VIETNAM NATIONAL UNIVERSITY – HO CHI MINH CITY

INTERNATIONAL UNIVERSITY

SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

Scalable and Distributed Computing

IT139IU

FINAL REPORT

Instructor:

Dr. Mai Hoang Bao An

Members - Student ID

Nguyen Huynh Ngan Anh - ITDSIU23003

Pham Tien Dat - ITITIU21172

Nguyen The Hao - ITDSIU22139

TABLE OF CONTENTS

CHAPTER I: INTRODUCTION

1. Abstract

This project aimed to apply the knowledge gained in the Scalable and Distributed Computing course to develop a real-time stock price forecasting system for Netflix using Apache Spark. The forecasting application provided investors with an intelligent and responsive tool for predicting stock price movements, while demonstrating the complex process of model training and real-time data processing through advanced machine learning algorithms, as shown in this report. The use of Spark for distributed data handling and scalable model deployment was explained in detail, in addition to the sophisticated forecasting techniques employed. The report concludes by evaluating the accuracy and performance of the models, analyzing the efficiency of the real-time pipeline, and proposing future enhancements for improved adaptability and precision.

2. Objectives

Via this project, our group aimed to:

- To build a forecasting system using machine learning on Apache Spark
- To enable real-time decision-making via a scalable pipeline

3. Scope of Work

- Analyze and preprocess historical Netflix stock data
- Handle missing values, stock splits, dividends, and timestamp alignment
- Extract and engineer features including technical indicators (e.g., RSI, moving averages)
- Develop LSTM forecasting models using Apache Spark
- Conduct backtesting on historical data

4. Project's contribution

- GitHub repository: https://github.com/swyrin/scalable-dist-comp-project
- Contribution table:

Name	Student ID	Contribution
Nguyen Huynh Ngan Anh	ITDSIU23003	+ **EDA Calculation**
Pham Tien Dat	ITITIU21172	+ **LSTM implementation** + **Write plotting code**
Nguyen The Hao	ITDSIU22139	+ **Handling Missing Values** + **Feature Engineering:**

CHAPTER II: PROJECT ANALYSIS

1. Data Investigation and Preprocessing

1.1. Data Sources and Structure

- The dataset used in this project was the historical Netflix stock market dataset [1], which was sourced publicly from Kaggle, covering the historical stock performance of Netflix Inc.
- This dataset includes daily trading records containing the following attributes: Date, Open, High, Low, Close, Adjusted Close, and Volume. The data spans several years, providing a comprehensive view of Netflix's market behavior over time. Each row represents a single trading day, and the Adjusted Close was used as the main target variable for forecasting tasks due to its accuracy in reflecting real stock value after accounting for corporate actions.
- The structure of the dataset is suitable for time series analysis, and it aligns well with Spark DataFrames for efficient distributed processing. The data is chronologically ordered and consistently formatted, enabling seamless transition into downstream analytics and model development.

```
df.show()
```

```
+----------+----------+----------+----------+----------+----------+--------+
|      Date|      Open|      High|       Low|     Close| Adj Close|  Volume|
+----------+----------+----------+----------+----------+----------+--------+
|2018-02-05|     262.0|267.899994|250.029999|254.259995|254.259995|11896100|
|2018-02-06|247.699997|266.700012|     245.0|265.720001|265.720001|12595800|
|2018-02-07|266.579987|272.450012|264.329987|264.559998|264.559998| 8981500|
|2018-02-08|267.079987|267.619995|     250.0|250.100006|250.100006| 9306700|
|2018-02-09|253.850006|255.800003|236.110001|249.470001|249.470001|16906900|
|2018-02-12|252.139999|259.149994|     249.0|257.950012|257.950012| 8534900|
|2018-02-13|257.290009|261.410004|254.699997|258.269989|258.269989| 6855200|
|2018-02-14|260.470001|269.880005|260.329987|     266.0|     266.0|10972000|
|2018-02-15|270.029999|     280.5|267.630005|280.269989|280.269989|10759700|
|2018-02-16|278.730011|281.959991|275.690002|278.519989|278.519989| 8312400|
|2018-02-20| 277.73999|285.809998|276.609985|278.549988|278.549988| 7769000|
|2018-02-21|282.070007|286.640015| 280.01001|281.040009|281.040009| 9371100|
|2018-02-22|283.880005|     284.5|274.450012|278.140015|278.140015| 8891500|
|2018-02-23|     281.0|     286.0|277.809998|285.929993|285.929993| 7301800|
|2018-02-26|    288.75|295.649994| 287.01001|294.160004|294.160004|10268600|
|2018-02-27|294.769989|297.359985|290.589996|290.609985|290.609985| 9416500|
|2018-02-28|293.100006|    295.75|290.779999|291.380005|291.380005| 7653500|
|2018-03-01|    292.75|    295.25|283.829987|290.390015|290.390015|11932100|
|2018-03-02|284.649994|301.179993|283.230011|301.049988|301.049988|13345300|
|2018-03-05|302.850006|316.910004|297.600006|     315.0|     315.0|18986100|
+----------+----------+----------+----------+----------+----------+--------+
only showing top 20 rows
```

