P ANIRUDH
2211CS010464PDF

Full-Scale Stock Analytics Using PySpark and Pandas: A Scalable Framework for Financial Insight Extraction

Abstract

This paper presents a scalable, reproducible framework for stock market analytics using PySpark and Pandas. Designed for educational and research contexts, the pipeline demonstrates how distributed computing and in-memory processing can be combined to extract actionable insights from historical stock data. The methodology includes return calculations, volatility analysis, moving averages, cumulative returns, and volume-based anomaly detection, supported by visualizations for interpretability. The framework is modular, extensible, and optimized for teaching time-series analysis and reproducible data science workflows.

---

1. Introduction

Financial markets generate vast amounts of time-series data that require scalable and interpretable analytics. Traditional tools such as Excel or standalone Python scripts often fall short in handling large datasets or ensuring reproducibility. This research integrates PySpark's distributed processing capabilities with Pandas' visualization strengths to build a robust pipeline for stock analytics. The project is designed to be teachable, reproducible, and extensible across multiple datasets and timeframes.

The primary goals of this framework are:

- To demonstrate scalable computation of financial indicators
- To visualize trends and anomalies for interpretability
- To support academic instruction in time-series analysis and PySpark window functions

---

2. Dataset Description

The dataset used in this study is a historical stock file (a.us.txt) containing daily trading data for a U.S.-listed company. It includes the following fields:

- Date: Trading date (YYYY-MM-DD)
- Open: Opening price
- High: Highest price of the day
- Low: Lowest price of the day
- Close: Closing price
- Volume: Number of shares traded

The dataset spans multiple years and is formatted as a CSV file with headers.

---

3. Tools and Technologies

Tool	Purpose
PySpark	Scalable data processing and window functions
Pandas	Data conversion and manipulation
Matplotlib	Line, bar, and scatter plots
Seaborn	Distribution plots and styling
Jupyter	Interactive development and teaching

---

4. Methodology

4.1 Data Preparation

- Loaded CSV into PySpark DataFrame
- Converted Date column to proper format
- Sorted chronologically and cached for performance

4.2 Analytical Metrics

4.2.1 Daily Return

[\text{Daily Return}t = \$frac{P_t - P$\{t-1\}\}\{P_\{t-1\}\}] Where (P_t) is the closing price on day (t), and (P_{t-1}) is the previous day's close.

4.2.2 Volatility (21-day rolling)

[\text{Volatility}$\{21\}$ = \$text{stddev}(\$text{Daily Return}$\{t-20 \rightarrow t\})$]

4.2.3 Moving Averages

- MA20: 20-day average of closing price
- MA50: 50-day average of closing price

[\text{MA}n = \$frac{1}{n} \$sum$\{i=0\}^\{n-1\}$ P_{t-i}]

4.2.4 Cumulative Return

[\text{Cumulative Return}t = \$exp\$left(\$sum$\{i=1\}^\{t\}$ \log\left(\frac{P_i}{P_{i-1}}\right)\right)]

4.2.5 Volume Analysis

- Identified top 10 days with highest trading volume
- Correlated volume spikes with price movement

4.2.6 Monthly Trends

- Grouped by calendar month to compute average close
- Revealed seasonal or cyclical behavior

---

5. Results

5.1 Daily Return and Volatility

Daily returns revealed short-term fluctuations, while rolling volatility highlighted risk periods. High volatility often coincided with volume spikes, suggesting market reactions to news or earnings.

5.2 Trend Detection

Moving averages smoothed price noise and revealed momentum shifts. Crossovers between MA20 and MA50 indicated potential buy/sell signals, consistent with technical analysis literature.

5.3 Cumulative Performance

Cumulative return plots showed long-term growth trajectories, useful for benchmarking investment strategies and comparing asset performance.

5.4 Volume Insights

Volume spikes correlated with price jumps or dips, suggesting institutional activity or news-driven movement. Scatter plots showed moderate correlation between volume and volatility.

5.5 Monthly Trends

Monthly average close prices revealed cyclical behavior, with certain months consistently outperforming others. This insight supports seasonal trading strategies and calendar-based forecasting.

---

6. Visualizations

Plot Type	Metrics Visualized
Line Plot	Daily Return, Cumulative Return, MA trends
Bar Plot	Volume spikes, Monthly averages
Scatter Plot	Volume vs Close Price, Return vs Volatility

All plots were generated using Pandas and Matplotlib for clarity and teaching impact. Each visualization was annotated with axis labels, titles, and legends to support interpretability.

---

7. Reproducibility

Project Structure

stock-analytics/

├── a.us.txt # Raw dataset

├── stock_analysis.ipynb # Main notebook

├── requirements.txt # Python dependencies

└── README.md # Project overview

Setup Instructions

git clone https://github.com/your-username/stock-analytics.git

cd stock-analytics

pip install -r requirements.txt

jupyter notebook

Dependencies

- Python ≥ 3.8
- PySpark ≥ 3.4
- Pandas ≥ 1.5
- Matplotlib ≥ 3.7
- Seaborn ≥ 0.12

---

8. Educational Value

This project is ideal for:

- Teaching time-series analysis
- Demonstrating PySpark window functions
- Visualizing financial metrics
- Introducing reproducible data science workflows
- Training students in Git-based project hygiene and modular analytics

---

9. Limitations and Future Work

- The current pipeline uses simple moving averages; future work may incorporate exponential moving averages (EMA), Bollinger Bands, and MACD.
- The dataset is limited to a single stock; future extensions could batch process multiple tickers and compare performance.
- Real-time data integration and dashboard deployment (e.g., via Streamlit) are planned for future iterations.

---

10. Conclusion

This research demonstrates a reproducible, scalable, and educational pipeline for stock analytics. By combining PySpark's distributed capabilities with Pandas' visualization strengths, the framework enables deep financial insight extraction and supports teaching, research, and investment strategy development. The modular design allows for easy extension to other datasets and indicators, making it a valuable tool for both academic and professional use.

References

- Real-Time Data Processing: An Analysis of PySpark's Capabilities
- PySpark with Pandas: A Comprehensive Guide
- Financial Time Series Analysis with Python
- GitHub: https://github.com/your-username/stock-analytics
