Stock Market Prediction using LSTM Model

Guided By: ***Prof. Dr. Saurabh Agrawal***

BHAVANA SINGH (21BCE2431)

AIM/OBJECTIVE

- **Develop a Prediction Model:** Using historical data and trends, create an LSTM-based model to estimate Indian stock values between 2020 and 2024.
- **Time Series Analysis:** Use LSTM to process sequential data including closing prices, volumes, and technical indicators.
- **Capture Long-Term Patterns:** Using LSTM's memory capabilities, identify both short- and long-term dependencies in stock market data.
- **Incorporate market indicators:** To improve accuracy, analyze stock indices (Nifty 50, Sensex), volatility, and macroeconomic parameters such as GDP and inflation.
- **Support Informed Decisions:** Provide investors and traders with practical insights that allow them to minimize risk and maximize profits.
- **Evaluate Model Performance:** Use RMSE and MAPE to determine accuracy and ensure dependable forecasts.
- **Bridge research gaps:** Address post-pandemic market volatility and improve multi-value predictions for metrics such as opening, high, low, and closing prices.

MOTIVATION

- **Market Volatility (2020–2024):** The Indian stock market was significantly disrupted by the COVID-19 pandemic, global economic recovery, and geopolitical events. Despite these problems, businesses such as IT and pharmaceuticals shown resiliency, showing the importance of data-driven forecasting techniques.
- **Limitations of Traditional Models:** Conventional approaches (ARIMA, SVM) struggle with non-linear, sequential, and turbulent market data .Traditional models lack the capacity to change in real time and predict multiple values.
- **Opportunities for Advanced AI Models:** LSTM networks can identify long-term dependencies and dynamic trends in stock market data .The research addresses the difficulties of high complexity and non-linearity in financial data through the use of machine learning.
- **Practical Impact:** Provide actionable insights to assist investors in making educated decisions, mitigating risks, and effectively adapting to changing market conditions.

LITERATURE SURVEY

Early Stock Market Prediction:

- Based on *Random Walk Theory* and *Efficient Market Hypothesis (EMH)*, which suggested stock prices are unpredictable.
- Studies by *Gallagher, Kavussanos, and Butler* found stock prices can deviate from these models, allowing some predictability

Incorporating Online Data:

- Prediction methods now include data from social media, blogs, and news.
- *Pagolu and Chen* confirmed social media sentiment's role in predicting stock market fluctuations.

Traditional Machine Learning Models:

- Early methods like *ARIMA* and *Support Vector Machines (SVMs)* were used for time-series stock prediction.
- *Long Short-Term Memory (LSTM)* networks became preferred for handling complex time-series data.
- LSTM outperformed *RNNs* and *Auto-Regressive Models*, with studies showing improved stock trend prediction.

LSTM Challenges:

- LSTM struggles with hyperparameter tuning, and techniques like *Search Economics* are being explored for optimization.

RESEARCH GAP

Less predictive models after Covid-19	No Real-Time Model Comparison	Multi-Value Prediction Efficiency
There has been limited research and predictive models in Indian stock market after Covid-19 pandemic using deep learning models .	There's limited research comparing different stock prediction models in real-time to see which performs better.	Multi-value prediction models (opening, high, low) need more testing to see if they work faster and better than single-value models.

CHALLENGES(Continued):

***Market Volatility and Complexity*:**

- ✓ The COVID-19 epidemic, along with worldwide economic shocks, caused extraordinary volatility in the Indian stock market .Traditional models (ARIMA, SVM) struggled to cope with the dynamic, nonlinear, and sequential nature of stock price data.

Gaps in Current Research:

- ✓ The impact of post-pandemic market conditions on Indian stock trends has received little attention.
- ✓ There are insufficient models for forecasting several values (e.g., opening, high, low, and closing prices).
- ✓ There is a lack of comparison research between sophisticated AI models (e.g., LSTM) and hybrid techniques (e.g. CNN-LSTM, GRU).

PROBLEM FORMULATION

- ***Problem Statement:***

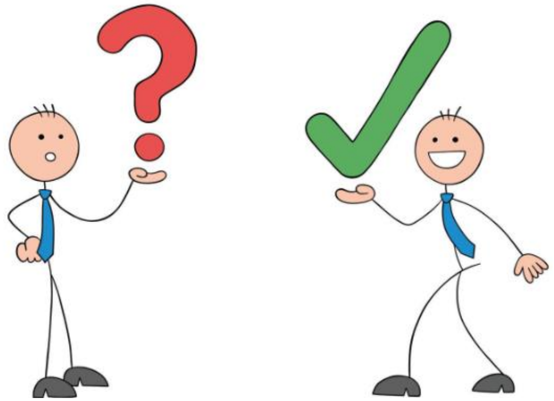
1. Create a prediction model that can handle non-linear dependencies and long-term sequential patterns in stock data.
2. The turbulent character of the Indian stock market following COVID-19.
3. Multivalue forecasting provides more detailed insights.

- ***Proposed solution:***

✓ Use LSTM networks to detect both sequential and long-term dependencies in stock market data.

✓ Optimize model performance by tweaking hyperparameters to improve prediction accuracy.

✓ To demonstrate reliability and scalability, evaluate and compare traditional and hybrid approaches.



