Stock market analysis and prediction

Team Name: The Bull Runners

Authors:

- Sandeep Pulavarthi : sandeeppulav@iisc.ac.in
- S Varun : varuns1@iisc.ac.in
- Arun Kumar A : aruna@iisc.ac.in
- Akhzar Farhan : akhzarfarhan@iisc.ac.in

Problem

- **Definition**

The goal of this project is to predict the future stock prices based on historical stock market data. Given a dataset of past stock prices, trading volumes, and other related features, we will develop a machine learning model to forecast future stock price movements or trends. Specifically, the task is to build a model that can predict either the next day's closing price or the future trend (e.g., whether the stock price will go up or down) for a given stock. The model will be implemented using PySpark to handle large datasets efficiently and perform distributed processing, enabling scalability and speed in training the model.

- **Problem Motivation**

1. Volatility and Financial Risk Management:

Predicting stock price trends is challenging due to the volatility of financial markets. Accurately forecasting stock price movements allows investors and traders to make better decisions, optimizing returns and reducing risks. By leveraging machine learning models, investors can improve the predictability of market movements based on historical trends.

2. Handling Large Datasets:

Financial data, such as stock prices and volumes, often spans decades or covers many different assets, resulting in large datasets. PySpark's ability to process data in parallel across clusters of machines allows for efficient handling of this massive amount of data, making it a suitable tool for the problem of stock price prediction, even without real-time data.

3. Improved Accuracy Over Traditional Methods:

Traditional stock price prediction methods, such as basic moving averages or linear regression, can struggle to capture the complexities in stock price behavior. Machine learning models can better identify complex, non-linear relationships in the data. By using PySpark to train models on large datasets, we can explore more sophisticated algorithms (e.g., Random Forest Regressor, ARIMA, etc) that can capture these complex relationships more effectively.

4. Forecasting Stock Trends and Market Behavior:

Understanding whether a stock will go up or down in the future is of great interest to investors. Even with historical data, trends such as momentum, volatility, and seasonality are valuable for determining future price directions. By training models with large volumes of historical data, we aim to identify patterns that can guide future trading or investment decisions.

6. Scalability and Parallel Processing:

PySpark is designed for scalable machine learning, and its distributed nature is an advantage when working with large stock market datasets. In this project, we aim to leverage PySpark's capabilities to handle large datasets of historical stock prices and optimize the training process without running into memory or processing bottlenecks, enabling a more efficient modeling pipeline.

- **Design Goals**:

1. Data Processing :
   Ensure the data set is free from errors and inconsistencies by handling missing values, removing outliers etc.
2. Feature Engineering:
   Create new features that can improve model performance.
3. Model Training :
   Evaluate and choose appropriate models for stock price forecasting such as Random Forest regression, ARIMA, etc. Train the model in distributed and non distributed methods and analyse the performances.
4. Visualisation and Reporting :
   Plot the key performance metrics and visualize the historical stock price trends alongside predicted price trajectories to allow for intuitive comparison between actual and predicted movements over time.

- **Features Required:**

Stock Price Data:

- Open Price: The price of the stock at the beginning of the trading day.
- Close Price: The price of the stock at the end of the trading day.
- High Price: The highest price the stock reached during the day.
- Low Price: The lowest price the stock reached during the day

- **Scalability/Performance Goals**

Handling Large Datasets Efficiently: Process vast amounts of historical stock data without performance issues.

Scaling with Multiple Stocks: Support analysis of multiple stocks simultaneously.

Fast Data Processing: Minimize preprocessing time for large datasets

Reducing Model Training Time: Speed up the training of complex models.

Model Prediction Speed: Ensure quick predictions, even for large datasets

Approach and Methods

- Feature Engineering:
  For the Random regressor, we have created new features called lags based on the 'Price' feature for the last 5 data points.
  For the Arima model, we have grouped the timestamps day wise and sampled from the grouping as per model needs.
- Train-Test Split:
  An 80:20 split is performed, where 80% of the data is used for training, and 20% is reserved for testing.
- Dataset Fractions:
  The models are trained and evaluated using different fractions of the dataset: [0.1, 0.2, 0.5, 0.8, 1.0], representing increasing dataset sizes.

