STOCK MARKET PREDICTION

A Project Report Submitted

In partial fulfilment of the requirements for the award of the degree of

Bachelor of Technology

In

Computer Science and Engineering

Submitted By:

Shubham Shaw

Vivek Mandal

Soham Biswas

Debanjan Mukherjee

Under the guidance of

Mr. Chiranjeet Sarkar, Dept. of CSE

Department of Computer Science and Engineering

Hooghly Engineering & Technology College, Hooghly

Affiliated to

Maulana Abul Kalam Azad University of Technology, West Bengal

2024-25

HOOGHLY ENGINEERING & TECHNOLOGY COLLEGE

Vivekananda Road, Pipulpati, P.O. & Dist-Hooghly (WB), Pin-712103

(Affiliated to Maulana Abul Kalam Azad University of Technology)

CERTIFICATE

This is to certify that the project entitled **"Stock Market Prediction"** has been submitted to the **Department of Computer Science and Engineering, Hooghly Engineering & Technology College, Hooghly** in partial fulfilment for the award of Bachelor of Technology in Computer Science and Engineering by the following 4th Year 8th Semester students under the supervision of **Mr. Chiranjeet Sarkar**, **Assistant Professor, CSE Department,** during a period from **February, 2025 to May, 2025.**

Their performances were found to be satisfactory.

Student Name (University Roll No.)

1. Shubham Shaw (17600121006)
2. Vivek Mandal (17600121007)
3. Soham Biswas (17600121008)
4. Debanjan Mukherjee (17600121046)

Mr. Chiranjeet Sarkar
Assistant Professor
Department of CSE

Dr. Sumanta Daw
HOD, Department of CSE

Prof. (Dr.) B. P. Pattanaik
Principal, HETC

ACKNOWLEDGEMENT

It gives us immense pleasure to announce the partial completion of our project on **“Stock Market Prediction”** and we are pleased to acknowledge our indebtedness to all the persons who directly or indirectly contributed in the development of this work and who influenced our thinking, behaviour and acts during the course of study.

We are thankful to our respected departmental HOD Dr. Sumanta Daw and Coordinator Mr. Dibyendu Samanta who granted all the facilities of the college to us for the fulfilment of the project.

We are thankful and express our sincere gratitude to our project guide **Mr. Chiranjeet Sarkar** who gave his valuable time to us for the sake of our project. He helped us each and every aspect of our project both academically and mentally.

Finally, the team expressed their gratitude to our respected Principal sir Prof. (Dr.) Bhabani Prasanna Pattanaik without his support our project would not have seen the light of success.

Student Name (with Roll No.)	**Signature**
1. Shubham Shaw (17600121006)	…………………………………………
2. Vivek Mandal (17600121007)	…………………………………………
3. Soham Biswas (17600121008)	…………………………………………
4. Debanjan Mukherjee (17600121046)	…………………………………………

4th Year, 8th Semester
Department of Computer Science and Engineering
Hooghly Engineering & Technology College
Academic Year: 2024-25

ABSTRACT

The stock market has long been a domain of immense interest, where accurate predictions can yield significant financial rewards. This project explores the potential of machine learning models, particularly Long Short-Term Memory (LSTM) networks, to forecast stock market trends. LSTMs, a variant of recurrent neural networks (RNNs), are uniquely suited to process sequential data, making them ideal for time series forecasting in financial markets. The project begins with data acquisition, utilizing reliable sources to gather historical stock market data. Preprocessing steps include normalization and transformation to prepare the data for analysis. Visualization techniques are employed to understand trends, identify patterns, and select relevant features. The core of the study is the implementation of the LSTM model using frameworks such as TensorFlow and Keras. The architecture leverages sequential layers with dense and dropout techniques to enhance model accuracy and mitigate overfitting. The model is trained on historical data and evaluated using metrics such as Mean Absolute Error (MAE) and Root Mean Squared Error (RMSE). Results indicate the effectiveness of LSTMs in capturing short-term and long-term dependencies within stock price data. The model demonstrates robust predictive capabilities, outperforming traditional statistical methods in several scenarios. Visualizations of predicted versus actual stock prices further validate the model’s accuracy. This study highlights the transformative potential of machine learning in financial forecasting. While challenges such as data volatility and external market factors persist, the integration of advanced algorithms like LSTMs provides a promising avenue for investors and analysts seeking data-driven insights. Future work could explore hybrid models and incorporate macroeconomic indicators for enhanced accuracy.

INDEX

FIGURE INDEX

1 INTRODUCTION

1.1 Background

Stock markets play a crucial role in the global economy, facilitating capital allocation and providing investment opportunities. However, the inherent volatility and complexity of the stock market make predicting price movements a challenging endeavor. Investors, traders, and financial analysts continuously strive to develop accurate methods for forecasting stock prices to maximize returns and minimize risks.

1.2 Motivation

Traditional forecasting models, such as moving averages, ARIMA, and exponential smoothing, often struggle to capture the intricate dependencies and non-linear relationships inherent in stock market data. This limitation has spurred the exploration of more sophisticated techniques, particularly machine learning, which excels at analyzing complex and non-linear patterns.

1.3 Objective

The primary objective of this project is to design and implement an LSTM-based model for predicting stock prices. The chosen dataset consists of historical stock data for TATAMOTORS.NS, obtained from Yahoo Finance.

1.4 Methodology

The methodology encompasses several key steps:

- Data preprocessing, including scaling and computing moving averages, to prepare the data for the LSTM model.
- Feature engineering to extract relevant information from the raw data.
- Model training and evaluation.

1.5 Significance

This project not only focuses on building an accurate predictive model but also aims to:

- Demonstrate the practicality and limitations of applying LSTMs in real-world financial forecasting.
- Gain insights into the strengths and weaknesses of the LSTM model in capturing stock market trends.
- Explore the broader implications of using machine learning for stock market prediction, acknowledging its limitations and potential.

1.6 Outline

The remainder of this report is organized as follows:

- Literature Review: Examines existing approaches to stock price prediction.
- Methodology: Details the data collection, preprocessing, and model implementation.
- Results: Discusses the model's performance and analyzes the results.
- Conclusion: Summarizes the findings and suggests directions for future research.

This project serves as a foundational step in understanding the capabilities and limitations of LSTM models in financial forecasting, paving the way for more advanced and integrated approaches.

2 REVIEW OF LITERATURE

2.1 Introduction to Stock Market Prediction

The field of stock market prediction has garnered significant interest due to its complexity and potential for financial gains. Traditional methods primarily relied on statistical techniques, such as autoregressive integrated moving average (ARIMA) models and exponential smoothing, which were effective for linear time series data but struggled with the non-linear relationships prevalent in financial markets.

2.2 Transition to Machine Learning

As limitations of traditional methods became apparent, researchers began exploring advanced computational techniques. Machine Learning (ML) models, including Support Vector Machines (SVMs), Decision Trees, and Random Forests, emerged as promising alternatives. These models improved accuracy by capturing non-linear patterns in data but faced challenges with temporal dependencies inherent in stock market data.

2.3 Advancements with Deep Learning

The introduction of Deep Learning techniques, particularly Recurrent Neural Networks (RNNs), marked a significant evolution in financial forecasting. RNNs, especially Long Short-Term Memory (LSTM) networks, have been widely adopted for time series analysis due to their ability to retain long-term dependencies. Research by Fischer and Krauss (2018) demonstrated that LSTMs outperformed traditional machine learning models in predicting stock market movements.

2.4 Performance Validation of LSTMs

Several studies have validated the effectiveness of LSTMs in stock market prediction: Patel et al. (2015) explored integrating LSTMs with other neural networks to enhance predictive performance, particularly for volatile stocks. Zhouet al. (2019) highlighted the use of attention mechanisms alongside LSTMs to improve the interpretability of predictions.

2.5 Hybrid Models and External Factors

Recent advancements have led to the development of hybrid models that combine deep learning techniques with external factors such as sentiment analysis and macroeconomic indicators. For instance, Bollen et al. (2011) investigated the influence of social media sentiment on stock prices, showing promising results when integrated with LSTM predictions. Additionally, ensemble models that combine multiple algorithms have demonstrated superior performance in capturing diverse market dynamics.

2.6 Ongoing Challenges

Despite these advancements, challenges remain in stock market prediction: The unpredictable nature of external events, such as geopolitical shifts and economic crises, can significantly impact prediction accuracy. Researchers emphasize the importance of data preprocessing, feature selection, and incorporating domain specific knowledge to enhance the robustness of predictions.

2.7 Conclusion and Future Directions

The literature underscores the transformative potential of deep learning, particularly LSTMs, in stock market prediction. While traditional methods laid the groundwork, modern advancements offer enhanced accuracy and reliability. Future research is likely to focus on hybrid approaches and integrating alternative data sources to further refine predictive capabilities.

3 PROPOSED WORK

The proposed work aims to develop a robust and efficient stock market prediction system using Long Short-Term Memory (LSTM) networks, a specialized type of recurrent neural network. LSTMs are well-suited for sequential data analysis due to their ability to model long-term dependencies. The key steps involved in this project are outlined below, along with a flowchart to visualize the workflow.

3.1 Data Acquisition

- Historical stock market data is collected from reliable sources such as Yahoo Finance or Alpha Vantage.
- The data includes open, high, low, close prices, and trading volume for selected stocks.

3.2 Data Preprocessing

- Handling missing values by interpolation or forward-fill methods
- Feature scaling using normalization or standardization techniques to prepare the data for LSTM input
- Splitting the dataset into training, validation, and testing subsets.

3.3 Feature Engineering

- Creating new features such as moving averages, Relative Strength Index (RSI), or Bollinger
- Bands to enhance model performance
- Analyzing correlations among features to reduce dimensionality if necessary.

3.4 Model Design and Training

Designing an LSTM model with:

- Input layer for sequential data.
- Bands to enhance model performance
- Hidden layers with LSTM cells to capture temporal dependencies.
- Dropout layers to prevent overfitting.
- Dense output layer to predict stock prices or trends.
- Training the model using the training data and validating it on unseen data to fine-tune hyperparameters.

3.5 Model Evaluation

- Using metrics such as Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), and R-squared for evaluation.
- Visualizing predicted versus actual stock prices to assess the model's accuracy.

3.6 Prediction and Insights

- Using the trained model to forecast future stock prices or trends.
- Providing actionable insights for investment decisions based on predictions.

4 PROJECT MODULES & FEATURES

4.1 Introduction

A vital component of the global economy, the stock market is a complicate system. Businesses, analysts, and investors have long been interested in making accurate stock market predictions. Large datasets may now be analyzed and data-driven predictions can be made thanks to developments in machine learning, especially the application of deep learning models like Long Short-Term Memory (LSTM). The goal of this project is to create an LSTM-based model that leverages historical stock data analysis to forecast stock prices. The intention is to offer information that will help with well-informed investment choices.

4.2 Aim

The aim of this project is to design and implement a machine learning model using LSTM to predict stock market prices with high accuracy, leveraging historical data and identifying trends for better decision-making.

4.3 Objective

4.3.1. To collect and preprocess historical stock price data using reliable sources such as Yahoo Finance.

4.3.2. To visualize trends and patterns in stock prices using statistical methods and moving averages.

4.3.3. To implement an LSTM model for time-series forecasting of stock prices.

4.3.4. To evaluate the performance of the model using appropriate metrics such as mean squared error (MSE).

4.3.5. To explore and analyze the practical applications of stock market prediction in financial decision-making.

4.4 Stock Market

The aggregate of buyers and sellers of stocks (also known as shares), which represent ownership claims on businesses, is known as a stock market, equity market, or

share market. This loose network of economic transactions is not a physical location or distinct entity. Securities that are listed on a public stock exchange as well as those that are only traded privately may be included. Shares of private enterprises that are offered for sale to investors via equity crowd funding platforms are instances of the latter. Common equity shares and other asset kinds, such as convertible and corporate bonds, are listed on stock exchanges.
One of the most researched issues, stock price prediction draws scholars from a wide range of disciplines. It is quite challenging to use basic time-series or regression approaches because of the stock market's volatility. Institutions of finance.

4.5 Motivation

Because the market is so dynamic and volatile, it has always been difficult to predict stock values. Complex patterns and connections in the data are frequently overlooked by traditional approaches. The ability of deep learning - more especially, LSTM network - to manage sequential data efficiently and spot patterns that are hard to spot using traditional methods is what inspired this study. Through the use of these cutting-edge technologies, this project seeks to increase the accessibility and accuracy of stock market forecasts, enabling investors to make informed choices.

