

Stock Price Prediction Using LSTM and Random Forest Regressor with yFinance API data

Team: Data Detectives
Milestone - 03

Team Members:
Greeshma Jale
Kalpana Bolla
Meghala Anumolu
Pravalika Bhupathi
Abhinaya Tanniru

April 19, 2024

1 Introduction

This technical document outlines the process of building a stock price prediction model using Long Short-Term Memory (LSTM) neural networks and Random Forest Regressor. The goal is to predict future stock prices based on historical data and technical indicators.

2 Data Collection and Preprocessing

Data Retrieval:

- Historical stock data for Microsoft (MSFT) is obtained using the yfinance library and saved as a CSV file.

Data Splitting:

- The dataset is split into training (80%) and test (20%) sets and stored in separate CSV files.

Preprocessing: :

- Numerical features are scaled using Min-Max scaling to a range of (0,1).
- Technical indicators such as Moving Averages (MA) and Relative Strength Index (RSI) are computed and added to the dataset.
- Rows with missing values resulting from rolling calculations are dropped.

The Jupyter Notebook interface shows the installation of the 'regressor' library. The code cell [6] contains the following commands:

```
[6] #!pip install yahooquery
    #!pip install yfinance
    !pip install regressor
```

The output shows the download of 'regressor-1.0.9-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (578 kB)' and the successful installation of the library. The next code cell [10] imports the necessary libraries:

```
[10] import os
    import numpy as np
    import pandas as pd
    import matplotlib.pyplot as plt
    from yahooquery import Ticker
    import yfinance as yf
    %matplotlib inline
```

The notebook status bar indicates '0s completed at 2:35 AM'.

Figure 1: Data obtained from Yahoo Finance Library.

The Jupyter Notebook interface shows the fetching of data for the ticker 'MSFT'. The code cell [8] sets the tickers:

```
[8] tickers = ['MSFT']
```

The code cell [9] contains the following commands to download data from yfinance and save it as a CSV file:

```
[9] folder = ("content/sample_data")
    #Downloading data from yfinance
    nosuccess = []
    for ticker in tickers:
        print("working on: ", ticker)
        try:
            ticker_obj = yf.Ticker(ticker)
            df = ticker_obj.history(period='10y', interval='1d')
            the_file = folder + ticker + ".csv"
            df.to_csv(the_file)
            print(ticker + " has been saved to ", folder)
        except FileNotFoundError:
            nosuccess.append(ticker)
            print("found no info on ", ticker)
            continue
```

The output shows the data for MSFT being saved to the folder 'content/sample_data'. The notebook status bar indicates '0s completed at 2:36 AM'.

Figure 2: Fetched data saved as a CSV file.

The Jupyter Notebook interface shows the splitting of the dataset into training and testing sets. The code cell [10] contains the following commands:

```
[10] df = pd.read_csv('content/sample_data/MSFT.csv')
    # Set the ratio for splitting (e.g., 80% training, 20% test)
    train_ratio = 0.8
    test_ratio = 1 - train_ratio

    # Calculate the number of rows for training and test sets
    train_size = int(len(df) * train_ratio)
    test_size = len(df) - train_size

    # Split the DataFrame into training and test sets
    train_df = df.head(train_size)
    test_df = df.tail(test_size)

    # Save the training and test sets to separate CSV files
    train_df.to_csv('train_data.csv', index=False)
    test_df.to_csv('test_data.csv', index=False)
    print("Splitting complete. Training and test data saved to train_data.csv and test_data.csv respectively.")
```

The output shows the splitting complete and the training and test data saved to 'train_data.csv' and 'test_data.csv' respectively. The notebook status bar indicates '0s completed at 2:37 AM'.

Figure 3: dataset is split into train and test sets and stored in separate CSV files.