- Machine learning methods:
  For Distributed Computing(Actual Model):
  Algorithm: Random Forest from PySpark's MLlib.
  Data Storage: Data is stored and processed using PySpark DataFrames.

  For Single-Node (Non-Distributed) Computing (For Comparison):
  Algorithm: Random Forest from scikit-learn.
  Data Storage: Data is stored and processed using standard Python DataFrames.

Evaluation:

Following evaluation metrics were used to assess the models' performance

- Mean Absolute Error (MAE): The average of the absolute differences between predicted and actual values.

$$\mathrm{MAE} = \frac{1}{n}\sum_{i=1}^{n} \left|Y_i - \widehat{Y_i}\right|$$

- Mean Squared Error (MSE): The average of the squared differences between predicted and actual values.

$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2$$

- Root Mean Squared Error (RMSE): The square root of the mean squared error, providing error in the same units as the target variable

$$RMSE = \sqrt{\sum_{i=1}^{n} \frac{(\hat{y}_i - y_i)^2}{n}}$$

- R-squared (R²): Measures the proportion of variance in the dependent variable that is predictable from the independent variables.

$$R^2 = 1 - \frac{\sum_{i=1}^{n} (y_i - \hat{y}_i)^2}{\sum_{i=1}^{n} (y_i - \bar{y})^2}$$

- Mean Absolute Percentage Error (MAPE): Measures the average magnitude of error produced by a model, or how far off predictions are on average.

$$\text{MAPE} = \frac{1}{n} \sum_{i=1}^{n} \left| \frac{Y_i - \hat{Y}_i}{Y_i} \right|$$

- Mean Absolute Scaled Error (MASE): Measure for determining the effectiveness of forecasts generated through an algorithm by comparing the predictions with the output of a naïve forecasting approach

$$MASE = \frac{\frac{1}{J} \sum_{j=T+1}^{T+J} |y_i - \hat{y}_i|}{\frac{1}{T-1} \sum_{i=2}^{T} |y_i - y_{i-1}|}$$

