**COLLEGE OF APPLIED BUSINESS AND TECHNOLOGY**

**Chabahil, Gangahity, Kathmandu**



**Stock Price Prediction**

**A Project Report**

**Submitted by:**

Alish Dahal (102)

Bibek Mainali (107)

Nischal Shrestha (119)

Sayastha Karki (127)

College Of Applied Business and Technology

B.Sc.CSIT 4th Semester

**Submitted to:**

College of Applied Business and Technology

Department of Science and Technology

**Table of Contents**

[ACKNOWLEDGEMENT ........................................................................................................](#_ACKNOWLEDGEMENT) I

[ABSTRACT .............................................................................................................................](#_ABSTRACT) II

[LIST OF ABBREVIATIONS ..................................................................................................III](#_LIST_OF_ABBREVIATIONS)

[CHAPTER 1: INTRODUCTION.......................................................................................... 1](#_INTRODUCTION)-3

Introduction ............................................................................................................... 1

Report Organization ...................................................................................................... 1

Problem Statement ........................................................................................................ 1

Objective ....................................................................................................................... 1

[CHAPTER 2: REQUIREMENT ANALYSIS AND FEASIBILITY STUDY .......................... 3](#_REQUIREMENT_ANALYSIS_AND)

Literature Review .......................................................................................................... 3

Requirement Analysis ................................................................................................... 3

Functional Requirement .................................................................................... 3 Non-Functional Requirement............................................................................ 4

Feasibility Study ........................................................................................................ 5-6

Technical Feasibility ......................................................................................... 5

Operational Feasibility ...................................................................................... 5

Economic Feasibility ......................................................................................... 6

[Chapter 3: SYSTEM DESIGN..................................................................................................](#_SYSTEM_DESIGN). 6

Methodology.................................................................................................................. 8

Data Collection.................................................................................................11

LSTM ………………………………………………………………………...12

Random Forest………………………………………………………………..14

XGBoost……………………………………………………………………...15

LightGBM……………………………………………………………………16

[Chapter 4: IMPLEMENTATION AND TESTING..................................................................](#_IMPLEMENTATION_AND_TESTING) 17

Implementation............................................................................................................ 17

Tools Used....................................................................................................... 18

Evaluation and Testing...... ....................................................................................... 19

[Chapter 5: CONCLUSION AND RECOMMENDATION](#_CONCLUSION_AND_RECOMMENDATION)...................................................... 23

[Conclusion ..................................................................................................................](#_Conclusion) 23

Future Enhancement …............................................................................................... 23

[REFERENCES](#_REFERENCES) ....................................................................................................................... 24

[APPENDIX ..........................................](#_APPENDIX)................................................................................... 25

## ACKNOWLEDGEMENT

It gives us immense pleasure to express our deepest sense of gratitude and sincere thanks to our highly respected and esteemed guide Mr. Tekendra Nath Yogi, for his valuable guidance, encouragement and help for completing this work. His useful suggestions for this whole work and co-operative behavior are sincerely acknowledged.

At the end we would like to express our sincere thanks to all our friends and others who helped us directly or indirectly during this project work.

Alish Dahal (102)

Bibek Mainali (107)

Nischal Shrestha (119)

Sayastha Karki (127)

## ABSTRACT

Stock Price Prediction Using Machine Learning: An Overview

Stock price prediction using machine learning is a complex yet promising endeavor in the realm of financial markets. The volatile and dynamic nature of stock markets, influenced by various factors such as economic indicators, geopolitical events, and company performance, makes accurate prediction challenging. However, the application of machine learning techniques, particularly Long Short Term Memory (LSTM) networks, has shown potential in forecasting stock prices to a certain extent.

**Keywords:**

machine learning, Python, Collab

## LIST OF ABBREVIATIONS

ANN: Artificial Neural Network

LSTM: Long Short Term Memory

RNN: Recurrent Neural Network

SVM: Support Vector Machine

## INTRODUCTION

**Introduction**

Stock price prediction using machine learning is a fascinating field that leverages advanced computational techniques to forecast the future values of stocks traded on the financial markets. It's a complex and challenging task due to the inherent volatility and uncertainty in stock markets, influenced by various factors such as economic indicators, market sentiment, geopolitical events, and company performance. Machine learning models, particularly Long Short Term Memory (LSTM) networks, offer a promising approach to analyze historical stock data and make informed predictions about future stock prices. We will use LSTM, SVM, Random Forest Model, and Gradient Boosting methods(XGBoost and LightGBM) to train on the historical stock price data to predict future stock closing prices.

**Objectives**

The objectives of this project are to:

* Financial Gain
* Risk Management
* Increased Accuracy

**Report Organization**

**The report is organized into 6 chapters, each of which discusses a different aspect of the project, from the problem statement, objectives, and scope to the recommendations for future work.**

**Chapter 1: Introduction**

* **This chapter introduces the project, including the problem statement, objectives, and scope.**
* **It also provides a brief overview of the state-of-the-art in stock price prediction sign classification.**

**Chapter 2: Related Work**

* **This chapter discusses the different approaches that have been used for stock price prediction, including the use of hand-crafted features and deep learning methods.**
* **It also discusses the challenges that still need to be addressed in order to improve the accuracy of stock price prediction systems.**

**Chapter 3: Methodology**

* **This chapter describes the methodology that was used for this project, including the data, RNN architecture, and training and evaluation procedures.**
* **It also discusses the challenges that were encountered during the implementation of the project.**

**Chapter 4: Results**

* **This chapter presents the results of the project, including the accuracy of the RNN model on a custom dataset of stock price predictions.**
* **It also discusses the implications of the results for the future of stock price prediction systems.**

**Chapter 5: Discussion**

* **This chapter discusses the limitations of the project and the challenges that still need to be addressed in order to improve the accuracy of stock price prediction systems.**
* **It also discusses the potential applications of the project in real-world stock price prediction recognition systems.**

**Chapter 6: Conclusion**

* **This chapter summarizes the main findings of the project and provides recommendations for future work.**

## REQUIREMENT ANALYSIS AND FEASIBILITY STUDY

**Literature Review**

Stock price prediction refers to the prediction of the trading operations at a certain time in the future. It is based on the historical and real data of the stock market according to a certain forecasting model. This prediction plays an important and positive role in improving the efficiency of the trading market and giving play to market signals. Accurate stock price forecast can help investors adjust their trading strategies in time, and effectively avoid investment risks, so as to obtain higher returns. Price prediction has long appeared in all kinds of trading markets. However, due to the influence of many factors, including not only the internal change rules of the stock market, but also the sudden impact of the external market, the prediction results of some existing stock price prediction models are not perfect. Using the existing technology and the improvement of the existing algorithm, the prediction result can be closer to the actual situation. Therefore, we need to further improve the algorithm and model, make use of the historical data given, and extract valuable data information to achieve more accurate stock price prediction.