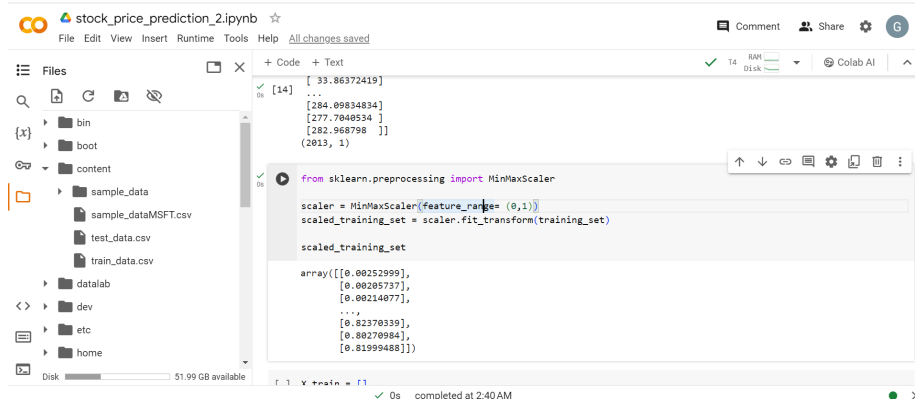


Figure 4: Numerical features are scaled using Min-Max scaling to a range of (0,1).

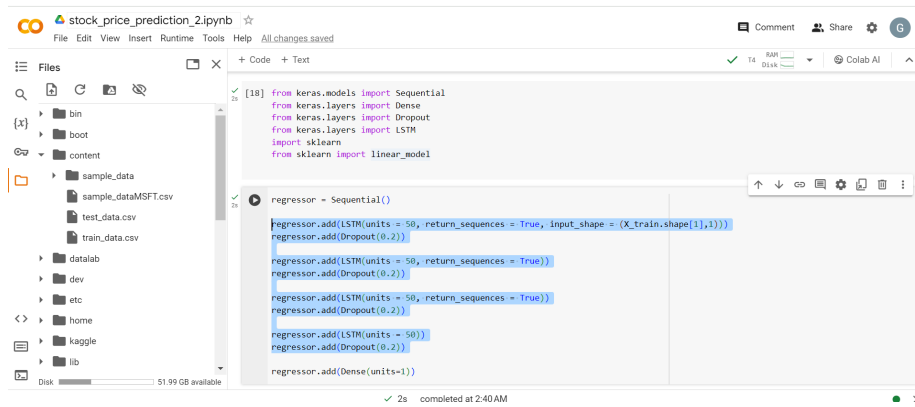


Figure 5: Four LSTM layers with 50 units each.

3 Model Preparation

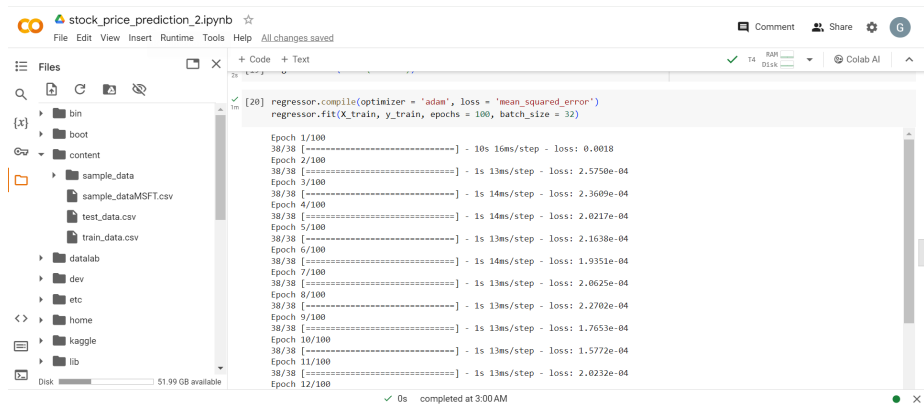
Long Short Term Memory (LSTM)

Architecture:

- Four LSTM layers with 50 units each are stacked, followed by a dropout rate of 0.2 after each LSTM layer to prevent overfitting.
- Input shape is defined based on the time steps and features of the training data ($X_{train}.shape[1], 1$).