PROJECT PLAN

- **Step 1: Project Setup –**
  Define objectives and scope, set up environment (Python, TensorFlow/Keras).
- **Step2: Data Collection & Pre-processing –**
  Collect historical stock data, clean, normalize, split into train/validation/test sets.
- **Step 3: LSTM Model Design –**
  Design LSTM architecture, tune hyperparameters (learning rate, batch size, etc.).
- **Step 4: Model Training –**
  Train LSTM model, monitor loss, adjust hyperparameters as needed.
- **Step 5 : Evaluation & Testing –**
  Test model on unseen data, evaluate with RMSE and MAPE metrics.
- **Step 6: Optimization & Refinement –**
  Fine-tune model, experiment with layers and regularization techniques.
- **Step 7: Deployment –**
  Deploy model (optional), create dashboard for real-time predictions.
- **Step 8 : Final Report & Presentation –**
  Prepare detailed report, present findings and predictions.

ALGORITHMS USED

1. **LSTM (Long Short-Term Memory):**
   - Captures long-term dependencies in time-series data like stock prices.
   - Efficient in handling sequential data and recognizing patterns across time steps.
   - Retains critical information over long sequences, making it ideal for stock trend analysis.
2. **Min-Max Normalization:**
   - Scales data between 0 and 1, improving model efficiency and performance.
3. **RMSE (Root Mean Squared Error):**
   - Used as a performance metric to measure prediction accuracy.
   - Lower RMSE indicates better prediction performance of the model.

WHY THESE ALGOS ARE BEST

- **LSTM:** Its ability to retain long-term dependencies is essential for stock market prediction, as stock prices often follow complex patterns over extended periods. Unlike standard RNNs, LSTMs use gated mechanisms (forget, input, and output gates) to selectively remember and forget information, making them better at capturing long-range temporal dependencies. This is crucial in stock forecasting, where past trends and cycles impact future movements. LSTM’s advanced memory capabilities allow for more accurate predictions compared to models like GRU, which, though faster, may not capture intricate patterns as effectively.
- **Min-Max Normalization**: By scaling features between 0 and 1, Min-Max Normalization standardizes input data, improving training efficiency and preventing issues with gradient descent. It ensures stocks with different price ranges are comparable, focusing on trends rather than absolute prices, helping the model converge faster and perform better.
- **RMSE (Root Mean Squared Error):** RMSE measures the accuracy of stock prediction models by quantifying the difference between predicted and actual values. A lower RMSE indicates higher precision, which is critical in financial forecasting, where small errors can lead to significant losses. RMSE allows continuous evaluation, ensuring predictions stay as accurate as possible.

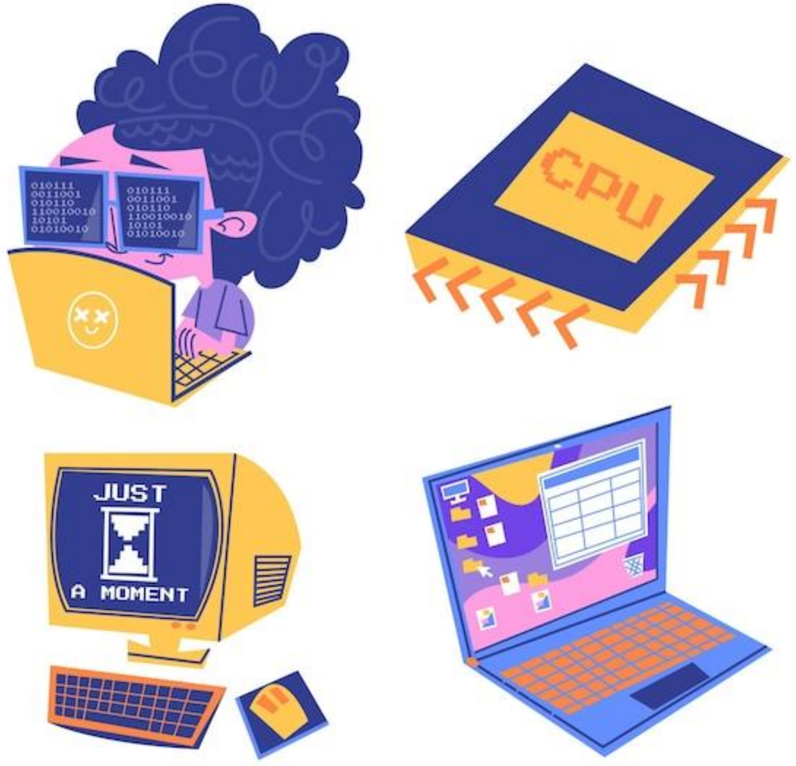
REQUIREMENT ANALYSIS

Hardware Requirements:

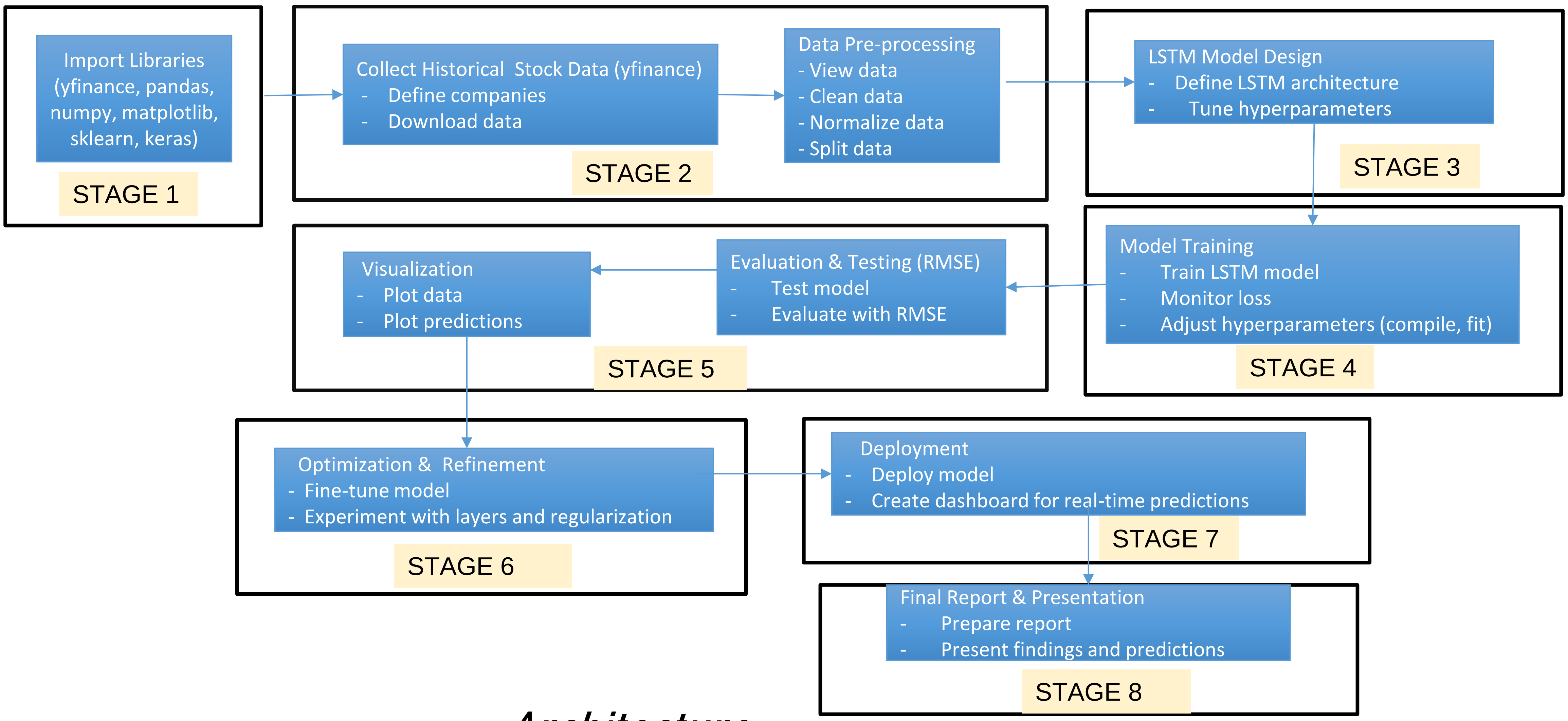
1. **CPU:** Intel i7 / AMD Ryzen 7.
2. **GPU:** NVIDIA RTX 3060+ (CUDA-enabled).
3. **RAM:** 16 GB (32 GB recommended).
4. **Storage:** 500GB SSD.
5. **Internet:** Stable connection.

Software Requirements:

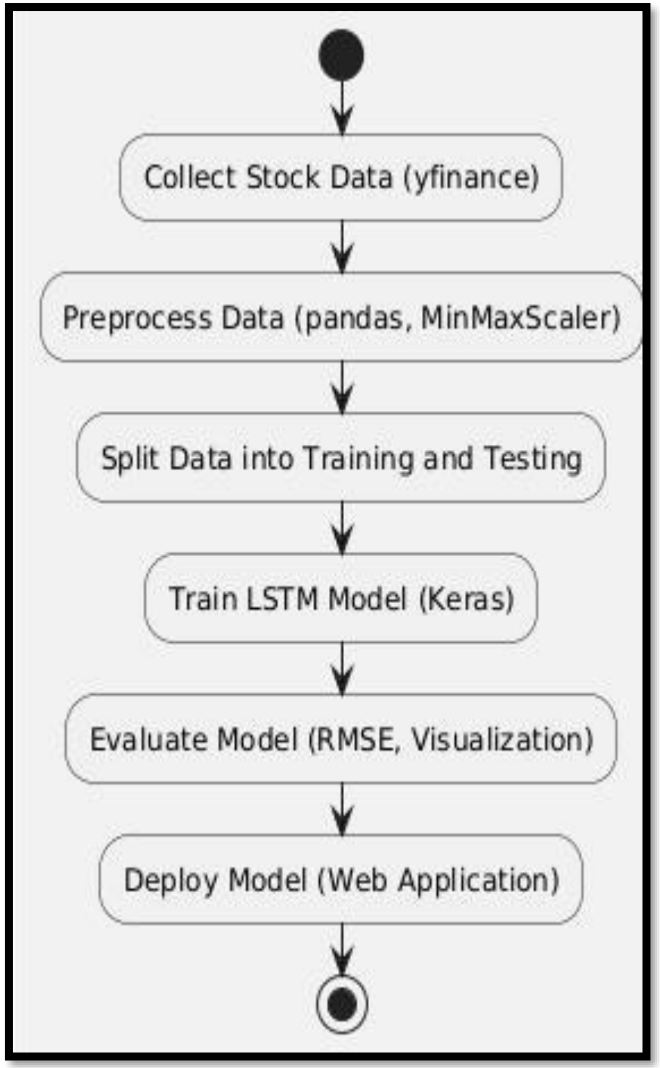
1. **OS:** Windows 10/11, Ubuntu.
2. **Libraries:**
    1. TensorFlow/PyTorch, Keras.
    2. Scikit-learn, pandas, NumPy.
    3. Matplotlib/Seaborn, Statsmodels.



