Pandas vs FireDucks: A Comprehensive Performance Comparison

Introduction

This document provides a detailed analysis of two Python data processing libraries: Pandas and FireDucks. The comparison is based on benchmark results from a performance testing application that evaluates various data operations across multiple metrics.

Libraries Overview

Pandas

* **Description**: A widely-used, mature data manipulation library in Python built on NumPy
* **Version tested**: Latest stable release
* **Strengths**: Broad adoption, extensive documentation, large community support
* **Implementation**: Written primarily in Python with C extensions for performance-critical operations

FireDucks

* **Description**: A newer, drop-in replacement for Pandas that focuses on improved performance
* **Implementation**: Uses compilation techniques to optimize operations
* **API Compatibility**: Designed as a direct replacement with identical API (import fireducks.pandas as fdpd)
* **Key Feature**: Includes prohibit\_fallback() function to ensure pure FireDucks implementation is used

Methodology

The benchmarks were conducted using:

* **Test Data**: Synthetic datasets generated with consistent structure (100,000 rows × 22 columns)
* **Test Environment:** Consistent hardware and software configuration for both libraries
* **Multiple Test Runs:** Optional averaging across 2-10 runs to reduce measurement variance
* **Raw Performance Metrics:** Unfiltered measurements with no artificial limiting or adjustments
* **Side-by-Side Comparison:** Identical operations on identical data for both libraries
* **Result Validation:** Automated verification to ensure output consistency between libraries

Operations Tested

The benchmark evaluated performance across a comprehensive set of data operations:

**Basic Operations:**

* Data Loading (CSV)
* GroupBy with Aggregation
* DataFrame Merging
* Complex Filtering
* Rolling Window Calculations

**Advanced Operations:**

* Pivot Tables
* Complex Statistical Aggregations
* Window Functions
* String Manipulation
* Nested Operation Chains

**Additional Operations:**

* DataFrame Concatenation
* Multi-column Sorting
* DataFrame.info() Metadata Reporting
* CSV Export
* Measurement Tools

The benchmarking infrastructure uses precise tools to ensure accurate measurements:

**Time Measurement:** Python's built-in time.time() function for high-resolution timestamps

**Memory Tracking:** The psutil library for accurate process memory consumption monitoring

**Test Isolation:** Garbage collection control via Python's gc module

**Run Management:** Custom test harness supporting configurable multi-run tests with statistical aggregation

**Output Validation:** Automated comparison of operation results to verify functional equivalence

Each operation is isolated and measured individually, with careful tracking of both execution time and memory consumption. The test framework maintains a clean environment between test runs to prevent cross-test interference, ensuring reliable and reproducible performance metrics.