Compilation:

- The LSTM model is compiled using the Adam optimizer and Mean Squared Error (MSE) loss function for regression tasks.



```
[20] regressor.compile(optimizer = 'adam', loss = 'mean_squared_error')
regressor.fit(X_train, y_train, epochs = 100, batch_size = 32)

Epoch 1/100
38/38 [=====] - 10s 16ms/step - loss: 0.0018
Epoch 2/100
38/38 [=====] - 1s 13ms/step - loss: 2.5750e-04
Epoch 3/100
38/38 [=====] - 1s 14ms/step - loss: 2.3609e-04
Epoch 4/100
38/38 [=====] - 1s 14ms/step - loss: 2.0217e-04
Epoch 5/100
38/38 [=====] - 1s 13ms/step - loss: 2.1638e-04
Epoch 6/100
38/38 [=====] - 1s 14ms/step - loss: 1.9351e-04
Epoch 7/100
38/38 [=====] - 1s 13ms/step - loss: 2.0625e-04
Epoch 8/100
38/38 [=====] - 1s 13ms/step - loss: 2.2702e-04
Epoch 9/100
38/38 [=====] - 1s 13ms/step - loss: 1.7653e-04
Epoch 10/100
38/38 [=====] - 1s 13ms/step - loss: 1.5772e-04
Epoch 11/100
38/38 [=====] - 1s 13ms/step - loss: 2.0232e-04
Epoch 12/100
```

Figure 6: LSTM model is compiled using the Adam optimizer and Mean Squared Error (MSE) loss function.

4 Training the LSTM Model

Data Transformation:

- Training data is reshaped to fit the LSTM input shape (samples, time steps, features).

Training Process:

- The LSTM model is trained using the training data over 100 epochs with a batch size of 32 to learn temporal dependencies and predict future stock prices.

5 Feature Engineering

Additional Features:

- Technical features like Moving Averages (MA) and Relative Strength Index (RSI) are computed and included in the dataset.

Normalization:

- Numerical features are normalized using Min-Max scaling to ensure uniform feature ranges across the dataset.

6 Training Random Forest Regressor

Model Selection:

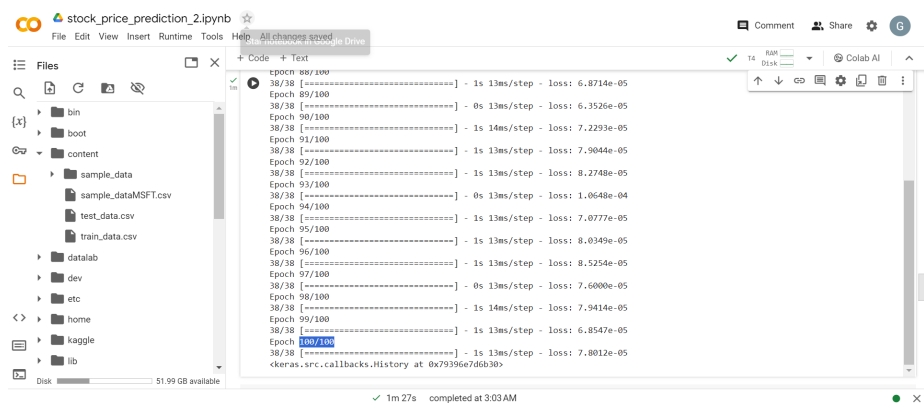


Figure 7: The LSTM model is trained using the training data over 100 epochs with a batch size of 32.