1.2. Data Cleaning and Processing Techniques

To ensure the dataset was suitable for machine learning and real-time forecasting, several preprocessing steps were applied:

- **Handling Missing Values:**

  The dataset was scanned for missing entries, particularly in price and volume columns. Missing values, if any is found, will be having its row dropped to maintain data semantics.

```
[134]  1  missing_values_count = df.select([sum(col(column).isNull().cast("int")).alias(column) for column in df.columns])
       2  missing_values_count.show()

+----+----+----+---+-----+---------+------+
|Date|Open|High|Low|Close|Adj Close|Volume|
+----+----+----+---+-----+---------+------+
|   0|   0|   0|  0|    0|        0|     0|
+----+----+----+---+-----+---------+------+

[135]  1  df = df.dropna()
```

- **Feature Engineering:**

  A significant part of the data processing involves calculating various technical indicators from the stock data. This is a form of feature engineering where new features are created from existing ones. The indicators calculated are:

  - Moving Averages (SMA)

```
[ ] # a. Moving Averages (SMA)
    window_20 = Window.orderBy("Date").rowsBetween(-19, 0)
    window_50 = Window.orderBy("Date").rowsBetween(-49, 0)

    df = df.withColumn("SMA_20", avg(col("Adj Close")).over(window_20)) \
           .withColumn("SMA_50", avg(col("Adj Close")).over(window_50))
```

  - Relative Strength Index (RSI)

```
[ ] # b. Relative Strength Index (RSI)
    df = df.withColumn("Change", col("Adj Close") - lag("Adj Close", 1).over(Window.orderBy("Date")))
    df = df.withColumn("Gain", when(col("Change") > 0, col("Change")).otherwise(0))
    df = df.withColumn("Loss", when(col("Change") < 0, -col("Change")).otherwise(0))

    window_14 = Window.orderBy("Date").rowsBetween(-13, 0)
    df = df.withColumn("AvgGain", avg(col("Gain")).over(window_14))
    df = df.withColumn("AvgLoss", avg(col("Loss")).over(window_14))
    df = df.withColumn("RS", col("AvgGain") / col("AvgLoss"))
    df = df.withColumn("RSI", 100 - (100 / (1 + col("RS"))))
```

- Stochastic Oscillator Bollinger Bands

```
[ ]  # c. Stochastic Oscillator
     window_14 = Window.orderBy("Date").rowsBetween(-13, 0)
     df = df.withColumn("High_14", max(col("High")).over(window_14))
     df = df.withColumn("Low_14", min(col("Low")).over(window_14))

     df = df.withColumn("Stochastic", ((col("Adj Close") - col("Low_14")) / (col("High_14") - col("Low_14"))) * 100)
```

```
[ ]  # d. Bollinger Bands
     df = df.withColumn("SMA_20", avg(col("Adj Close")).over(window_20))
     df = df.withColumn("STD_20", stddev(col("Adj Close")).over(window_20))
     df = df.withColumn("Upper_Band", col("SMA_20") + (col("STD_20") * 2))
     df = df.withColumn("Lower_Band", col("SMA_20") - (col("STD_20") * 2))
```

- **Data Transformation for LSTM:**
  - The "Date" column is cast to a timestamp and the data is ordered by date.

```
[ ]  df_input = df.select("Date", "Adj Close")
     df_input = df_input.withColumn("Date", col("Date").cast("timestamp")).orderBy("Date")
```

  - The Spark DataFrame is converted to a Pandas DataFrame.

```
[ ]  pf = df.toPandas()
```

  - The "Adj Close" column is scaled using MinMaxScaler.

```
[ ]  import numpy as np
     from sklearn.preprocessing import MinMaxScaler

     scaler = MinMaxScaler()
     scaled_data = scaler.fit_transform(pf[["Adj Close"]])

     def create_sequences(data, window_size):
         X, y = [], []
         for i in range(window_size, len(data)):
             X.append(data[i - window_size:i])
             y.append(data[i])
         return np.array(X), np.array(y)

     window_size = 20
     X, y = create_sequences(scaled_data, window_size)
     X = X.reshape((X.shape[0], X.shape[1], 1))
```

  - Sequences of data are created with a defined window size, which is a common technique for time series data preparation for LSTM models.