# **Problem Statement**

Investors face significant risk when investing in the stock market.

While there are many success stories of investors making huge profits in the stock market, there are also stories of investors losing everything due to poor decisions. The strategy that investors take to the stock market, rather than the stock market itself, is the issue. Many investors are unclear about whether it is feasible to buy a particular stock or what is required to be successful in the market.

**Objective of study**

### To use machine learning methods to predict stock market using historical data

**Non-Functional Requirements**

The stock price prediction system must meet the following non-functional requirements:

* Accuracy: The system must be able to accurately identify stock price predictions, with a high confidence score.
* Speed: The system must be able to classify stock price predictions in real time.
* Robustness: The system must be able to handle variations in market crash and market increase.
* Scalability: The system must be able to scale to handle a large number of stock price predictions.

**Data Requirements**

The stock price prediction system will require a dataset of stock price predictions. The dataset should include datasets of stock price predictions from a variety of time, days and years. The dataset should also include the ground truth labels for such data which indicate the type of stock price prediction.

**Hardware Requirements**

The stock price prediction system will require a computer with a GPU. The GPU will be used to accelerate the train.

**Software Requirements**

The stock price prediction system will require the following software:

* Pandas
* numpy
* Svm
* Random forest model
* Gradient Boosting Methods

**Deployment Requirements**

The stock price prediction system can be deployed on a variety of platforms, including:

* On-premises servers
* Cloud-based platforms

**Testing Requirements**

The stock price prediction system will be tested using the following methods:

* Unit testing
* Integration testing
* System testing
* User acceptance testing

**Maintenance Requirements**

The stock price prediction system will require regular maintenance to ensure that it continues to function properly. The maintenance tasks will include:

* Updating the dataset
* Troubleshooting any problems that arise

**Feasibility Study**

This feasibility study will assess the feasibility of developing a stock price prediction system using historical price data The study will consider the following factors:

**Technical Feasibility**

There are a number of publicly available datasets of stock price predictions that can be used for training and testing.

The project uses a number of Python libraries, including NumPy, Pandas, Pickle, Matplotlib, sklearn, XGboost and Light GBM and Keras. These libraries provide a variety of features that are essential for the project, such as data manipulation, data visualization, and model training.

**Operational Feasibility**

The operational feasibility of the project is also high. The RNN-based stock price prediction system can be deployed on a variety of platforms, including embedded systems, cloud-based systems, and mobile devices. Additionally, the system can be easily integrated with existing stock management systems.

**Economic Feasibility**

The economic feasibility of the project is also good. The development costs of the project will depend on the complexity of the system and the hardware platform that is used and the size of the data sets. However, in general, the development of such a system can be relatively cost-effective.

In addition to the three sections mentioned above, here are some other factors that should be considered when assessing the feasibility of the project:

* The availability of funding
* The regulatory environment
* The level of public acceptance

**Methodology**

First, we need to load the stock data from a CSV file. We will be using the 'NEPSE\_10years’ file from nepse website from 2013 to 2023. The data will be stored in a Pandas Data Frame, and we will extract the 'Close' column for further processing.

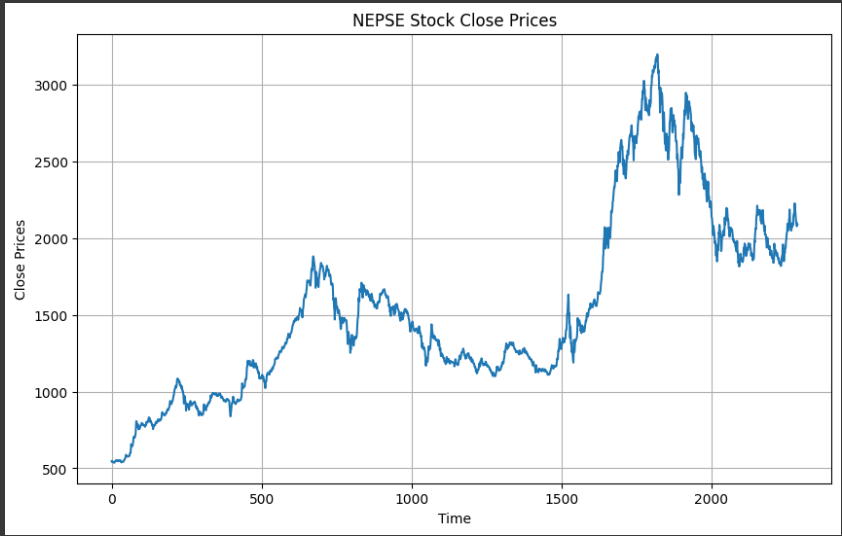


Figure 1 Nepse stock closing price

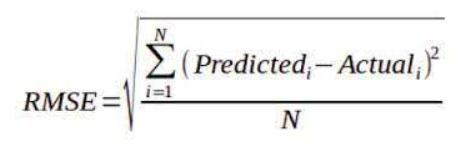
We will move onto some data pre-processing steps.  This is necessary because it is essential for improving data quality and handling missing values and outliers.It involves normalization to bring features to a common scale, pre-processing enhances model performance and ensures data compatibility with analysis techniques.  Here we are reshaping the data to have a single feature and normalize it to the range [0, 1]. This normalization step is essential to ensure that all features have the same scale and to improve the model's convergence. Before Data processing Numpy is imported in Python for its efficient numerical computing capabilities. It provides N-dimensional arrays for handling large datasets and a wide range of mathematical functions for array operation.