where

Y is actual stock price

Ŷ is predicted stock price

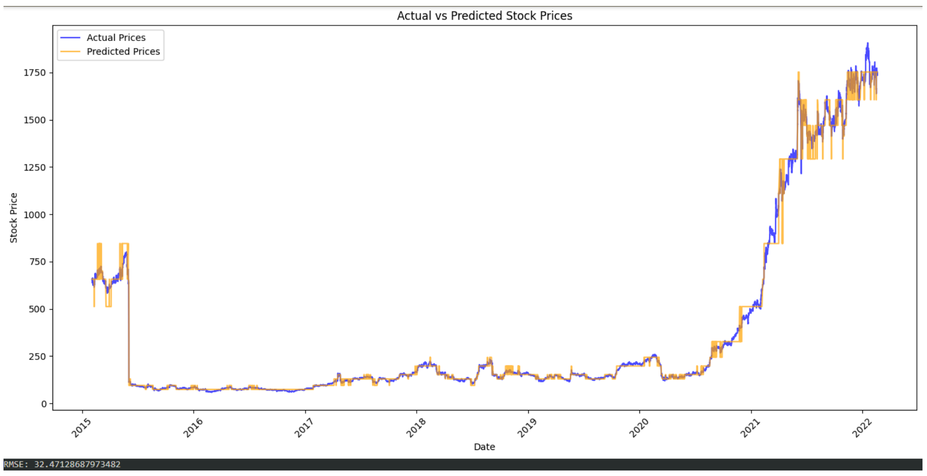
n is no of data points

T is size of train data

J is size of test data

Observations & Results

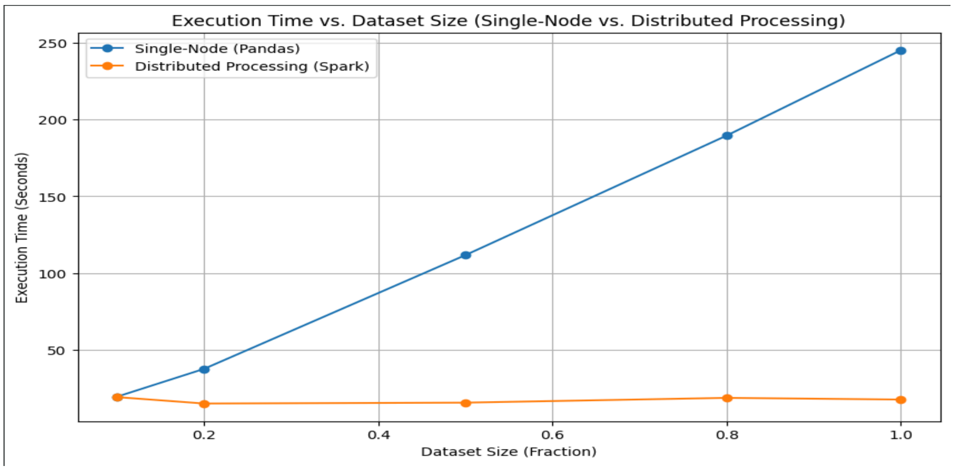
Plot 1 : Actual Stock Price vs Predicted Stock prices for the Test Data



```
RMSE: 32.47128687973482
```

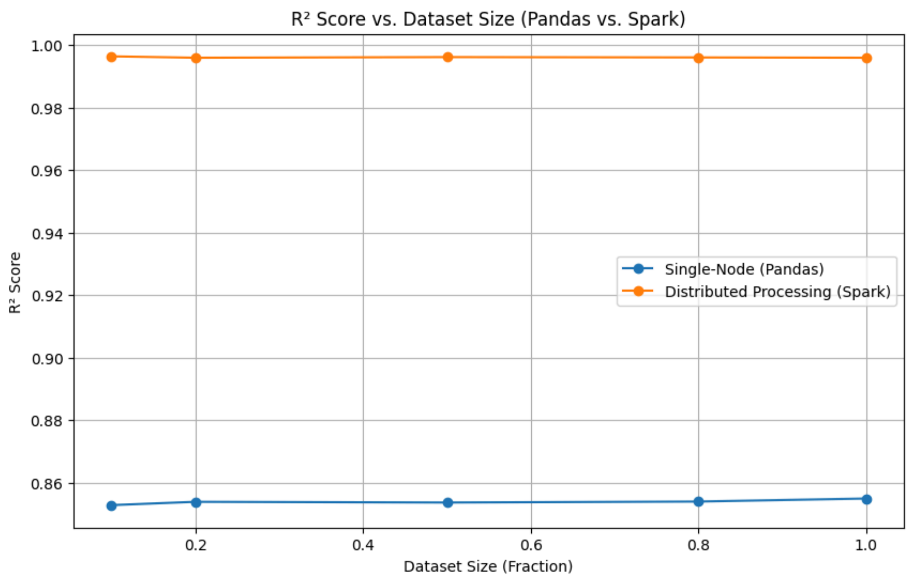
The Spark MLlib Random Forest model demonstrates strong performance in capturing the overall trend of stock prices, as indicated by the close alignment of the predicted prices (orange line) with the actual prices (blue line) over time. The model effectively adapts to long-term upward trends, particularly during periods of rapid price escalation, such as 2020–2021. However, deviations between predicted and actual prices are more noticeable during periods of high volatility, suggesting that the model struggles with short-term price fluctuations and may not fully account for rapid or unexpected changes in stock prices. The **Root Mean Square Error (RMSE) of 32.47** provides a quantitative measure of prediction accuracy, which, while reasonably low relative to the scale of stock prices, highlights potential areas for improvement, particularly in scenarios characterized by high volatility. Overall, the Random Forest algorithm's capability to model non-linear patterns is evident in its ability to track broader trends, though enhancements may be needed to address its limitations in short-term prediction precision.