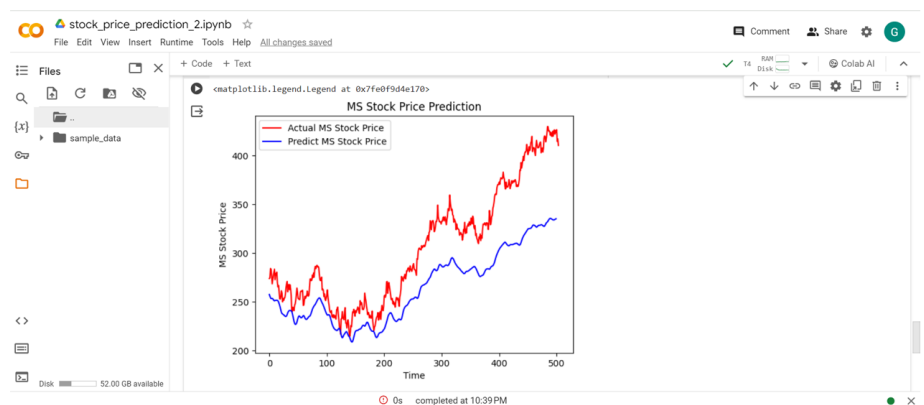


Figure 8: Graph showing the actual and predicted stock prices

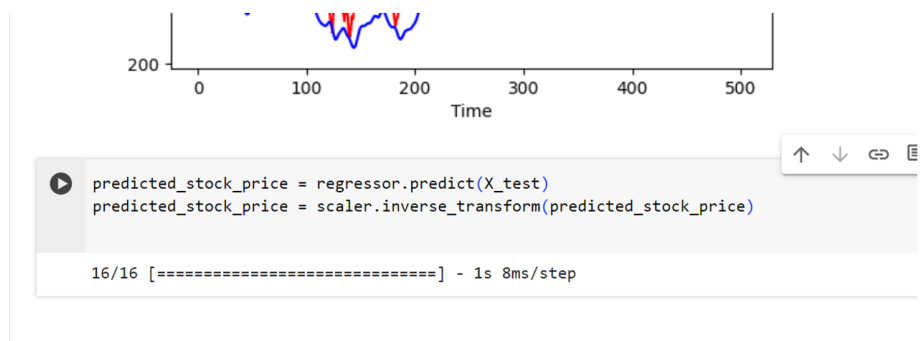


Figure 9: shows the number of batches processed (16/16) and the time taken per batch is indicated in milliseconds (8ms/step).

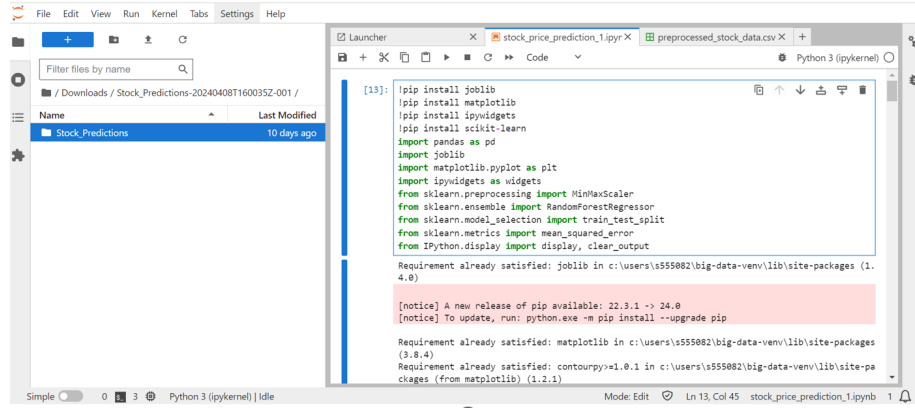


Figure 10: Packages for Random Forest Regressor successfully installed.