2. Feature Engineering

2.1. Selected Features and Rationale

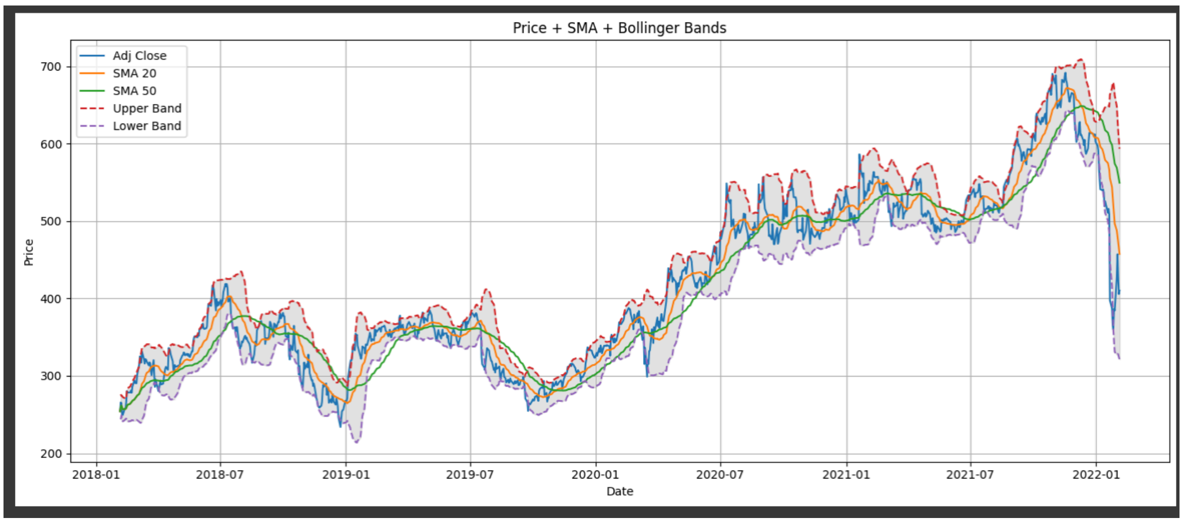
The features selected for model training were based on their predictive value and relevance to financial time series forecasting. These include:

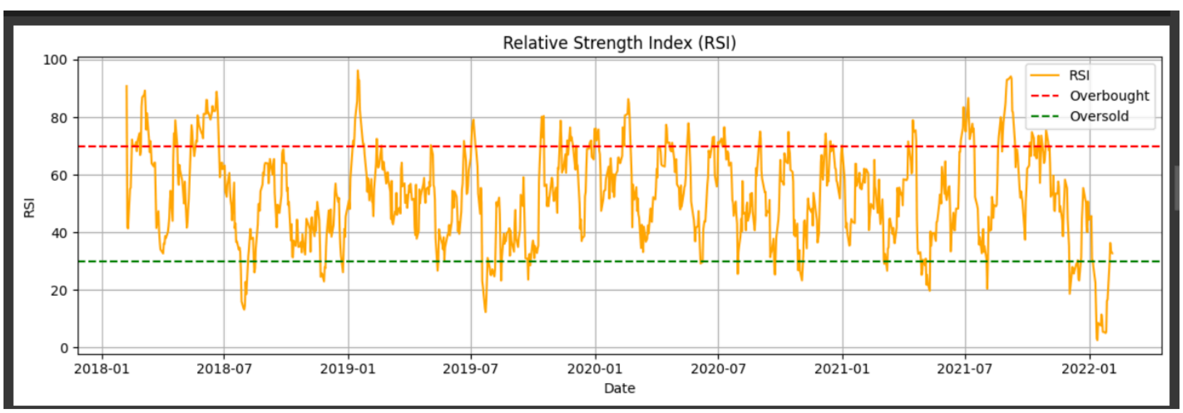
- The date of the stock price record, for historical purposes.
- The Adjacent Close, to reduce noise with other attributes, and leave out real-life human interferences.