We will split the data into training and testing sets. Here, we assign 80% of the data for training and the remaining 20% for testing. The training data will be used to train our models, while the testing data will be used to evaluate their performance. Train data is used to learn patterns and features and test data is used to evaluate models performance on unseen data. This ensures the models' generalization capability.

LSTM Model: We will start with a Long Short-Term Memory (LSTM) model, which is a type of recurrent neural network (RNN) suitable for sequential data like stock prices. LSTM models are known for their ability to capture temporal dependencies and make accurate predictions. In this code TensorFlow is an open-source deep learning library developed by Google. It allows you to build and train various machine learning and deep learning models efficiently using neural networks.

Sequential is a class in tensor flow’s Keras API that allows you to create a linear stack of layers in a neural network. Dense is a layer type in TensorFlow used for fully connected layers in neural networks, where every neuron is connected to every neuron in the previous and next layer. Dropout is a regularization technique used to prevent overfitting in neural networks. It randomly sets a fraction of input units to zero during the training, which helps in reducing the co-adaptation of neurons and improves the generalization of the model. Adam  is an optimization algorithm used for training neural networks. It is an adaptive learning rate optimization algorithm that combines the benefits of AdaGrad and RMSprop. It dynamically adjusts the learning rates for each parameter, making it well suited for a wide range of deep learning tasks. We define a function, create\_lstm\_model, to create the LSTM model with the given hyperparameters. The function takes the number of LSTM units, activation function, and learning rate as inputs. The model consists of an LSTM layer followed by a dense output layer. We use the Adam optimizer and mean squared error loss for training the model.

Next, we define a set of hyperparameters for tuning the LSTM model. We specify different values for the number of LSTM units, activation functions, and learning rates. We will perform a grid search to find the best combination of hyperparameters. We initialize a variable, best\_rmse, with a high value to keep track of the best root mean squared error (RMSE) achieved by the LSTM model. Root Mean Square Error is calculated with the following formula :



We will update this value as we find better models during the grid search. Now, we iterate over all possible combinations of hyperparameters and train the LSTM models. For each combination, we create the LSTM model, train it on the training data, and make predictions on the testing data. We calculate the RMSE between the actual and predicted values and update the best\_rmse and best\_lstm\_model variables if we find a model with a lower RMSE. After finding the best LSTM model, we make predictions on the entire dataset and inverse normalize the predictions to obtain the actual stock prices.

SVM Model: We will now explore Support Vector Machines (SVM). SVM is a popular non-linear regression model that finds the best hyperplane to separate the data points. We initialize an SVM model and define a set of hyperparameters to tune using a grid search. The grid search selects the best hyperparameters based on the negative mean squared error scoring. We fit the SVM model to the data and make predictions on the entire dataset.

Random Forest Model: Next, we explore the Random Forest model, which is an ensemble model consisting of multiple decision trees. Random Forest models are known for their robustness and ability to handle complex relationships in the data. We initialize a Random Forest model and define a set of hyperparameters for tuning. Similar to the SVM model, we use a grid search to find the best combination of hyperparameters based on the negative mean squared error. We fit the Random Forest model to the data and make predictions on the entire dataset.

Gradient Boosting Methods: Lastly, we explore two popular gradient boosting methods, XGBoost and LightGBM. These models are known for their efficiency and excellent performance in many machine learning tasks. We follow a similar approach for both XGBoost and LightGBM models. We initialize the models and define a set of hyperparameters for tuning using a grid search. We fit the models to the data and make predictions on the entire dataset.

To evaluate the models' performance, we calculate the root mean squared error (RMSE) between the actual and predicted stock prices. Lower RMSE values indicate better model performance. We plot the actual and predicted values for each model to visualize their predictions.Finally, we plot the predictions made by the best LSTM model alongside the actual stock prices.

So far, we have  explored various machine-learning models for predicting stock prices. We started with an LSTM model to capture temporal dependencies in the data. Then, we explored other models like Support Vector Machines, Random Forest, XGBoost, and LightGBM. We evaluated their performance using the root mean squared error (RMSE) metric and visualized the predictions. The LSTM model demonstrates superior predictive performance, exhibiting the lowest Root Mean Square Error (RMSE) among all the models. This highlights the strength and effectiveness of the LSTM model in forecasting stock prices accurately. Next, we will do forecast using LSTM Model.

**Data Collection**

The preprocessed dataset was downloaded from Nepal Stock Exchange (NEPSE) website. The data set is from the year 2013 to 2023 over the span of ten years. The dataset was divided into train set and test set. The train set of data were used to train the model and the test set of data were used to test the accuracy of the model. 

Figure 2: - Preprocessed Training Data

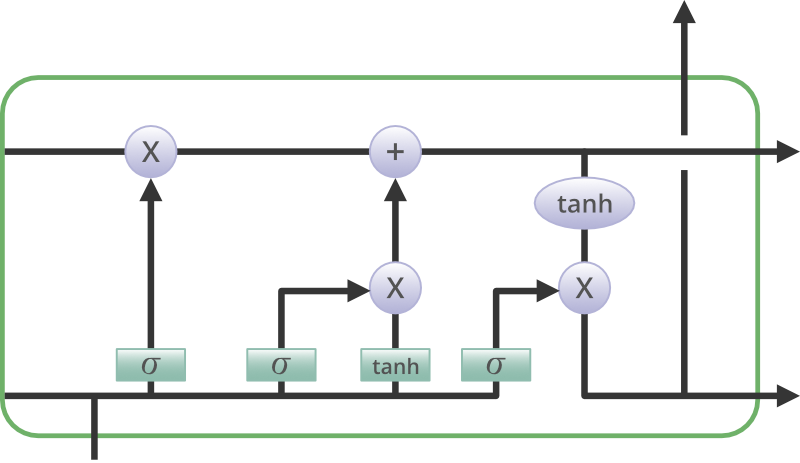
**Long Short Term Memory**

Long Short Term Memory is a kind of recurrent neural network. In RNN output from the last step is fed as input in the current step. LSTM was designed by Hochreiter & Schmidhuber. It tackled the problem of long-term dependencies of RNN in which the RNN cannot predict the word stored in the long-term memory but can give more accurate predictions from the recent information. As the gap length increases RNN does not give an efficient performance. LSTM can by default retain the information for a long period of time. It is used for processing, predicting, and classifying on the basis of time-series data.