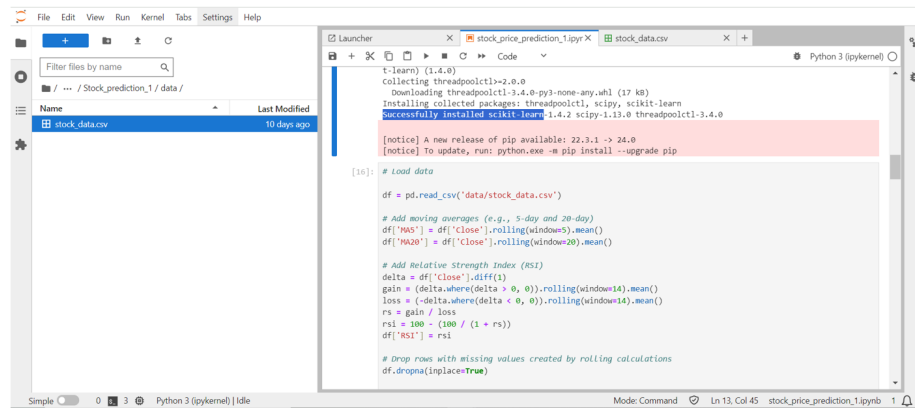


Figure 11: Loading MSFT stock data

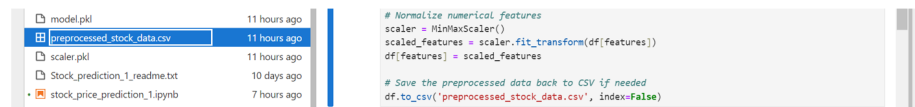


Figure 12: Preprocessed data generated.

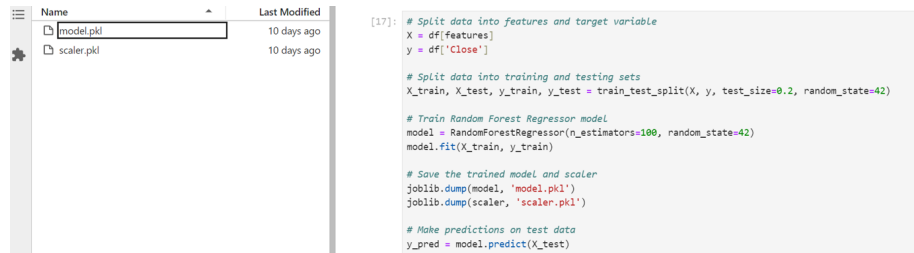


Figure 13: Splitting the dataset(train and test sets) and training the Random Forest Regressor Model.

- A Random Forest Regressor model from sklearn is trained using preprocessed features and target variables (closing prices).

Model Configuration:

- The Random Forest model is configured with 100 decision trees (`n_estimators=100`) and a specified random state for reproducibility.

7 Model Evaluation and Prediction

- **Model Saving:** Trained LSTM and Random Forest models, along with the scaler used for normalization, are saved for future use.
- **Prediction Phase:** Predictions are generated on the test data using the trained Random Forest model to forecast future stock prices.
- **Visualization:** Actual vs. predicted stock prices are plotted to evaluate model performance and accuracy in capturing stock price trends.

8 Setting up Python, jupyter lab, Google Colab

- **Python Installation:** Download the Python installer from the official website, run it, and verify the installation by checking the Python version in PowerShell.
- **Virtual Environment Setup:** After Python installation, create a virtual environment using `python -m venv big-data-venv`.
- **Windows Terminal Installation:** Download Windows Terminal from GitHub releases and install it with default options.
- **Windows Terminal Configuration:** Open Windows Terminal, add a new profile, and modify the command line argument to include `NoExit-File%userprofile%\big-data-venv \Scripts \activate.ps1`.