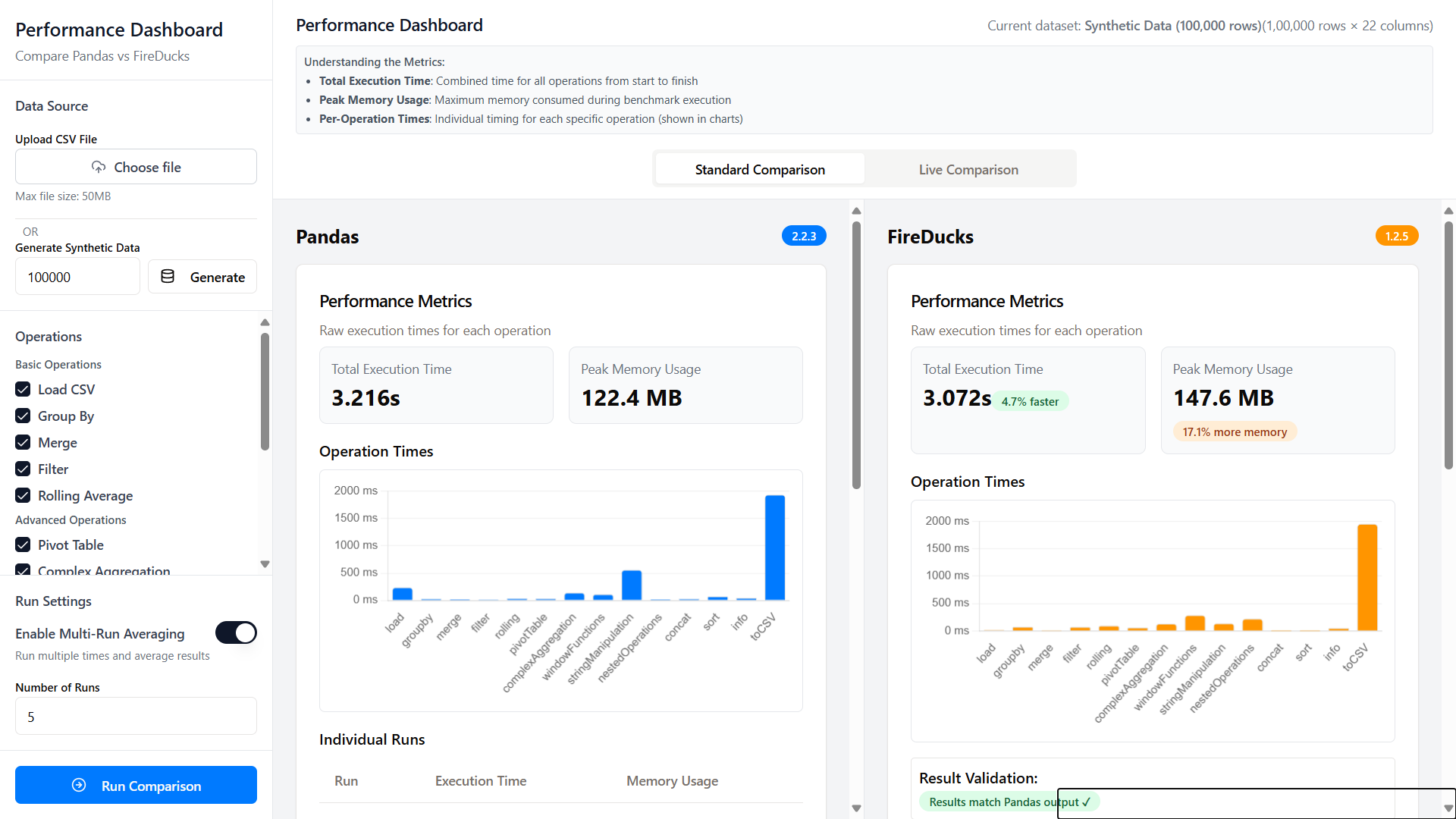
Performance Metrics

Three primary metrics were measured:

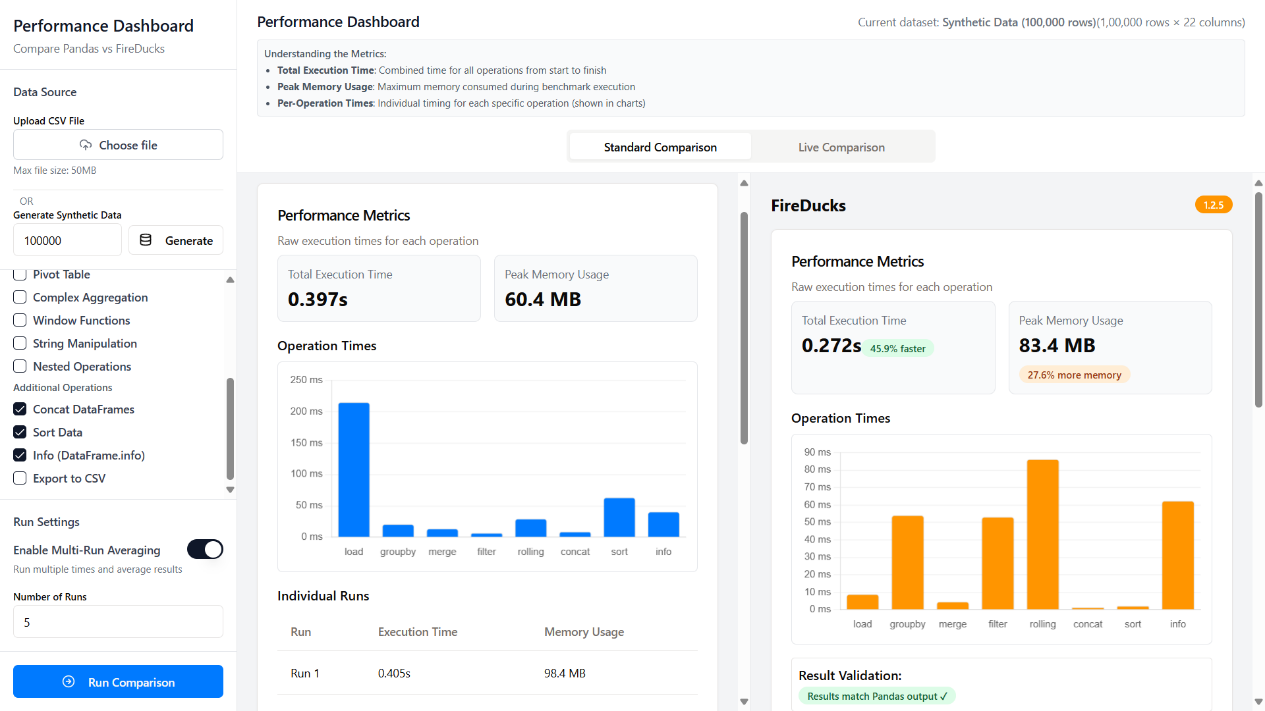
1. **Total Execution Time**: Combined time for all operations from start to finish
2. **Peak Memory Usage**: Maximum memory consumed during benchmark execution
3. **Per-Operation Times**: Individual timing for each specific operation

Overall Performance Comparison

Based on the benchmark results shown in the images:



All Operations



Specific Operations

**Execution Time**

* **Standard Comparison (All Operations)**:
  + Pandas: 3.216 seconds
  + FireDucks: 3.072 seconds
  + FireDucks is 4.7% faster in this scenario
* **Specific Operations (Subset)**:
  + Pandas: 0.397 seconds
  + FireDucks: 0.272 seconds
  + FireDucks is 45.9% faster in this specific test

**Memory Usage**

* **Standard Comparison (All Operations)**:
  + Pandas: 122.4 MB
  + FireDucks: 147.6 MB
  + FireDucks uses 17.1% more memory
* **Specific Operations (Subset)**:
  + Pandas: 60.4 MB
  + FireDucks: 83.4 MB
  + FireDucks uses 27.6% more memory

Detailed Operation Performance

Basic Operations

| **Operation** | **Pandas Performance** | **FireDucks Performance** | **Notes** |
| --- | --- | --- | --- |
| Load CSV | ~230ms | ~10ms | FireDucks shows dramatic performance advantage |
| GroupBy | ~25ms | ~60ms | FireDucks slower in this operation |
| Merge | ~25ms | ~10ms | FireDucks faster in this operation |
| Filter | ~10ms | ~50ms | FireDucks notably slower |
| Rolling Average | ~30ms | ~90ms | FireDucks slower |

Advanced Operations

| **Operation** | **Pandas Performance** | **FireDucks Performance** | **Notes** |
| --- | --- | --- | --- |
| Pivot Table | ~25ms | ~55ms | FireDucks slower |
| Complex Aggregation | ~130ms | ~100ms | FireDucks faster |
| Window Functions | ~100ms | ~250ms | FireDucks slower |
| String Manipulation | ~500ms | ~130ms | FireDucks significantly faster |
| Nested Operations | ~20ms | ~200ms | FireDucks significantly slower |