2.2. Technical Indicators Used

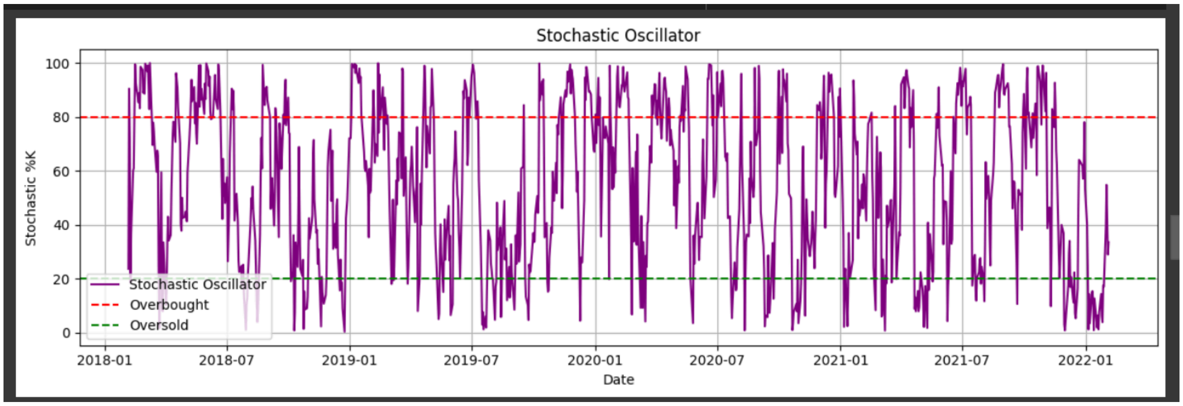
To strengthen the feature set, several technical indicators were engineered:

- **Moving Averages (SMA/EMA)**: Smooth short-term noise and highlight trends
- **Relative Strength Index (RSI)**: Measures recent gains/losses to identify overbought or oversold conditions
- **Bollinger Bands**: Visualize price volatility and potential reversal points.





- **Stochastic Oscillator**: Indicates momentum by comparing a closing price to a range over time



3. Model Development

3.1. Data Splitting Strategy

Given the time-dependent nature of the data, the dataset was split chronologically to avoid look-ahead bias and ensured realistic model evaluation:

- **Training set**: 80% of the data from the beginning of the timeline
- **Testing set**: Remaining 20%, representing the most recent data
- **Window size**: 20, which means 20 of previous data.

Implement LSTM memory window

```
import numpy as np
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()
scaled_data = scaler.fit_transform(pf[["Adj Close"]])

def create_sequences(data, window_size):
    X, y = [], []
    for i in range(window_size, len(data)):
        X.append(data[i - window_size:i])
        y.append(data[i])
    return np.array(X), np.array(y)

window_size = 20
X, y = create_sequences(scaled_data, window_size)
X = X.reshape((X.shape[0], X.shape[1], 1))
```

```python
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout

split = int(0.8 * len(X))
X_train, X_test = X[:split], X[split:]
y_train, y_test = y[:split], y[split:]
```

3.2. Algorithms Implemented

LSTM: A Sequential LSTM model is implemented for predicting the stock price. This is a type of recurrent neural network (RNN) specifically designed for sequential data like time series. The model has two LSTM layers and two Dropout layers before a final Dense layer for the prediction.

```python
model = Sequential()
model.add(LSTM(50, return_sequences=True, input_shape=(X.shape[1], 1)))
model.add(Dropout(0.2))
model.add(LSTM(50))
model.add(Dropout(0.2))
model.add(Dense(1))
```

4. Model Evaluation

4.1. Evaluation Metrics

Both MAE and RMSE are common metrics for evaluating regression models, including time series forecasting models like the LSTM used in the notebook. Lower values for both MAE and RMSE indicate better model performance:

- **Mean Absolute Error (MAE)**: This metric measures the average magnitude of the errors between the predicted and actual values. It provides a direct measure of the average prediction error in the same units as the target variable.
- **Root Mean Squared Error (RMSE)**: This metric is the square root of the average of the squared differences between the predicted and actual values. RMSE gives more weight to larger errors and is also in the same units as the target variable.

4.2. Backtesting Methodology

A simple time-based split for backtesting is a common and basic form of backtesting for time series models. It simulates how the model would perform on future data based on its training on past data:

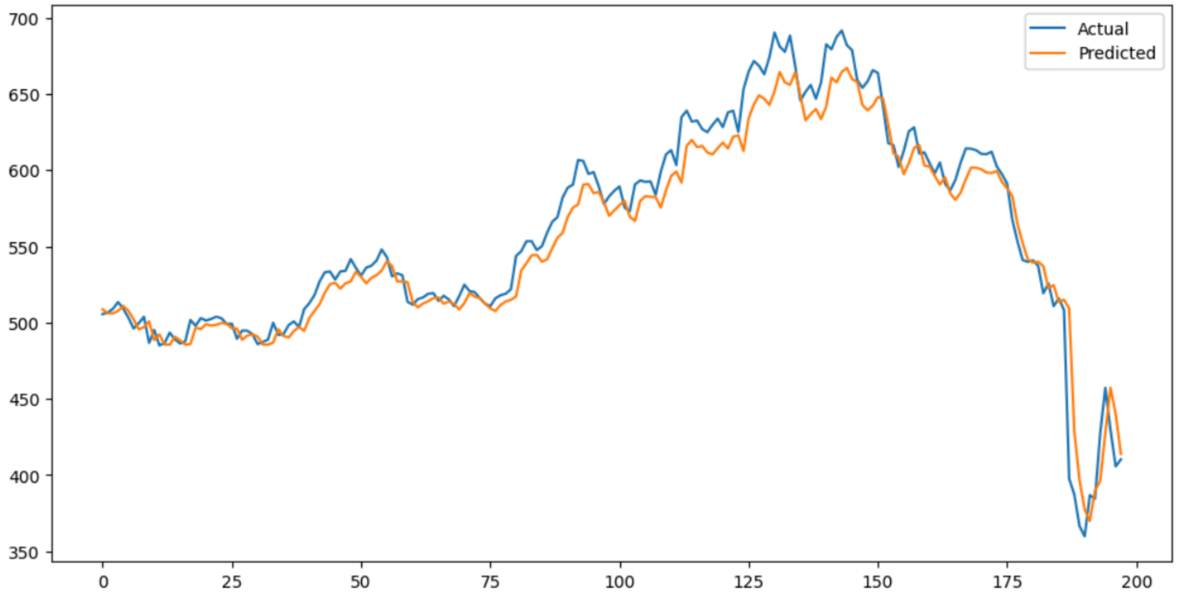
- **Train/Test Split**: The data is split into training and testing sets based on time. The first 80% of the data (chronologically) is used for training the LSTM model. The remaining 20% of the data is used for testing and evaluating the model's predictions.
- **Prediction**: The trained model makes predictions on the test set.
- **Evaluation**: The predictions are compared against the actual values in the test set to calculate performance metrics (MSE). The process incorporated Adam optimizer and set to run at 100 epochs, yielding around 0.0020 of loss value:

```python
model.compile(optimizer="adam", loss="mean_squared_error")
model.fit(X_train, y_train, epochs=100, batch_size=32, verbose=1)
```

```
Epoch 1/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 5s 21ms/step - loss: 0.0646
Epoch 2/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 1s 20ms/step - loss: 0.0056
Epoch 3/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 1s 22ms/step - loss: 0.0039
Epoch 4/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 1s 21ms/step - loss: 0.0031
Epoch 5/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 1s 21ms/step - loss: 0.0036
Epoch 6/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 1s 21ms/step - loss: 0.0033
Epoch 7/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 1s 20ms/step - loss: 0.0030
Epoch 8/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 1s 21ms/step - loss: 0.0032
Epoch 9/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 1s 20ms/step - loss: 0.0030
Epoch 10/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 1s 21ms/step - loss: 0.0030
Epoch 11/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 1s 20ms/step - loss: 0.0028
Epoch 12/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 1s 22ms/step - loss: 0.0027
Epoch 13/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 1s 20ms/step - loss: 0.0026
```

4.3. Result Analysis

- **Plotting**: A plot is generated to visualize the *actual Adj Close* prices and the *predicted Adj Close* prices on the test set. This provides a visual representation of how closely the predictions follow the actual trend.



- **Metrics**: The calculated and actual values are printed. These numerical metrics quantify the prediction errors.

CHAPTER III: CONCLUSION

1. Accomplishments:

- We have succeeded in writing a model that correctly estimates Stock Market changes of NFLX - as shown in the above figure, showing potential of a distributed computation model (Spark), incorporating Python programming languages and many of its ecosystems: matplotlib, pandas.
- This opens a future of a personal assistant that can help investors and share buyers to reinforce their decision on whether to support NFLX, together with a forecasting model for potential investors.

2. Future Improvements

- **Web application:** due to lack of human resource, timing constraints and upcoming final examination, we are unable to complete the project at its full potential. However, the web application is still one of our aims and will be implemented in the future, on Streamlit.
- **Model comparison:** we aim to implement several models (such as Linear Regression, Decision Tree Regressor) to show out the strengths and weaknesses of all models used for prediction.

3. References

[1] *Netflix stock price prediction*. (2022, February 5). Kaggle.
https://www.kaggle.com/datasets/jainilcoder/netflix-stock-price-prediction

[2] Adam: A Method for Stochastic Optimization. (2014, 22 Dec).
https://arxiv.org/abs/1412.6980

[3] Long Short-Term Memory (LSTM). NVIDIA
https://developer.nvidia.com/discover/lstm