Long Short-Term Memory (LSTM) is a type of Recurrent Neural Network (RNN) that is specifically designed to handle sequential data, such as time series, speech, and text. LSTM networks are capable of learning long-term dependencies in sequential data, which makes them well suited for tasks such as language translation, speech recognition, and time series forecasting.

A traditional RNN has a single hidden state that is passed through time, which can make it difficult for the network to learn long-term dependencies. LSTMs address this problem by introducing a memory cell, which is a container that can hold information for an extended period of time. The memory cell is controlled by three gates: the input gate, the forget gate, and the output gate. These gates decide what information to add to, remove from, and output from the memory cell.

The input gate controls what information is added to the memory cell. The forget gate controls what information is removed from the memory cell. And the output gate controls what information is output from the memory cell. This allows LSTM networks to selectively retain or discard information as it flows through the network, which allows them to learn long-term dependencies.

LSTMs can be stacked to create deep LSTM networks, which can learn even more complex patterns in sequential data. LSTMs can also be used in combination with other neural network architectures, such as Convolutional Neural Networks (RNNs) for image and video analysis. LSTM has a chain structure that contains four neural networks and different memory blocks called **cells.**

**Random Forest Algorithm**

Random Forest is a popular machine learning algorithm that belongs to the supervised learning technique. It can be used for both Classification and Regression problems in ML. It is based on the concept of **ensemble learning,** *which is a process of*combining multiple classifiers to solve a complex problem and to improve the performance of the model. Instead of relying on one decision tree, the random forest takes the prediction from each tree and based on the majority votes of predictions, and it predicts the final output.



Fig: Working of Random Forest algorithm.

**XGBoost:**

XGBoost is an implementation of Gradient Boosted decision trees. XGBoost models majorly dominate in many Kaggle Competitions.

In this algorithm, decision trees are created in sequential form. Weights play an important role in XGBoost. Weights are assigned to all the independent variables which are then fed into the decision tree which predicts results. The weight of variables predicted wrong by the tree is increased and these variables are then fed to the second decision tree. These individual classifiers/predictors then ensemble to give a strong and more precise model. It can work on regression, classification, ranking, and user-defined prediction problems.

#### Mathematics behind XgBoost

Before beginning with mathematics about Gradient Boosting, Here’s a simple example of a CART that classifies whether someone will like a hypothetical computer game X. The example of tree is below:

The prediction scores of each individual decision tree then sum up to get  If you look at the example, an important fact is that the two trees try to *complement* each other. Mathematically, we can write our model in the form



where, K is the number of trees, f is the functional space of F, F is the set of possible CARTs. The objective function for the above model is given by:



where, first term is the loss function and the second is the regularization parameter.

**LightGBM**

LightGBM is a gradient-boosting framework based on decision trees to increase the efficiency of the model and reduces memory usage.   
It uses two novel techniques:

* Gradient-based One Side Sampling(GOSS)
* Exclusive Feature Bundling (EFB)

These techniques fulfill the limitations of the histogram-based algorithm that is primarily used in all GBDT (Gradient Boosting Decision Tree) frameworks. The two techniques of GOSS and EFB described below form the characteristics of the LightGBM Algorithm. They comprise together to make the model work efficiently and provide it a cutting edge over other GBDT frameworks.

Different data instances have varied roles in the computation of information gain. The instances with larger gradients(i.e., under-trained instances) will contribute more to the information gain. GOSS keeps those instances with large gradients (e.g., larger than a predefined threshold, or among the top percentiles), and only randomly drops those instances with small gradients to retain the accuracy of information gain estimation. This treatment can lead to a more accurate gain estimation than uniformly random sampling, with the same target sampling rate, especially when the value of information gain has a large range.

High-dimensional data are usually very sparse which provides us the possibility of designing a nearly lossless approach to reduce the number of features. Specifically, in a sparse feature space, many features are mutually exclusive, i.e., they never take nonzero values simultaneously. The exclusive features can be safely bundled into a single feature (called an Exclusive Feature Bundle).  Hence, the complexity of histogram building changes from *O(data × feature)* to *O(data × bundle)*, while *bundle<<feature*. Hence, the speed of the training framework is improved without hurting accuracy.

## IMPLEMENTATION AND TESTING

**Implementation**

A simple GUI was developed in web which has a feature of choose file to upload image for classification. To develop a working system implementation was done in 3 phases. First phase was the implementation of convolutional Neural Network along with the preprocessors of image data. Second phase was the implementation of backend API along with database. Finally, third phase was implementation of frontend i.e. a website for the classification model.

**Tools Used**

**HTML**

HTML was used to design website for classification of signs. HTML gave the structure for the webpages. It was used to format the texts and images of the web pages. Formatting pages, creating hyperlinks and web forms are all done using HTML.