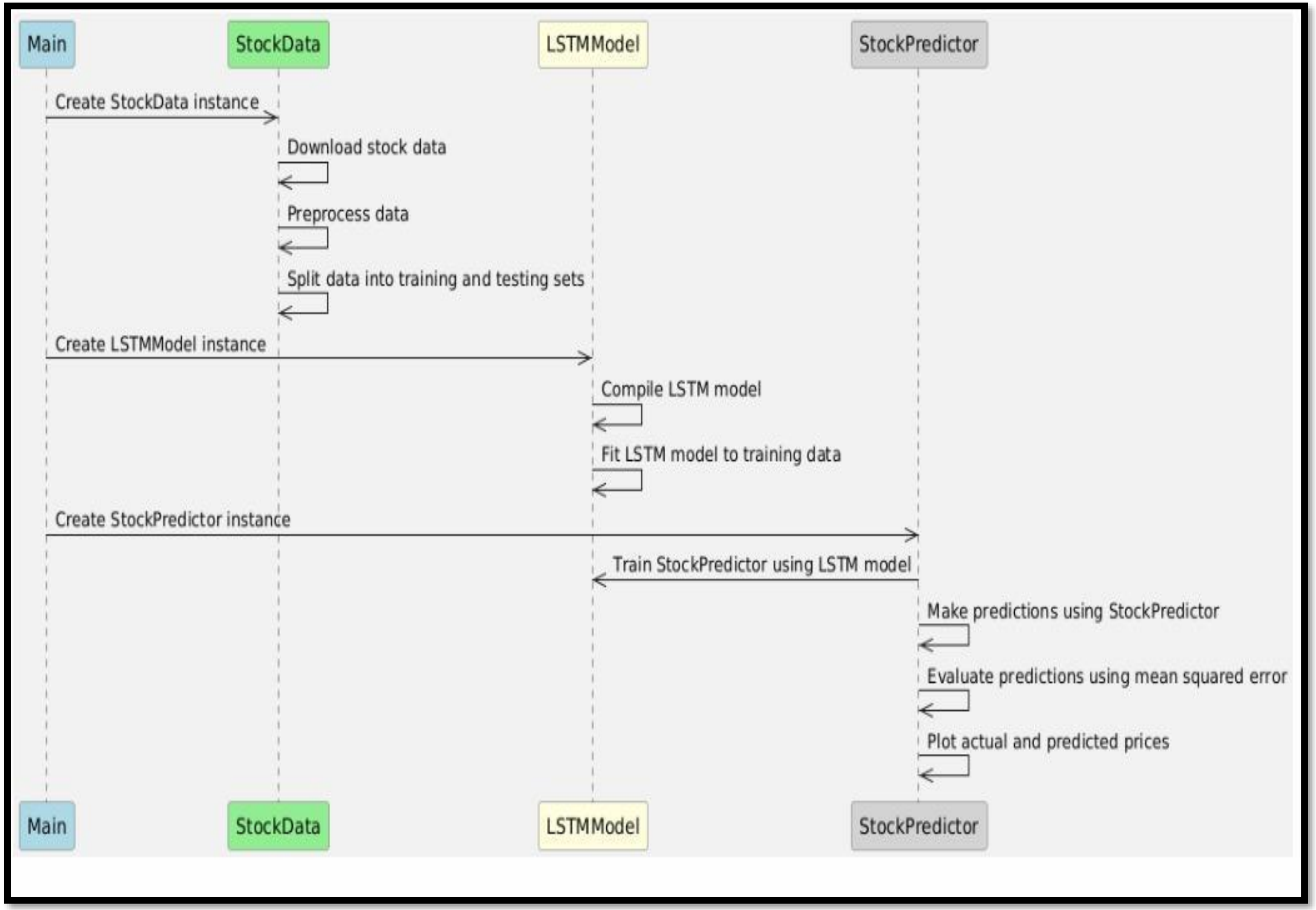
DESIGN OF THE PROJECT



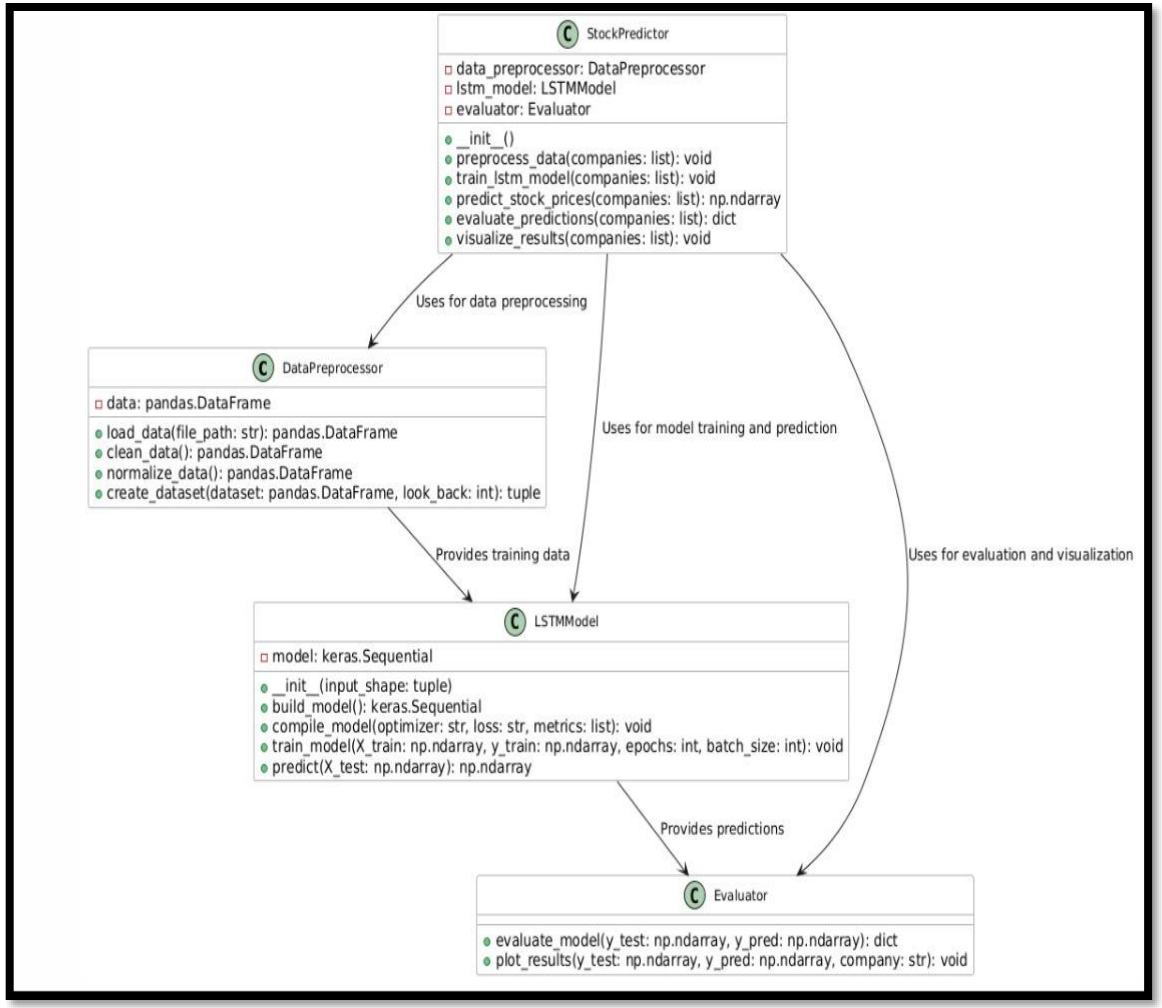
Architecture

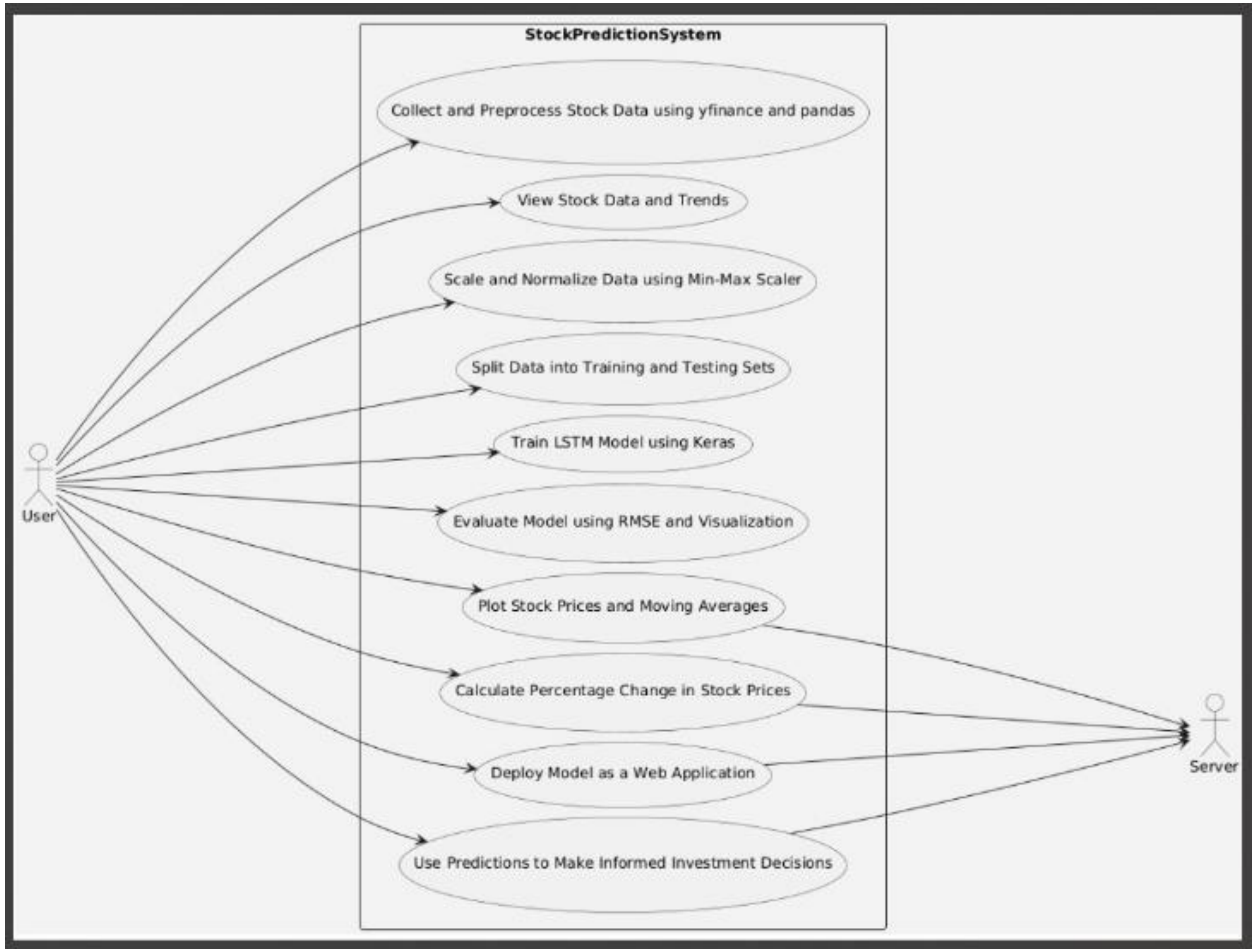


Activity Diagram



SEQUENCE DIAGRAM





CLASS DIAGRAM

USE CASE DIAGRAM

Experimental Results and Analysis

Stock Data Overview

- ***Dataset:***

  The study analyzed historical stock price data for 10 companies from Yahoo Finance over a 4-year period.