Plot 2 : Execution Time vs Dataset Size (Single-Node vs Distributed Processing)



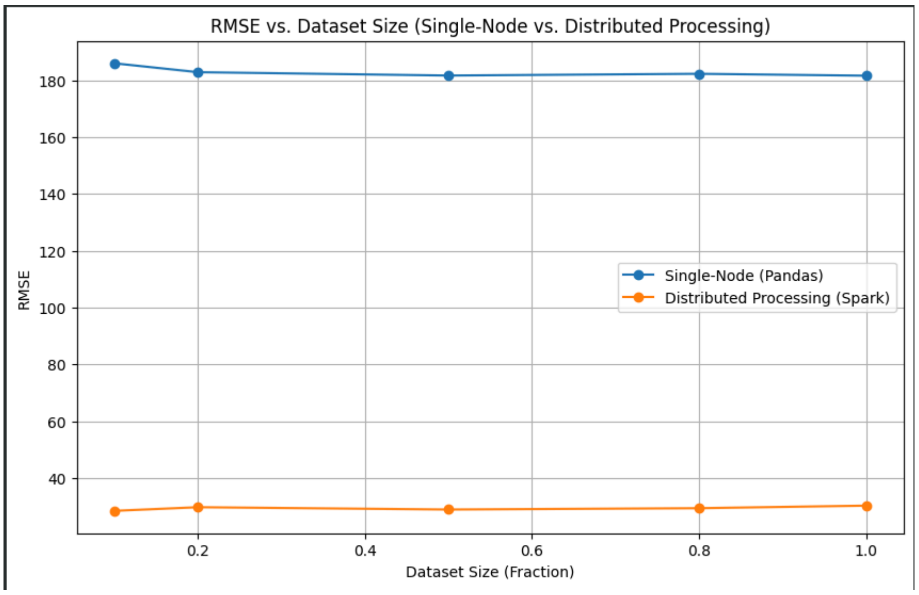
The execution time for single-node processing, where the Random Forest algorithm from scikit-learn is used and the dataset is loaded using Pandas, increases linearly with dataset size. In contrast, the execution time for distributed processing, leveraging the Random Forest implementation from PySpark and datasets loaded as Spark DataFrames, remains nearly constant, showing only a marginal increase as the dataset size grows. For smaller dataset fractions (e.g., 0.2), the performance difference between the scikit-learn and PySpark implementations is minimal; however, as the dataset size approaches its maximum (e.g., 1.0), the gap widens significantly, with the scikit-learn and Pandas-based approach taking substantially more time. PySpark's distributed architecture demonstrates superior scalability, efficiently handling larger datasets without significant performance overhead, whereas the scikit-learn and Pandas-based single-node approach, being in-memory and non-distributed, struggles to maintain performance as dataset size increases

Plot 3 : R2 score vs Dataset Size (Single-Node vs Distributed Processing)



The R² score for Spark's MLlib Random Forest remains consistently close to 1.0 across all dataset sizes (fractions from 0.2 to 1.0), indicating near-perfect predictive accuracy and showcasing its robustness in handling large datasets without degradation in performance. In contrast, the R² score for the Pandas-based implementation using scikit-learn's **Random Forest remains around 0.86**, with limited improvement as dataset size increases, suggesting that the single-node approach is constrained by computational and memory limitations. This disparity highlights Spark's distributed architecture, which processes data in parallel across clusters, ensuring high scalability and consistent predictive accuracy even for large-scale datasets. Meanwhile, the Pandas-based single-node implementation struggles to fully exploit the dataset's complexity and information due to bottlenecks in memory and processing capacity, resulting in capped performance regardless of dataset size.