5 System Analysis

5.1 Objective

The objective of the system is to give an approximate idea of where the stock market might be headed. It does not give a long-term forecasting of a stock value. There are way too many reasons to acknowledge for the long-term output of a current stock. Many things and parameters may affect it on the way due to which long term forecasting is just not feasible.

5.2 Proposed System

LSTM: LSTM stands for Long Short-Term Memory, a type of artificial recurrent neural network (RNN) architecture used in the field of deep learning. LSTMs are well-suited to processing and predicting sequences of data, such as time-series data, natural language, or any data where context and order are important.

Linear Regression: Linear Regression is a statistical method used to model the relationship between a dependent variable and one or more independent variables by fitting a line (or a hyperplane in the case of multiple variables) that best represents the data.

Advantages of LSTM over Linear Regression: LSTM offers several key advantages over Linear Regression when dealing with sequential data:

1. Handling Long-Term Dependencies:

LSTM: Excels at capturing and utilizing long-term dependencies in data. This is crucial for tasks like natural language processing, where understanding the context of a sentence or paragraph requires remembering information from earlier parts.

Linear Regression: Primarily focuses on linear relationships between variables, making it less effective at capturing complex, long-term patterns in sequential data.

2. Non-Linear Relationships:

LSTM: Can model complex, non-linear relationships within the data due to its non-linear activation functions. This allows it to capture intricate patterns and dependencies that linear regression cannot.

Linear Regression: Assumes a linear relationship between variables, limiting its ability to model complex, non-linear patterns often found in real-world data.

3. Handling Time-Series Data:

LSTM: Specifically designed to handle time-series data, making it well-suited for tasks like stock market prediction, weather forecasting, and anomaly detection.

Linear Regression: While it can be applied to time-series data, its performance may be limited when the data exhibits strong temporal dependencies or non-linear patterns.

4. Flexibility and Adaptability:

LSTM: Can be combined with other neural network architectures, such as convolutional neural networks (CNNs), to create hybrid models that leverage the strengths of both. This allows for even more complex and powerful models.

Linear Regression: A simpler model with limited flexibility for combining with other architectures.

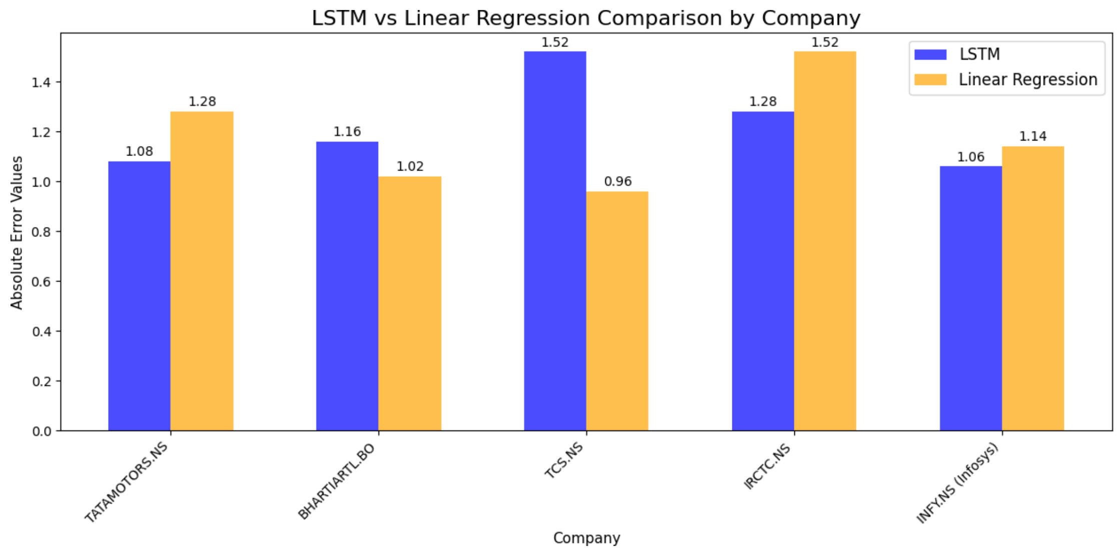


Figure 1: Comparing Linear Regression & LSTM Absolute Error

6 System Implementation

6.1 Introduction

Transforming the stock market prediction system's basic design into a fully functioning and operational model is the main goal of this project's execution phase. Data preparation, feature engineering, model training, and evaluation are just a few of the elements that must be integrated into a coherent workflow during this phase. The system efficiently analyzes time-series data and forecasts stock values by utilizing Long Short-Term Memory (LSTM) networks, a kind of recurrent neural network.

The first step in the implementation process is gathering historical stock data from trustworthy sources. Next, the data is cleaned and normalized so that it can be analyzed. Designing the LSTM model, training it on preprocessed data, and finetuning its hyperparameters to maximize its performance are all crucial components of the implementation. In order to interpret outcomes, the system also includes evaluation metrics and visualizations.

6.2 Architecture

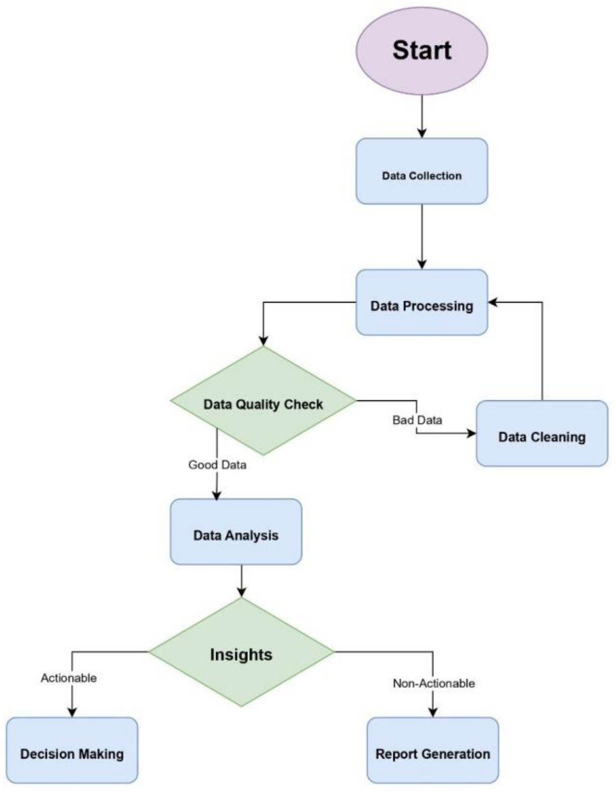


Figure 2 Flowchart

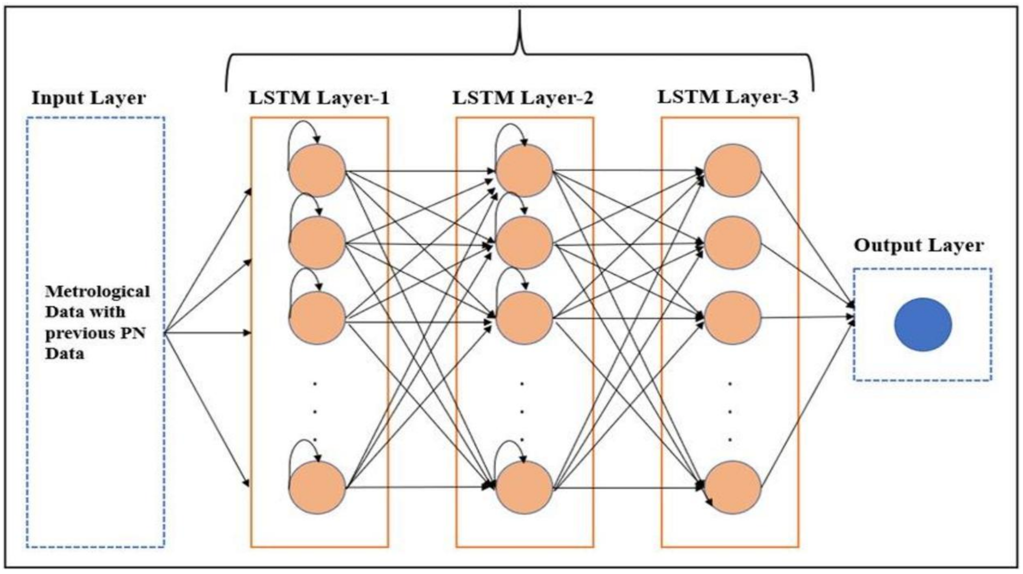


Fig. 2. Working of LSTM Model

6.3 Module Description

1. Input Data: The raw stock market data is collected, which typically includes attributes like opening price, closing price, highest and lowest prices of the day, and trading volume.
We fetch historical stock data for a specific company (e.g., TCS) from a data source like Yahoo Finance. This serves as the foundational dataset for all subsequent processes.

2. Preprocessing: Preprocessing is the step where the raw data is cleaned and prepared for analysis. This includes handling missing or null values, transforming data to a suitable scale (e.g., normalizing or standardizing values) and organizing data into meaningful formats for further processing.
We clean the dataset by removing missing values to ensure data integrity. Normalize the data to scale all values between 0 and 1, making it easier for the LSTM model to process and learn.

3. Splitting into Training and Testing Datasets: The dataset is split into two parts:

- **Training Dataset**: Used to train the machine learning model.
- **Testing Dataset**: Used to evaluate the model's performance on unseen data.

The first 80% of the stock prices are allocated for training the model, and the remaining 20% is set aside for testing. This ensures that the model is tested on data it hasn't encountered during training, providing a realistic evaluation of its performance.

4. Feature Extraction: Feature extraction focuses on identifying and isolating important information or patterns from the data that can help the model make predictions.
The LSTM model captures sequential patterns in stock prices by analyzing a sliding

window of historical prices (e.g., the past 100 days). These patterns allow the model to identify trends or dependencies in the data that contribute to more accurate predictions.

5. LSTM Model: Long Short-Term Memory (LSTM) is a specialized type of recurrent neural network (RNN) designed to capture long-term dependencies in sequential data. Unlike linear regression, which assumes a straight-line relationship, LSTM can model complex temporal patterns in stock prices.

6. Prediction: The trained model generates predictions based on the input data. This is the final output of the process, where the model forecasts future stock prices. The LSTM model predicts the next day's or next time period's stock price based on the past 100 days of data. These predictions are evaluated against the actual test data to measure accuracy and reliability.

7 Code & Output (Model Creations)

1. Library Installation

```
!pip install numpy
!pip install pandas
!pip install matplotlib
!pip install yfinance
!pip install datetime
!pip install tensorflow
!pip install sklearn
!pip install keras
```

- numpy: For numerical computations.
- pandas: For handling tabular data.
- matplotlib: For data visualization.
- yfinance: For downloading stock market data.
- datetime: To handle dates.
- tensorflow, sklearn, keras: For machine learning and deep learning.

2. Importing Libraries

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import yfinance as yf
from datetime import datetime
```

- numpy and pandas handle data processing.
- matplotlib is used to plot graphs.
- yfinance fetches stock market data.
- datetime helps to manage date and time information.

3. Setting the Date Range and Stock Symbol

```
end = datetime.today().strftime('%Y-%m-%d')
stock = 'TCS.NS'
```

- start: Beginning of the company.
- The current date (formatted as YYYY-MM-DD).
- The ticker symbol for Tata Motors on the NSE (National Stock Exchange).

4. Fetching Stock Data

```
data = yf.download(stock, end=end)
data.reset_index(inplace=True)
data
```

- yf.download(): Downloads stock market data for the specified stock, start, and end dates.
- data.reset_index(): Resets the dataframe index so that the date becomes a column.

5. Splitting the Data into Training and Testing Sets

```
data.dropna(inplace=True)
data_train = pd.DataFrame(data.Close[0:int(len(data)*0.80)])
data_test = pd.DataFrame(data.Close[int(len(data)*0.80)
               :int(len(data))])

data_train.shape[0]
data_test.shape[0]
```

- dropna(): Removes rows with missing values.
- data_train: First 80% of the closing prices for training.
- data_test: Remaining 20% for testing.

6. Scaling the Data

```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler(feature_range=(0, 1))
data_train_array = scaler.fit_transform(data_train)
x = []
y = []
for i in range(100, data_train_array.shape[0]):
   x.append(data_train_array[i-100:i])
   y.append(data_train_array[i, 0])
x, y = np.array(x), np.array(y)
```

- MinMaxScaler: Scales data to the range [0, 1] for better performance in neural networks.
- data_train: akes 100 previous time steps (x) to predict the next time step (y).

7. Building the LSTM Model

```
from tensorflow.keras.layers import Dense, Dropout, LSTM

from tensorflow.keras.models import Sequential

from tensorflow.keras.callbacks import ModelCheckpoint,
EarlyStopping, ReduceLROnPlateau

model = Sequential()

model.add(LSTM(units=50, activation='tanh',
return_sequences=True, input_shape=(x.shape[1], 1)))

model.add(Dropout(0.1))

model.add(LSTM(units=100, activation='tanh'))

model.add(Dropout(0.2))
model.add(Dense(units=50, activation='tanh'))

model.add(Dropout(0.1))

model.add(Dense(units=1))Sequential: Defines a sequential
  model.
```

- LSTM: Adds two LSTM layers with 50 and 100 units respectively.
- Dropout: Prevents overfitting by randomly ignoring a fraction of units.
- Dense: Fully connected output layer with one unit for prediction

8. Model Compilation and Training.

