International Journal of Research Publication and Reviews

Journal homepage: www.ijrpr.com ISSN 2582-7421

ENHANCING TIME SERIES PRODUCT DEMAND FORECASTING USING DEEP LEARNING MODELS

Mr. T.S.Subramani. [a] ***, R.Suwaytha*** [b] ***, J.Vallarasu*** [c] ***, S.Monika*** [d] ***, S.Mohamed Roshan Ashraf*** [e]

[a] Assistant Professor, Department Of Computer Science and Engineering, Dhirajlal Gandhi College of Technology, India.
[b, c, d, e] Student, Department Of Computer Science and Engineering, Dhirajlal Gandhi College of Technology, India.

ABSTRACT:

Stock price prediction is the process of using historical stock price data to forecast the future prices of a stock. The goal of stock price prediction is to identify patterns and trends in the historical data that can be used to predict the future prices with reasonable accuracy. There are many different approaches to stock price prediction, including technical analysis, fundamental analysis, and machine learning algorithms. Technical analysis involves the use of charts and technical indicators to identify trends and patterns in the historical stock price data. Fundamental analysis, on the other hand, involves the analysis of the financial and economic factors that can impact the stock price, such as earnings reports, economic indicators, and industry trends. Stock price prediction is an important area of research in finance and economics, with many different algorithms being used to predict future prices. One such algorithm is the Long Short-term memory (LSTM) Regression algorithm, which is a type of artificial neural network. This project presents a study of LSTM Regression algorithm for stock price prediction. The algorithm is trained on historical stock price data and used to predict the future prices of a stock. The performance of the algorithm is evaluated using various metrics, including the Mean Squared Error (MSE) and the coefficient of determination (R-squared). The results of the study show that LSTM Regression algorithm is a powerful tool for stock price prediction, and can provide accurate and reliable predictions for a wide range of stocks. The study also highlights the importance of selecting appropriate input features, and the need to carefully tune the hyperparameters of the algorithm to achieve optimal performance. Overall, the study demonstrates the potential of LSTM Regression algorithm for stock price prediction, and provides insights into its strengths and limitations

INTRODUCTION

Stock price prediction plays a crucial role in financial markets, helping investors and traders make informed decisions. It involves analyzing historical stock price data to identify trends and patterns that can be used to forecast future stock prices. Traditional methods such as technical analysis and fundamental analysis have been widely used for this purpose.Technical analysis relies on historical price movements, charts, and indicators to predict future trends, while fundamental analysis considers financial statements, economic factors, and industry trends. However, these approaches have limitations in handling complex market behaviors and large datasets, leading to the exploration of machine learning techniques for more accurate predictions. With advancements in artificial intelligence, machine learning models have become increasingly popular for stock price prediction. These models can process vast amounts of data and detect hidden patterns that traditional methods might overlook. Techniques such as linear regression, Autoregressive Integrated Moving Average (ARIMA), and Artificial Neural Networks (ANNs) have been applied to stock price forecasting.Among these, deep learning models like Recurrent Neural Networks (RNNs) and Long Short-Term Memory (LSTM) networks have shown significant potential due to their ability to capture sequential dependencies in stock price data. LSTM, a specialized type of RNN, is particularly effective in handling time-series data by retaining important information over long periods, making it a strong candidate for stock market predictions. This project proposes a stock price prediction system using the LSTM regression algorithm. The system leverages historical stock price data and financial indicators to train the model and generate future price predictions. Stock price prediction plays a crucial role in financial markets, helping investors and traders make informed decisions. It involves analyzing historical stock price data to identify trends and patterns that can be used to forecast future stock prices. Traditional methods such as technical analysis and fundamental analysis have been widely used for this purpose.Technical analysis relies on historical price movements, charts, and indicators to predict future trends, while fundamental analysis considers financial statements, economic factors, and industry trends. However, these approaches have limitations in handling complex market behaviors and large datasets, leading to the exploration of machine learning techniques for more accurate predictions. With advancements in artificial intelligence, machine learning models have become increasingly popular for stock price prediction. These models can process vast amounts of data and detect hidden patterns that traditional methods might overlook. Techniques such as linear regression, Autoregressive Integrated Moving Average (ARIMA), and Artificial Neural Networks (ANNs) have been applied to stock price forecasting.Among these, deep learning models like Recurrent Neural Networks (RNNs) and Long Short-Term Memory (LSTM) networks have shown significant potential due to their ability to capture sequential dependencies in stock price data. LSTM, a specialized type of RNN, is particularly effective in handling time-series

data by retaining important information over long periods, making it a strong candidate for stock market predictions. This project proposes a stock price prediction system using the LSTM regression algorithm. The system leverages historical stock price data and financial indicators to train the model and generate future price predictions.

LITERATURE SURVEY

2.1. NEW PRODUCT DEMAND FORECASTING IN RETAIL: APPLYING MACHINE LEARNING TECHNIQUES TO FORECAST DEMAND FOR NEW PRODUCT PURCHASING DECISIONS

In retail, it's vital to minimize food spoilage and working capital, and to maximize product availability and sales. To make it possible, retailers must know what the proper order quantity is when making purchase decisions. Hence, demand forecasting plays a crucial role in retail industry. However, generating accurate forecasts is difficult, especially when it comes to new products for which there is no historical data available. In addition, the most common new product forecast methods are at least partly judgemental and thus inefficient. Consequently, there is a clear need for accurate and efficient method for forecasting new product demand. This thesis addresses the phenomena by applying and evaluating five machine learning models for new product demand forecasting for purchase decisions in retail and selecting the model that performs best.The forecasting problem is formulated as a classification task in which the objective is to forecast the magnitude of sales for new products.

2.2. FOG COMPUTING ENABLED LOCALITY BASED PRODUCT DEMAND PREDICTION AND DECISION-MAKING USING REINFORCEMENT LEARNING

Artificial Intelligence (AI) is a trending technology, which retrieves meaningful information by processing data generated from widely deployed IoT sensors and devices in smart supermarkets. Wide range of Machine Learning (ML) methods such as Supervised Learning (SL), Unsupervised Learning (UL) and Reinforcement Learning (RL) algorithms assist in extracting various parameters to build the decision making and prediction model. Wastage of perishable and non-perishable products due to manual monitoring in shopping malls creates huge revenue loss in supermarket industry. Besides, internal and external factors such as calendar events and weather condition contribute to excess wastage of products in different regions of supermarket. It is a challenging job to know about the wastage of the products manually in different supermarkets region-wise. Therefore, the supermarket management needs to take appropriate decision and action to prevent the wastage of products. The fog computing data centers located in each region can collect, process and analyze data for demand prediction and decision making.

SYSTEM STUDY

3.1. EXISTING SYSTEM

Stock price prediction has traditionally relied on statistical and machine learning models, each offering specific benefits while also presenting limitations. One of the most basic approaches is the use of moving averages, which smooth out short-term fluctuations by averaging past prices over a defined period.Similarly, linear regression is frequently employed to fit a trend line through historical data to estimate future prices. Another popular statistical model is ARIMA (Autoregressive Integrated Moving Average), which incorporates both trends and seasonality in the time-series data, offering better forecasting than simpler models. However, despite their simplicity and interpretability, these models are often insufficient for capturing the intricate dynamics of financial markets. The primary issue with traditional models is their inability to handle nonlinearity and volatility present in real-world stock markets. As a result, they struggle in unpredictable or highly volatile conditions. With the rise of artificial intelligence, more advanced machine learning techniques have emerged to improve prediction accuracy. Techniques such as Artificial Neural Networks (ANNs) have been employed to model complex relationships in data. Deep learning models like Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs) further enhanced predictive capabilities by processing temporal and spatial patterns in stock price data.Among these models, Long Short-Term Memory (LSTM) networks, a specialized form of RNN, have gained significant attention for their effectiveness in stock price forecasting. LSTMs are uniquely capable of remembering long-term dependencies in time-series data, which is critical for understanding financial trends over time. Despite their strengths, existing systems based on these technologies still face several challenges, such as difficulty handling large-scale datasets, high computational costs, and increased false positive rates. Additionally, many models struggle with unlabeled datasets, limiting their adaptability across various market conditions.

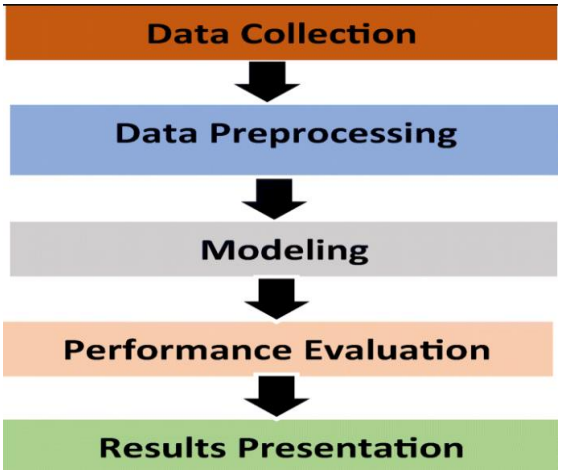


Figure 3.1.1: Existing System Processing Using

3.2. PROPOSED SYSTEM

The proposed system leverages the Long Short-Term Memory (LSTM) Regression algorithm to improve the accuracy and reliability of stock price prediction. Unlike traditional statistical models, LSTM is a deep learning technique capable of capturing long-term dependencies in sequential data. This makes it highly suitable for analyzing time-series stock data, which often exhibits complex patterns and non-linear trends.The system begins with data acquisition from reliable financial sources, ensuring high-quality input for training and testing. Following data collection, the system applies preprocessing techniques such as data cleaning, normalization, and splitting the dataset into training and testing sets. To further enhance model performance, advanced feature extraction methods are used to identify the most relevant financial indicators. These extracted features help the model learn important market signals, enabling it to understand the stock price fluctuations over time. By focusing on meaningful features, the system reduces noise and improves learning efficiency. Once the model is trained, it is used to generate future stock price predictions with greater precision than existing methods. The approach includes careful tuning of hyperparameters and optimizing the LSTM network architecture to prevent overfitting and reduce false positives. By addressing the shortcomings of traditional and earlier machine learning methods, the proposed system achieves better prediction accuracy and lower computational complexity. Overall, it provides a robust, adaptive, and efficient framework for stock market forecasting.

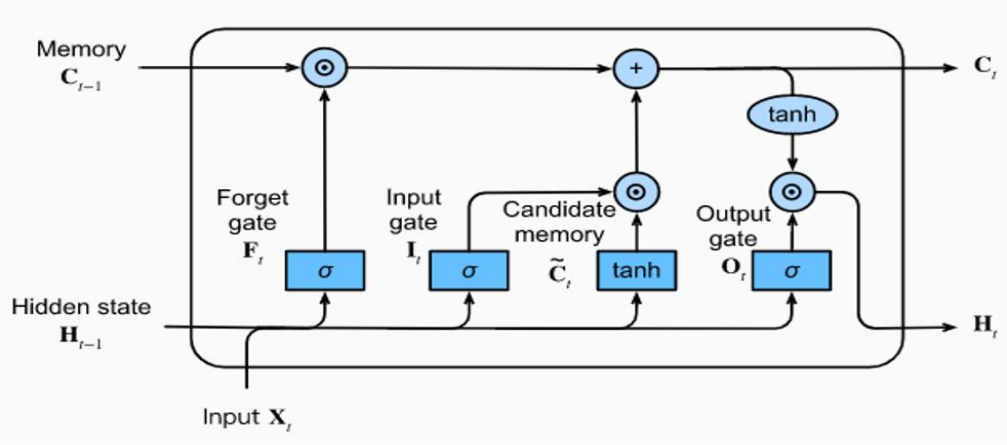


Figure 3.2.1: PROPOSED SYSTEM PROCESS

SYSTEM SPECIFICATION

4.1 HARDWARE REQUIREMENTS

- Processor :Intelcore processor 2.6.0 GHZ
- RAM : 1GB
- Hard disk : 160 GB

- Compact Disk : 650 Mb
- Keyboard : Standard keyboard
- Monitor :15-inch color monitor

4.2 SOFTWARE REQUIREMENTS

- Operating system : Windows OS
- Front End : PYTHON
- Back End : MYSQL
- IDE : PYCHARM

MODULES IMPLEMENTATION

5.1 LIST OF MODULES

- Datasets Acquisition
- Preprocessing
- Features Extraction
- LSTM Based Model Construction
- Price Prediction

5.2 MODULES DESCRIPTION

5.2.1 DATASETS ACQUSITION

A data set (or dataset, although this spelling is not present in many contemporary dictionaries like Merriam-Webster) is a collection of data. Most commonly a data set corresponds to the contents of a single database table, or a single statistical data matrix, where every column of the table represents a particular variable, and each row corresponds to a given member of the data set in question. The data set may comprise data for one or more members, corresponding to the number of rows. The term data set may also be used more loosely, to refer to the data in a collection of closely related tables, corresponding to a particular experiment or event. In this module, we can upload the datasets related to stock data. Data can be collected from Kaggle Web data sources. It contains the attributes such as opening price, closing price, date and soon.

5.2.2 PREPROCESSING

Data pre-processing is an important step in the [data mining] process. The phrase "garbage in, garbage out" is particularly applicable to data mining and machine learning projects. Data-gathering methods are often loosely controlled, resulting in out-of-range values, impossible data combinations, missing values, etc. Analyzing data that has not been carefully screened for such problems can produce misleading results. Thus, the representation and quality of data is first and foremost before running an analysis. If there is much irrelevant and redundant information present or noisy and unreliable data, then knowledge discovery during the training phase is more difficult. Data preparation and filtering steps can take considerable amount of processing time. In this module, we can eliminate the irrelevant values and also estimate the missing values of data. Finally provide structured datasets.

5.2.3 FEATURES EXTRACTION

Feature extraction is an essential step in stock price prediction that involves transforming raw financial data into a set of features that can be used as input to a machine learning model. Features extraction in stock datasets is the process of selecting and creating relevant variables or attributes from the raw stock market data. These features, also known as predictors or input variables, are used to train machine learning models for stock market analysis and prediction. The choice of features will depend on the specific problem being solved and the type of deep learning model being used. It's important to carefully select and engineer relevant features as they can significantly impact the performance of the machine learning model. Fundamental analysis involves examining a company's financial statements and other economic factors to evaluate its intrinsic value. These factors may include earnings, revenue, profit margin, and market share. By analyzing these factors, investors can gain insights into the future performance of a company and its stock price.

5.2.4 LSTM BASED MODEL CONSTRUCTION

LSTM is a type of artificial neural network that can be used for stock price prediction. The basic idea behind using an LSTM for stock price prediction is to train the network on historical stock price data and use the trained network to make predictions about future stock prices. The LSTM should have

one or more hidden layers, and the number of neurons in each layer will depend on the complexity of the problem and the amount of data have. Once the data has been prepared and the features selected, the LSTM regressor can be trained using the backpropagation algorithm. During training, the LSTM regressor learns to map the input features to the target variable (i.e., stock price) by adjusting the weights of the neurons in the network. After training, the performance of the LSTM regressor can be evaluated using a validation set or cross-validation. This can help to identify any issues with the model, such as overfitting or underfitting, and can guide further improvements to the model.Root Mean Squared Error (RMSE) is a popular performance metric used in machine learning to measure the difference between predicted and actual values. In the context of a Long Short-Term memory (LSTM) model, RMSE can be used to evaluate the performance of the model. The RMSE error rate in an LSTM model is calculated by taking the square root of the average of the squared differences between the predicted values and the actual values.

This can be expressed mathematically as:

RMSE = sqrt((1/n) * sum((y_pred - y_actual)^2))

where:

y_pred is the predicted value

y_actual is the actual value

n is the number of samples in the dataset

In an LSTM model, the goal is to minimize the RMSE between the predicted output and the actual output. This is achieved by adjusting the weights and biases in the model during the training process, using techniques such as gradient descent. It is important to note that RMSE is just one of many performance metrics that can be used to evaluate the performance of an MLP model.

5.2.5 PRICE PREDICTION

In stock dataset analysis using LSTMs, the model is trained on historical stock prices and stock market indicators and then used to make predictions on future prices. The accuracy of predictions can be improved by fine-tuning the model, adding more indicators, and increasing the size of the training dataset. However, stock market predictions are inherently uncertain and LSTM models are not a guarantee of future performance. And also calculate the future price predictions based on LSTM regressor.

5.2.6 FORECASTING AND BOT CONSTRUCTION

In this module, we can provide the days to forecast the stock the price for each stock, days may be 10 days, 15 days or 30 days. Then provide the chat interface to ask the questions about stock data. And also send the questions to experts about stock, then manually answer the questions.

SYSTEM DESIGN

6.1 SYSTEM ARCHITECTURE

The architecture of the proposed stock price prediction system is designed to integrate various modules for seamless data processing and prediction. It begins with the data acquisition module, where stock market data is collected from various sources such as APIs or web scraping platforms. The collected data is then pre-processed to remove inconsistencies and scale the features appropriately. After preprocessing, relevant features are extracted to create a dataset suitable for training the LSTM-based model. The LSTM model is trained on the processed data, learning the temporal patterns of stock prices to make accurate predictions.Additionally, a chatbot is integrated into the system to answer user queries related to stock prices, trends, and forecasts, enhancing user interaction and accessibility to the predictions.The system begins with the Data Acquisition Module, which collects stock market data from multiple sources such as APIs and web scraping tools.Next, the Data Preprocessing Module cleans the data by handling missing values. removing inconsistencies, and scaling features to ensure compatibility with the model.To enhance user experience, the system also integrates a Chatbot Interface that allows users to interact with the system. The chatbot provides responses to queries related to stock prices, market trends, and forecasted values, thereby improving accessibility and usability of the prediction outputs.

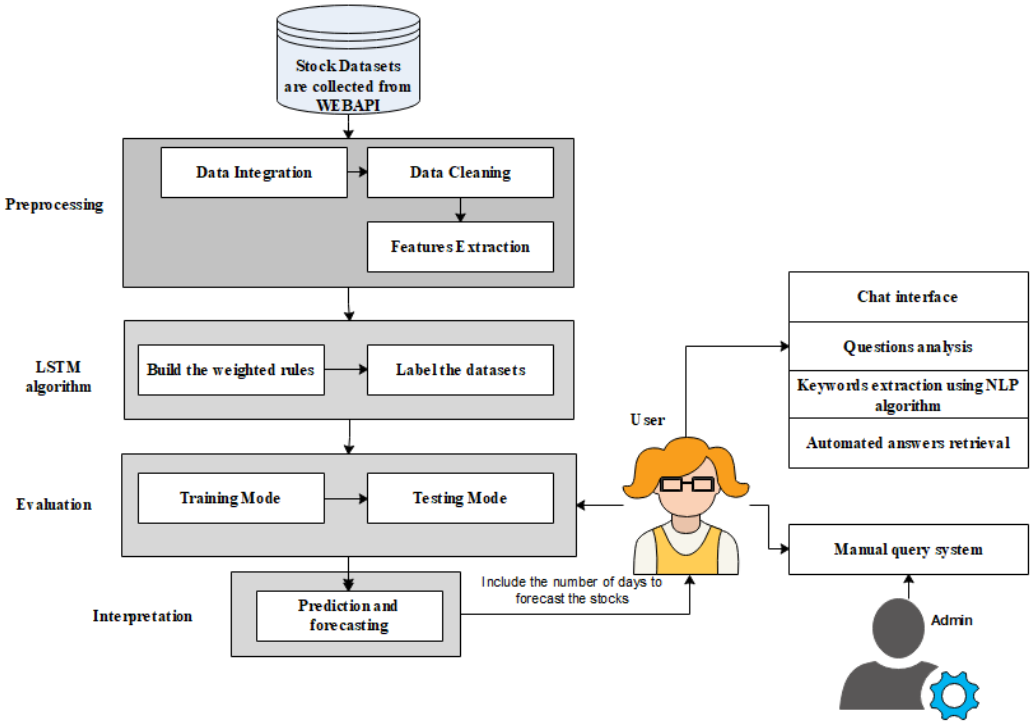


Figure 6.1: System architecture

7.CONCLUSION AND FUTURE ENHANCEMENTS

CONCLUSION

The study on stock price prediction using the LSTM Regression algorithm demonstrates the effectiveness of deep learning in capturing complex patterns and long-term dependencies in financial data. By utilizing historical stock prices and relevant financial indicators, the proposed system achieves improved accuracy in forecasting stock price trends. Compared to traditional methods such as moving averages and linear regression, LSTM proves to be more efficient in handling nonlinear relationships and reducing prediction errors. The evaluation metrics, including Mean Squared Error (MSE) and R-squared, indicate that the model provides reliable predictions, making it a valuable tool for investors and financial analysts.Despite its advantages, stock price prediction remains a challenging task due to market volatility and unpredictable external factors. While the LSTM model enhances forecasting accuracy, incorporating additional data sources such as news sentiment analysis and macroeconomic indicators could further improve performance. Continuous model training and parameter optimization are essential to adapt to changing market conditions. Overall, this project highlights the potential of deep learning in financial forecasting and provides a foundation for future advancements in stock market analysis.

FUTURE ENHANCEMENTS

- **Integration of Sentiment Analysis :**Incorporating sentiment analysis from financial news, social media, and expert opinions can enhance prediction accuracy. By analyzing public sentiment, the model can better anticipate market fluctuations influenced by investor behavior.
- **Incorporation of Macroeconomic Indicators :**Adding external economic factors such as GDP growth, inflation rates, and interest rates can provide a more comprehensive understanding of market trends.This will help improve the model's ability to predict price movements under different economic conditions.
- **Hybrid Model Implementation :**Combining LSTM with other machine learning models, such as Random Forest or XGBoost, can improve prediction performance.A hybrid model can leverage the strengths of multiple algorithms to enhance accuracy and reduce errors.
- **Real-Time Prediction System :**Developing a real-time stock prediction system with continuous data streaming will make the model more practical for investors. By integrating real-time data feeds, the system can provide instant insights and timely decision-making support.
- **Optimization of Model Performance :**Fine-tuning hyperparameters and exploring different neural network architectures can further enhance the efficiency and accuracy of the LSTM model. This will help reduce computational complexity while maintaining high prediction accuracy.

8.APPENDIX -1 SOURCE CODE

```
from flask import Flask, render_template, request, session, flash, send_file
import mysql.connector
import os
import pandas as pd
import numpy as np
import math
import datetime as dt
import matplotlib.pyplot as plt
from sklearn.metrics import mean_squared_error, mean_absolute_error, explained_variance_score, r2_score
from sklearn.metrics import mean_poisson_deviance, mean_gamma_deviance, accuracy_score
from sklearn.preprocessing import MinMaxScaler
import tensorflow as tf
from keras.models import Sequential
from keras.layers import Dense, Dropout
from keras.layers import LSTM
import matplotlib.pyplot as plt
from itertools import cycle
import plotly.graph_objects as go
import plotly.express as px
import plotly.io as pio

pio.renderers.default = "browser"
# pio.renderers.default = 'png'
from plotly.subplots import make_subplots
import io
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import plotly.graph_objects as go
import plotly.io as pio
import pytz
import yfinance as yf
import plotly.express as px
# Set the start and end date for the historical data
from datetime import datetime as dt
import pytz

from chatterbot import ChatBot
from chatterbot.trainers import ListTrainer
from requests import get
from bs4 import BeautifulSoup
import os
from flask import Flask, render_template, request, jsonify

english_bot = ChatBot('Bot',
                      storage_adapter='chatterbot.storage.SQLStorageAdapter',
                      logic_adapters=[
                          {
                              'import_path': 'chatterbot.logic.BestMatch'
```

```
            },

        ],
        trainer='chatterbot.trainers.ListTrainer')
english_bot.set_trainer(ListTrainer)
app = Flask(__name__)
app.config['SECRET_KEY'] = 'aaa'


@app.route('/')
def home():
    return render_template('index.html')

@app.route('/AdminLogin')
def AdminLogin():
    return render_template('AdminLogin.html')

@app.route('/Chat')
def Chat():
    return render_template('chat.html')

@app.route('/NewUser')
def NewUser():
    return render_template('NewUser.html')

@app.route('/UserLogin')
def UserLogin():
    return render_template('UserLogin.html')
@app.route('/NewExpert')
def NewExpert():
    return render_template('NewExpert.html')

@app.route('/ExpertLogin')
def ExpertLogin():
    return render_template('ExpertLogin.html')

@app.route("/ask", methods=['GET', 'POST'])
def ask():
    message = str(request.form['messageText'])
    bott = ''
    bott1 = ''
    sresult1 = ''
    bot_response = english_bot.get_response(message)

    print(bot_response)
    conn1 = mysql.connector.connect(user='root', password='', host='localhost', database='1StockNew')
    cur1 = conn1.cursor()
    cur1.execute(
        "SELECT  * from  chattb where Query='" + message + "'")
    data = cur1.fetchone()
```

```
    if data:
        bb = "Expert Name "+ str(data[1])
        bott ="Answer "+str(data[3])

        return jsonify({'status': 'OK', 'answer': bb + bott})

    while True:

        if bot_response.confidence > 0.5:

            bot_response = str(bot_response)
            print(bot_response)
            return jsonify({'status': 'OK', 'answer': bot_response})

        elif message == ("bye") or message == ("exit"):

            bot_response = 'Hope to see you soon' + '<a href="http://127.0.0.1:5000/UserHome">Exit</a>'

            print(bot_response)
            return jsonify({'status': 'OK', 'answer': bot_response})
            break
        else:
            try:
                url = "https://en.wikipedia.org/wiki/" + message
                page = get(url).text
                soup = BeautifulSoup(page, "html.parser")
                p = soup.find_all("p")
                return jsonify({'status': 'OK', 'answer': p[1].text})

            except IndexError as error:

                bot_response = 'Sorry i have no idea about that.'
                print(bot_response)
                return jsonify({'status': 'OK', 'answer': bot_response})
    # return render_template("index.html")
@app.route("/adminlogin", methods=['GET', 'POST'])
def adminlogin():
    error = None
    if request.method == 'POST':
        if request.form['uname'] == 'admin' and request.form['password'] == 'admin':
            conn = mysql.connector.connect(user='root', password='', host='localhost',
                                           database='1StockNew')
            cur = conn.cursor()
            cur.execute("SELECT * FROM regtb")
            data = cur.fetchall()
            return render_template('AdminHome.html', data=data)

        else:
            return render_template('index.html', error=error)
```

```
@app.route("/AdminHome")
def AdminHome():
    conn = mysql.connector.connect(user='root', password='', host='localhost', database='1StockNew')
    cur = conn.cursor()
    cur.execute("SELECT * FROM regtb")
    data = cur.fetchall()
    return render_template('AdminHome.html', data=data)

@app.route("/ExpertInfo")
def ExpertInfo():
    conn = mysql.connector.connect(user='root', password='', host='localhost', database='1StockNew')
    cur = conn.cursor()
    cur.execute("SELECT * FROM experttb")
    data = cur.fetchall()
    return render_template('ExpertInfo.html', data=data)

@app.route("/newex", methods=['GET', 'POST'])
def newex():
    if request.method == 'POST':
        name = request.form['uname']
        mobile = request.form['mobile']
        email = request.form['email']
        uname = request.form['username']
        password = request.form['password']

        conn = mysql.connector.connect(user='root', password='', host='localhost', database='1StockNew')
        cursor = conn.cursor()
        cursor.execute(
            "INSERT INTO experttb VALUES ('" + name + "','" + email + "','" + mobile + "','" + uname + "','" + password + "')")
        conn.commit()
        conn.close()
        flash('New Expert  Register successfully')
    return render_template('ExpertLogin.html')

@app.route("/exlogin", methods=['GET', 'POST'])
def exlogin():
    if request.method == 'POST':
        username = request.form['uname']
        password = request.form['password']
        session['ename'] = request.form['uname']

        conn = mysql.connector.connect(user='root', password='', host='localhost', database='1StockNew')
        cursor = conn.cursor()
        cursor.execute("SELECT * from experttb where UserName='" + username + "' and password='" + password + "'")
        data = cursor.fetchone()
        if data is None:

            flash('Username or Password is wrong')
            return render_template('ExpertLogin.html')
```

```
        else:
            session['mob'] = data[4]
            conn = mysql.connector.connect(user='root', password='', host='localhost',
                                           database='1StockNew')
            cur = conn.cursor()
            cur.execute("SELECT * FROM experttb where UserName='" + username + "' and password='" + password + "'")
            data = cur.fetchall()

            flash("you are successfully logged in")
            return render_template('ExpertHome.html', data=data)
@app.route('/ExpertHome')
def ExpertHome():
    uname = session['ename']
    conn = mysql.connector.connect(user='root', password='', host='localhost', database='1StockNew')
    cur = conn.cursor()
    cur.execute("SELECT * FROM experttb where UserName='" + uname + "' ")
    data = cur.fetchall()
    return render_template('ExpertHome.html', data=data)

@app.route('/QueryInfo')
def QueryInfo():
    conn = mysql.connector.connect(user='root', password='', host='localhost', database='1StockNew')
    cur = conn.cursor()
    cur.execute("SELECT * FROM Querytb where Answer ='' ")
    data = cur.fetchall()
    uname = session['ename']

    conn = mysql.connector.connect(user='root', password='', host='localhost', database='1StockNew')
    cur = conn.cursor()
    cur.execute("SELECT * FROM Querytb where Answer !='  and ExpertName='" + uname + "' ")
    data1 = cur.fetchall()

    return render_template('QueryInfo.html', data=data, data1=data1)

@app.route("/ans")
def ans():
    id = request.args.get('id')
    session['id'] = id
    return render_template('Answer.html')

@app.route("/ChatTrain")
def ChatTrain():
    name = session['ename']
    conn = mysql.connector.connect(user='root', password='', host='localhost', database='1StockNew')
    cur = conn.cursor()
    cur.execute("SELECT * FROM chattb where ExpertName='" + name + "' ")
    data = cur.fetchall()
    return render_template('ChatTrain.html', data=data)

@app.route("/chattrain", methods=['GET', 'POST'])
```

```
def chattrain():
    if request.method == 'POST':
        name = session['ename']
        Query = request.form['Query']
        Answer = request.form['Answer']
        import datetime
        date = datetime.datetime.now().strftime('%d-%b-%Y')

        conn = mysql.connector.connect(user='root', password='', host='localhost', database='1StockNew')
        cursor = conn.cursor()
        cursor.execute(
            "INSERT INTO chattb VALUES ('','" + name + "','" + Query + "','" + Answer + "','" + date + "')")
        conn.commit()
        conn.close()
        flash('Chatbot train successfully')

    conn = mysql.connector.connect(user='root', password='', host='localhost', database='1StockNew')
    cur = conn.cursor()
    cur.execute("SELECT * FROM chattb where ExpertName='" + name + "' ")
    data = cur.fetchall()
    return render_template('ChatTrain.html', data=data)

@app.route("/Remove")
def Remove():
    id = request.args.get('id')
    name = session['ename']
    conn = mysql.connector.connect(user='root', password='', host='localhost', database='1StockNew')
    cursor = conn.cursor()
    cursor.execute(
        "delete from  chattb  where id='" + id + "'")
    conn.commit()
    conn.close()

    conn = mysql.connector.connect(user='root', password='', host='localhost', database='1StockNew')
    cur = conn.cursor()
    cur.execute("SELECT * FROM chattb where ExpertName='" + name + "' ")
    data = cur.fetchall()
    return render_template('ChatTrain.html', data=data)

@app.route("/answer", methods=['GET', 'POST'])
def answer():
    if request.method == 'POST':
        name = session['ename']
        id = session['id']
        Query = request.form['Query']

        conn = mysql.connector.connect(user='root', password='', host='localhost', database='1StockNew')
        cursor = conn.cursor()
        cursor.execute(
            "update  Querytb set Answer='" + Query + "',ExpertName='" + name + "' where id='" + id + "'")
```

```
        conn.commit()
        conn.close()
        flash('Answer Info  Update successfully')

    conn = mysql.connector.connect(user='root', password='', host='localhost', database='1StockNew')
    cur = conn.cursor()
    cur.execute("SELECT * FROM Querytb where Answer =" '')
    data = cur.fetchall()
    uname = session['ename']

    conn = mysql.connector.connect(user='root', password='', host='localhost', database='1StockNew')
    cur = conn.cursor()
    cur.execute("SELECT * FROM Querytb where Answer !='  and ExpertName='" + uname + "' ")
    data1 = cur.fetchall()
    return render_template('QueryInfo.html', data=data, data1=data1)

@app.route("/newuser", methods=['GET', 'POST'])
def newuser():
    if request.method == 'POST':
        name = request.form['uname']
        mobile = request.form['mobile']
        email = request.form['email']
        uname = request.form['username']
        password = request.form['password']

        conn = mysql.connector.connect(user='root', password='', host='localhost', database='1StockNew')
        cursor = conn.cursor()
        cursor.execute(
            "INSERT INTO regtb VALUES ('" + name + "','" + email + "','" + mobile + "','" + uname + "','" + password + "')")
        conn.commit()
        conn.close()
        flash('New User  Register successfully')

    return render_template('UserLogin.html')

@app.route("/userlogin", methods=['GET', 'POST'])
def userlogin():
    if request.method == 'POST':
        username = request.form['uname']
        password = request.form['password']
        session['uname'] = request.form['uname']

        conn = mysql.connector.connect(user='root', password='', host='localhost', database='1StockNew')
        cursor = conn.cursor()
        cursor.execute("SELECT * from regtb where UserName='" + username + "' and password='" + password + "'")
        data = cursor.fetchone()
        if data is None:

            flash('Username or Password is wrong')
            return render_template('UserLogin.html')
```

```
        else:
            session['mob'] = data[4]
            conn = mysql.connector.connect(user='root', password='', host='localhost',
                                           database='1StockNew')
            cur = conn.cursor()
            cur.execute("SELECT * FROM regtb where UserName='" + username + "' and password='" + password + "'")
            data = cur.fetchall()

            flash("you are successfully logged in")
            return render_template('UserHome.html', data=data)
@app.route('/Search')
def Search():
    from gnewsclient import gnewsclient

    client = gnewsclient.NewsClient(language='english',
                                    location='india',
                                    topic='Business',
                                    max_results=20)

    news_list = client.get_news()

    return render_template('Search.html', data=news_list)

@app.route('/UserHome')
def UserHome():
    uname = session['uname']
    conn = mysql.connector.connect(user='root', password='', host='localhost',
                                   database='1StockNew')
    cur = conn.cursor()
    cur.execute("SELECT * FROM regtb where UserName='" + uname + "' ")
    data = cur.fetchall()
    return render_template('UserHome.html', data=data)

@app.route('/Prediction')
def Prediction():
    return render_template('Prediction.html')

@app.route("/predict", methods=['GET', 'POST'])
def predict():
    if request.method == 'POST':
        ptdate = request.form['days']
        symbol = request.form['symbol']

        tz = pytz.timezone("America/New_York")
        start_date = tz.localize(dt(2022, 1, 1))
        end_date = tz.localize(dt.today())
        # start_date = date(2023, 1, 1)
        # end_date = date(2023, 4, 20)

        #symbol = "AAPL"
```

```
print(start_date, end_date)
# from_date = start_date.split('T')[0]
# to_date = end_date.split('T')[0]
# print(from_date, to_date)
df1 = yf.download(symbol, start_date, end_date)
maindf = df1.reset_index()
maindf['symbol'] = symbol
print(maindf)

hovertext = []
for i in range(len(maindf['Open'])):
    hovertext.append('Open: ' + str(maindf['Open'][i]) + '<br>Close: ' + str(maindf['Close'][i]))
pio.templates.default = "plotly_dark"
fig = go.Figure(data=go.Candlestick(x=maindf['Date'],
                                    open=maindf['Open'],
                                    high=maindf['High'],
                                    low=maindf['Low'],
                                    close=maindf['Close'], text=hovertext,
                                    hoverinfo='text'))
fig.update(layout_xaxis_rangeslider_visible=False)
fig.update_layout(title_text="Stock history")
fig.show()

# maindf = pd.read_csv('Data/' + dateset + '.csv')
print(maindf)

print('Total number of days present in the dataset: ', maindf.shape[0])
print('Total number of fields present in the dataset: ', maindf.shape[1])

print(maindf.shape)

sd = maindf.iloc[0][0]
ed = maindf.iloc[-1][0]

print('Starting Date', sd)
print('Ending Date', ed)

maindf['Date'] = pd.to_datetime(maindf['Date'], format='%Y-%m-%d')

closedf = maindf[['Date', 'Close']]

closedf = closedf[closedf['Date'] > '2022-02-02']
close_stock = closedf.copy()
print("Total data for prediction: ", closedf.shape[0])

del closedf['Date']
scaler = MinMaxScaler(feature_range=(0, 1))
closedf = scaler.fit_transform(np.array(closedf).reshape(-1, 1))
print(closedf.shape)
```

```
training_size = int(len(closedf) * 0.60)
test_size = len(closedf) - training_size
train_data, test_data = closedf[0:training_size, :], closedf[training_size:len(closedf), :1]
print("train_data: ", train_data.shape)
print("test_data: ", test_data.shape)

def create_dataset(dataset, time_step=1):
    dataX, dataY = [], []
    for i in range(len(dataset) - time_step - 1):
        a = dataset[i:(i + time_step), 0]  ###i=0, 0,1,2,3-----99   100
        dataX.append(a)
        dataY.append(dataset[i + time_step, 0])
    return np.array(dataX), np.array(dataY)

time_step = 15
X_train, y_train = create_dataset(train_data, time_step)
X_test, y_test = create_dataset(test_data, time_step)

print("X_train: ", X_train.shape)
print("y_train: ", y_train.shape)
print("X_test: ", X_test.shape)
print("y_test", y_test.shape)

X_train = X_train.reshape(X_train.shape[0], X_train.shape[1], 1)
X_test = X_test.reshape(X_test.shape[0], X_test.shape[1], 1)

print("X_train: ", X_train.shape)
print("X_test: ", X_test.shape)

model = Sequential()

model.add(LSTM(10, input_shape=(None, 1), activation="relu"))

model.add(Dense(1))

model.compile(loss="mean_squared_error", optimizer="adam", metrics=['accuracy'])
history = model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=200, batch_size=32, verbose=1)

acc = history.history['accuracy']
loss = history.history['loss']
epochs = range(1, len(acc) + 1)

acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
epochs = range(len(acc))

plt.plot(epochs, acc, label='Training Accuracy')
plt.plot(epochs, val_acc, label='Validation Accuracy')
plt.title('Training and Validation Accuracy')
plt.xlabel('Epochs')
```

```
plt.ylabel('Accuracy')
plt.legend()
plt.grid(True)
# plt.show()

# Plot Model Loss
loss_train = history.history['loss']
loss_val = history.history['val_loss']
plt.plot(epochs, loss_train, label='Training Loss')
plt.plot(epochs, loss_val, label='Validation Loss')
plt.title('Training and Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.grid(True)
plt.show()

train_predict = model.predict(X_train)
test_predict = model.predict(X_test)
train_predict.shape, test_predict.shape

train_predict = scaler.inverse_transform(train_predict)
test_predict = scaler.inverse_transform(test_predict)
original_ytrain = scaler.inverse_transform(y_train.reshape(-1, 1))
original_ytest = scaler.inverse_transform(y_test.reshape(-1, 1))

print("Train data RMSE: ", math.sqrt(mean_squared_error(original_ytrain, train_predict)))
print("Train data MSE: ", mean_squared_error(original_ytrain, train_predict))

print("-------------------------------------------------------------------------------------")
print("Test data RMSE: ", math.sqrt(mean_squared_error(original_ytest, test_predict)))
print("Test data MSE: ", mean_squared_error(original_ytest, test_predict))

print("Train data explained variance regression score:",
    explained_variance_score(original_ytrain, train_predict))
print("Test data explained variance regression score:",
    explained_variance_score(original_ytest, test_predict))

print("Train data R2 score:", r2_score(original_ytrain, train_predict))
print("Test data R2 score:", r2_score(original_ytest, test_predict))

look_back = time_step
trainPredictPlot = np.empty_like(closedf)
trainPredictPlot[:, :] = np.nan
trainPredictPlot[look_back:len(train_predict) + look_back, :] = train_predict
print("Train predicted data: ", trainPredictPlot.shape)

# shift test predictions for plotting
testPredictPlot = np.empty_like(closedf)
testPredictPlot[:, :] = np.nan
```

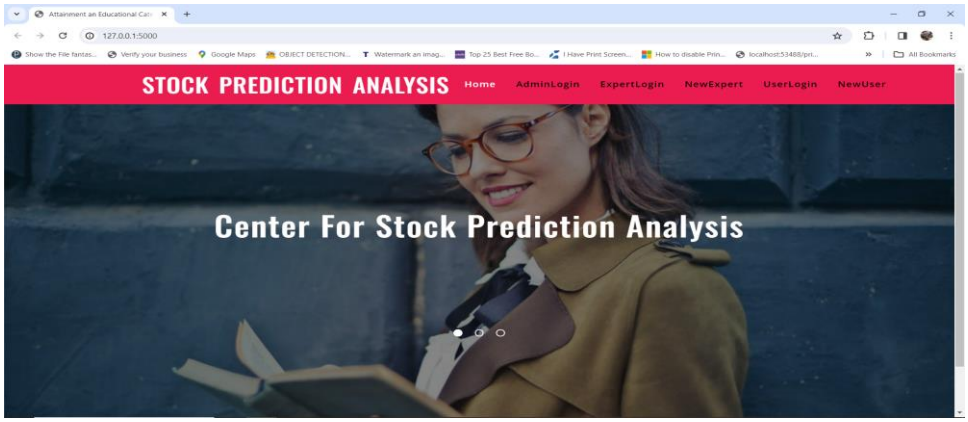
```
testPredictPlot[len(train_predict) + (look_back * 2) + 1:len(closedf) - 1, :] = test_predict
print("Test predicted data: ", testPredictPlot.shape)

names = cycle(['Original close price', 'Train predicted close price', 'Test predicted close price'])

plotdf = pd.DataFrame({'date': close_stock['Date'],
                       'original_close': close_stock['Close'],
                       'train_predicted_close': trainPredictPlot.reshape(1, -1)[0].tolist(),
                       'test_predicted_close': testPredictPlot.reshape(1, -1)[0].tolist()})

fig = px.line(plotdf, x=plotdf['date'], y=[plotdf['original_close'], plotdf['train_predicted_close'],
                                           plotdf['test_predicted_close']],
              labels={'value': 'Stock price', 'date': 'Date'})
fig.update_layout(title_text='Comparison between original close price vs predicted close price',
                  plot_bgcolor='white', font_size=15, font_color='white', legend_title_text='Close Price')
fig.for_each_trace(lambda t: t.update(name=next(names)))

fig.update_xaxes(showgrid=False)
fig.update_yaxes(showgrid=False)
fig.show()

# fig.write_image("images/fig31.png")
# img = mpimg.imread('images/fig31.png')
# imgplot = plt.imshow(img)
# plt.show()

x_input = test_data[len(test_data) - time_step:].reshape(1, -1)
print(x_input)
temp_input = list(x_input)
temp_input = temp_input[0].tolist()

lst_output = []
n_steps = time_step
i = 0

pred_days = int(ptdate)

while i < pred_days:

    if len(temp_input) > time_step:

        x_input = np.array(temp_input[1:])
        # print("{} day input {}".format(i,x_input))
        x_input = x_input.reshape(1, -1)
        x_input = x_input.reshape((1, n_steps, 1))

        yhat = model.predict(x_input, verbose=0)
        # print("{} day output {}".format(i,yhat))
        temp_input.extend(yhat[0].tolist())
        temp_input = temp_input[1:]
```

```
        # print(temp_input)

        lst_output.extend(yhat.tolist())
        i = i + 1

    else:

        x_input = x_input.reshape((1, n_steps, 1))
        yhat = model.predict(x_input, verbose=0)
        temp_input.extend(yhat[0].tolist())

        lst_output.extend(yhat.tolist())
        i = i + 1
print("Output of predicted next days: ", len(lst_output))
last_days = np.arange(1, time_step + 1)
day_pred = np.arange(time_step + 1, time_step + pred_days + 1)
print(last_days)
print(day_pred)
temp_mat = np.empty((len(last_days) + pred_days + 1, 1))
temp_mat[:] = np.nan
temp_mat = temp_mat.reshape(1, -1).tolist()[0]
last_original_days_value = temp_mat
next_predicted_days_value = temp_mat

last_original_days_value[0:time_step + 1] = \
    scaler.inverse_transform(closedf[len(closedf) - time_step:]).reshape(1, -1).tolist()[0]
next_predicted_days_value[time_step + 1:] = \
    scaler.inverse_transform(np.array(lst_output).reshape(-1, 1)).reshape(1, -1).tolist()[0]

new_pred_plot = pd.DataFrame({
    'last_original_days_value': last_original_days_value,
    'next_predicted_days_value': next_predicted_days_value
})

names = cycle(['Last 15 days close price', 'Predicted next' + ptdate + 'days close price'])
fig = px.line(new_pred_plot, x=new_pred_plot.index, y=[new_pred_plot['last_original_days_value'],
                                       new_pred_plot['next_predicted_days_value']],
              labels={'value': 'Stock price', 'index': 'Timestamp'})
fig.update_layout(title_text='Compare last 15 days vs next ' + ptdate + ' days',
                  plot_bgcolor='white', font_size=15, font_color='white', legend_title_text='Close Price')
fig.for_each_trace(lambda t: t.update(name=next(names)))
fig.update_xaxes(showgrid=False)
fig.update_yaxes(showgrid=False)
fig.show()
# fig.write_image("images/fig19.png")
# img = mpimg.imread('images/fig19.png')
# imgplot = plt.imshow(img)
# plt.show()

lstmdf = closedf.tolist()
```

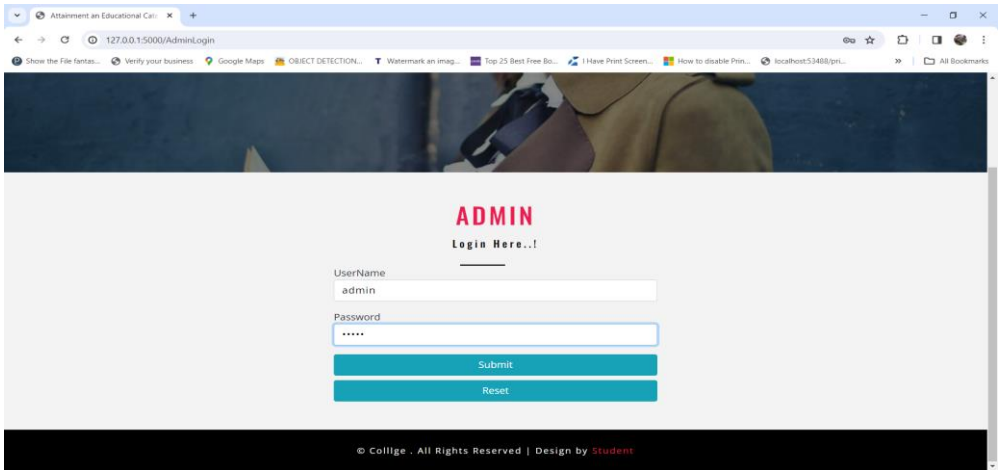
```
        lstmdf.extend((np.array(lst_output).reshape(-1, 1)).tolist())
        lstmdf = scaler.inverse_transform(lstmdf).reshape(1, -1).tolist()[0]

        names = cycle(['Close price'])
        fig = px.line(lstmdf, labels={'value': 'Stock price', 'index': 'Timestamp'})
        fig.update_layout(title_text='Plotting whole closing Stock price with prediction',
                          plot_bgcolor='white', font_size=15, font_color='white', legend_title_text='Stock Price')
        fig.for_each_trace(lambda t: t.update(name=next(names)))
        fig.update_xaxes(showgrid=False)
        fig.update_yaxes(showgrid=False)
        fig.show()
        # fig.write_image("images/fig20.png")
        # img = mpimg.imread('images/fig20.png')
        # imgplot = plt.imshow(img)
        # plt.show()
        name = session['uname']
        import datetime
        date = datetime.datetime.now().strftime('%Y-%m-%d')
        conn = mysql.connector.connect(user='root', password='', host='localhost', database='1StockNew')
        cursor = conn.cursor()
        cursor.execute(
            "INSERT INTO stocktb VALUES ('" + name + "','" + symbol + "','" + date + "','1')")
        conn.commit()
        conn.close()
        return render_template('Prediction.html')
@app.route('/Query')
def Query():
    uname = session['uname']
    conn = mysql.connector.connect(user='root', password='', host='localhost', database='1StockNew')
    cur = conn.cursor()
    cur.execute("SELECT * FROM Querytb where UserName='" + uname + "' ")
    data = cur.fetchall()
    return render_template('NewQuery.html', data=data)
@app.route('/Recommend')
def Recommend():
    conn = mysql.connector.connect(user='root', password='', host='localhost', database='1StockNew')
    cur = conn.cursor()
    cur.execute("SELECT UserName,StockName,date, sum(coun) as count FROM stocktb group by UserName,StockName,date ")
    data = cur.fetchall()
    return render_template('Recommend.html', data=data)
@app.route("/newquery", methods=['GET', 'POST'])
def newquery():
    if request.method == 'POST':
        name = session['uname']
        Query = request.form['Query']
        conn = mysql.connector.connect(user='root', password='', host='localhost', database='1StockNew')
        cursor = conn.cursor()
        cursor.execute(
            "INSERT INTO Querytb VALUES ('" + name + "','" + Query + "',',')")
        conn.commit()
```

```
        conn.close()
        flash('New Query  Register successfully')
    uname = session['uname']
    conn = mysql.connector.connect(user='root', password='', host='localhost', database='1StockNew')
    cur = conn.cursor()
    cur.execute("SELECT * FROM Querytb where UserName='" + uname + "' ")
    data = cur.fetchall()
    return render_template('NewQuery.html', data=data)
def sendmsg(targetno, message):
    import requests
    requests.post(
"http://smsserver9.creativepoint.in/api.php?username=fantasy&password=596692&to=" + targetno + "&from=FSSMSS&message=Dear user  your msg is " + message + " Sent By FSMSG FSSMSS&PEID=1501563800000030506&templateid=1507162882948811640")
if __name__ == '__main__':
    app.run(debug=True, use_reloader=True)
```

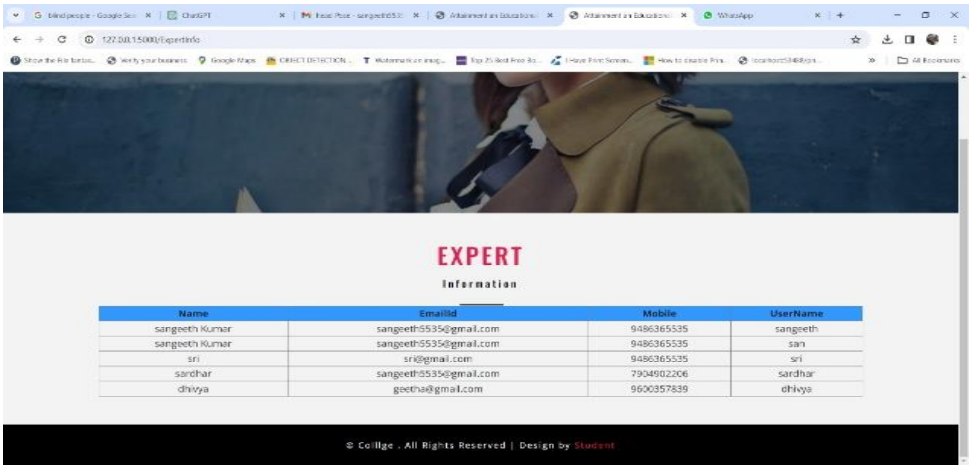
APPENDIX -II SCREENSHOTS



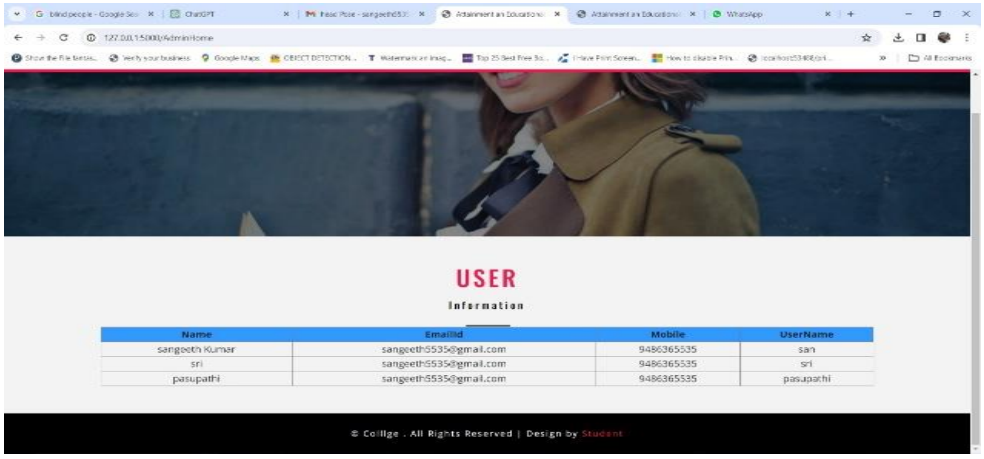
8.1 HOME PAGE



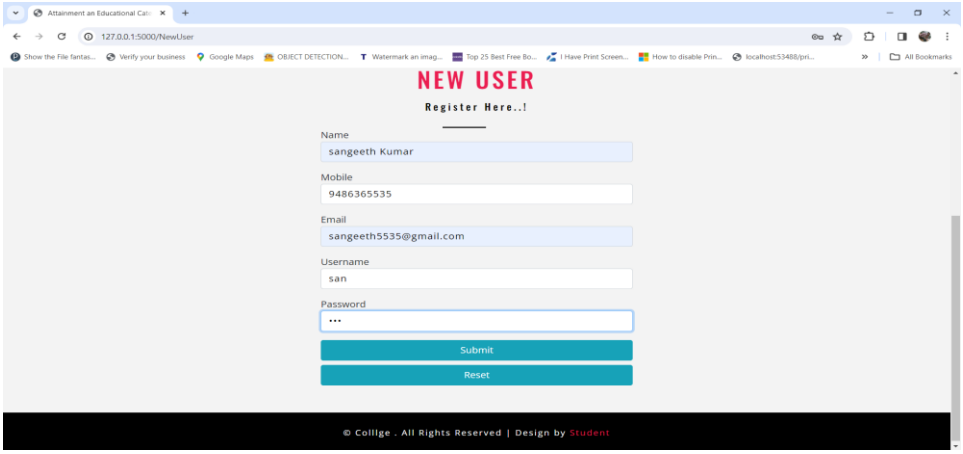
9.2 ADMIN LOGIN PAGE



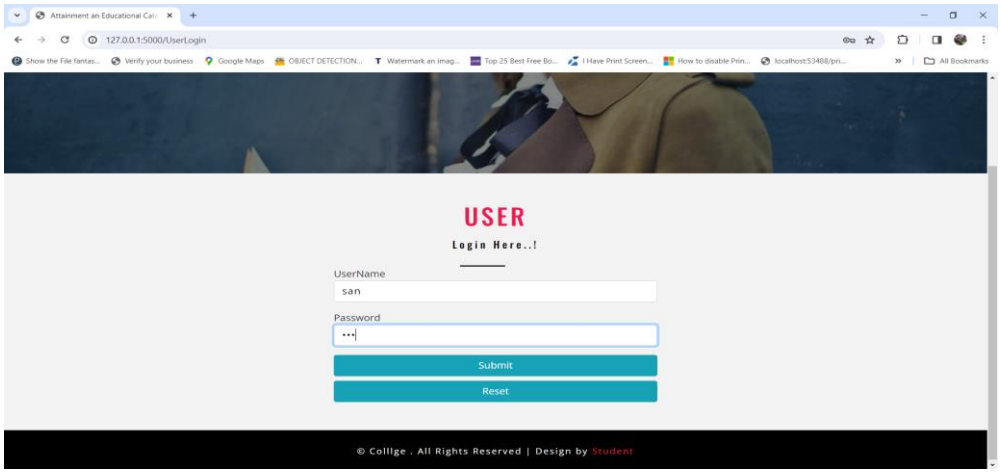
9.3 EXPERT INFORMATION PAGE



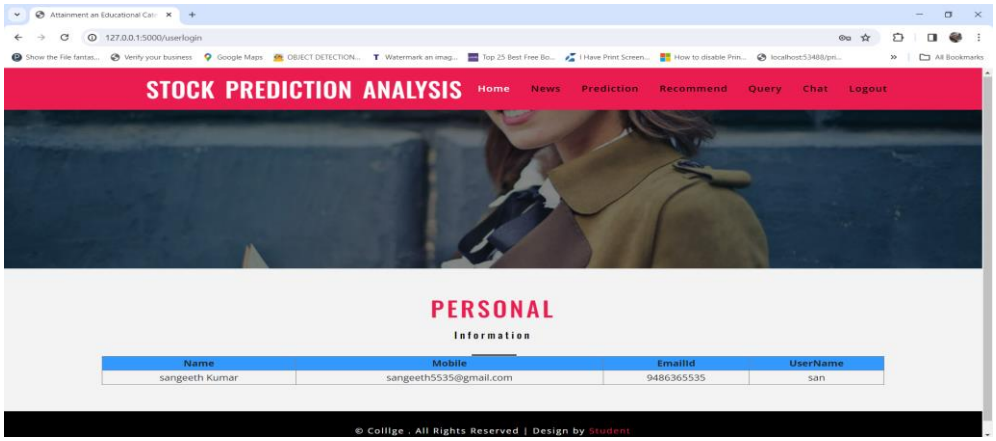
9.4 USER INFORMATION PAGE



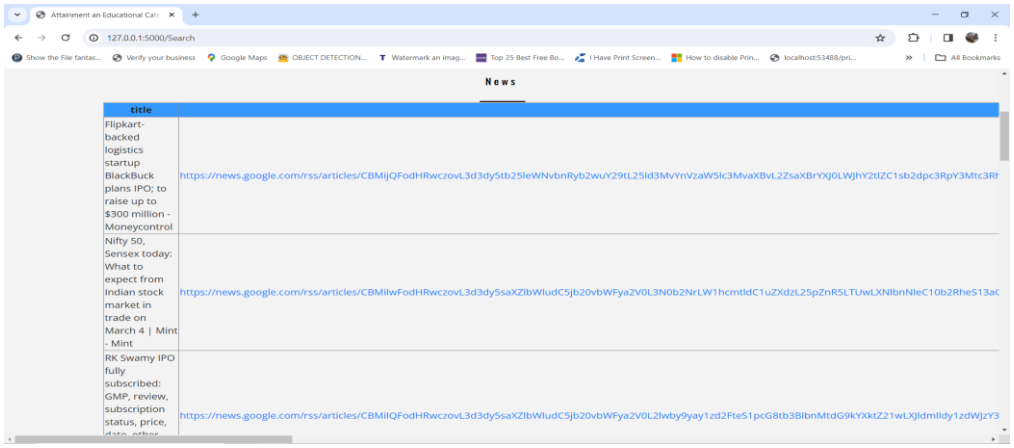
9.5 NEW USER REGISTER PAGE



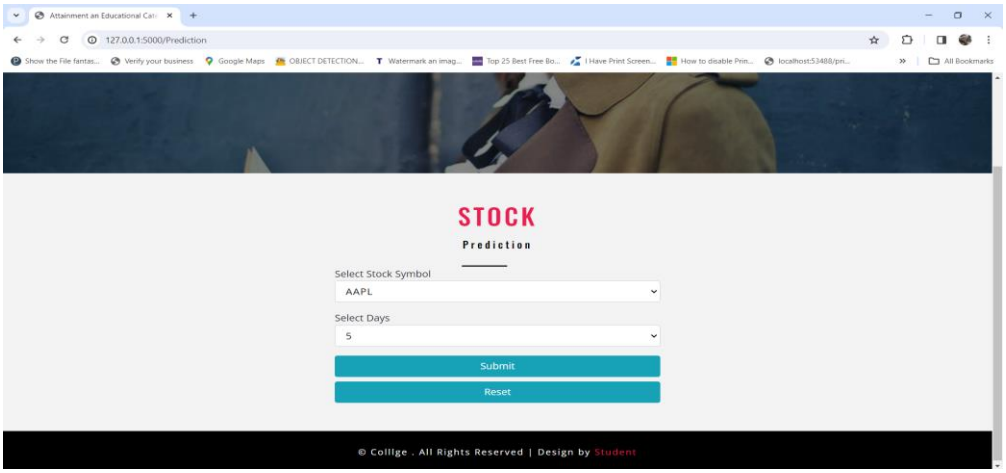
9.6 USER LOGIN PAGE



9.7 PERSONAL INFORMATION PAGE

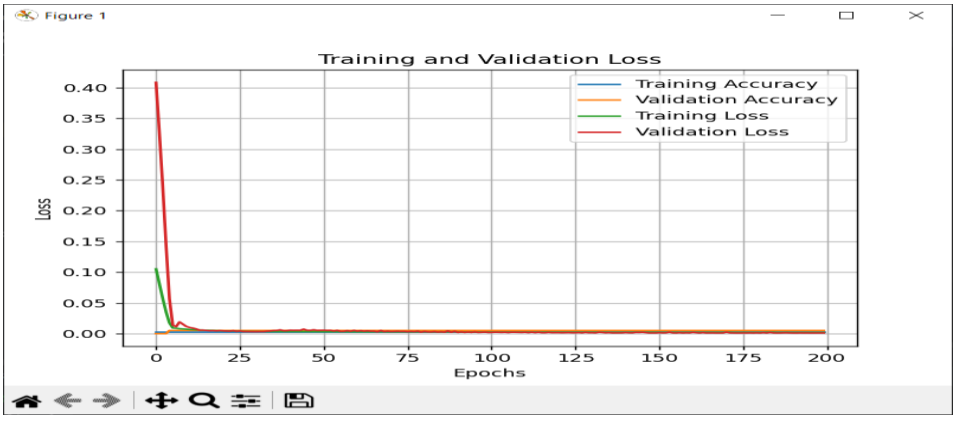


9.8 NEWS PAGE



9.9 STOCK PREDICTION PAGE

9.10 STOCK HISTORY PAGE

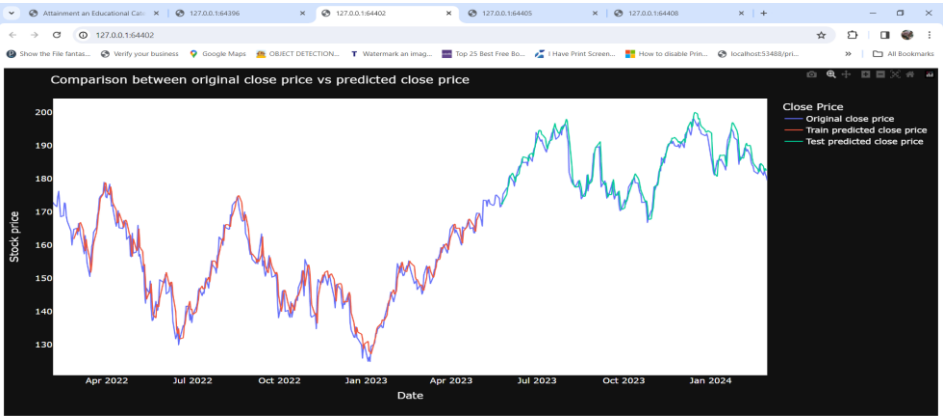


9.11 TRAINING AND VALIDATION LOSS PAGE

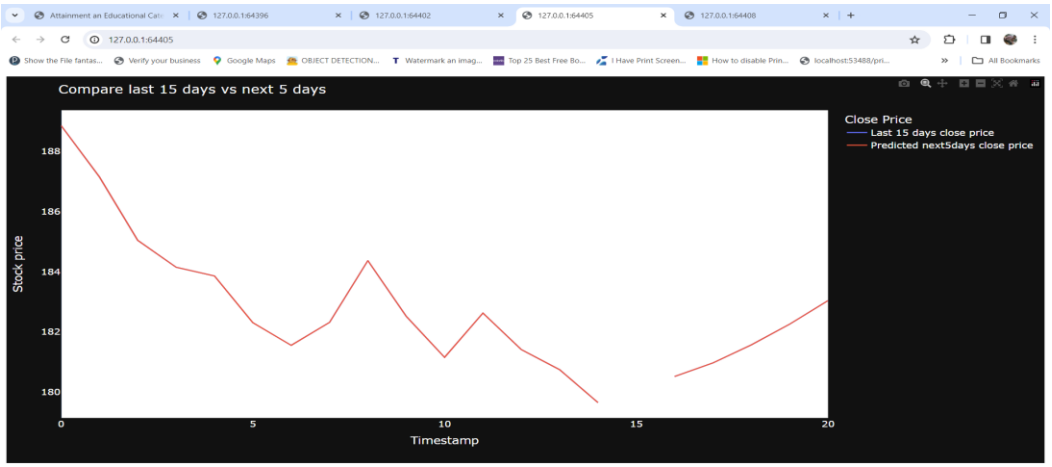
Train data RMSE: 3.3034219465711305
Train data MSE: 10.912596557087797

Test data RMSE: 2.48652360449766
Test data MSE: 6.1827996357240345
Train data explained variance regression score: 0.9268782771793389
Test data explained variance regression score: 0.9020263758804979
Train data R2 score: 0.9230559963083107
Test data R2 score: 0.8886800065732013
Train predicted data: (521, 1)

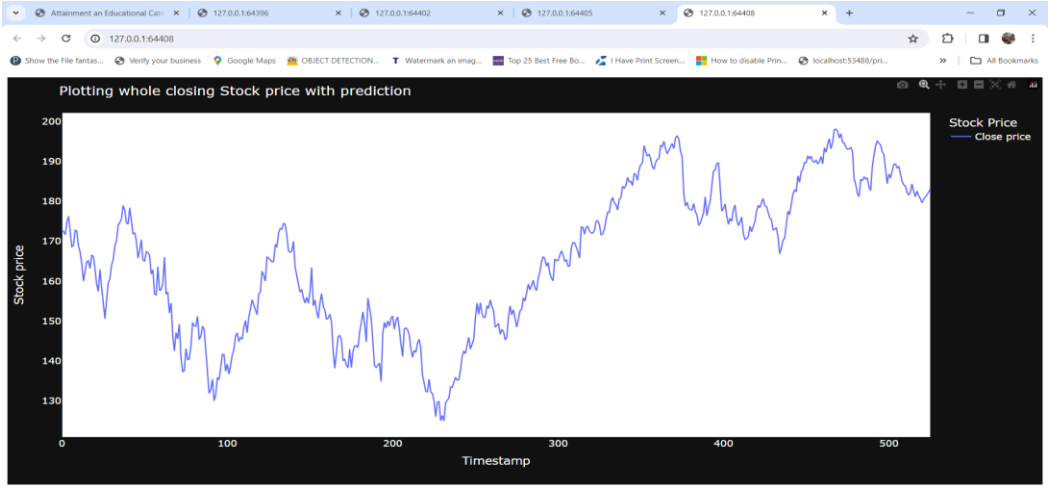
Test predicted data: (521, 1)
[[0.87330696 0.8500478 0.82117928 0.80900252 0.8050349 0.78382813
0.77356681 0.78396508 0.81201252 0.78670139 0.76809418 0.78820639
0.77165137 0.76248461 0.74757154]]



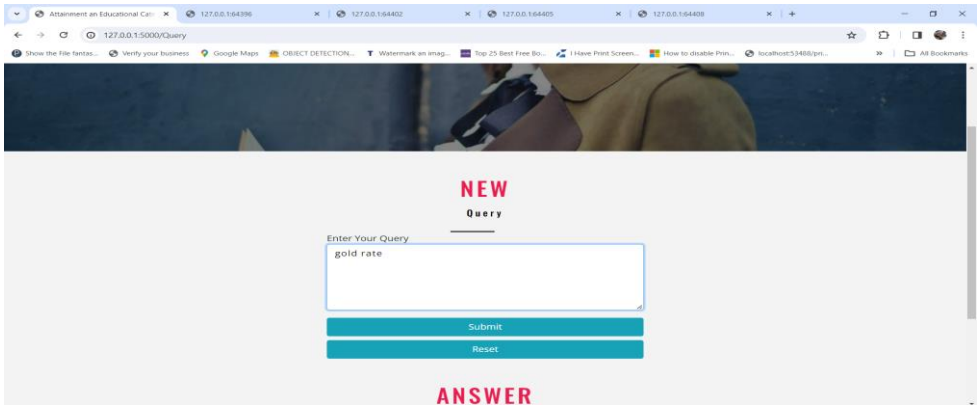
9.12 COMPARISION BETWEEN ORIGINAL CLOSE PRICE VS PREDICTED CLOSE PRICE



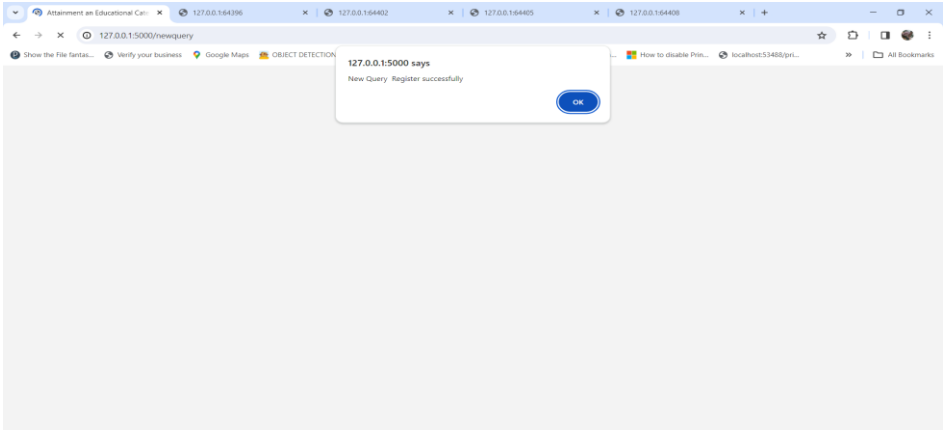
9.13 COMPARE LAST 15 DAYS VS NEXT 5 DAYS



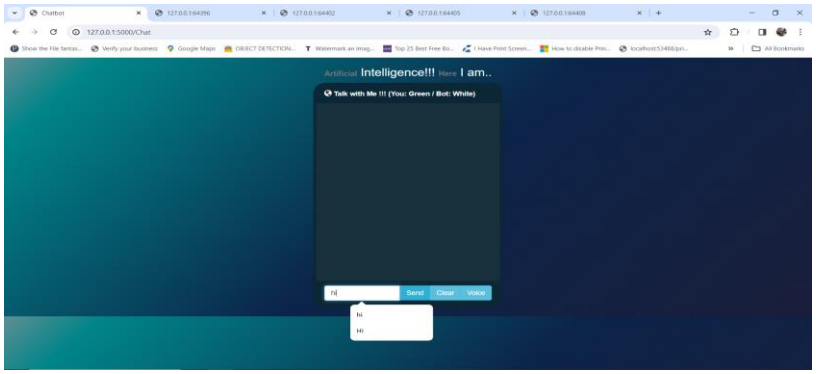
9.14 TIMESTAMP



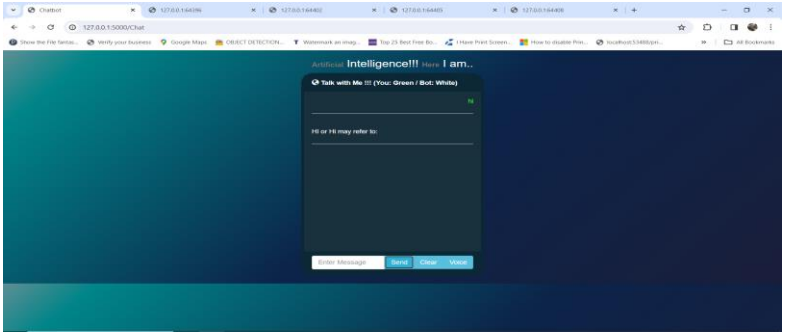
9.15 NEW QUERY PAGE



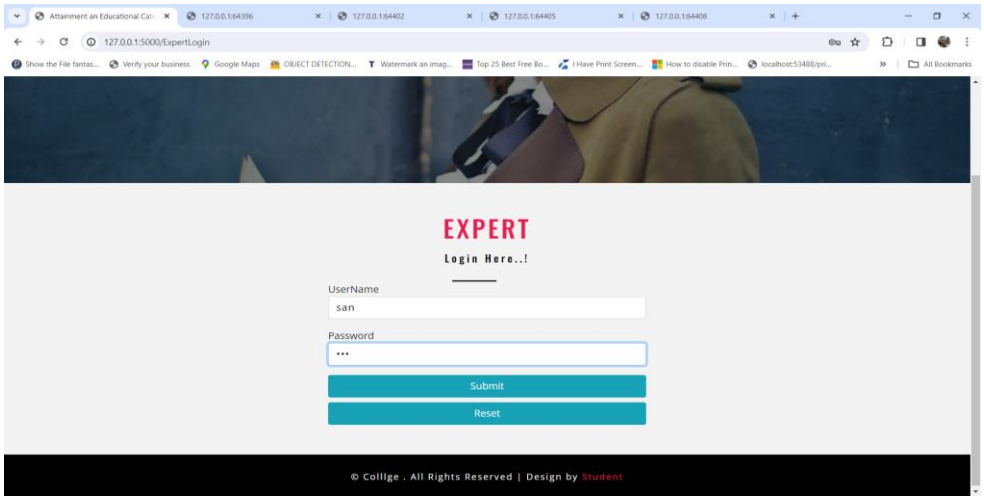
9.16 NEW QUERY REGISTER SUCCESSFULLY PAGE



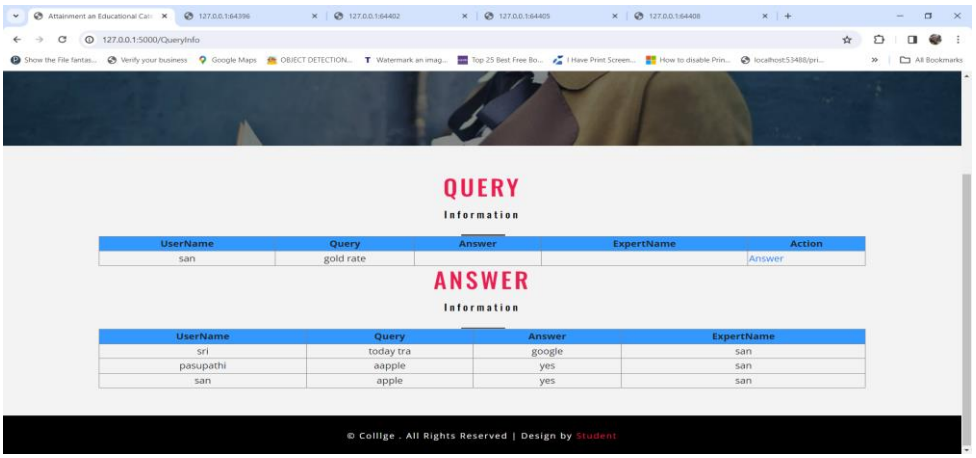
9.17 ARTIFICAL INTELLIGENCE PAGE



9.18 ARTIFICAL INTELLIGENCE PAGE



9.19 EXPERT LOGIN PAGE



9.20 QUERY AND ANSWER INFORMATION PAGE

10.REFERENCES

1. Heino, Aino. New Product Demand Forecasting in Retail: Applying Machine Learning Techniques to Forecast Demand for New Product Purchasing Decisions. MS thesis. 2021.
2. Neelakantam, Gone, Djeane Debora Onthoni, and Prasan Kumar Sahoo. "Fog computing enabled locality-based product demand prediction and decision-making using reinforcement learning." Electronics 10.3 (2021): 227.
3. Ulrich, Matthias, et al. "Distributional regression for demand forecasting in e-grocery." European Journal of Operational Research 294.3 (2021): 831-842.
4. Agbonlahor, Osayi Victor. A comparative study on machine learning and deep learning techniques for predicting big Mart item outlet sales. Diss. Dublin Business School, 2020.
5. Kohli, Pahul Preet Singh, et al. "Stock prediction using machine learning algorithms." Applications of Artificial Intelligence Techniques in Engineering. Springer, Singapore, 2019. 405-414.
6. Bian, Ruijie, Pamela Murray-Tuite, and Brian Wolshon. "Predicting Grocery Store Visits During the Early Outbreak of COVID-19 with Machine Learning." Transportation Research Record (2021): 03611981211043538.
7. Boyapati, Sai Nikhil, and Ramesh Mummidi. "Predicting sales using Machine Learning Techniques." (2020).
8. Chen, Shile, and Changjun Zhou. "Stock prediction based on genetic algorithm feature selection and long short-term memory neural network." IEEE Access 9 (2020): 9066-9072.
9. Pawar, Vijay. "Stock Positions Analysis and Prediction using Machine Learning" International journal 2022
10. Liu, Yixu. "Grocery Sales Forecasting." 2022 International Conference on Creative Industry and Knowledge Economy (CIKE 2022). Atlantis Press, 2022.