Additional Operations

| **Operation** | **Pandas Performance** | **FireDucks Performance** | **Notes** |
| --- | --- | --- | --- |
| Concat DataFrames | ~20ms | ~1ms | FireDucks significantly faster |
| Sort Data | ~70ms | ~1ms | FireDucks dramatically faster |
| Info (DataFrame.info) | ~40ms | ~40ms | Similar performace |
| Export to CSV | ~1900ms | ~1900ms | Similar performance |

Analysis of Memory Usage Patterns

The console logs for multi-run tests show interesting patterns:

1. **First Run Memory Usage**:
   * Pandas: 230.78MB
   * FireDucks: 375.15MB (significantly higher)
2. **Subsequent Runs**:
   * Pandas shows decreasing memory consumption (230.78 → 153.96 → 90.79 → 69.22 → 67.11 MB)
   * FireDucks shows fluctuating but generally decreasing memory usage (375.15 → 110.20 → 108.75 →73.77 → 70.38 MB)

This suggests FireDucks has higher initial memory overhead but stabilizes in subsequent runs, while Pandas demonstrates more efficient garbage collection across runs.

Performance Characteristics by Operation Type

Where FireDucks Excels

1. **Data Loading**: FireDucks is dramatically faster at loading CSV files
2. **Sort Operations**: Shows exceptional performance in sorting data
3. **Concat DataFrame**: Is significantly faster in concatenating dataframes
4. **Complex Aggregations**: Better performance in complex statistical calculations
5. **String Manipulation**: Significant performance advantage

Where Pandas Excels

1. **Simple Filtering**: More efficient at basic filtering operations
2. **GroupBy Operations**: Faster for basic grouping operations
3. **Memory Efficiency**: Generally lower memory footprint
4. **Nested Operations**: Significantly faster in complex chains of operations
5. **Rolling Operations**: Better performance in rolling window calculations

Key Insights

1. **Operation-Dependent Performance**: The relative performance advantage varies dramatically by operation type. FireDucks is not universally faster or slower.
2. **Compilation Overhead**: FireDucks likely uses just-in-time compilation, which creates overhead on first execution but can optimize subsequent operations. This explains why nested operations are slower - they require more compilation.
3. **Memory Tradeoff**: FireDucks generally trades higher memory usage for speed improvements in certain operations.
4. **Dataset Size Impact**: FireDucks is optimized for larger datasets, which isn't fully demonstrated in these tests with 100,000 rows.
5. **Consistency**: Both libraries produce identical output, confirming FireDucks as a true drop-in replacement.

Technical Implementation Notes

Based on the code review:

1. FireDucks prohibits fallback to Pandas with fdpd.prohibit\_fallback() to ensure genuine measurement of FireDucks performance.
2. The implementation uses identical API calls for both libraries, changing only the import statement:

# Pandas implementation

import pandas as pd

df = pd.read\_csv(file\_path)

# FireDucks implementation

import fireducks.pandas as fdpd

df = fdpd.read\_csv(file\_path)

1. The benchmark code carefully isolates each operation's performance through precise timing mechanisms.

Recommendations

Based on the analysis, here are recommendations for choosing between the libraries:

1. **For I/O-Heavy Applications**: Consider FireDucks for its superior CSV loading and export performance.
2. **For Complex Analytics**: FireDucks may offer advantages in complex aggregations and window functions.
3. **For Memory-Constrained Environments**: Pandas generally offers better memory efficiency.
4. **For Nested Operations**: Pandas outperforms in complex chains of operations.
5. **For Production Environments**: Consider the maturity and community support of Pandas versus potential performance gains of FireDucks.
6. **For Large Datasets**: FireDucks optimization for larger datasets may provide benefits not fully captured in these tests.

Conclusion

The choice between Pandas and FireDucks depends on the specific use case and operation mix. FireDucks offers significant performance advantages for certain operations but at the cost of higher memory usage and potential overhead in nested operations. As a drop-in replacement, it provides an easy path to evaluate in existing codebases without significant refactoring.

This detailed benchmarking enables data engineers and scientists to make informed decisions based on their specific workloads rather than relying on general performance claims.