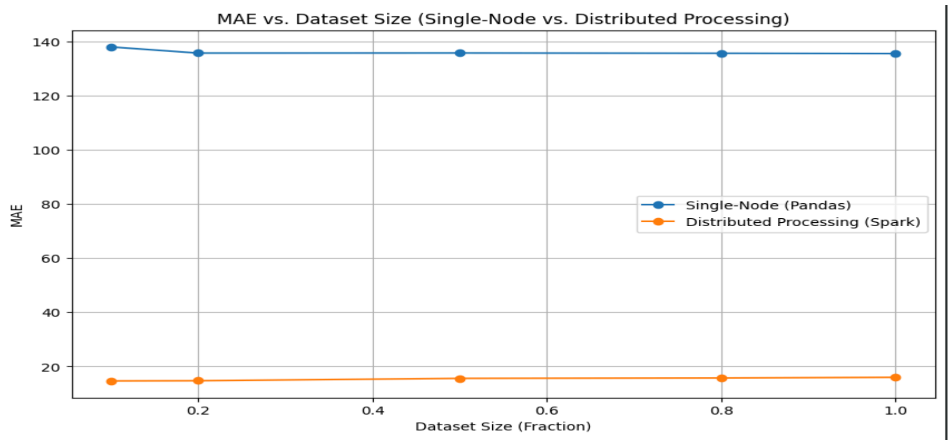
Plot 4 : RMSE vs Dataset Size (Single-Node vs Distributed Processing)



The curve representing distributed processing with Spark MLlib exhibits a significantly lower and nearly constant RMSE across all dataset sizes (fractions from 0.2 to 1.0), demonstrating superior predictive accuracy and robustness compared to the single-node implementation using scikit-learn and Pandas, which consistently shows higher RMSE values regardless of dataset size. This highlights

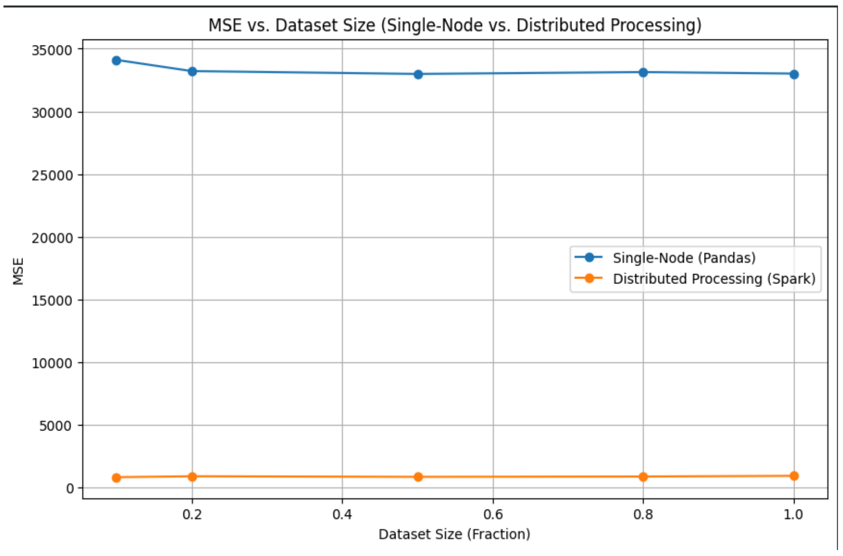
Spark's distributed architecture, which efficiently handles large-scale datasets by leveraging parallel computation and robust data processing, resulting in better model generalization and scalability. In contrast, the single-node approach is constrained by resource limitations, unable to improve performance or fully utilize the complexity of larger datasets, as reflected by its consistently high RMSE and lack of scalability.