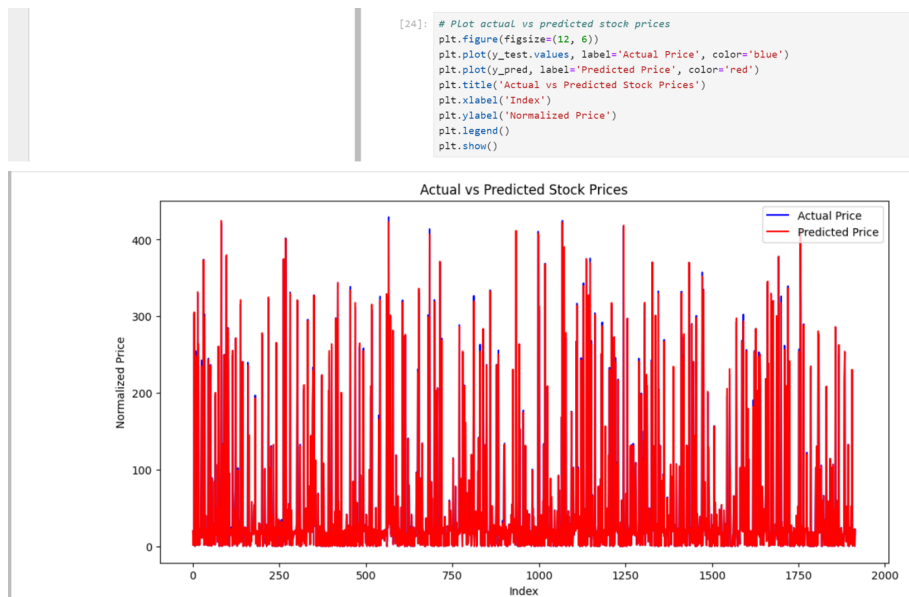


Figure 14: Plotting actual vs Predicted stock prices.

- **JupyterLab Installation:** In the newly configured profile in Windows Terminal, install JupyterLab using `pip install jupyterlab`. If there are installation issues, try `pip install jupyterlab==3.5.3` to resolve them.
- **Accessing JupyterLab:** Launch JupyterLab by running `jupyter lab` and access it via `localhost:8888/lab` in a web browser.
- **Setting Google Colab:** To open and import a dataset in Google Colab, you can follow these steps. First, open Google Colab by navigating to <https://colab.research.google.com/> in your web browser. Once the Colab interface loads, create a new notebook by clicking on the "File" menu and selecting "New notebook." Next, you can import datasets into your Colab notebook from various sources such as Google Drive, GitHub, or directly from your local machine.

9 Metrics

- **Data Quality:** The project ensures data quality by retrieving accurate historical stock data from a reliable source (Yahoo Finance), splitting the dataset into complete training and test sets, and applying consistent preprocessing steps to handle missing values and scale numerical features.
- **5Vs (Volume, Velocity, Variety, Veracity, Value):** Volume: historical stock data for Microsoft (MSFT) using the `yfinance` library. Veracity:

preprocessed data. Value: accurately forecasting future stock prices based on historical data and technical indicators.

- **Processing Time:** Used machine learning models to handle and process large datasets effectively.
- **Resource Utilization:** Resource utilization in this project is optimized to ensure efficient usage of memory and computational resources during model training and evaluation, maximizing performance while minimizing resource wastage.
- **Security:** Security measures such as data encryption, access control, and regular updates are implemented to ensure the confidentiality and integrity of sensitive data used in the stock price prediction framework.
- **Cost:** Involves the time and resources required for data collection, pre-processing, model training, and evaluation.

10 Conclusion

- **Data Preprocessing:** Data preprocessing is essential for ensuring data quality and preparing it for model training.
- **Model Construction:** Building models using LSTM and Random Forest allows for capturing temporal dependencies and non-linear relationships in the data.
- **Feature Engineering:** Feature engineering enriches the dataset with additional insights, aiding in capturing important patterns and trends.
- **Training Methodologies:** Training methodologies, including configuration of layers and optimization techniques, are crucial for effective model learning.
- **Model Evaluation and Prediction:** Model evaluation through metrics and visualization provides insights into the accuracy and effectiveness of the predictive framework.
- **Robust Framework:** Usage of LSTM and Random Forest models offers a robust and versatile framework for stock price prediction.

11 Citations:

- **Python Reference:** <https://docs.python.org/3/reference/index.html>
- **Git URL:** <https://github.com/KalpanaBolla/StockPricePrediction>
- **Google Colab:** <https://colab.google/>