations - like using the lower function for case-insensitive comparisons - required extra attention, as the behavior could differ subtly between the two systems. Overall, the transition demanded a deeper understanding of distributed computing paradigms and more intricate performance tuning.