```
model.compile(optimizer='adam', loss='mean_squared_error')
checkpoint = ModelCheckpoint('best_model.keras',
monitor='val_loss', save_best_only=True, mode='min')
early_stop = EarlyStopping(monitor='val_loss', patience=10,
mode='min')
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.5,
patience=10, min_lr=1e-6)
```

- optimizer='adam': Specifies the Adam optimizer for updating weights during training.
- loss='mean_squared_error': The loss function is Mean Squared Error (MSE), commonly used for regression problems.

```
model.fit(x, y, epochs=50, batch_size=32, verbose=1)
```

- x and y: Input features and corresponding target values for training.
- epochs=50: The model will iterate 50 times over the entire training dataset.
- batch_size=32: The model processes 32 samples in each training iteration.
- verbose=1: Displays a progress bar during training.

```
model.summary()
```

- Prints a summary of the model architecture, including the number of layers, their shapes, and the total number of parameters.

9. Prepare Data for Testing

```
pas_100_days = data_train.tail(100)
data_test = pd.concat([pas_100_days, data_test], ignore_index=
True)
```

- data_train.tail(100): Extracts the last 100 rows from the training data.
- pd.concat(): Combines the last 100 rows of training data with the test data to ensure continuity in sequences.

```
data_test_array = scaler.fit_transform(data_test)
```

- Scales the combined test data to normalize values using the previously defined scaler (e.g., MinMaxScaler).

10. Create Sequences for Test Data

```
x = []
y = []
for i in range(100, data_test.shape[0]):
   x.append(data_train_array[i-100:i])
   y.append(data_train_array[i, 0])
x, y = np.array(x), np.array(y)
```

- Creates input sequences (x) of length 100 and corresponding target values (y).
- range(100, data_test.shape[0]): Loops from the 100th row to the end of the test data.
- data_train_array[i-100:i]: Collects 100 rows before the current row as the input sequence.
- data_train_array[i, 0]: Takes the current row's first column as the target value. Predicts output for the test sequences using the trained model.

11. Calculate and Display MAPE

```
from sklearn.metrics import mean_absolute_percentage_error
mape = mean_absolute_percentage_error(data_test.values[100:],
                                      predicted_test_price)
print(f"Mean Absolute Percentage Error (MAPE): {mape * 100:.2f}%")
```

- mean_absolute_percentage_error: Computes the Mean Absolute Percentage Error (MAPE) between actual and predicted prices.
- Prints the MAPE as a percentage.

12. Display Predictions

```
Last_date = data['Date'].iloc[-1].strftime('%Y-%m-%d')
for i in range(1, 11):
   future_date = datetime.strptime(Last_date, '%Y-%m-%d') +
                pd.DateOffset(days=i)
```

```
print(f"{i}\t{future_date.strftime('%Y-%m-%d')}\t
   {future_predictions[i-1][0]}")
```

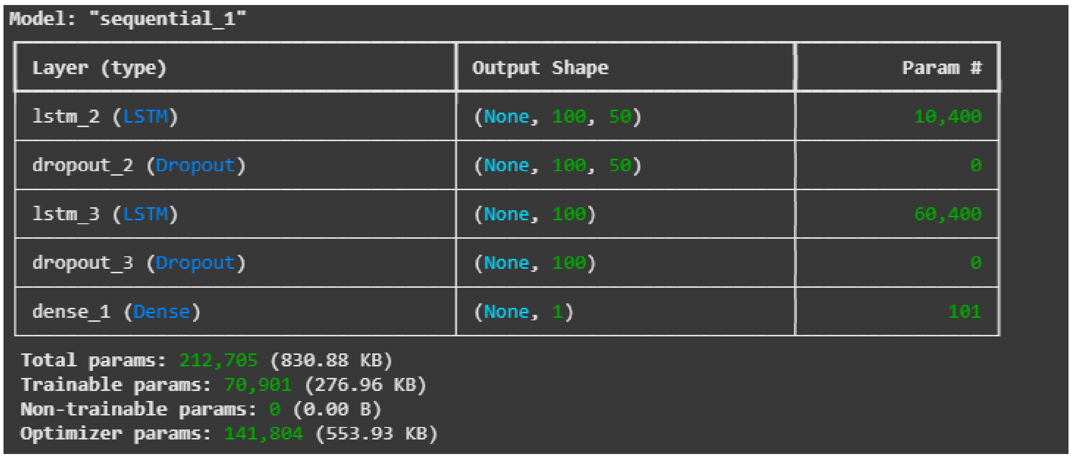
• Calculates the dates for the next 10 days.

Prints the predicted closing price for each day

13. Saving model

```
from tensorflow.keras.models import load_model
model = load_model('best_model.keras')
```

- Purpose:
  - Loads a pre-trained LSTM model from a file for making predictions.
- **from tensorflow.keras.models import load_model**:
  - Imports the load_model function used to load saved Keras models from disk.
- **model = load_model('best_model.keras')**:
  - Loads the model saved in the file named 'best_model.keras'.
  - Assigns it to the variable model, making it ready for use (e.g. prediction or evaluation)..



```
Model: "sequential_1"
┌──────────────────────────────┬─────────────────────┬──────────────┐
│ Layer (type)                 │ Output Shape        │      Param # │
├──────────────────────────────┼─────────────────────┼──────────────┤
│ lstm_2 (LSTM)                │ (None, 100, 50)     │       10,400 │
├──────────────────────────────┼─────────────────────┼──────────────┤
│ dropout_2 (Dropout)          │ (None, 100, 50)     │            0 │
├──────────────────────────────┼─────────────────────┼──────────────┤
│ lstm_3 (LSTM)                │ (None, 100)         │       60,400 │
├──────────────────────────────┼─────────────────────┼──────────────┤
│ dropout_3 (Dropout)          │ (None, 100)         │            0 │
├──────────────────────────────┼─────────────────────┼──────────────┤
│ dense_1 (Dense)              │ (None, 1)           │          101 │
└──────────────────────────────┴─────────────────────┴──────────────┘
 Total params: 212,705 (830.88 KB)
 Trainable params: 70,901 (276.96 KB)
 Non-trainable params: 0 (0.00 B)
 Optimizer params: 141,804 (553.93 KB)
```

Fig 4: Model details

8 Research Report Survey

8.1 Research paper on Machine Learning Based Framework for a Stage-Wise Classification of Date Palm White Scale Disease

Authors: Abdelaaziz Hessane, Ahmed El Youssefi

Objective

The study explores machine learning frameworks for identifying White Scale Disease (WSD) in palm plants.
Models Used: - **SVM (Support Vector Machine):** Finds the optimal hyperplane to separate data into classes. - **KNN (K-Nearest Neighbors):** Classifies based on neighbors' labels. - **RF (Random Forest):** Ensemble of decision trees for improved accuracy and reduced overfitting. - **LightGBM (Light Gradient Boosting Machine):** Efficient gradient boosting with lower memory usage.
Colour Models: - **GLCM (Gray Level Co-occurrence Matrix):** Analyzes image textures. - **HSV (Hue, Saturation, Value):** Provides better color representation by separating intensity and color information.

Conclusion

The SVM classifier achieved the highest accuracy (98.29%) using combined GLCM and HSV features. The study highlights the effectiveness of supervised (SVM, KNN) and ensemble (RF, LightGBM) learning models, emphasizing the advantages of the HSV color model for disease detection.

9 Financial Market Prediction Using LSTM Networks

Authors: Thomas Fischer, Christopher Krauss

9.1 Introduction

This study applies LSTM networks to predict stock price movements in the S&P 500 index (1992–2015). LSTMs outperformed traditional models, achieving daily returns of 0.46% and a Sharpe Ratio of 5.8.

9.2 Key Concepts

LSTM Networks: Suitable for time-series data, LSTMs address vanishing gradients with memory cells and gates to retain long-term dependencies. Trained on 240-day stock return sequences, LSTMs excel at identifying profitable patterns.
Stock Patterns: LSTMs identified volatile stocks with reversal behavior, aiding the development of simplified trading strategies based on recent losses or gains.

9.3 Conclusion

LSTM models capture complex dependencies in financial data, outperforming Random Forests, DNNs, and Logistic Regression. Insights into stock patterns demonstrate the model's utility for trading strategies.

10 XGBoost: A Scalable Tree Boosting System

Authors: Tianqi Chen, Carlos Guestrin

10.1 Introduction

XGBoost is a gradient boosting framework optimized for speed and scalability, widely used in classification, regression, and ranking tasks.

10.2 Key Concepts and Innovations

System Optimizations: - Sparsity-aware learning for efficient handling of missing values. Weighted quantile sketch for accurate split point approximation. Out-of-core computation for large datasets.
Algorithmic Features: - Regularization to prevent overfitting. Shrinkage and column subsampling for stability.

10.3 Conclusion

XGBoost achieves state-of-the-art performance in large-scale applications, ex- celling in speed and accuracy. Its optimizations make it a powerful tool in modern machine learning.

11 Fraud Detection and Financial Classification in Financial Applications using Deep Learning.

11.1 Deep Learning for Financial Applications: A Survey

Authors: Nguyen, T. T., Choi, H., & Park, J. H. (2019)

Main Focus

This survey explores the application of Machine Learning (ML) and Deep Learning (DL) techniques in financial applications, focusing on areas such as fraud detection, credit scoring, and algorithmic trading. With the availability of large financial datasets and computational power, ML and DL play a crucial role in enhancing financial predictions and decision-making processes.

Key Financial Applications

- **Fraud Detection:** - ML and DL techniques help identify suspicious financial transactions. **LSTM Networks** model sequential patterns in transaction data to detect anomalies. **CNNs** are effective for image-based fraud detection, such as check fraud.
- **Credit Scoring:** - DL models like Feedforward Neural Networks (FNNs) and Gradient Boosting Machines (GBMs) enhance the accuracy of credit scoring. - They analyze borrower characteristics and risk, aiding financial institutions.
- **Algorithmic Trading:** - Deep Reinforcement Learning models enable real-time decision-making for trading. - **GANs** generate synthetic data, helping develop robust strategies for volatile markets.

Key Models for Financial Predictions

- **LSTM:** Effective in modeling sequential data, tracking transaction patterns over time.
- **CNN:** Adapted for fraud detection scenarios involving image data, like detecting fraudulent checks.
- **GANs:** Generate synthetic data to create realistic transaction scenarios for training robust models.

Challenges and Future Directions

- **Data Privacy:** Ensuring compliance while handling sensitive financial data.
- **Model Interpretability:** Making DL models transparent for regulatory use.
- **Real-Time Processing:** Addressing scalability for large datasets and high-speed decision-making.

Conclusion: The survey highlights significant advancements in financial applications using DL techniques like LSTM, CNN, and GANs. Future research should focus on improving interpretability and regulatory compliance to further enhance their applicability in finance.

12 Cybersecurity and Intrusion Detection Using Deep Learning

12.1 Applying Deep Learning for Network Intrusion Detection: A Comprehensive Review

Authors: Vinayakumar, R., Soman, K. P., & Poornachandran, P. (2019)

Main Focus

This review investigates the application of ML and DL in Network Intrusion Detection Systems (NIDS) for cybersecurity. With the increasing sophistication of cyber threats, DL methods provide accurate and efficient anomaly detection, threat prediction, and classification.

Key Applications of Intrusion Detection

- **Anomaly-Based Detection:** Focuses on deviations from normal behavior patterns. - Uses models like **RNNs** and **LSTMs** for time-sequenced data.
- **Signature-Based Detection:** Relies on known threat signatures. **CNNs** process structured data to detect complex attack patterns.
- **Hybrid Detection:** - Combines anomaly- and signature-based approaches. Uses **Autoencoders** and **GANs** to learn mixed behavior patterns.

Key Deep Learning Models for Intrusion Detection

- **CNN:** Extracts features from structured network data.
- **RNN and LSTM:** Analyze time-based activities to identify gradual intrusions.

- **Autoencoders:** Highlight anomalies by reconstructing normal network traffic.
- **GANs:** Augment datasets and improve detection of zero-day attacks.

Challenges and Future Directions

- **Data Quality:** Addressing imbalanced datasets and rare attack types.
- **Real-Time Processing:** Scaling DL models for high-speed networks.
- **Interpretability:** Ensuring transparency in detection results for sensitive applications.

Conclusion: This review underscores the potential of DL models like CNNs, LSTMs, Autoencoders, and GANs for network intrusion detection. Future research should enhance model interpretability, address data imbalance, and focus on real- time performance to advance DL's role in cybersecurity.

13 Deep Neural Networks for YouTube Recommendations

Authors: Covington, P., Adams, J., & Sargin, E.

13.1 Introduction

In the 2016 paper *Deep Neural Networks for YouTube Recommendations*, Covington, Adams, and Sargin present the architecture of YouTube's recommendation system, which employs deep neural networks (DNNs) to personalize video suggestions. This recommendation system focuses on efficiently narrowing down billions of potential content options for users while maintaining a high likelihood of engagement. The methodologies described in the paper have broad relevance, influencing recommendation systems in e-commerce, streaming services, and applications such as traffic prediction and smart cities.

13.2 Key Components of YouTube's Recommendation System

The architecture of YouTube's recommendation system is built around two primary stages:

- **Candidate Generation:** The system reduces the large pool of available videos to a smaller set of candidates. Using user interaction data—such as watch history, searches, and engagement patterns—the candidate generation model leverages DNNs to infer user-video affinity, narrowing down choices to a few hundred videos.
- **Ranking:** Once candidates are generated, the ranking model scores each video to prioritize those most likely to interest the user. This stage uses features like watch history, video popularity, context (e.g., time of day or device type), and user engagement patterns to refine the list, balancing relevance and diversity.

13.3 Deep Learning and Model Architecture

The model architecture applies DNNs for both candidate generation and ranking. These DNNs efficiently handle large-scale data and adapt dynamically to shifting user preferences. Features derived from YouTube's extensive dataset—such as video metadata, user demographics, and contextual data—enhance the quality.

13.4 Broader Applications and Relevance

The techniques used in YouTube's recommendation model extend beyond video recommendations, offering insights for:

- **E-commerce and Streaming Services:** Similar architectures predict user preferences in online shopping and media streaming, tailoring recommendations based on browsing or viewing history.
- **Traffic Prediction and Smart Cities:** DNN-based recommendation systems optimize traffic predictions, forecast congestion patterns, manage transportation demands, and streamline urban planning in smart cities.

13.5 Conclusion

The paper by Covington et al. introduces a scalable, adaptable architecture for recommendation systems, revolutionizing content presentation on platforms with diverse user bases. The use of DNNs in candidate generation and ranking offers a powerful, flexible solution applicable across industries, from entertainment to urban planning, highlighting the growing role of machine learning in enhancing user experiences.

14 Revisiting Spatial-Temporal Similarity: A Deep Learning Framework for Traffic Prediction

Authors: Yao, H., Tang, X., Wei, H., Zheng, G., & Li, Z.

14.1 Introduction

In the paper *Revisiting Spatial-Temporal Similarity: A Deep Learning Framework for Traffic Prediction*, Yao et al. (2019) propose an innovative framework to capture spatial and temporal features for accurate traffic flow prediction. This framework addresses urban traffic complexities and has applications in smart city planning, autonomous driving, and efficient urban management. By focusing on spatial-temporal similarity, the model enhances traffic predictions and supports better urban infrastructure planning.

14.2 Model Overview

- **Spatial-Temporal Representation:** The framework combines spatial data, which reflects physical road connectivity, with temporal data, which captures time-dependent patterns like rush hours and seasonal trends. This holistic approach improves prediction accuracy.
- **Deep Learning Architecture:** The model integrates Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs). CNNs capture spatial dependencies between areas (e.g., intersections and road segments), while RNNs (typically LSTMs) learn sequential patterns in traffic flow. This combination captures complex, dynamic traffic patterns.

14.3 Key Contributions and Features

- **Spatial-Temporal Similarity Module:** This module generalizes patterns observed in one area to similar regions, reducing the need for extensive data and enabling faster predictions.

- **Attention Mechanism:** The attention mechanism prioritizes critical spatial temporal patterns, improving model efficiency and interpretability. This feature helps urban planners identify factors driving traffic congestion in specific regions.

14.4 Applications and Relevance

The framework has significant implications in several domains:

- **Smart City Planning:** It supports urban planners by informing data-driven decisions on road expansions, signal timing adjustments, and congestion management.
- **Autonomous Driving:** Real-time traffic predictions aid autonomous vehicles in navigation, rerouting, and ensuring safety in dynamic urban environments.
- **Urban Management:** The model enables dynamic traffic management, such as rerouting public transport during peak hours and identifying high-emission areas for eco-friendly policies.

14.5 Conclusion

Yao et al.'s (2019) deep learning framework effectively captures spatial-temporal dependencies for accurate traffic prediction. By integrating CNNs and RNNs, along with spatial-temporal similarity and attention mechanisms, the model delivers enhanced performance. This framework holds immense potential for smart cities, autonomous driving, and real-time urban management, setting a benchmark for spatial-temporal modeling in traffic systems.

15 PLATFORM & TECHNOLOGY

15.1 Programming Language

- Python: The primary programming language due to its extensive libraries for data analysis, machine learning, and deep learning.

15.2 Frameworks and Libraries to Build Environment (Requirements.txt)

```
bcrypt==4.3.0
cachetools==5.5.2
cryptography==44.0.2
cycler==0.12.1
DateTime==5.5
joblib==1.4.2
keras==3.9.2
matplotlib==3.10.1
ml_dtypes==0.5.1
mysql-connector==2.2.9
numpy==2.1.3
pandas==2.2.3
pillow==11.2.1
PyMySQL==1.1.1
python-dateutil==2.9.0.post0
scikit-learn==1.6.1
scipy==1.15.2
streamlit==1.45.0
tensorboard==2.19.0
tensorboard-data-server==0.7.2
tensorflow==2.19.0
tensorflow-io-gcs-filesystem==0.31.0
yfinance==0.2.59
```

- **Bcrypt** — Provides secure password hashing using the bcrypt algorithm.
- **cryptography** — Supplies a wide range of low-level cryptographic primitives for secure communications.
- **PyMySQL / mysql-connector**— Python libraries used to connect and interact with MySQL databases.
- **cachetools** — Implements extensible memorizing collections and decorators for efficient caching.
- **DateTime** — Offers object-oriented date and time operations (often substituted with Python's built-in datetime module).
- **python-dateutil**— Provides advanced extensions to the standard datetime module, including parsing and timezone support.
- **numpy**— Core library for numerical computing with support for multi-dimensional arrays and mathematical operations.
- **pandas** — Powerful data manipulation and analysis library based on NumPy.

- **scikit-learn** — Provides machine learning algorithms and tools for data mining and analysis.
- **scipy**— Extends NumPy with additional modules for optimization, integration, interpolation and statistics.
- **ml_dtypes**— Supplies TensorFlow-compatible NumPy data types for machine learning workloads.
- **joblib** — Enables efficient disk-based caching and parallel execution for computational tasks.
- **tensorflow** — End-to-end open-source platform for building & training deep learning models.
- **tensorflow-io-gcs-filesystem** — Provides file system support for TensorFlow I/O, including Google Cloud Storage.
- **keras**— High-level API for building and training deep learning models on top of TensorFlow.
- **tensorboard** — Visualization toolkit for monitoring & debugging machine learning training.
- **tensorboard-data-server** — Backend server that supports the TensorBoard interface.
- **matplotlib** — Comprehensive library for creating static, animated, and interactive 2D visualizations.
- **pillow** — Python Imaging Library (PIL) fork, used for opening, manipulating & saving image files.
- **cycler** — Provides a composable way to configure matplotlib's property cycling.
- **streamlit** — Framework for building interactive & responsive web applications directly from Python scripts.
- **yfinance**— Utility for downloading historical market data from Yahoo Finance.

15.3 Database used in Application

```
def get_db_connection():
    try:
        connection = mysql.connect(
            host="metro.proxy.rlwy.net",
            port=58802,
            user="root",
            password="MavluVOwwCWmmiiiXnvjihnIqOyuopVe",
            database="railway"
        )
        return connection
    except Exception as e:
        st.error(f"Connection error: {e}")
             st.stop()
```

- Connects to the remote MySQL database using provided credentials.
- If the connection fails, it shows an error message in Streamlit and stops the app using st.stop().

```
def authenticate_user(username, password):
    username = username.strip().lower()
    conn = get_db_connection()
    try:
        cursor = conn.cursor()
        query = "SELECT COUNT(*) FROM userlogin WHERE username =%s
     AND userPassword=%s"
        cursor.execute(query, (username, password))
        result = cursor.fetchone()
        return result and result[0] == 1
    except Exception as e:
        st.error(f"Authentication error: {e}")
        return False
    finally:
        conn.close()
```

- Normalizes the username (trims spaces, makes lowercase).
- Checks if the given username and password exist in the userlogin table.
- Returns True if the user is authenticated, False otherwise

```
def register_user(username, password):

  username = username.strip().lower()

  conn = get_db_connection()

  try:

      cursor = conn.cursor()

     cursor.execute("SELECT COUNT(*) FROM userlogin WHERE
userName=%s", (username,))

      if cursor.fetchone()[0] > 0:

          st.warning("Username already exists.")

          return False

      cursor.execute("INSERT INTO userlogin (userName,
userPassword) VALUES (%s, %s)", (username, password))

      conn.commit()

      st.success("Account created! Please log in.")

      return True

  except Exception as e:

      st.error(f"Registration error: {e}")

      return False

  finally:

          conn.close()
```

- Checks if the username already exists.
- If not, inserts the new user into the userlogin table.
- Displays appropriate Streamlit messages (warning, success, or error).

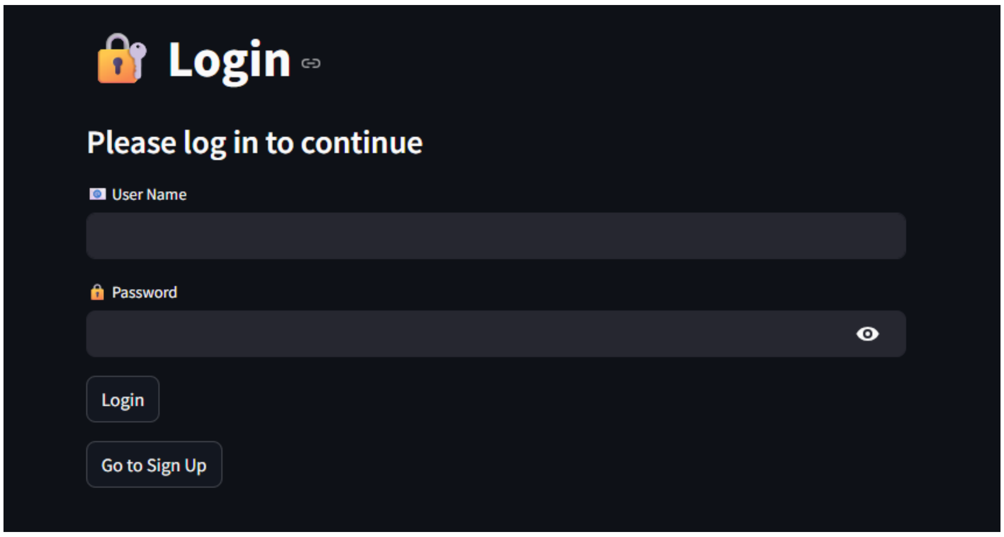
```
def insert_history(username, company_name, company_ticker, investment_amount, final_value, total_profit):

connection = get_db_connection()

if connection:

cursor = connection.cursor()

try:

query = """

INSERT INTO history (username, company_name, company_ticker, investment_amount, final_value, total_profit)

VALUES (%s, %s, %s, %s, %s, %s)

"""

cursor.execute(query, (username, company_name, company_ticker, investment_amount, final_value,  total_profit))

connection.commit()

st.success("Transaction history saved successfully.")

except mysql.connector.Error as err:

st.error(f"Error saving transaction: {err}")

finally:

cursor.close()

connection.close()
```

- Inserts a record into the history table when a user performs a stock-related transaction.
- Parameters include the username, company info, investment amount, result, and profit.
- On success, shows a Streamlit success message.