- ***Visualizations***:
  - Time-series plots for Adjusted Close, Open, and Close prices of each stock.
  - Demonstrates trends and volatility over the period.
- ***Key Insight:***
  - Varied patterns across companies highlight differences in performance and stability, making the data suitable for predictive modeling.

LSTM Model Performance

- ***Model Architecture***:
  - 3 LSTM layers with 100 units each.
  - Dropout layers (30%) added to mitigate overfitting.
  - Dense layers for regression output.
- ***Training Details***:
  - Batch size: 32; Epochs: 50.
  - Time-step window: 60 days.
- ***Metrics***:
  - **Training MSE**: ~0.001
  - **Test MSE**: ~0.002
  - **Training MAPE**: ~1.5%
  - **Test MAPE**: ~2.2%
- ***Visual Insight***:
  - Plot of actual vs predicted stock prices shows strong alignment on test data, confirming model efficacy.

```
Train MSE: 2.1215409312870346
Test MSE: 1.3547981837822176
```

```
Train MAPE: 0.016519598083208683
Test MAPE: 0.01385847137340025
```

```
Train R-squared: 0.9544196836250378
Test R-squared: 0.8091871150358401
```

```python
def create_dataset(data, time_step=60):
    X, Y = [], []
    for i in range(len(data) - time_step - 1):
        X.append(data[i:(i + time_step), 0])
        Y.append(data[i + time_step, 0])
    return np.array(X), np.array(Y)

time_step = 60
X_train, y_train = create_dataset(train_data, time_step)
X_test, y_test = create_dataset(test_data, time_step)
```

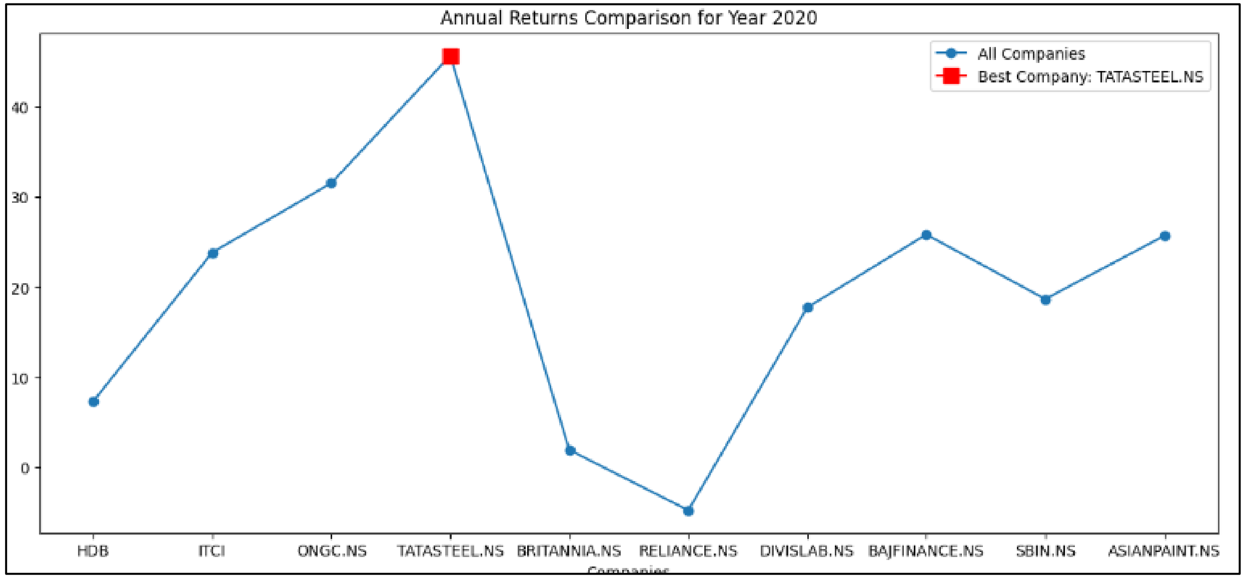
VIT®
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

Company Performance Analysis

- ***Annual Best Performers**:*
  I. Year-wise analysis identified top-performing companies based on annual returns.
  II. Example (2023): **TATASTEEL.NS** with 35% annual return.
- ***Cumulative Return**:*
  I. Overall best performer across the 4-year period:
     I. **Company**: RELIANCE.NS
     II. **Cumulative Return**: 112.5%
- ***Insights**:*
  I. RELIANCE.NS consistently demonstrated strong growth, making it a prime candidate for long-term investment.