**CSS**

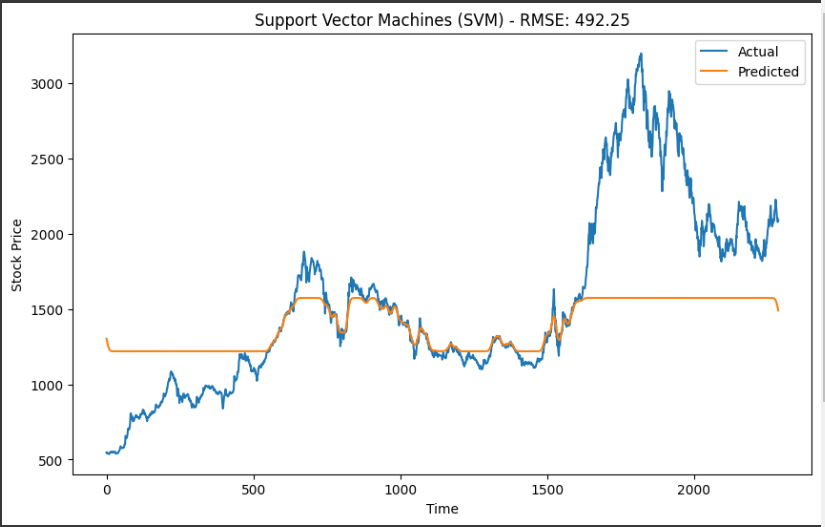
CSS was used to stylize the website. Arranging the images and texts of the webpage in organized manner, decorating HTML elements and managing the elements are all done using CSS.

**JavaScript**

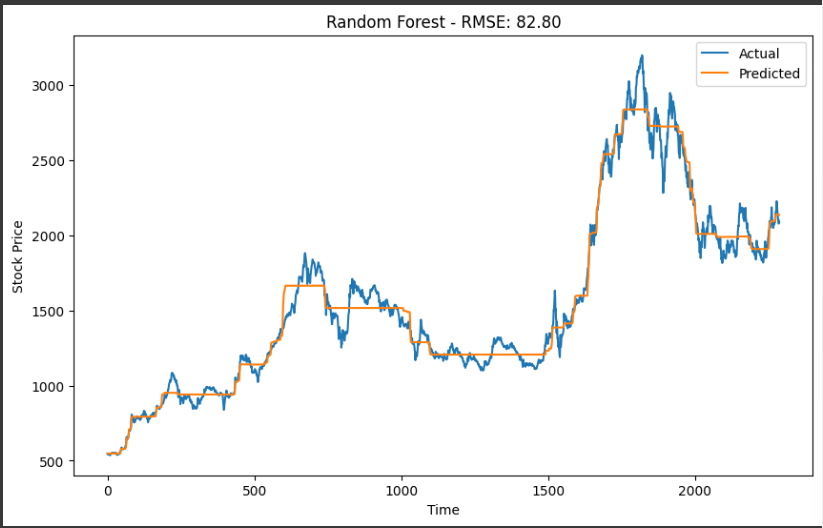
JavaScript was used to request backend for data and display data properly on the website. AJAX principle was followed in order to consume backend API‟s data. Not only it was used to request for data but also it was used to manage active user’s session.

**Evaluation and testing**

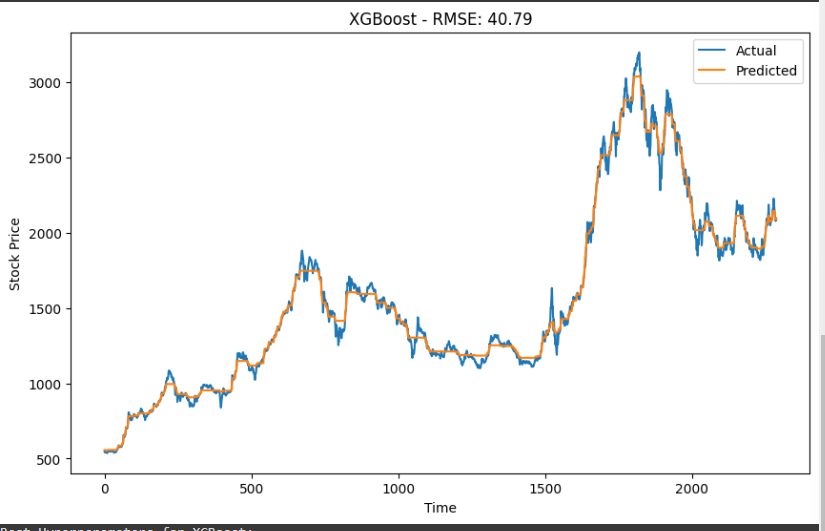
To evaluate the models' performance, we calculate the root mean squared error (RMSE) between the actual and predicted stock prices. Lower RMSE values indicate better model performance. We plot the actual and predicted values for each model to visualize their predictions.