Plot 5 : MAE vs Dataset Size (Single-Node vs Distributed Processing)



The distributed processing approach using Spark MLlib demonstrates significantly lower and nearly constant Mean Absolute Error (MAE) across all dataset sizes (fractions from 0.2 to 1.0), indicating consistent and robust predictive accuracy. In contrast, the single-node implementation using scikit-learn's Random Forest with Pandas exhibits consistently higher MAE values, with minimal variation as dataset size increases, underscoring its limitations in predictive accuracy. Spark's distributed architecture enables superior performance by efficiently handling larger datasets while maintaining scalability and accuracy, whereas the single-node approach shows no significant improvement in MAE with larger datasets, likely due to memory and computational constraints, highlighting its inability to fully leverage the complexity of larger data volumes.

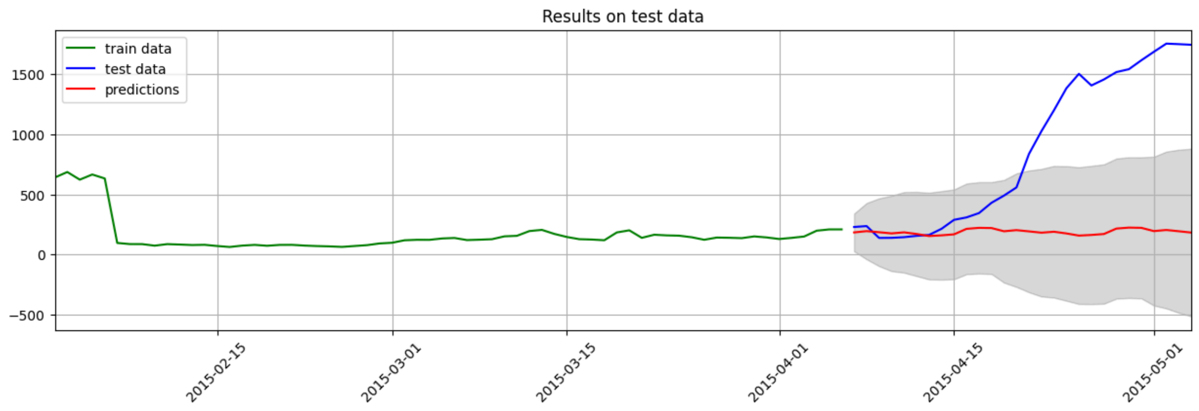
Plot 6 : MSE vs Dataset Size (Single-Node vs Distributed Processing)



The curve for distributed processing using Spark demonstrates a significantly lower and relatively constant Mean Squared Error (MSE) across all dataset sizes (fractions from 0.2 to 1.0), indicating consistent and robust predictive performance. In contrast, the single-node implementation using scikit-learn and Pandas exhibits a consistently higher MSE, remaining near 35,000 regardless of dataset size, highlighting its limitations in predictive accuracy. Spark's distributed Random Forest model showcases superior scalability and robustness, effectively handling larger datasets with minimal variation in MSE, reflecting the capability of its distributed architecture to maintain high accuracy even as data volume increases.

Along with the Random Forest model, we have implemented ARIMA model (Auto Regressive Integrated Moving Average) which is a standard algorithm used in financial analytics to forecast stock prices. Even though pyspark's built-in library doesn't support the ARIMA model, we have tried to group the data and parallelise the operations using third party libraries.

Plot 7 : ARIMA model predictions



The ARIMA model combines three components : AutoRegression (AR), Integration (I), and Moving Average (MA)—to model and predict time series data effectively. From the above plot, we can observe that the model tries to maintain the predictions around moving averages but the model struggles with short-term price fluctuations and may not fully account for rapid or unexpected changes in stock prices. This model has a **Mean Absolute Percentage Error of 0.592** which is good but further improvement can be achieved by training on more stocks to achieve more accurate predictions in high volatile cases.

Summary:

In this project, we developed stock forecasting models leveraging both traditional machine learning techniques and statistical models. We used Random Forest Regression and Autoregressive Integrated Moving Average (ARIMA) models to predict the stock prices, incorporating both distributed and non distributed approaches for enhanced performance and stability.

Future scope of this project could be to deploy the model with user interface. Real time data integration and addition of alternative data such as sentiment analysis and macroeconomic factors to enhance model predictions.