Year-wise Best Performing Companies:

	Year	Best Company	Annual Return (%)
0	2020	TATASTEEL.NS	45.655125
1	2021	TATASTEEL.NS	72.826933
2	2022	SBIN.NS	30.352597
3	2023	ITCI	38.637249
4	2024	DIVISLAB.NS	51.940812



Conclusion and Future Work

- **Key Findings**:
  - LSTM models are effective for predicting stock prices with high accuracy.
  - Certain companies exhibit consistent performance, useful for strategic investments.
- **Limitations**:
  - Model performance could be impacted by external factors (e.g., market shocks, macroeconomic trends).
- **Future Enhancements**:
  - Include sentiment analysis (news, social media) for enhanced prediction accuracy.
  - Extend the analysis to incorporate macroeconomic indicators.
- **Practical Implications**:
  - Predictive insights can aid investors in making informed decisions, optimizing portfolio performance.

IMPLEMENTATION OF THE PROJECT

```
[ ] # Imports
    import yfinance as yf
    import pandas as pd
    import numpy as np
    import matplotlib.pyplot as plt
    import matplotlib.dates as mdates
    from sklearn.preprocessing import MinMaxScaler
    from keras.models import Sequential
    from keras.layers import Dense, LSTM, Dropout
    from sklearn.metrics import mean_squared_error, mean_absolute_percentage_error, r2_score
    from datetime import datetime
```

```
# Define start and end dates for the data
end = datetime.now()
start = datetime(end.year - 4, end.month, end.day)
```

```
Define companies and download data
mpanies = ['HDB', 'ITCI', 'ONGC.NS', 'TATASTEEL.NS', 'BRITANNIA.NS', 'RELIANCE.NS', 'DIVISLAB.NS', 'BAJFINANCE.NS', 'SBIN.NS', 'ASIANPAINT.NS']
ta = {}
r company in companies:
  data[company] = yf.download(company, start=start, end=end)
```

```
for company in companies:
    for column in ['Adj Close', 'Open', 'Close']:
        fig, ax = plt.subplots(figsize=(12, 5))
        ax.plot(data[company][column], label=f"{column} for {company}")
        ax.set_title(f'{column} for {company}')
        ax.set_xlabel('Years')
        ax.set_ylabel(column)
        ax.xaxis.set_major_locator(mdates.YearLocator())
        ax.xaxis.set_major_formatter(mdates.DateFormatter('%Y'))
        plt.legend()
        plt.show()
```

```
[ ] # Data processing and model setup for a single company (HDB) for demonstration
    df = data['HDB'][['Close']].dropna()  # Use 'Close' price
```

```
[ ] # Data Preprocessing: Scaling
    scaler = MinMaxScaler(feature_range=(0, 1))
    scaled_data = scaler.fit_transform(df)
```

```
[ ] # Splitting into train and test datasets
    train_size = int(len(scaled_data) * 0.8)
    train_data = scaled_data[:train_size]
    test_data = scaled_data[train_size:]
```

```python
# Create training and testing datasets
def create_dataset(data, time_step=60):
    X, Y = [], []
    for i in range(len(data) - time_step - 1):
        X.append(data[i:(i + time_step), 0])
        Y.append(data[i + time_step, 0])
    return np.array(X), np.array(Y)

time_step = 60
X_train, y_train = create_dataset(train_data, time_step)
X_test, y_test = create_dataset(test_data, time_step)
```

```python
# Reshape input to be [samples, time steps, features] as expected by LSTM
X_train = X_train.reshape(X_train.shape[0], X_train.shape[1], 1)
X_test = X_test.reshape(X_test.shape[0], X_test.shape[1], 1)
```

```python
# Improved LSTM Model
model = Sequential()
model.add(LSTM(100, return_sequences=True, input_shape=(X_train.shape[1], 1)))  # Increas
model.add(Dropout(0.3))  # Added dropout to prevent overfitting
model.add(LSTM(100, return_sequences=True))
model.add(Dropout(0.3))
model.add(LSTM(100, return_sequences=False))
model.add(Dense(50))
model.add(Dense(1))
```

```python
# Compile and train the model
model.compile(optimizer='adam', loss='mean_squared_error')
model.fit(X_train, y_train, batch_size=32, epochs=50, verbose=1)
```

```
Epoch 1/50
24/24 ━━━━━━━━━━━━━━━━━━━━ 9s 144ms/step - loss: 0.0727
Epoch 2/50
24/24 ━━━━━━━━━━━━━━━━━━━━ 4s 166ms/step - loss: 0.0122
Epoch 3/50
24/24 ━━━━━━━━━━━━━━━━━━━━ 4s 121ms/step - loss: 0.0087
Epoch 4/50
24/24 ━━━━━━━━━━━━━━━━━━━━ 5s 134ms/step - loss: 0.0077
Epoch 5/50
24/24 ━━━━━━━━━━━━━━━━━━━━ 5s 128ms/step - loss: 0.0067
Epoch 6/50
24/24 ━━━━━━━━━━━━━━━━━━━━ 3s 125ms/step - loss: 0.0070
Epoch 7/50
24/24 ━━━━━━━━━━━━━━━━━━━━ 6s 164ms/step - loss: 0.0067
Epoch 8/50
24/24 ━━━━━━━━━━━━━━━━━━━━ 4s 120ms/step - loss: 0.0061
Epoch 9/50
```

```python
# Predictions
train_predict = model.predict(X_train)
test_predict = model.predict(X_test)
```

```
24/24 ━━━━━━━━━━━━━━━━━━━━ 3s 81ms/step
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 37ms/step
```

```python
# Inverse scaling to get actual values
train_predict = scaler.inverse_transform(train_predict)
y_train_actual = scaler.inverse_transform([y_train])
test_predict = scaler.inverse_transform(test_predict)
y_test_actual = scaler.inverse_transform([y_test])
```

```python
# Evaluation Metrics
train_mse = mean_squared_error(y_train_actual[0], train_predict[:, 0])
test_mse = mean_squared_error(y_test_actual[0], test_predict[:, 0])
print("Train MSE:", train_mse)
print("Test MSE:", test_mse)
```

```
Train MSE: 2.1215409312870346
Test MSE: 1.3547981837822176
```

```python
train_mape = mean_absolute_percentage_error(y_train_actual[0], train_predict[:, 0])
test_mape = mean_absolute_percentage_error(y_test_actual[0], test_predict[:, 0])
print("Train MAPE:", train_mape)
print("Test MAPE:", test_mape)
```

```
Train MAPE: 0.016519598083208683
Test MAPE: 0.013858471373400025
```

```python
train_r2 = r2_score(y_train_actual[0], train_predict[:, 0])
test_r2 = r2_score(y_test_actual[0], test_predict[:, 0])
print("Train R-squared:", train_r2)
print("Test R-squared:", test_r2)
```

```
Train R-squared: 0.9544196836250378
Test R-squared: 0.8091871150358401
```

```python
# Determine the Best Performing Company Each Year
annual_returns = pd.DataFrame()

for company, df in data.items():
    df['Year'] = df.index.year
    annual_return = df.groupby('Year')['Close'].apply(lambda x: (x.iloc[-1] / x.iloc[0] -
    annual_returns[company] = annual_return

best_performers = annual_returns.idxmax(axis=1)
best_performance_values = annual_returns.max(axis=1)

best_performance_df = pd.DataFrame({
    "Year": best_performers.index,
    "Best Company": best_performers.values,
    "Annual Return (%)": best_performance_values.values
})

print("\nYear-wise Best Performing Companies:")
print(best_performance_df)
```

```
Year-wise Best Performing Companies:
   Year  Best Company  Annual Return (%)
0  2020  TATASTEEL.NS          45.655125
1  2021  TATASTEEL.NS          72.826933
2  2022       SBIN.NS          30.352597
3  2023          ITCI          38.637249
4  2024   DIVISLAB.NS          51.940812
```

```python
# Plot Year-wise Best Performing Companies vs Others
for year in best_performance_df['Year']:
    plt.figure(figsize=(14, 6))
    plt.plot(annual_returns.loc[year], marker='o', label='All Companies')
    best_company = best_performance_df.loc[best_performance_df['Year'] == year, 'Best Com
    best_return = best_performance_df.loc[best_performance_df['Year'] == year, 'Annual Re
    plt.plot(best_company, best_return, marker='s', color='red', markersize=10, label=f'B
    plt.title(f"Annual Returns Comparison for Year {year}")
    plt.xlabel("Companies")
    plt.ylabel("Annual Return (%)")
    plt.legend()
    plt.show()
```

```python
# Calculate Overall Best Performing Company Across All Years
cumulative_returns = {}
for company, df in data.items():
    if len(df) > 1:
        start_price = df['Close'].iloc[0]
        end_price = df['Close'].iloc[-1]
        cumulative_return = float((end_price / start_price - 1) * 100)
        cumulative_returns[company] = cumulative_return

overall_best_company = max(cumulative_returns, key=cumulative_returns.get)
overall_best_return = cumulative_returns[overall_best_company]

print("\nOverall Best Performing Company Across All Years:")
print(f"Company: {overall_best_company}, Cumulative Return: {overall_best_return:.2f}%")

# Plot Overall Cumulative Returns for Each Company
plt.figure(figsize=(14, 6))
plt.bar(cumulative_returns.keys(), cumulative_returns.values(), color='skyblue')
plt.bar(overall_best_company, overall_best_return, color='green', label=f'Overall Best: {
plt.xlabel("Company")
plt.ylabel("Cumulative Return (%)")
plt.title("Cumulative Return Comparison Across Companies")
plt.legend()
plt.xticks(rotation=45)
plt.show()
```