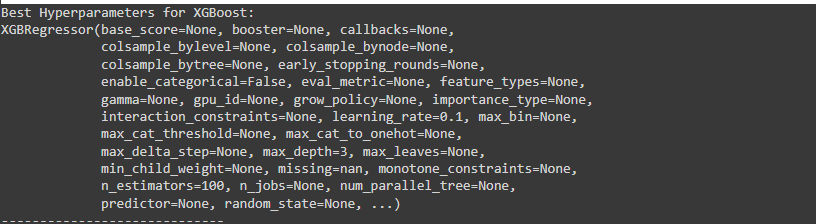


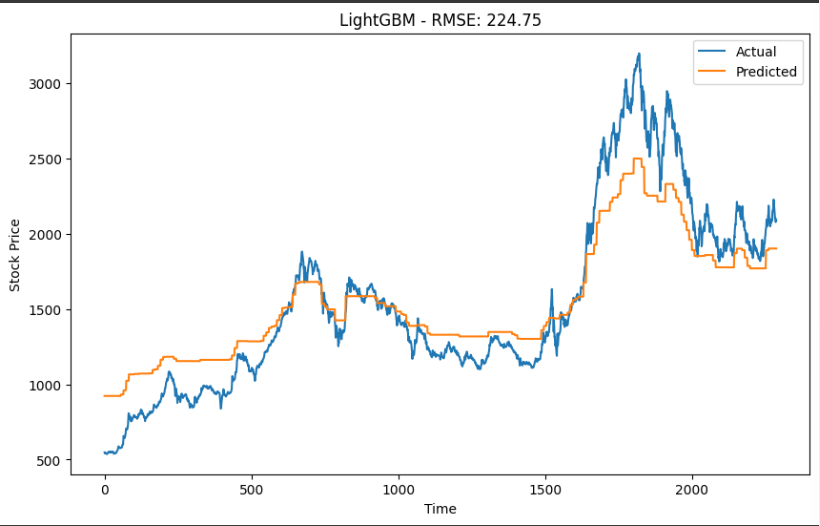








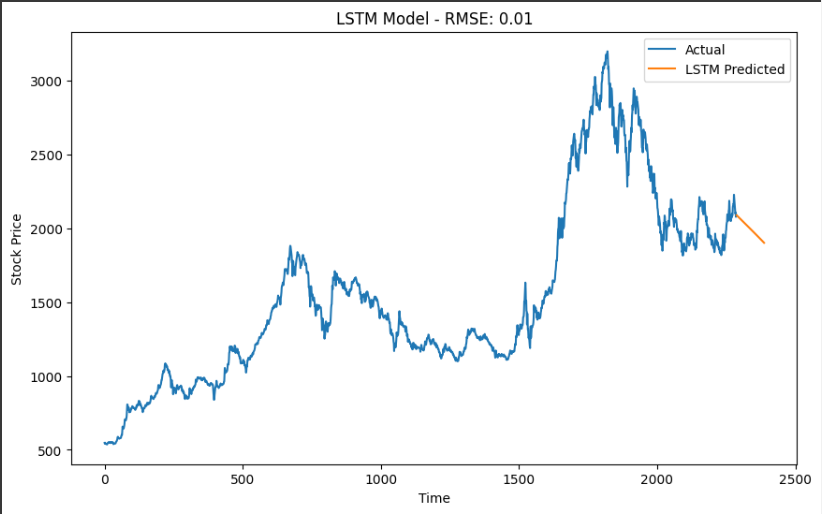








Finally, we plot the predictions made by the best LSTM model alongside the actual stock prices over 100 days.



## CONCLUSION AND RECOMMENDATION

## Conclusion

In this report, development of a system to classify stock price predictions and then into

predefine categories is documented. Historical data can be used to train. The stock price predictions beside the roads are displayed to the user also in low vision. The main goal of our project “Detecting different Stock price predictions from historical data” was achieved by using LSTM, gradient boosting. Using flask web framework of python saved a lot of time required to develop backend which allowed spare time to focus on the non-functional requirement too.

**Future Enhancements**

The system can be further enhanced by adding more feature or adding more data. Adding various sign in the dataset and optimizing algorithms for better efficiency.

## REFERENCES

Sonkavde, G.; Dharrao, D.S.; Bongale, A.M.; Deokate, S.T.; Doreswamy, D.; Bhat, S.K. Forecasting Stock Market Prices Using Machine Learning and Deep Learning Models: A Systematic Review, Performance Analysis and Discussion of Implications. Int. J. Financial Stud. **2023**, 11, 94. <https://doi.org/10.3390/ijfs11030094>

.

Faruque, Md Omar & Strasser, Thomas & Lauss, Georg & Marandi, V. & Forsyth, Paul & Dufour, Christian & Dinavahi, Venkata & Monti, A. & Kotsampopoulos, Panos & Martinez, Juan & Strunz, K. & Saeedifard, Maryam & Wang, Xiaoyu & Shearer, David & Paolone, Mario & Brandl, Ron & Matar, Mahmoud & Davoudi, Ali & Iravani, Reza. (2015). Real-Time Simulation Technologies for Power Systems Design, Testing, and Analysis. IEEE Power and Energy Technology Systems Journal. 2. 63 - 73. 10.1109/JPETS.2015.2427370.

[3] Stock price predictions Preprocessed Dataset: https://nepsealpha.com/nepse-data

## APPENDIX

**Code for model:**

*"""Stock Prediction using Machine Learning.ipynb  
  
Automatically generated by Colaboratory.*import pandas as pd  
import numpy as np  
# Load the stock data  
  
data = pd.read\_csv("nepse\_10 years.csv")  
close\_prices\_NEPSE = data['Close']  
  
# Reverse the order of the data  
close\_prices\_NEPSE\_reverse = close\_prices\_NEPSE.iloc[::-1]  
  
# Reset index to maintain the correct time series order in the plot  
close\_prices\_NEPSE\_reverse.reset\_index(drop=True, inplace=True)  
  
# Plot the line chart  
import matplotlib.pyplot as plt  
plt.figure(figsize=(10, 6))  
plt.plot(close\_prices\_NEPSE\_reverse)  
plt.xlabel('Time')  
plt.ylabel('Close Prices')  
plt.title('NEPSE Stock Close Prices')  
plt.grid(True)  
plt.show()  
  
# Data preprocessing  
data = close\_prices\_NEPSE\_reverse.values.reshape(-1, 1) # Reshape the data  
data\_normalized = data / np.max(data) # Normalize the data  
  
# Split the data into training and testing sets  
train\_size = int(len(data\_normalized) \* 0.8)  
train\_data = data\_normalized[:train\_size]  
test\_data = data\_normalized[train\_size:]  
  
from tensorflow.keras.models import Sequential  
from tensorflow.keras.layers import LSTM, Dense, Dropout  
from tensorflow.keras.optimizers import Adam  
  
# Function to create LSTM model  
def create\_lstm\_model(units, activation, learning\_rate):  
 model = Sequential()  
 model.add(LSTM(units=units, activation=activation, input\_shape=(1, 1)))  
 model.add(Dense(units=1))  
 optimizer = Adam(learning\_rate=learning\_rate)  
 model.compile(optimizer=optimizer, loss='mean\_squared\_error')  
 return model  
  
# Define hyperparameters for tuning  
lstm\_units = [50, 100, 200]  
lstm\_activations = ['relu', 'tanh']  
learning\_rates = [0.001, 0.01, 0.1]  
epochs = 100  
batch\_size = 32  
  
