PROJECT: Billion-Row Analyzer

DuckDB vs Pandas - Big Data Analytics on Consumer Hardware

---

1. AIM

To build a high-performance, local data analytics dashboard capable of processing millions of taxi trip records instantly on consumer hardware without relying on cloud clusters or heavy database servers.

---

2. OBJECTIVES

- **Benchmark Performance**: Quantify the speed difference between traditional in-memory processing (Pandas) and modern in-process OLAP engines (DuckDB)
- **Demonstrate 'Zero-Copy' Analysis**: Show how to query large Parquet files directly from disk without loading them entirely into RAM
- **Create an Interactive Tool**: Build a user-friendly GUI (Streamlit) that allows non-coders to explore Big Data dynamically
- **Run Big Data Workloads Locally**: Execute sophisticated analytics on a standard laptop (Ryzen 5, 16GB RAM) without expensive cloud infrastructure

---

3. WHY THIS PROJECT?

In the Data Science industry, Pandas is the default tool. However, Pandas struggles with datasets larger than available RAM (Memory Error) and is slow for aggregations on millions of rows. This project proves that by switching to DuckDB (a columnar, vectorized SQL engine), we can:

1. Analyze datasets larger than RAM
2. Achieve 10x-50x faster query speeds
3. Run Big Data workloads locally on a laptop without expensive cloud infrastructure

---

4. DATASET INFORMATION

Property	Details
Name	NYC Yellow Taxi Trip Records (January 2023)
Source	NYC Taxi & Limousine Commission (TLC)
Format	Apache Parquet (Columnar storage format, highly compressed and optimized for read-heavy analytics)
Size	~47MB (compressed), representing millions of rows
Key Columns	tpep_pickup_datetime, trip_distance, passenger_count, total_amount, tip_amount, PULocationID, DOLocationID

---

5. TECHNICAL ARCHITECTURE

5.1 Core Technologies

- **DuckDB**: The 'Engine' - Uses vectorized execution to process data in chunks (batches) rather than row-by-row, utilizing CPU cache efficiently. It queries the Parquet file directly without loading the entire dataset into memory.
- **Pandas**: Used as the 'Baseline' for speed comparison and for handling small result sets for visualization and post-processing.
- **Streamlit**: A Python framework to turn data scripts into shareable web apps. It manages the frontend UI, handles user interactions, and displays results.
- **Plotly Express**: For generating interactive charts (bar charts, scatter plots) that users can zoom/hover over for deeper insights.

5.2 Workflow

1. **User Input**: User moves a slider on the Streamlit web interface (e.g., 'Filter trips between 0 and 20 miles')
2. **Query Generation**: Python dynamically constructs a SQL query based on these inputs
3. **Execution**: DuckDB executes this SQL against the taxi_data.parquet file on the disk
4. **Visualization**: The aggregated small results are sent to Plotly/Streamlit to render charts

---

6. CODE EXPLANATION

6.1 Benchmark Script (src/benchmark.py)

This script purely measures speed between Pandas and DuckDB:

run_pandas_benchmark():

- `pd.read_parquet(DATA_PATH)`: This is the bottleneck. It reads the entire file from disk and loads it into RAM.
- `df.groupby(...)`: Performs the calculation using standard CPU processing.

run_duckdb_benchmark():

- `duckdb.query(...)`: Does not load the file. It scans only the necessary columns (passenger_count, trip_distance) directly from the disk.
- **Result**: DuckDB is vastly faster because it ignores unused data columns and uses parallel processing.

6.2 Dashboard App (src/app.py)

This is the interactive component:

- **@st.cache_resource**: We don't want to reconnect to the database every time the user clicks a button. This "decorator" tells Streamlit: "Run this function once, save the connection, and reuse it."
- **Dynamic SQL Construction**: Python injects the values from the sidebar slider (distance_filter) directly into the SQL string, making the dashboard responsive to user inputs.
- **The Visualization Trick**: You cannot plot 1 million dots on a scatter plot; the browser will crash. Solution: USING SAMPLE 1000. We use SQL to randomly pick only 1000 rows for the Scatter Plot (Tip vs Distance), giving visual trends without performance cost.
- **SQL Playground**: st.text_area creates a box for raw SQL. con.execute(user_query) takes whatever the user typed and runs it as code.

---

7. GIT & VERSION CONTROL

Command	Description
`git init`	Initializes a new empty repository
`git config --global user.name "Name"`	Sets your identity for commits
`git config --global user.email "Email"`	Sets your email for commits
`git status`	Shows which files have changed
`git add .`	Stages ALL changed files
`git commit -m "message"`	Saves staged files with message
`git remote add origin URL`	Connects local folder to GitHub
`git push -u origin main`	Uploads commits to GitHub
`pip freeze > requirements.txt`	Updates list of libraries used

---

8. HOW TO RUN (QUICK GUIDE)

Step 1: Activate Virtual Environment

.\venv\Scripts\activate

Step 2: Run Benchmark (Terminal Output)

python src/benchmark.py

Step 3: Run Dashboard (Browser UI)

streamlit run src/app.py

Step 4: Stop Dashboard

Press `Ctrl + C` in the terminal.

Document prepared for professional distribution and PDF conversion