```
def seek_history(username):
    connection = get_db_connection()
    if connection:
        cursor = connection.cursor()
        try:
            query = "SELECT username, company_name, company_ticker, investment_amount, final_value, total_profit,timestamp FROM history WHERE username = %s order  by Id desc"
            cursor.execute(query, (username,))
            history_data = cursor.fetchall()
            if history_data:
                # Adjust the column names to match the actual table schema
                df_history = pd.DataFrame(history_data,
                    columns=["Username", "Company Name", "Company Ticker", "Investment Amount", "Final Value", "Total Profit", "Transaction Date"])
                st.dataframe(df_history)
            else:
                st.warning("No history found.")
        except mysql.MySQLError as err:
            st.error(f"Error fetching transaction history: {err}")
        finally:
            cursor.close()
            connection.close()
```

- Fetches all transaction history for the specified user, ordered by most recent (ORDER BY Id DESC).
- Converts the result into a pandas DataFrame and displays it in a table (st.dataframe()).
- If no data is found, shows a warning.

15.4 Login Page.


Login

Please log in to continue

User Name

Password

Fig 5: Project Login Page

```
def login_page():
    st.set_page_config(page_title="Login", page_icon="",
layout="centered")
    st.title(" Login")
    st.subheader("Please log in to continue")

    email = st.text_input(" User Name", key="login_email")
    password = st.text_input(" Password", type="password")

    if st.button("Login"):
        if authenticate_user(email, password):
            st.session_state.username = email
            st.success("Login successful!")
            st.session_state.authenticated = True
            st.session_state.page = "main"
            st.rerun()
        else:
            st.error("Invalid UserName or password.")

    if st.button("Go to Sign Up"):
        st.session_state.page = "signup"
        st.rerun()
```

The login_page() function:

- Sets page config: Title as "Login", key icon, centered layout.
- Displays title/subheader: Shows " Login" and a prompt.
- Takes user input:
  - **email:** Text input for username/email.
  - **password:** Password input (masked).
- **Login logic:**
  - On clicking Login:
    - Calls **authenticate_user(email, password)**
    - If valid:
      - Stores username in session
      - Sets **authenticated = True**
      - Navigates to "**main**" page and reruns app
    - If invalid: Shows error message.
  - Sign-up redirect:
  - On clicking Go to Sign Up, sets page to "signup" and reruns.

15.5 Signup Page

```
def signup_page():
    st.set_page_config(page_title="Sign Up", page_icon="",
     layout="centered")
    st.title(" Sign Up")
    st.subheader("Create a new account")

    email = st.text_input(" User Name", key="signup_email")
    password = st.text_input(" Password", type="password",
     key="signup_password")
    confirm_password = st.text_input(" Confirm Password",
     type="password", key="signup_confirm")

    if st.button("Create Account"):
        if not email or not password:
            st.warning("All fields are required.")
            elif password != confirm_password:
                st.warning("Passwords do not match.")

            else:
                register_user(email, password)
                st.rerun()

        if st.button("Back to Login"):
            st.session_state.page = "login"
                st.rerun()
```

Here's a brief bullet-point explanation of the signup_page() function:

- **Sets page config**: Title as "Sign Up", pencil icon, centered layout.
- **Displays title/subheader**: Shows " Sign Up" and a prompt to create an account.
- **Takes user input**:
  - email: Text input for username/email.
    - password: Password input (masked).
    - confirm_password: Confirm password input (masked).
  - **Account creation logic**:
  - On clicking **Create Account**:
- If any field is empty: shows a warning.
  - If passwords don't match: shows a warning.
    - If valid: calls register_user(email, password) and reruns app.
- **Back to login**:
  - On clicking **Back to Login**, sets page to "login" and reruns.

15.6 Home Page

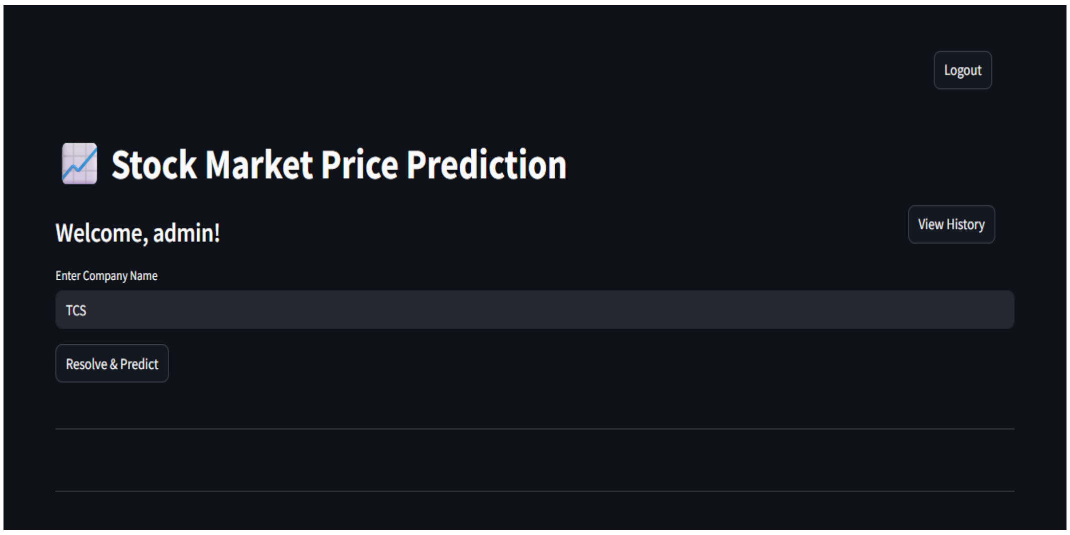


Fig 6: Project Home page

Logout:

```
cols = st.columns([10, 1])
    with cols[1]:
        if st.button("Logout", key="logout"):
            st.session_state.authenticated = False
            st.session_state.page = "login"
            st.session_state.predict_clicked = False
            st.session_state.resolved_ticker = ""
            st.session_state.username = ""
                st.rerun()
```

Here's a brief bullet-point explanation of the provided code:

- Creates layout with columns: st.columns([10, 1]) makes two columns—left (wide) and right (narrow).
- **Places Logout button:** Inside the narrow column (cols[1]).
- **Logout logic on click:**
- Sets authenticated to False (logs user out).
    - Redirects to the "login" page.
    - Resets prediction state and ticker.
    - Clears the logged-in username.

- Reruns the app to apply changes.

History:

```
def history_page():
    st.set_page_config(page_title="History",page_icon="",
     layout="wide")
    st.title("History")
    st.subheader("Your Investment History")

    if "username" in st.session_state:
        username = st.session_state.username
        seek_history(username)
    else:
        st.warning("You must be logged in to view your
history .")
    if st.button("Back to Main"):
        st.session_state.page = "main"
        st.rerun()
```

Here's a brief bullet-point explanation of the history_page() function:

- **Page setup:** Sets title, icon, and layout to "wide" using st.set_page_config.,
- **Displays title and subheader:** " History" and "Your Investment History".
- **Checks login status:**
    - If "username" is in st.session_state, it calls seek_history (username) to display user's investment history.
    - Otherwise, shows a warning that login is required.
- **Back button:**
    - If "Back to Main" is clicked, it sets the current page to "main" and reruns.

Searching Company's Ticker ID

```
import yfinance as yf

import requests


# Static fallback dictionary

company_to_ticker_map = {

    "TCS": "TCS.NS",
"INFOSYS": "INFY.NS",
"WIPRO": "WIPRO.NS",
"HDFC BANK": "HDFCBANK.NS",
"RELIANCE": "RELIANCE.NS",
"ICICI BANK": "ICICIBANK.NS",
"BHARTI AIRTEL": "BHARTIARTL.NS",
"APPLE": "AAPL",
"MICROSOFT": "MSFT",
"AMAZON": "AMZN",
"GOOGLE": "GOOG",
"META": "META",
"TESLA": "TSLA",
"BERKSHIRE HATHAWAY": "BRK-A",
"VISA": "V",
"JPMORGAN CHASE": "JPM",
"JOHNSON & JOHNSON": "JNJ",
"WALMART": "WMT",
"PROCTER & GAMBLE": "PG",
"MASTERCARD": "MA",
"BANK OF AMERICA": "BAC",
"NVIDIA": "NVDA",
"HOME DEPOT": "HD",
"ADOBE": "ADBE",
"CISCO": "CSCO",
"NETFLIX": "NFLX",
"PEPSICO": "PEP",
"LARSEN & TOUBRO": "LT.NS",
"AXIS BANK": "AXISBANK.NS",
        "KOTAK MAHINDRA BANK": "KOTAKBANK.NS",
        "ULTRATECH CEMENT": "ULTRACEMCO.NS",
        "HCL TECHNOLOGIES": "HCLTECH.NS",
        "MARUTI SUZUKI": "MARUTI.NS",
        "TATA MOTORS": "TATAMOTORS.NS",
        "TATA STEEL": "TATASTEEL.NS",
        "MAHINDRA & MAHINDRA": "M&M.NS",
        "ASIAN PAINTS": "ASIANPAINT.NS",
        "SUN PHARMACEUTICAL": "SUNPHARMA.NS",
        "DIVI'S LABORATORIES": "DIVISLAB.NS",
        "BRITANNIA INDUSTRIES": "BRITANNIA.NS",
        "NESTLE INDIA": "NESTLEIND.NS",
        "ADANI PORTS": "ADANIPORTS.NS",
        "STATE BANK OF INDIA": "SBIN.NS",
        "POWER GRID": "POWERGRID.NS",
        "SHREE CEMENT": "SHREECEM.NS",
        "INDUSIND BANK": "INDUSINDBK.NS",
        "BAJAJ FINANCE": "BAJFINANCE.NS"
    }

def resolve_company_to_ticker(company_name):
    # Input validation
    company_name = company_name.upper()
    if not company_name or not isinstance(company_name, str):
        return None

    # Normalize company name for lookup
    company_name = company_name.strip().upper()
    # Try using static map first
    if company_name in company_to_ticker_map:
        return company_to_ticker_map[company_name]
    else:
        # Try yfinance's search API
            try:

     url=f"https://query2.finance.yahoo.com/v1/finance/search?q=
     {company_name}"
                headers = {
                    "User-Agent": "Mozilla/5.0
(compatible; MyApp/1.0)"
                }
                response = requests.get(url, headers=headers)
                response.raise_for_status()  # Raise an error  for
bad status codes
                data = response.json()
                if data.get("quotes"):
                    return data["quotes"][0]["symbol"]
            except requests.RequestException as e:
                print(f"Error during API request: {str(e)}")
            except (KeyError, IndexError) as e:
                print(f"Error processing API response:  {str(e)}")
    return None
```

Here's a **brief bullet-point explanation** of the `resolve_company_to_ticker()` function & related code:

- **Imports:**
  - `yfinance` for financial data (not used directly here but may be used elsewhere).
  - `requests` to make API calls.

- **Static map:**
  - A dictionary `company_to_ticker_map` maps known company names to their stock ticker symbols.
- Function `resolve_company_to_ticker(company_name):
  - Converts input to uppercase and strips whitespace for consistency.
  - Checks static map first:
    - If company name exists, returns its mapped ticker.
  - If not found, attempts dynamic lookup via Yahoo Finance API:
    - Sends a GET request to Yahoo's search API with user-agent header.
    - Parses the JSON response and returns the first symbol if available.
  - **Error handling:**
    - Catches and logs request or parsing errors.
  - **Fallback:**
    - Returns `None` if no match is found.

Retrieving Data from yfinance.

```
if st.session_state.predict_clicked and st.session_state.resolved_ticker:
        stock = st.session_state.resolved_ticker
        data = yf.download(stock, end=end)
        if data.empty:
            st.error("No stock data found. Please check the company
name or ticker.")
            return
    # Fetch and display historical data
        data = yf.download(stock, end=end)
        st.subheader("Historical Data")
            st.write(data) do the same
```

Here’s a brief bullet-point explanation of the given code:

- **Condition:**
    - Checks if prediction was triggered (predict_clicked) & a stock ticker was resolved (resolved_ticker).
- **Download stock data:**
    - Uses yfinance.download() to fetch historical stock data for the resolved ticker up to the specified end date.
- **Check for data:**
    - If no data is returned (data.empty), shows an error message and exits the function.
- **Display data:**
    - Re-downloads the data (redundantly) and displays it under "Historical Data" heading.

Spliting data into training and testing:

```
#train data from start to today
        data_train=pd.DataFrame(data.Close[0:int(len(data)0.80)])
        data_test=pd.DataFrame(data.Close[int(len(data)0.80):int
(len(data))])
```

Here's a brief bullet-point explanation of the code:

Purpose:

* Splits stock price data into training and testing datasets.

* **data.Close:**

* Uses only the closing prices from the full dataset.

* Training data (data_train):

* Contains the first 80% of the closing price data.

* Testing data (data_test):

* Contains the remaining 20% of the closing price data .

Feature Scaling:

```
from sklearn.preprocessing import MinMaxScaler

scaler=MinMaxScaler(feature_range=(0,1))

data_train_array=scaler.fit_transform(data_train)

x=[]

y=[]

for i in range(100,data_train_array.shape[0]):

  x.append(data_train_array[i-100:i])

  y.append(data_train_array[i,0])

x,y=np.array(x),np.array(y)
```

Here's a brief bullet-point explanation of the code:

Purpose:

* Prepares the training data for the LSTM model using feature scaling and sequence creation.

* **MinMaxScaler:**

* Scales the training data to a range between 0 and 1.

* **data_train_array:**

* Transformed/scaled training dataset.

* **Loop (sequence creation):**

* Creates input sequences of 100 time steps (x) and the next value as target (y), for each point after the first 100.

* **x, y:**

* Final NumPy arrays used to train the LSTM model.

Moving Average of 100 Days & 200 Days with Closing Price:

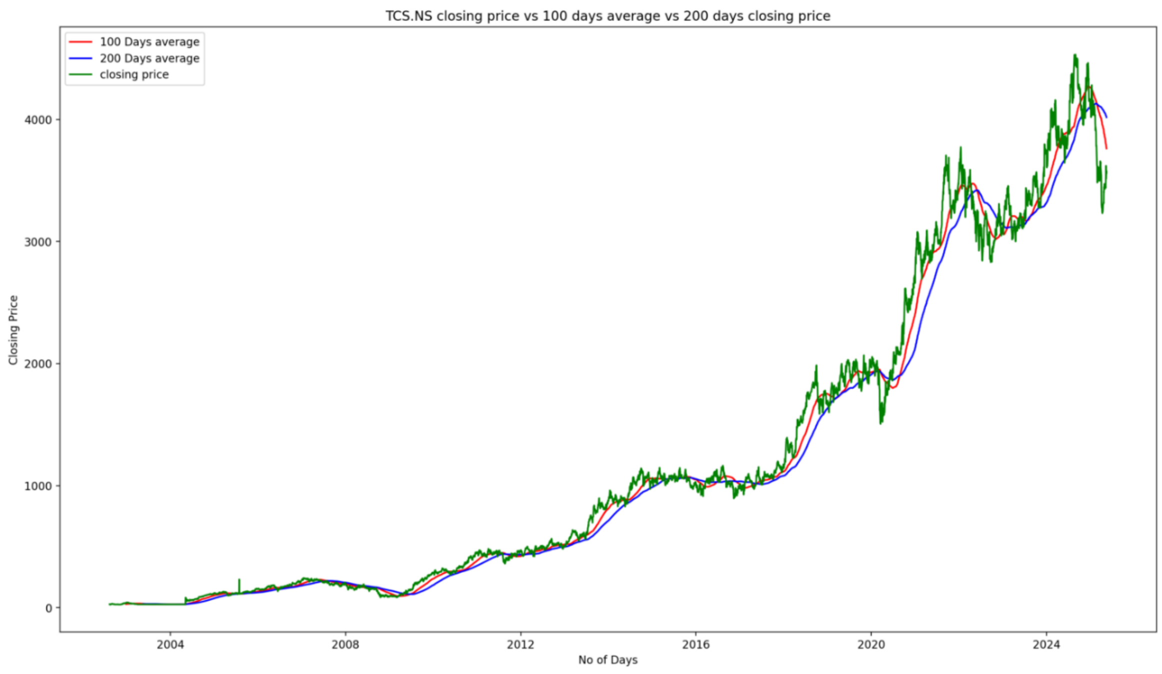


Fig 7: Moving average with closing price

```
    st.subheader("Moving Average 100 and 200 days")

    ma_100_days = data['Close'].rolling(100).mean()

    ma_200_days=data.Close.rolling(200).mean()

    fig1=plt.figure(figsize=(18,10))

    plt.plot(ma_100_days,'r',label="100 Days average")

    plt.plot(ma_200_days,'b',label="200 Days average")

    plt.plot(data.Close,'g',label='closing price')

    plt.legend()

    plt.xlabel('No of Days')

    plt.ylabel('Closing Price')

    plt.title(f'{stock} closing price vs 100 days average vs 200
days closing price')

        st.pyplot(fig1)
```

Here’s a brief bullet-point explanation of the code:

- **Purpose:**
    - Visualizes stock closing prices along with 100-day and 200-day moving averages.

- **Moving Averages:**
    - ma_100_days: Rolling average of the closing price over the last 100 days.
    - ma_200_days: Rolling average of the closing price over the last 200 days.
- **Plot:**
    - Creates a Matplotlib figure fig1 with:
        - Red line for 100-day average.
        - Blue line for 200-day average.
        - Green line for actual closing prices.
        - Adds labels, legend, and title for clarity.
- **Streamlit:**
    - st.pyplot(fig1): Displays the plot in the Streamlit app.

Using model data for data prediction:

```
using model data for data prediction

[12:36]

st.subheader("Prediction vs actual price")

        pas_100_days=data_train.tail(100)

        data_test=pd.concat([pas_100_days,data_test],
    ignore_index=True)

        data_test_array=scaler.fit_transform(data_test)

        x=[]

        y=[]

        for i in range(100,data_test_array.shape[0]):

          x.append(data_train_array[i-100:i])

          y.append(data_train_array[i,0])
```

```
                x,y=np.array(x),np.array(y)

                ypredict=model.predict(x)

                scale=1/scaler.scale

                y_predict=y_predictscale

     y=yscale
```

Here’s a brief bullet-point explanation of the code:

- **Purpose:**
  - To compare the predicted stock prices against actual prices using the trained LSTM model.
- **Preparing Test Data:**
  - pas_100_days: Last 100 days from training data are included as context.,
  - data_test: Combined with the test set to maintain continuity for prediction input.
  - data_test_array: Scaled using MinMaxScaler to normalize data.
- **Input Sequence Creation:**
  - Loops from day 100 onward to create:
    - x: Sequences of the previous 100 data points.
    - y: Actual next day's price.,
    - Converts x and y to NumPy arrays for model input.
- **Prediction & Rescaling:**
  - y_predict: Predictions made by the LSTM model.,
  - scale = 1 / scaler.scale_: Used to revert scaled predictions and labels back to original values.
  - y_predict and y are both scaled back to real price values.

Predicted Result vs Test Result.

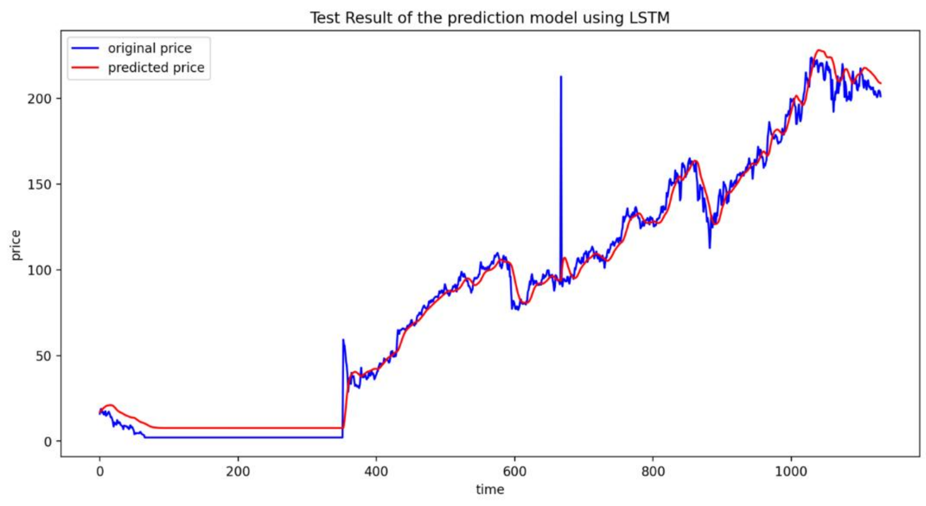


Fig 8: Model Testing and Result

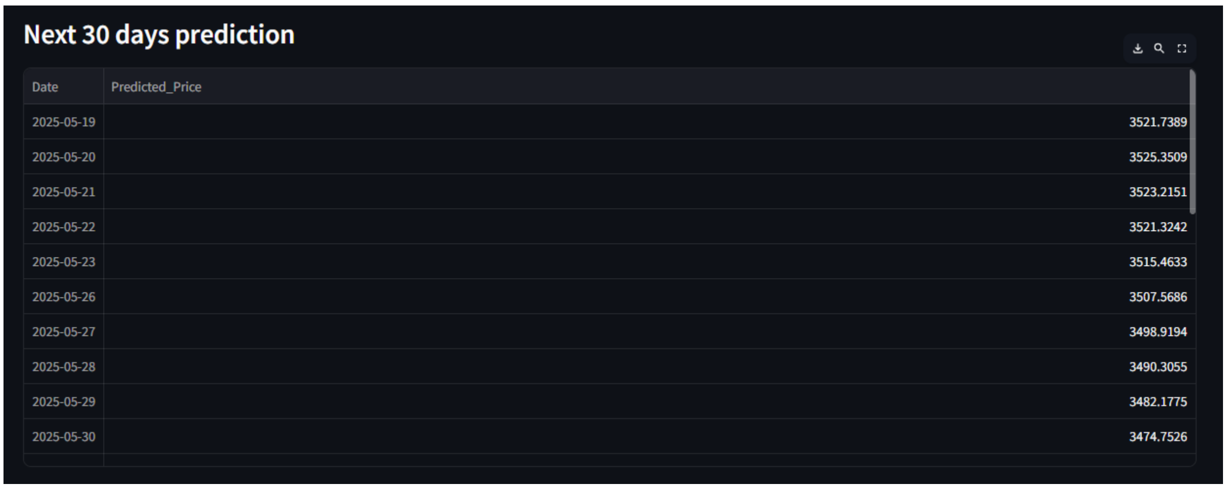
```
fig2=plt.figure(figsize=(12,6))
    plt.plot(y,'b',label='original price')
    plt.plot(y_predict,'r',label='predicted price')
    plt.xlabel('time')
    plt.ylabel('price')
    plt.title("Test Result of the prediction model using LSTM")
    plt.legend()
    st.pyplot(fig2)
```

Here’s a brief bullet-point explanation of the given code block:

- **Purpose:**
  - ○ To visualize and compare the actual vs predicted stock prices using a line chart.
- **Plotting with Matplotlib:**
  - ○ fig2 = plt.figure(figsize=(12,6)):
    - ▪ Creates a new figure with a specific size for the plot.
  - ○ plt.plot(y, 'b', label='original price'):
    - ▪ Plots the actual stock prices in blue.

- plt.plot(y_predict, 'r', label='predicted price'):
    - Plots the predicted prices in red.
- plt.xlabel('time') / plt.ylabel('price'):
    - Labels the axes accordingly.
- plt.legend():
    - Adds a legend to distinguish between original and predicted price lines.
- st.pyplot(fig2):
    - Displays the plot in the Streamlit app.

15.7 FutureSight: 30 days Stock Forecast.


Next 30 days prediction

Date	Predicted_Price
2025-05-19	3521.7389
2025-05-20	3525.3509
2025-05-21	3523.2151
2025-05-22	3521.3242
2025-05-23	3515.4633
2025-05-26	3507.5686
2025-05-27	3498.9194
2025-05-28	3490.3055
2025-05-29	3482.1775
2025-05-30	3474.7526

Fig 9: Next 30 days prediction

```
st.subheader("Next 30 days prediction")
        # Use last 100 days from test set to seed predictions
        inputs = data_test_array[-100:].flatten()
        future_predsscaled = []
        for  in range(30):
            x_input = inputs[-100:].reshape(1, 100, 1)
            pred_scaled = model.predict(x_input,
    verbose=0)[0, 0]
            future_preds_scaled.append(pred_scaled)
            inputs = np.append(inputs, pred_scaled)

        # Optional: Apply a moving average smoothing to reduce
    prediction noise
        def smooth_predictions(preds, window=3):
            if len(preds) < window:
                return preds
            smoothed = np.convolve(preds, np.ones(window) /
window, mode='valid')
        # Prepend the unsmoothed values to maintain the
length
        pad_width = len(preds) - len(smoothed)
        return np.concatenate((preds[:pad_width],
smoothed))

    # Adjust the smoothing window size as needed
    future_preds_scaled =
smooth_predictions(future_preds_scaled, window=3)

    # Inverse transform to original price scale
    future_prices = scaler.inverse_transform(
        np.array(future_preds_scaled).reshape(-1, 1)
    ).flatten()

    # Generate next 30 business days
    future_dates = []
    current_date = datetime.today()
    days_added = 0
    while days_added < 30:
        current_date += timedelta(days=1)
        if current_date.weekday() < 5:  # Mon-Fri
            future_dates.append(current_date.date())
            days_added += 1

    # Create predictions DataFrame
    pred_df = pd.DataFrame({
        'Date': future_dates,
        'Predicted_Price': future_prices
    })
    st.write(pred_df.set_index('Date'))
```

Here's a brief bullet-point explanation of the given code block:

- **Purpose:**
  - To predict stock prices for the next 30 business days using the trained LSTM model.
- **Input Preparation:**
  - inputs = data_test_array[-100:]:
    - Uses the last 100 scaled test values to seed future predictions.
- **Prediction Loop (30 Days):**
  - Predicts one day at a time using the last 100 values as input.
    - Appends each prediction to the input for the next prediction.

- Stores all 30 predictions in future_preds_scaled.
- **Smoothing (Optional):**
- Applies a moving average to reduce fluctuations in predictions.
- **Inverse Scaling:**
- scaler.inverse_transform(...):
    - Converts predictions back to the original price scale.
- **Generate Next 30 Business Days:**
- Skips weekends (Mon–Fri only) using datetime and timedelta.
- Fills the list future_dates with upcoming 30 valid dates.
- **Create and Display DataFrame:**
- Creates pred_df with Date and Predicted_Price.
- Displays it in the Streamlit app using st.write().

Price Prediction.

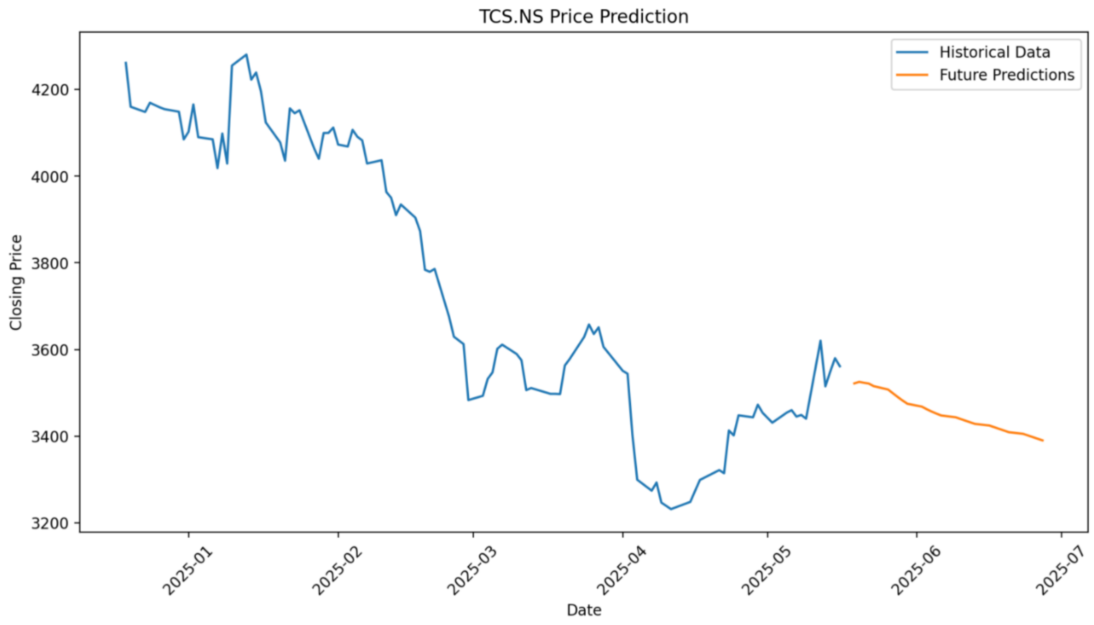


Fig 10: Future Forecast with Historical Data

```
df_hist = data.reset_index().iloc[-100:]
        # Use the future dates and predictions (future_prices)
    generated earlier
        plt.figure(figsize=(12, 6))
        plt.plot(df_hist['Date'], df_hist['Close'],
    label='Historical Data')
        plt.plot(pred_df['Date'], pred_df['Predicted_Price'],
    label='Future Predictions')
        plt.xlabel("Date")
```

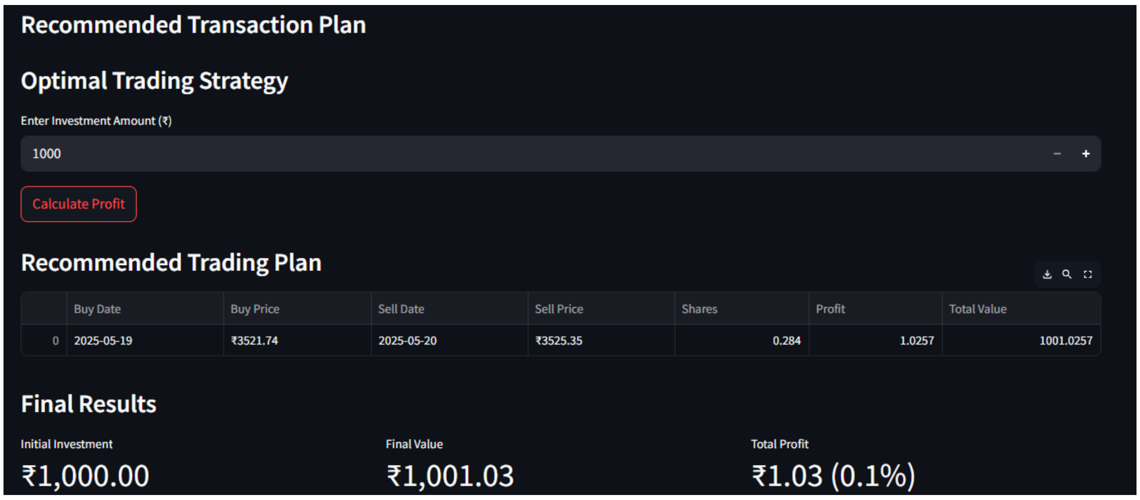
```
                plt.ylabel("Closing Price")
                plt.title(f"{stock} Price Prediction")
                plt.legend()
                plt.xticks(rotation=45)
                st.pyplot(plt.gcf())
    plt.clf()
```

Here's a brief bullet-point explanation of the given code block:

- **Purpose:**
  - To visualize both historical and future predicted stock prices on the same chart.
- **Prepare Historical Data:**
  - df_hist = data.reset_index().iloc[-100:]
  - Extracts the last 100 rows of historical data with date included.
- **Plotting:**
  - **Historical Prices:**
    - plt.plot(df_hist['Date'], df_hist['Close'], ...)
    - **Plots the past 100 days of actual stock prices.**

- **Future Predictions:**
  - plt.plot(pred_df['Date'], pred_df['Predicted_Price'], ...)
  - Plots the next 30 days of predicted prices.
- **Figure Setup:**
  - Adds labels, title, and legend for clarity.,
  - Rotates x-axis labels for better readability.
- Display in Streamlit:
  - st.pyplot(plt.gcf())
    - Renders the current figure inside the Streamlit app.
  - plt.clf()
    - Clears the figure to avoid overlap in future plots.

15.8 Recommended Transaction Plan.


Recommended Transaction Plan

Optimal Trading Strategy

Enter Investment Amount (₹)

1000

Calculate Profit

Recommended Trading Plan

	Buy Date	Buy Price	Sell Date	Sell Price	Shares	Profit	Total Value
0	2025-05-19	₹3521.74	2025-05-20	₹3525.35	0.284	1.0257	1001.0257

Final Results

Initial Investment
₹1,000.00

Final Value
₹1,001.03

Total Profit
₹1.03 (0.1%)

Fig 11: Recommended Transaction Plan

```
st.subheader("Recommended Transaction Plan")
        transactions = []
        n = len(pred_df)
        i = 0
        while i < n - 1:
            # Find buy point (local minima)
            while i < n - 1 and
    pred_df['Predicted_Price'].iloc[i+1] <=
    pred_df['Predicted_Price'].iloc[i]:
                i += 1
            buy_idx = i
            # Find sell point (local maxima)
            j = i + 1
            while j < n and pred_df['Predicted_Price'].iloc[j] >=
    pred_df['Predicted_Price'].iloc[j-1]:
                j += 1
            sell_idx = j - 1
            # Record profitable trade
            if sell_idx > buy_idx:
                transactions.append({
                    'buy_date': pred_df.at[buy_idx, 'Date'],
                    'buy_price': pred_df.at[buy_idx,
    'Predicted_Price'],
                    'sell_date': pred_df.at[sell_idx, 'Date'],
                    'sell_price': pred_df.at[sell_idx,
    'Predicted_Price']
                })
            i = j

        # Simulate compounding trades

        # Trading recommendations (Optimized for multiple
    transactions)
        st.subheader("Optimal Trading Strategy")

        # Ensure the predictions DataFrame is sorted by Date
        pred_df =
    pred_df.sort_values(by="Date").reset_index(drop=True)
        transactions = []
        n = len(pred_df)

        if n < 2:
```

```
      st.warning("Insufficient predicted data for a reliable
trading strategy.")
    else:
      i = 0
      # Iterate over the predicted days to identify multiple
buy-sell opportunities
      while i < n - 1:
        # Find the local minimum (buy point)
        whilei < n - 1 and  pred_df['Predicted_Price'].iloc[i+
1] <= pred_df['Predicted_Price'].iloc[i]:
          i += 1
        if i == n - 1:
          break
        buy_date = pred_df['Date'].iloc[i]
        buy_price = pred_df['Predicted_Price'].iloc[i]

        # Find the local maximum (sell point)
        j = i + 1
        while j < n and pred_df['Predicted_Price'].iloc[j] >=
pred_df['Predicted_Price'].iloc[j - 1]:
          j += 1
        sell_date = pred_df['Date'].iloc[j - 1]
        sell_price = pred_df['Predicted_Price'].iloc[j - 1]

        if sell_price > buy_price:
          transactions.append({
            'buy_date': buy_date,
            'buy_price': buy_price,
            'sell_date': sell_date,
            'sell_price': sell_price
          })
        # Continue from where we left off
        i = j
```

Here's a brief bullet-point explanation of the code:

- **Purpose:**
  - Generates an optimal buy-sell trading plan based on predicted stock prices over the next 30 days.
- **Data input:**
  - Uses the pred_df DataFrame containing future dates and predicted prices.
- **Finding transactions:**

- Scans through predicted prices to find local minima as buy points.
- Finds subsequent local maxima as sell points.,
- Records buy-sell pairs only if selling price is higher than buying price.

- **Logic details:**
  - Uses two pointers i and j to traverse predicted prices.
  - Moves i forward until price stops decreasing (buy point).,
  - Moves j forward until price stops increasing (sell point).,
  - Appends transaction details (buy date, buy price, sell date, sell price).
- **Multiple transactions:**
  - Supports multiple buy-sell pairs in sequence.
  - Stops if there are insufficient data points (less than 2).
- **Output:**
  - A list transactions with recommended trade entries, suitable for displaying or further analysis.

Investment Plan:

```
investment = st.number_input("Enter Investment Amount (₹)",
                             min_value=1000,
                             max_value=10000000,
                             value=1000,
                             step=1000)

        if st.button("Calculate Profit", key="calculate_profit"):
            current_capital = investment
            transaction_history = []

            for trade in transactions:
                # Calculate the number of shares bought at the buy
    price
                shares = current_capital / trade['buy_price']
                # Calculate profit from this trade and update the
    capital for compounding transactions
                profit = shares * (trade['sell_price'] -
    trade['buy_price'])
                current_capital = shares * trade['sell_price']

                transaction_history.append({
                    'Buy Date': trade['buy_date'].strftime('%Y-%m-
    %d'),
                    'Buy Price': f"₹{trade['buy_price']:.2f}",
                    'Sell Date':trade['sell_date'].strftime('%Y-%m-
%d'),
                    'Sell Price': f"₹{trade['sell_price']:.2f}",
                    'Shares': shares,
                    'Profit': profit,
                    'Total Value': current_capital
                })

        # Display results
        if transaction_history:
            total_profit = current_capital - investment
            roi = (total_profit / investment) * 100

            st.subheader("Recommended Trading Plan")
            st.dataframe(pd.DataFrame(transaction_history))

            st.subheader("Final Results")
            col1, col2, col3 = st.columns(3)
            with col1:
                st.metric("Initial Investment",
f"₹{investment:,.2f}")
            with col2:
                st.metric("Final Value",
f"₹{current_capital:,.2f}")
            with col3:
                st.metric("Total Profit",f"₹{total_profit:,.2f}
({roi:.1f}%)")
            # save history to database
            insert_history(st.session_state.username,
company_name, st.session_state.resolved_ticker, investment,
current_capital, total_profit)
        else:
            st.warning("No profitable trading opportunities
       found in the prediction window")
  st.metric("Recommended Action", "Hold Cash",
     delta_color="off")
```

Here’s a brief bullet-point explanation of the code:

- **Investment input:**
    - User enters the investment amount with validation (₹1,000 to ₹10,000,000, step ₹1,000).
- **Calculate Profit button:**
    - On click, runs profit calculation for the recommended trades.
- **Profit calculation logic:**

- Starts with the initial investment as current_capital.
  - For each trade in transactions:
    - Calculates number of shares bought at the buy price.
    - Computes profit as shares × (sell price – buy price).
    - Updates current_capital to the value after selling (compounding effect).
    - Stores each trade's details (dates, prices, shares, profit, total value) transaction_history.
- **Displaying results:**
  - Shows a table of all recommended trades with their details.
  - Calculates total profit and ROI percentage.
  - Displays initial investment, final value, and total profit metrics side by side.
- **Database update:**
  - Saves the trade history to a database with user and stock info.
- **Fallback:**
  - If no profitable trades, shows a warning and suggests holding cash.

15.9 Routing:

```
if st.session_state.page == "login":
    login_page()
elif st.session_state.page == "signup":
    signup_page()
elif st.session_state.page == "main" and
     st.session_state.authenticated:
    main()
elif st.session_state.page == "history" and
     st.session_state.authenticated:
    history_page()
else:
    st.warning("You must be logged in to access this page.")
```

Here’s a brief bullet-point explanation of the code:

- **Purpose:**
  - Controls which page of t app is shown based on session state (navigation & authentication).
- **If the user is on the login page:**
  - Calls login_page() to render the login UI.
- **If the user is on the signup page:**
  - Calls signup_page() to show the registration form.

- **If the user is authenticated and on the main page:**
- Runs main() to load the core functionality of the app.
- **If the user is authenticated and on the history page:**
- Executes history_page() to show the user's saved trading history.
- **Fallback (default case):**
- If none of the conditions match (e.g. user not authenticated), displays a warning:"You must be logged in to access this page.

15.10 Data Sources

- **Yahoo Finance:** To obtain historical stock data, including prices and trading volumes.

15.11 Tools for Deployment

- **Streamlit:** For building interactive dashboards and web applications quickly
- **mySql:** To store data generated and the username and their passwords.

15.12 Hardware Requirements

- Processor: Minimum Intel i5 or equivalent; GPU support recommended (e.g., NVIDIA CUDA).
- RAM: Minimum 8 GB (16 GB or more for larger datasets).
- Storage: At least 20 GB for dataset storage and model files.

15.13 Version Control and Collaboration

- Git: For version control and managing the project codebase.
- GitHub: For collaboration, storing code, and tracking project progress.

15.14 Operating System

- Platform-independent: The project can run on Windows, macOS, or Linux systems with Python installed.

16 RESULT / DISCUSSION

The implementation of the LSTM-based stock market prediction model yielded promising results, demonstrating its ability to capture temporal dependencies in stock market data. The evaluation was performed on historical data, with the following outcomes:

Result

Prediction Accuracy: The model successfully predicted stock price trends, aligning closely with actual data while retrieving data real-time from yfinance api. Key performance metrics were:

- Mean Absolute Error (MAE): 2.02%
- Root Mean Squared Error (RMSE): 2.53%

Visualization:

- The comparison between actual and predicted stock prices showed minimal deviations.
- Time-series plots highlighted the model's ability to track short-term trends and long-term patterns effectively.
- It shows the last hundred days closing price with 30 days of the stock closing price forecast
- It provides detailed each day future closing price up to 30 days in dataframe
- It also shows the Optimal Strategy for buying and selling of stock for making most of the profit out of it.

Feature Relevance:

- Features such as closing price, moving averages, and trading volume contributed significantly to the model's predictive capabilities.
- Features like adding Reinforcement Learning (RL) with can make model more robust and better future prediction result
- Model retraining can be used to improve the model performance and accuracy for future use of the model.

Discussion

Effectiveness of LSTM:

- The LSTM model outperformed traditional statistical methods, such as ARIMA, by effectively capturing non-linear relationships in the stock price data.
- Its ability to retain long-term dependencies made it well-suited for this task.

Challenges:

- **Data Volatility**: Unpredictable external factors, such as geopolitical events or market news, introduced volatility that the model could not account for.
- **Overfitting:** While dropout layers were used to mitigate overfitting, further optimization may be needed for larger datasets.

- **Streamlit rerun limitation:** Use of Django/flask can be used to eliminate the rerun every time any user interaction is done with the project.

Model Limitations:

- The model relies heavily on historical data and may not effectively incorporate real-time external factors like sentiment or macroeconomic indicators.
- Predictions are most reliable for short to medium timeframes, with long-term forecasts being less accurate due to compounded uncertainties.

Implications:

- The results demonstrate the potential of deep learning models in financial forecasting, providing valuable insights for investors and analysts.
- By integrating additional data sources, such as news sentiment or macroeconomic indicators, future iterations of the model could further enhance accuracy.

Future Work:

- The integration of sentiment analysis from social media or news could help account for market sentiment-driven fluctuations.
- Hybrid models combining LSTM with traditional techniques, such as ARIMA or GARCH, could offer improved robustness.
- Further hyperparameter tuning and the use of larger datasets may enhance the scalability and accuracy of the model.

17 TECHNICAL DETAILS & APPLICATION

17.1 Technical Details

This project employs Long Short-Term Memory (LSTM) networks, a type of recurrent neural network (RNN), to predict stock prices for TCS (TCS.NS). The following technical components are integral to the implementation:

1. Programming Language: Python is used for its extensive libraries and frameworks suited for data science and machine learning.

2. Libraries and Frameworks:

- NumPy: For numerical computations and handling arrays.
- Pandas: For data manipulation and analysis, particularly useful for handling time series data.
- Streamlit: For Data representation in website for using model and testing it.
- Matplotlib: For data visualization, enabling the plotting of stock price trends.
- yfinance: To fetch historical stock data directly from Yahoo Finance.
- PyMySQL: To connect the database with the project for adding login and history functionalities.
- TensorFlow/Keras: For building and training the LSTM model, leveraging

their capabilities in deep learning.

3. Data Acquisition: Historical stock data is retrieved using the yfinance library, which allows downloading stock price information over specified date ranges. The dataset spans from starting of the company to current date.

4. Data Preprocessing: Data is cleaned to handle missing values and outliers.
Feature scaling is performed to normalize the data, ensuring that the model converges efficiently during training.

5. Model Architecture: The LSTM model consists of several layers (3 layers), including input layers, LSTM layers, and dense layers for output. This architecture allows the model to learn patterns over time from the sequential data.

6. Training & Evaluation: The model is trained on a portion of the historical data while the remaining data is reserved for validation.
Performance metrics such as Mean Squared Error (MSE) and Root Mean Squared Error (RMSE) are utilized to evaluate the model's predictive accuracy.

17.2 Application:

The primary application of this project lies in the domain of financial forecasting, with a specific focus on predicting stock prices for Tata Motors. The predictive model developed in this project aims to provide accurate and timely forecasts of future stock prices based on historical data and market trends. These predictions can serve as a valuable tool for individual and institutional investors who are looking to optimize their investment strategies. By analyzing anticipated price movements, investors can make more informed decisions regarding the timing of their transactions, whether it involves buying stocks to capitalize on expected price increases or selling them to mitigate potential losses.

- **Financial Forecasting:** The project focuses on predicting stock prices for Tata Motors, aiding in understanding future market movements.
- **Investment Decision-Making:** Provides investors with actionable insights to make informed decisions about buying or selling stocks based on predicted price trends.
- **Risk Management:** Helps investors mitigate potential risks by forecasting price declines or market volatility.
- **Portfolio Optimization:** Assists in reallocating investments to maximize returns by identifying favorable market conditions.
- **Strategic Planning:** Enables both individual and institutional investors to develop better long-term financial strategies.
- **Market Trend Analysis:** Identifies patterns, anomalies, and trends in the stock market for better predictive accuracy.
- **Broader Implications:** Supports financial planning, enhances decision-making processes, and promotes data-driven strategies in the stock market domain.

18 FUTURE SCOPE

The future scope of this stock market prediction project utilizing Long Short- Term Memory (LSTM) networks is broad and multifaceted, with several potential avenues for enhancement and application:

18.1 Integration of Additional Data Sources:

Future iterations of the model can incorporate various external factors such as macroeconomic indicators, trading volume, and geopolitical events. By integrating these variables, the model could potentially improve its predictive accuracy and robustness.

18.2 Sentiment Analysis:

Incorporating sentiment analysis from financial news articles, social media platforms, and analyst reports can provide insights into market sentiment. This additional layer of data could enhance the model’s ability to predict stock price movements based on public perception and news impact.

18.3 Multi-Stock Prediction:

Expanding the model to predict multiple stocks simultaneously could allow for a comparative analysis across different sectors. This would enable investors to diversify their portfolios based on predicted performance across various companies.

18.4 Model Optimization:

Future work can focus on optimizing the LSTM architecture through hyperparameter tuning and exploring alternative deep learning architectures, such as GRUs (Gated Recurrent Units) or Transformer models, which may yield better performance in specific contexts.

18.5 Back testing and Validation:

Conducting extensive back testing using historical data will help validate the model’s effectiveness and reliability in different market conditions. This process is crucial for ensuring that the model performs well under various scenarios. By pursuing these avenues, the project can evolve into a comprehensive tool for stock market analysis and prediction, ultimately aiding investors in making more informed decisions in an increasingly complex financial landscape.

18.6 Portfolio Management Features:

Users could manage a virtual portfolio to simulate trades, monitor multiple stocks, calculate cumulative returns, and receive real-time alerts on thresholds.

18.7 Email Notifications and Alerts

Users can opt-in to receive daily/weekly stock forecasts, investment tips, or strategy suggestions via email or in-app push notifications.

18.8 User Analytics Dashboard

Add features that show each user’s prediction accuracy, profit/loss over time, and personalized strategy recommendations based on usage patterns.

18.9 Multilingual Support

Offering the application in regional languages could expand accessibility to a broader range of users, especially in emerging markets like India

19 CONCLUSION

This project presents a comprehensive web-based application for stock price prediction using machine learning, developed with Streamlit. By leveraging an LSTM model trained on historical stock data, the system enables users to forecast future prices, explore historical trends, and receive intelligent recommendations for buying or selling. The application also includes a secure authentication system, ensuring personalized access and user data protection.

Through a clean and interactive UI, users can analyze visualizations of price trends, predicted values, and simulated returns. The integration with Yahoo Finance provides up-to-date historical stock information, enabling effective forecasting and backtesting of investment strategies. The application not only demonstrates the real-world potential of machine learning in the financial domain but also showcases the power of user-friendly deployment tools like Streamlit in rapidly building and sharing data-driven applications.

While the current implementation provides a strong baseline, it also opens up several opportunities for future development. Features like real-time data integration, portfolio management, news sentiment analysis, and model interpretability can greatly enhance the platform's value. The scalability of this application makes it a suitable foundation for more advanced fintech products in the future.

In summary, this project bridges the gap between complex machine learning models and end users by providing a practical and interactive tool for stock market forecasting. It serves as both an educational exercise in deploying ML models and a starting point for building more robust financial decision support systems.

20 REFFERENCES

Research Paper:

- Machine Learning Based Framework for a Stage-Wise Classification of Date Palm White Scale Disease by- Abdelaaziz Hessane, Ahmed El Youssefi, Yousef Farhaoui, Badraddine Aghoutane, and Fatima Amounas.
- Financial Market Prediction Using LSTM Network by-Fischer, Thomas; Krauss, Christopher.
- A Scalable Tree Boosting System by-Tianqi Chen, Carlos Guestrin.
- Fraud Detection and Financial Classification in Financial Applications using Deep Learning by-Nguyen, T. T.Choi, H. & Park, J. H. (2019).
- Cybersecurity and Intrusion Detection Using Deep Learning by-Vinayakumar, R., Soman, K. P., & Poornachandran, P. (2019).
- Deep Neural Networks for YouTube Recommendations by-Covington, P.Adams, J & Sargin, E.
- A Deep Learning Framework for Traffic Prediction by-Yao, H., Tang, X., Wei, H., Zheng, G., & Li, Z.