# Perform hyperparameter tuning for LSTM model  
best\_rmse = float('inf')  
best\_lstm\_model = None  
  
from sklearn.metrics import mean\_squared\_error  
  
for units in lstm\_units:  
 for activation in lstm\_activations:  
 for learning\_rate in learning\_rates:  
 # Create and train LSTM model  
 model = create\_lstm\_model(units=units, activation=activation, learning\_rate=learning\_rate)  
 model.fit(train\_data[:-1].reshape(-1, 1, 1), train\_data[1:], epochs=epochs, batch\_size=batch\_size, verbose=0)  
  
 # Predict on test data  
 test\_predictions = model.predict(test\_data[:-1].reshape(-1, 1, 1)).flatten()  
  
 # Calculate RMSE  
 rmse = np.sqrt(mean\_squared\_error(test\_data[1:], test\_predictions))  
  
 # Check if current model has lower RMSE  
 if rmse < best\_rmse:  
 best\_rmse = rmse  
 best\_lstm\_model = model  
  
# Predict on the entire dataset using the best LSTM model  
all\_lstm\_predictions = best\_lstm\_model.predict(data\_normalized[:-1].reshape(-1, 1, 1)).flatten()  
  
# Inverse normalize the LSTM predictions  
all\_lstm\_predictions = all\_lstm\_predictions \* np.max(data)  
  
from sklearn.svm import SVR  
from sklearn.model\_selection import GridSearchCV  
  
# Support Vector Machines (SVM) Model  
svm\_model = SVR()  
  
svm\_params = {  
 'C': [0.1, 1, 10],  
 'gamma': [0.01, 0.1, 1]  
}  
  
svm\_grid\_search = GridSearchCV(svm\_model, svm\_params, scoring='neg\_mean\_squared\_error')  
svm\_grid\_search.fit(np.arange(len(close\_prices\_NEPSE\_reverse)).reshape(-1, 1), close\_prices\_NEPSE\_reverse)  
svm\_best\_model = svm\_grid\_search.best\_estimator\_  
svm\_predictions = svm\_best\_model.predict(np.arange(len(close\_prices\_NEPSE\_reverse)).reshape(-1, 1))  
  
from xgboost import XGBRegressor  
from lightgbm import LGBMRegressor  
from sklearn.ensemble import RandomForestRegressor  
  
# Random Forest Model  
rf\_model = RandomForestRegressor()  
  
rf\_params = {  
 'n\_estimators': [50, 100, 200],  
 'max\_depth': [None, 5, 10]  
}  
  
rf\_grid\_search = GridSearchCV(rf\_model, rf\_params, scoring='neg\_mean\_squared\_error')  
rf\_grid\_search.fit(np.arange(len(close\_prices\_NEPSE\_reverse)).reshape(-1, 1), close\_prices\_NEPSE\_reverse)  
rf\_best\_model = rf\_grid\_search.best\_estimator\_  
rf\_predictions = rf\_best\_model.predict(np.arange(len(close\_prices\_NEPSE\_reverse)).reshape(-1, 1))  
  
# Gradient Boosting Methods (XGBoost)  
xgb\_model = XGBRegressor()  
  
xgb\_params = {  
 'learning\_rate': [0.1, 0.01, 0.001],  
 'max\_depth': [3, 5, 7]  
}  
  
xgb\_grid\_search = GridSearchCV(xgb\_model, xgb\_params, scoring='neg\_mean\_squared\_error')  
xgb\_grid\_search.fit(np.arange(len(close\_prices\_NEPSE\_reverse)).reshape(-1, 1), close\_prices\_NEPSE\_reverse)  
xgb\_best\_model = xgb\_grid\_search.best\_estimator\_  
xgb\_predictions = xgb\_best\_model.predict(np.arange(len(close\_prices\_NEPSE\_reverse)).reshape(-1, 1))  
  
# Gradient Boosting Methods (LightGBM)  
lgbm\_model = LGBMRegressor()  
  
lgbm\_params = {  
 'learning\_rate': [0.1, 0.01, 0.001],  
 'max\_depth': [3, 5, 7]  
}  
  
lgbm\_grid\_search = GridSearchCV(lgbm\_model, lgbm\_params, scoring='neg\_mean\_squared\_error')  
lgbm\_grid\_search.fit(np.arange(len(close\_prices\_NEPSE\_reverse)).reshape(-1, 1), close\_prices\_NEPSE\_reverse)  
lgbm\_best\_model = lgbm\_grid\_search.best\_estimator\_  
lgbm\_predictions = lgbm\_best\_model.predict(np.arange(len(close\_prices\_NEPSE\_reverse)).reshape(-1, 1))  
  
import matplotlib.pyplot as plt  
from sklearn.metrics import mean\_squared\_error  
  
# Function to calculate RMSE  
def rmse(y\_true, y\_pred):  
 return np.sqrt(mean\_squared\_error(y\_true, y\_pred))  
  
# List of model names and predictions  
model\_names = ['Support Vector Machines (SVM)', 'Random Forest', 'XGBoost', 'LightGBM']  
predictions = [svm\_predictions, rf\_predictions, xgb\_predictions, lgbm\_predictions]  
best\_models = [svm\_best\_model, rf\_best\_model, xgb\_best\_model, lgbm\_best\_model]  
  
# Truncate actual values to match the length of predictions  
actual\_values\_reverse = close\_prices\_NEPSE\_reverse[-len(svm\_predictions):]  
  
# Evaluate models and plot graphs  
for i, model\_name in enumerate(model\_names):  
 model\_prediction = predictions[i]  
 model\_prediction\_truncated = model\_prediction[-len(actual\_values\_reverse):] # Truncate predicted values  
 model\_rmse = rmse(actual\_values\_reverse, model\_prediction\_truncated)  
  
 # Plotting actual and predicted values  
 plt.figure(figsize=(10, 6))  
 plt.plot(actual\_values\_reverse, label='Actual')  
 plt.plot(model\_prediction\_truncated, label='Predicted')  
 plt.title(f"{model\_name} - RMSE: {model\_rmse:.2f}")  
 plt.xlabel('Time')  
 plt.ylabel('Stock Price')  
 plt.legend()  
 plt.show()  
  