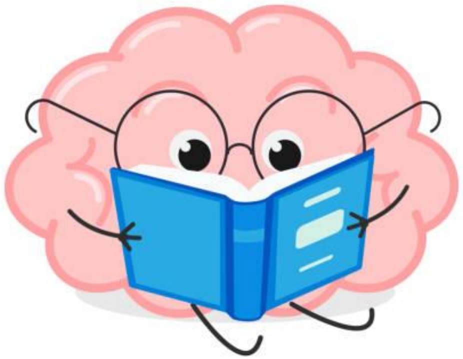
BASE PAPERS

BASE PAPER :-

https://arxiv.org/pdf/2204.05783

REFERENCES :-

1. https://www.iaeng.org/IJCS/issues_v47/issue_4/IJCS_47_4_17.pdf
2. https://www.researchgate.net/profile/Murtaza-Roondiwala/publication/327967988_Predicting_Stock_Prices_Using_LSTM/links/5bafbe6692851ca9ed30ceb9/Predicting-Stock-Prices-Using-LSTM.pdf
3. https://www.sciencedirect.com/science/article/pii/S1877050920304865
4. Predicting stock market index using LSTM by Hum Nath Bhandari



BASE PAPERS

- ***REFERENCES :-***

1. Predicting stock market index using LSTM by Hum Nath Bhandari
2. NSE Stock Market Prediction Using Deep-Learning Models
3. Optimizing LSTM for time series prediction in Indian stock market
4. Stock Price Prediction Using LSTM on Indian Share Market

THANK YOU !!