 # Print the best hyperparameters for the model  
 best\_model = best\_models[i]  
 print(f"Best Hyperparameters for {model\_name}:")  
 print(best\_model)  
 print("-----------------------------")  
  
# Plotting LSTM predictions  
plt.figure(figsize=(10, 6))  
plt.plot(actual\_values\_reverse, label='Actual')  
plt.plot(all\_lstm\_predictions, label='LSTM Predicted')  
plt.title(f"LSTM Model - RMSE: {best\_rmse:.2f}")  
plt.xlabel('Time')  
plt.ylabel('Stock Price')  
plt.legend()  
plt.show()  
  
import pandas as pd  
import numpy as np  
import matplotlib.pyplot as plt  
from sklearn.metrics import mean\_squared\_error  
from sklearn.model\_selection import GridSearchCV  
from sklearn.svm import SVR  
from sklearn.ensemble import RandomForestRegressor  
from xgboost import XGBRegressor  
from lightgbm import LGBMRegressor  
import tensorflow as tf  
from tensorflow import keras  
from tensorflow.keras import layers  
from keras.models import Sequential  
#from tensorflow import Sequential  
#from tensorflow import LSTM, Dense  
from tensorflow.keras.layers import LSTM , Dense  
#from tensorflow import Adam  
from keras.models import Model  
from keras.optimizers import adam  
  
# Load the stock data  
# file\_path = r'E:\non-credt\stock prediction\nepse\_10years.csv'  
data = pd.read\_csv("nepse\_10 years.csv")  
close\_prices\_NEPSE = data['Close']  
  
# Reverse the order of the data  
close\_prices\_NEPSE\_reverse = close\_prices\_NEPSE.iloc[::-1]  
  
# Reset index to maintain the correct time series order in the plot  
close\_prices\_NEPSE\_reverse.reset\_index(drop=True, inplace=True)  
  
# Data preprocessing  
data = close\_prices\_NEPSE\_reverse.values.reshape(-1, 1) # Reshape the data  
data\_normalized = data / np.max(data) # Normalize the data  
  
# Split the data into training and testing sets  
train\_size = int(len(data\_normalized) \* 0.8)  
train\_data = data\_normalized[:train\_size]  
test\_data = data\_normalized[train\_size:]  
  
# Function to create LSTM model  
def create\_lstm\_model(units, activation, learning\_rate):  
 model = Sequential()  
 model.add(LSTM(units=units, activation=activation, input\_shape=(1, 1)))  
 model.add(Dense(units=1))  
 optimizer = Adam(learning\_rate=learning\_rate)  
 model.compile(optimizer=optimizer, loss='mean\_squared\_error')  
 return model  
  
# Define hyperparameters for tuning  
lstm\_units = [50, 100, 200]  
lstm\_activations = ['relu', 'tanh']  
learning\_rates = [0.001, 0.01, 0.1]  
epochs = 200  
batch\_size = 32  
  
# Perform hyperparameter tuning for LSTM model  
best\_rmse = float('inf')  
best\_lstm\_model = None  
  
for units in lstm\_units:  
 for activation in lstm\_activations:  
 for learning\_rate in learning\_rates:  
 # Create and train LSTM model  
 model = create\_lstm\_model(units=units, activation=activation, learning\_rate=learning\_rate)  
 model.fit(train\_data[:-1].reshape(-1, 1, 1), train\_data[1:], epochs=epochs, batch\_size=batch\_size, verbose=0)  
  
 # Predict on test data  
 test\_predictions = model.predict(test\_data[:-1].reshape(-1, 1, 1)).flatten()  
  
 # Calculate RMSE  
 rmse = np.sqrt(mean\_squared\_error(test\_data[1:], test\_predictions))  
  
 # Check if current model has lower RMSE  
 if rmse < best\_rmse:  
 best\_rmse = rmse  
 best\_lstm\_model = model  
  
# Predict on the entire dataset using the best LSTM model  
all\_lstm\_predictions = best\_lstm\_model.predict(data\_normalized[:-1].reshape(-1, 1, 1)).flatten()  
  
# Inverse normalize the LSTM predictions  
all\_lstm\_predictions = all\_lstm\_predictions \* np.max(data)  
  
# Calculate the scaling factor based on the maximum value of the original data  
scaling\_factor = np.max(close\_prices\_NEPSE\_reverse)  
  
# Function to predict future stock prices using the LSTM model  
def predict\_future\_lstm(model, data, num\_predictions, scaling\_factor):  
 predictions = []  
  
 # Get the last data point from the input data  
 last\_data\_point = data[-1]  
  
 for \_ in range(num\_predictions):  
 # Predict the next time step  
 prediction = model.predict(last\_data\_point.reshape(1, 1, 1))  
 predictions.append(prediction[0, 0])  
  
 # Update last\_data\_point to include the predicted value for the next iteration  
 last\_data\_point = np.append(last\_data\_point[1:], prediction)  
  
 # Inverse normalize the predictions  
 predictions = np.array(predictions) \* scaling\_factor  
  
 return predictions  
  
# Predict the next 100 days using the LSTM model  
num\_predictions = 100  
lstm\_predictions = predict\_future\_lstm(best\_lstm\_model, data\_normalized, num\_predictions, scaling\_factor)  
  
# Plot the LSTM predictions for the next 100 days  
plt.figure(figsize=(10, 6))  
plt.plot(close\_prices\_NEPSE\_reverse, label='Actual')  
plt.plot(np.arange(len(close\_prices\_NEPSE\_reverse), len(close\_prices\_NEPSE\_reverse) + num\_predictions), lstm\_predictions, label='LSTM Predicted')  
plt.title(f"LSTM Model - RMSE: {best\_rmse:.2f}")  
plt.xlabel('Time')  
plt.ylabel('Stock Price')  
plt.legend()  
plt.show()  
  
# Print the predicted stock prices for the next 100 days using LSTM  
print("Predicted stock prices for the next 100 days:")  
for i, prediction in enumerate(lstm\_predictions, start=1):  
 print(f"Day {i}: {prediction:.2f}")

Output:

