InvestX – FAC

Finance and Analytics Club

Mentors: Shivansh Gupta | Aayush Gupta | Hamza

Know the Project

What we'll learn...

01 First Part

Stock price prediction using LSTM

02 Second Part

Building Option Chain Strategy

Timeline

Week1

- Intro to Financial Markets
- Stock Listing Process
- Technical and Fundamental Analysis
- Technical Indicators: Types and Applications
- Basic Python and Data Analysis Using Python

Week2

- MachineLearning: basics and shared lectures on Neural Networks
- Lectures on the LSTM model
- Implementation for prediction of stock price for multiple parameter variation in price
- Coding out the Technical Indicators in Python
- Deducing the optimized parameters to be used for stock prediction

Timeline

Week3

- Application of the LSTM model on a sample dataset
- Identification of the right technical indicators for prediction
- Integration of the technical indicators in the ML model
- Gauging accuracy of the model

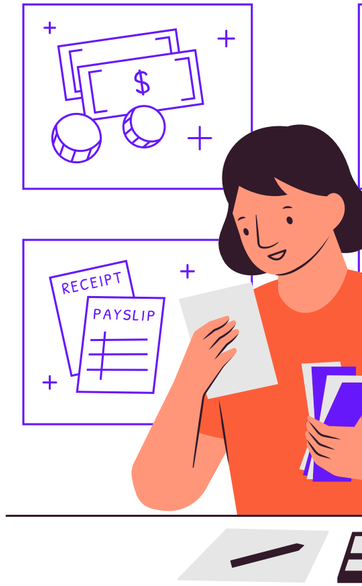
Week4

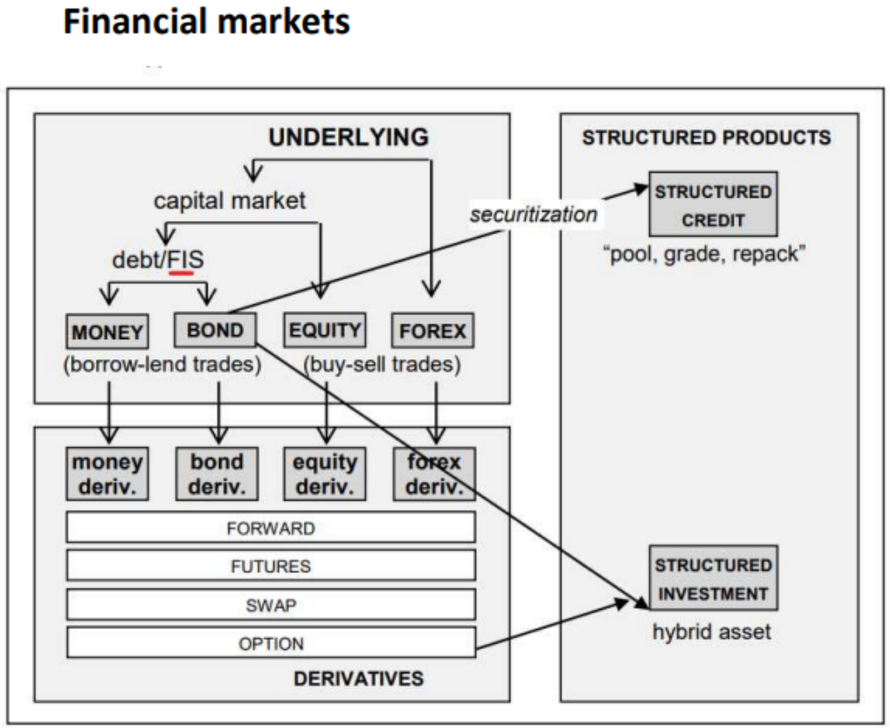
- Set up Selenium and Web Driver
- Navigate to the Opstra website
- Find and interact with the necessary elements
- Extract the option chain data
- Process and save the data

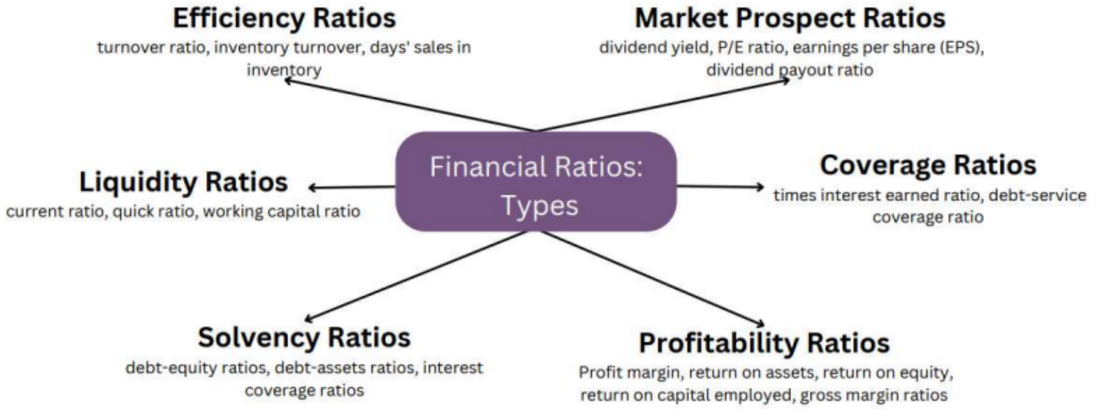
Intro to financial market

What we learned...

- About financial market basics
- IPO Process in India
- Factors affecting market movements
- Basics of fundamental analysis
- Intro to Technical Indicators







ASSIGNMENT-1

CODE FOR TECHNICAL INDICATORS

MOMENTUM , VOLUME, VOLATILITY

3 INDICATOR FROM EACH BY EACH MENTEE

CODING

Writing a Python code for all 3 types of indicator which were Alloted

OPTIMIZING PARAMETERS

Change the value of the parameters like lookback time period, open/ high/low/close etc.

OBSERVATION

Observe which parameter give the best suited result. and which one is the most effective

Average Directional Index Indicator

Average Directional Index

```
plus_DM = np.where(Stock.High.diff()>Stock.Low.diff(), Stock.High.diff(),0)
plusDM=pd.DataFrame(plus_DM)
minus_DM = np.where(Stock.Low.diff()>Stock.High.diff(), Stock.Low.diff(),0)
minusDM = pd.DataFrame(minus_DM)


True_range = pd.DataFrame([Stock.High - Stock.Low, abs(Stock.High - Stock.Close.shift(1)),abs(Stock.Low - Stock.Close.shift(
smooth_tr = True_range.rolling(24).mean()
smooth_plus_DM = plusDM.ewm(span=24).mean()
smooth_minus_DM = minusDM.ewm(span=24, min_periods=24).mean()

DI=abs((smooth_plus_DM - smooth_minus_DM)/ (smooth_plus_DM + smooth_minus_DM))*100
ADX = (DI.shift(1)*13 + DI)/14
ADX
```

[276]

	0
0	NaN
1	NaN
2	NaN
3	NaN
4	NaN
...	...
524	19.944184
525	22.636107
526	3.719402

ode + Markdown | Run All | Clear All Outputs | Outline ...

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

```
Stock = yf.download("^NSEI", start='2021-04-01',
                    end='2023-05-23');
Stock
```

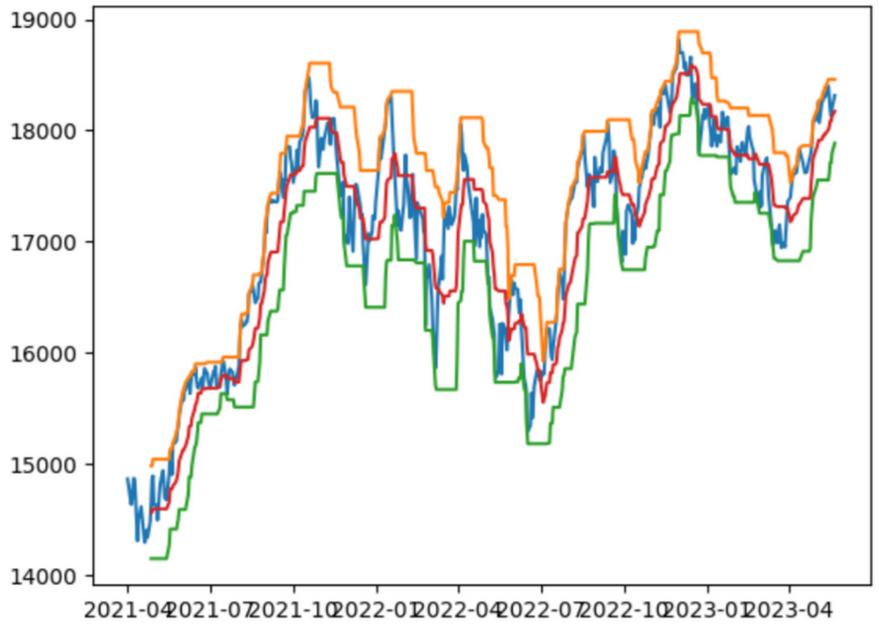
[**********100%**********] 1 of 1 completed

Date	Open	High	Low	Close	Adj Close	Volume
2021-04-01	14798.400391	14883.200195	14692.450195	14867.349609	14867.349609	445000
2021-04-05	14837.700195	14849.849609	14459.500000	14637.799805	14637.799805	509700
2021-04-06	14737.000000	14779.099609	14573.900391	14683.500000	14683.500000	475300
2021-04-07	14716.450195	14879.799805	14649.849609	14819.049805	14819.049805	0
2021-04-08	14875.650391	14984.150391	14821.099609	14873.799805	14873.799805	514800
...	...	...	...	...	...	...
2023-05-16	18432.349609	18432.349609	18264.349609	18286.500000	18286.500000	219500
2023-05-17	18300.449219	18309.000000	18115.349609	18181.750000	18181.750000	229900
2023-05-18	18287.500000	18297.199219	18104.849609	18129.949219	18129.949219	272100
2023-05-19	18186.150391	18218.099609	18060.400391	18203.400391	18203.400391	260900
2023-05-22	18201.099609	18335.250000	18178.849609	18314.400391	18314.400391	262600

529 rows × 6 columns

Donchian Channel

```
In [273…
plt.plot( Stock.Close)
plt.plot(don_channel)
plt.show()
```



Donchian Channel

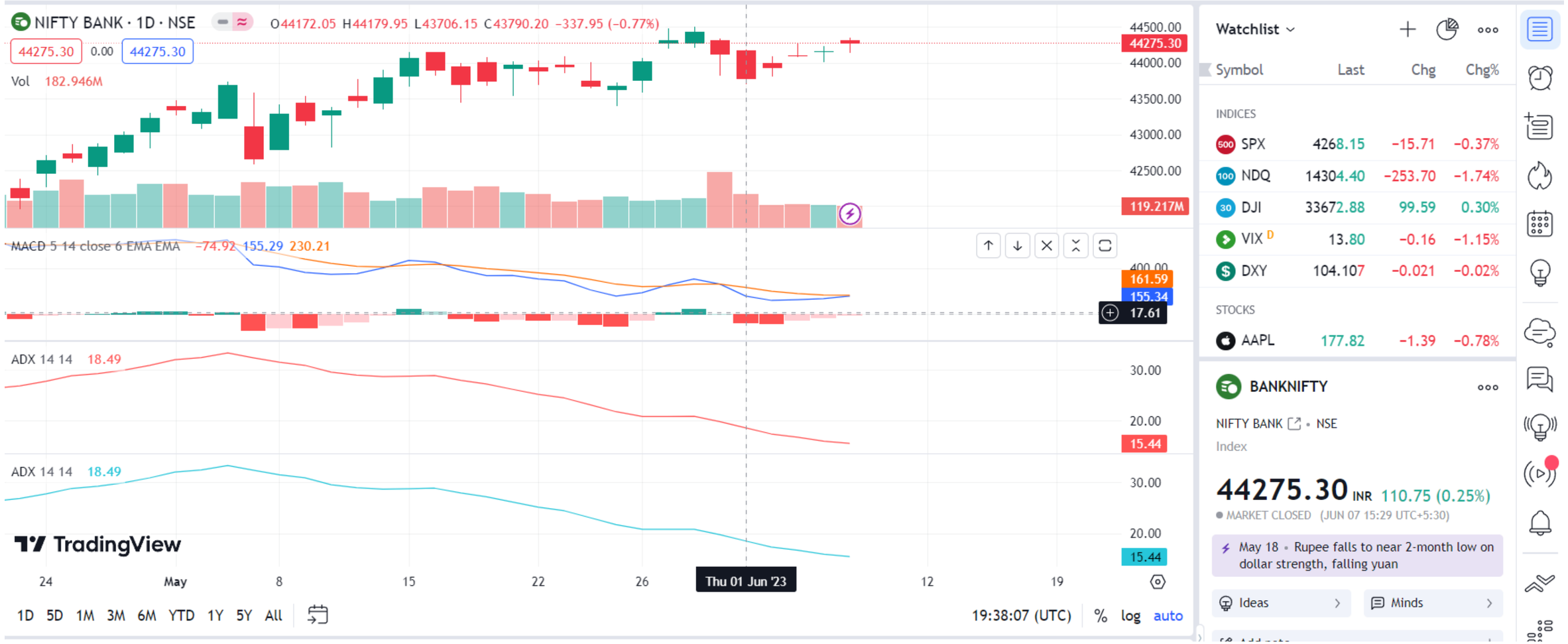
```
In [272…
upper=Stock.High.rolling(16).max()
lower=Stock.Low.rolling(16).min()
mid=(upper + lower)/2
don_channel=pd.DataFrame()
don_channel['upper']=upper
don_channel['lower']=lower
don_channel['mid']=mid
don_channel
```

Out[272…

Date	upper	lower	mid
2021-04-01	NaN	NaN	NaN
2021-04-05	NaN	NaN	NaN
2021-04-06	NaN	NaN	NaN
2021-04-07	NaN	NaN	NaN
2021-04-08	NaN	NaN	NaN
...	...	...	...
2023-05-16	18458.900391	17612.500000	18035.700195
2023-05-17	18458.900391	17711.199219	18085.049805
2023-05-18	18458.900391	17711.199219	18085.049805
2023-05-19	18458.900391	17797.900391	18128.400391
2023-05-22	18458.900391	17885.300781	18172.100586

Optimising the parameters of the indicators

With the help of 'Trading View' we applied indicators on the existing data and change the parameters, analysed the graph and achieved the optimum value to get better returns.



What we learnt from assignment 1?

- Basic knowledge of stock analysis
- Learnt about the technical indicators and the fundamentals
- Learnt about how indicators work to give better results
- Optimizing the standard indicators like RSI, Bollinger bands, etc.
- Writing Python code for indicators and using them
- Learnt the basics of Python and the data analysis using Python

Assignment -2

Calculating efficiency of the LSTM Model

LSTM Model

Long short-term memory

Data Preparation

We downloaded the required data from yFinance and made it into CSV file for better handling, and modified data according to our needs using panda dataframes, and fit the data with the help of scaler.fit.

Model Architecture

We defined lookover period, number of past days we want to use to predict the future. Dropout layers for avoiding overfitting, added layers using dense function for changing the dimension of the vectors by using every neuron.

Data Formatting

We took the first 150 data for training the model, and tail 50 data for testing the model i.e. predicted for last 50 datas and compared with real data for finding efficienct of model.
Then compiled the model.

Model Training

Then we trained the model by trying to minimize derived loss function.

Model Evaluation

Then we evaluated the efficiency of our model by finding some parameters like R^2, MAPE, MSE and MAE.

What we learnt from assignment 2?

1.) Predicting future stock price on the basis of past days stock prices with the help of Machine Learning using in-built python library (Several libraries, including numpy, pandas, yfinance, matplotlib, scikit-learn, and tensorflow.keras, were imported to manage data manipulation, visualisation, machine learning modelling, and evaluation.)

2.) LSTM Model Architecture: Defining of Sequential model using tensorflow.keras.models.Sequential and LSTM layers. To prevent overfitting, the model architecture incorporates two LSTM layers of differing sizes and a dropout layer. The final layer is a dense layer containing the same number of neurons as the target variables.

3.) Writing python code for stock price prediction and theoretical understanding of the RNN LSTM model under Machine learning along with its code implementation.

What we learnt from assignment 2?

4.) Basics of neural network(weights, biases, activations), convolution (image processing) - padding, striding, pooling layers,etc. Model Compilation and Training: Used the Adam optimizer and mean squared error (MSE) loss function to compile the model. The model was then trained using the fit method, with the number of epochs, batch size, validation split, and verbosity specified.

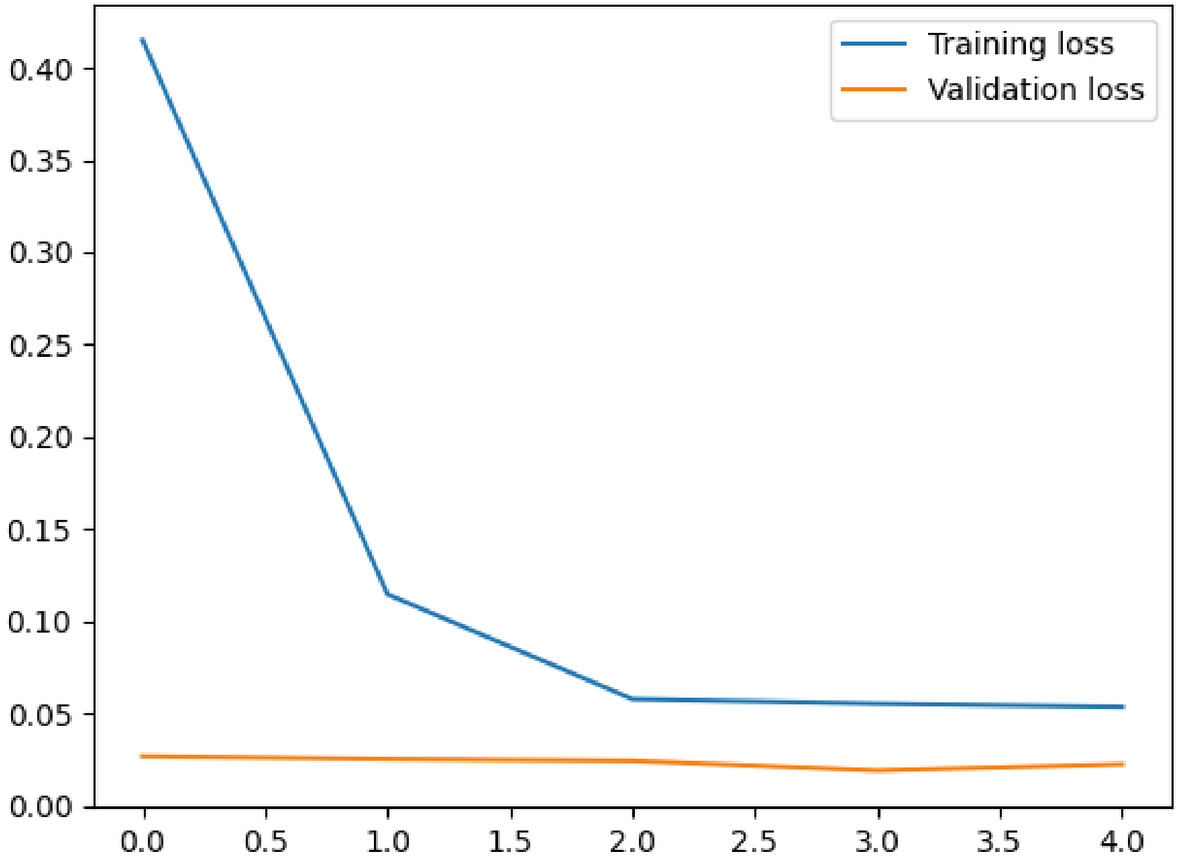
5.) Understanding of Backpropagation algorithm which propagates from output layers to hidden intermediate layers of the neural network involving iterative setting of weights and biases on the basis of gradients to minimise the error in the cost function.

What we learnt from assignment 2?

6.)The stock itself matters a lot in predicting data. Highly volatile stock like Tesla is harder to predict accurately, and more stable stock like Lockheed Martin is easier to predict more accurately. With the same model, accuracy changed with the selected stock as well.

7.) The model struggles a bit with extraordinary events. Like in the third assignment, we first took HDFC's stocks. During COVID, there was a heavy fall in its share price, but in real life it recovered soon. The Model on the other hand predicted lower values compared to the real ones. Upon testing the same on Lockheed Martin (which did not have that big of a drop), the accuracy of the model was higher.

8.)Adding tons of hidden layers does not directly mean a better model (i added like 3-4 extra ones just to see what changes. There was no significant increase in accuracy)



Model Training

Minimizing Cost Function

Feeding the input sequences to the model and adjusting the model's weights iteratively to minimize the defined loss function.

Result

R2 is 0.6561012189740019

Mean absolute error is : 0.9707654487059361

Mean square error is : 1.358796141885165

MAPE is 0.007836394796547803

Ass2.py > y_pred_future

```python
    model.add(Dropout(0.2))
    model.add(Dense(trainY.shape[1]))

    model.compile(optimizer='adam', loss='mse')
    model.summary()

    history = model.fit(trainX, trainY, epochs=5, batch_size=3,
                        validation_split=0.1, verbose=1)

    plt.plot(history.history['loss'], label='Training loss')
    plt.plot(history.history['val_loss'], label='Validation loss')
    plt.legend()
    plt.show()

    us_bd = CustomBusinessDay(calendar=USFederalHolidayCalendar())

    df2 = df.head(150)
    train_dates = pd.to_datetime(df2['Date'])
    print(train_dates.tail(15))

    n_past = 1
    n_days_for_prediction = 50  # let us predict past 50 days

    predict_period_dates = pd.to_datetime(df['Date'].tail(50)).tolist()
    # predict_period_dates = pd.date_range(
    #     list(train_dates)[-n_past], periods=n_days_for_prediction, freq='M'
    print(predict_period_dates)

    prediction = model.predict(trainX[-n_days_for_prediction:])
    prediction_copies = np.repeat(prediction, df_for_training.shape[1], axis=
    y_pred_future = scaler.inverse_transform(prediction_copies)[:, 9]

    forecast_dates = []
    for time_i in predict_period_dates:
        forecast_dates.append(time_i.date())

    df_forecast = pd.DataFrame(
        {'Date': np.array(forecast_dates), 'price': y_pred_future})
    df_forecast['Date'] = pd.to_datetime(df_forecast['Date'])


    original = df[['Date', 'price']].iloc[-50:]
    original['Date'] = pd.to_datetime(original['Date'])
    print(original)
    print(df_forecast)

    original.plot(x='Date', y='price')
    # original = pd.concat([original, df_forecast], axis=0) kyu bhai ??
    original.plot(x='Date', y='price')

    error = mean_absolute_error(original['price'], df_forecast['price'])
```

```
13 2015-12-31  126.922768
14 2016-01-29  127.006218
15 2016-02-29  127.151352
16 2016-03-31  127.294121
17 2016-04-29  127.485336
18 2016-05-31  127.250404
19 2016-06-30  127.129097
20 2016-07-29  126.742683
21 2016-08-31  126.410378
22 2016-09-30  126.279190
23 2016-10-31  126.546272
24 2016-11-30  126.317039
25 2016-12-30  125.682289
26 2017-01-31  125.193146
27 2017-02-28  125.288704
28 2017-03-31  125.094337
29 2017-04-28  124.366753
30 2017-05-31  123.701103
31 2017-06-30  123.574501
32 2017-07-31  123.628067
33 2017-08-31  123.785637
34 2017-09-29  123.632141
35 2017-10-31  123.628120
36 2017-11-30  123.773262
37 2017-12-29  123.519234
38 2018-01-31  123.494110
39 2018-02-28  124.298271
40 2018-03-30  124.636276
41 2018-04-30  124.844368
42 2018-05-31  124.643402
43 2018-06-29  124.292946
44 2018-07-31  123.905617
45 2018-08-31  123.339348
46 2018-09-28  123.060532
47 2018-10-31  122.858620
48 2018-11-30  122.688919
49 2018-12-31  122.286598
*****************************************
R2 is  0.6561012189740019
Mean absolute error is :  0.9707654487059361
Mean square error is :  1.358796141885165
MAPE is  0.007836394796547803
*****************************************

C:\Users\Rachit\OneDrive\Desktop\code\InvestX>
```

Ln 90, Col 1 (65 selected) Spaces: 4 UTF-8 CRLF Python 3.11.2 64-bit

Training our model along with different technical indicators and predicting their future movements

Some of the Technical Indicator Used

MACD

Moving average convergence divergence

RSI

Relative Strength Index

ADX

Average Directional Index

Volume Price Trend

Average Directional Index

Donchian Channel

Bollinger Bands

Preparing training data

We have taken the training price data and calculated values of technical indicators

```python
# RSI
def calculate_rsi(prices):
    price_diff = prices.diff()
    gains = price_diff.where(price_diff > 0, 0)
    losses = -price_diff.where(price_diff < 0, 0)

    avg_gain = gains.rolling(window=14).mean()
    avg_loss = losses.rolling(window=14).mean()

    rs = avg_gain / avg_loss
    rsi = 100 - (100 / (1 + rs))

    return rsi
```

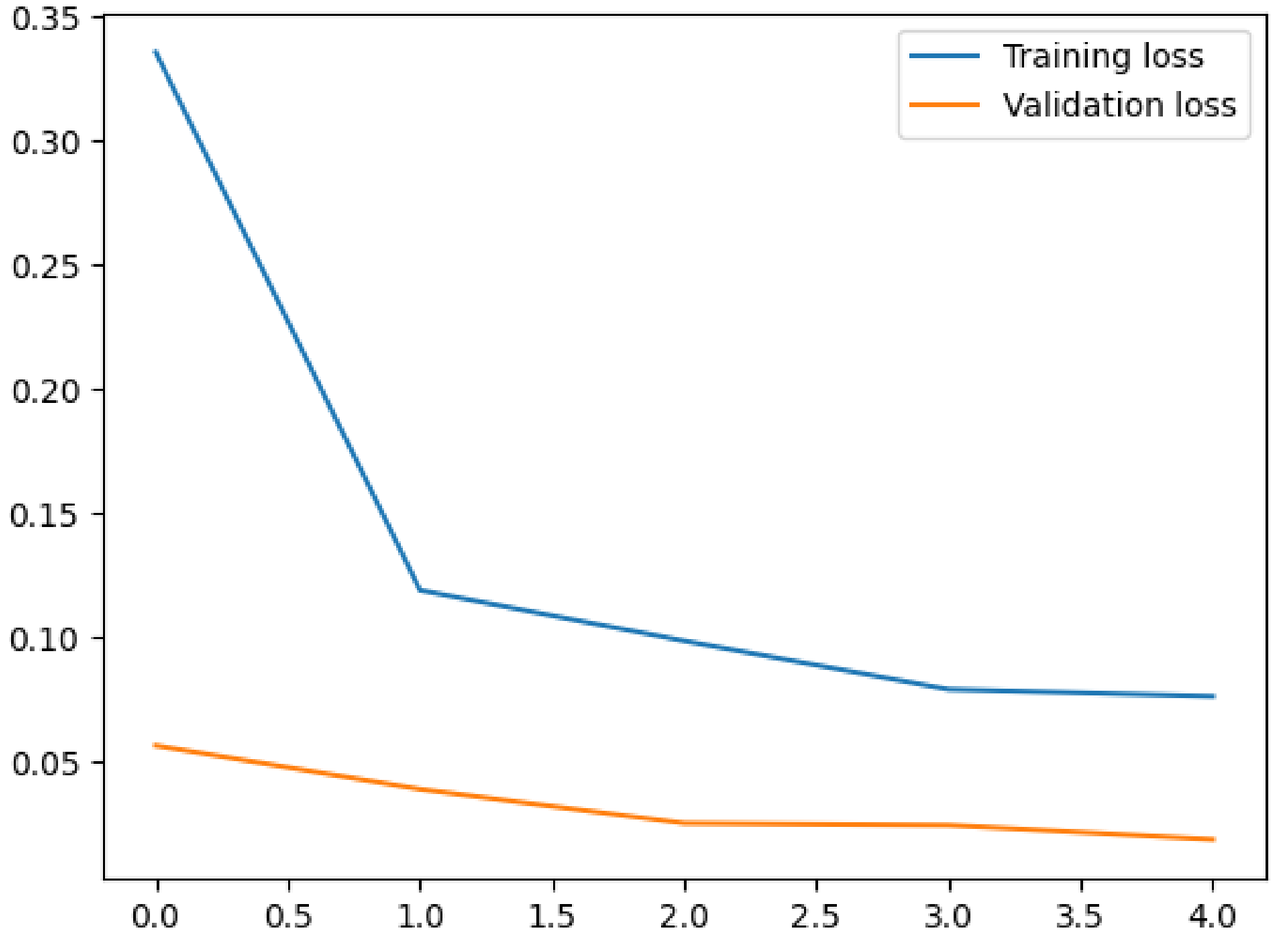
```python
# Bollinger Bands
def calculate_bollinger_bands(data):
    rolling_mean = data['Close'].rolling(window=20).mean()
    rolling_std = data['Close'].rolling(window=20).std()

    upper_band = rolling_mean + (rolling_std * 2)
    lower_band = rolling_mean - (rolling_std * 2)

    return rolling_mean, upper_band, lower_band
```

```python
# MACD
def calculate_macd(data, short_period, long_period, signal_period):
    short_ema = data['Close'].ewm(span=short_period, adjust=False).mean()
    long_ema = data['Close'].ewm(span=long_period, adjust=False).mean()

    macd_line = short_ema - long_ema
    signal_line = macd_line.ewm(span=signal_period, adjust=False).mean()
    macd_histogram = macd_line - signal_line

    return macd_line, signal_line, macd_histogram


macd_line, signal_line, macd_histogram = calculate_macd(data, 12, 26, 9)
```

Model training

Now building and training our model with the training data generated

Minimizing cost function



Results:

upto 94.96 %

Efficiency

```
1/9 [==>...........................] - ETA: 6s
6/9 [====================>.........] - ETA: 0s
9/9 [==============================] - 1s 11ms/step
0.30820775900256 89
Efficiency = 94.95573006349392 %
```

Assignment -3

Training our model along with different technical indicators, clubbing two or more indicators together, replacing the existing fundamental data of csv file from previous assignment with that of indicator to increase the efficiency of the model we developed in assignment 2.

Code sample

read the full code here

assignmen3.py > ...

```python
scaler = StandardScaler()
scaler = scaler.fit(df_for_training)
df_for_training_scaled = scaler.transform(df_for_training)

trainX = []
trainY = []

n_future = 1   # Number of days we want to look into the future based on the past days.
n_past = 14 # Number of past days we want to use to predict the future.

for i in range(n_past, len(df_for_training_scaled) - n_future +1):
    trainX.append(df_for_training_scaled[i - n_past:i, 0:df_for_training.shape[1]])
    trainY.append(df_for_training_scaled[i + n_future - 1:i + n_future, 9])

trainX, trainY = np.array(trainX), np.array(trainY)

print('trainX shape == {}.'.format(trainX.shape))
print('trainY shape == {}.'.format(trainY.shape))

df_for_training_scaled

trainY

model = Sequential()
model.add(LSTM(64, activation='relu', input_shape=(trainX.shape[1], trainX.shape[2]), return_sequences=True))
model.add(LSTM(32, activation='relu', return_sequences=False))
model.add(Dropout(0.2))
model.add(Dense(trainY.shape[1]))

model.compile(optimizer='adam', loss='mse')
```

Share Ln 51, Col 1 Spaces: 4 UTF-8 LF Python 3.10.8 64-bit Go Live

Assignment –4

Use Selenium & Python to scrape data from the option chain for various dates present on Opstra's Strategy builder.

Objective:

Extracting Option chain data of Indicies from Opstra using selenium

We used **Selenium**

Selenium is a popular open-source library in Python used for automating web browsers. It allows developers and testers to simulate user interactions with web applications, navigate web pages, fill in forms, click buttons, and extract data. Selenium is widely used for web scraping, automated testing, and other tasks that involve interacting with web pages programmatically.

Web Automation with Selenium:

Selenium is used for automating web browsers. It allows you to write scripts in Python that mimic user interactions with a web browser, such as clicking buttons, filling out forms, navigating through pages, and extracting information. This automation process is useful for various tasks, including web scraping, testing web applications, and performing repetitive actions.

Option Chain Data:

In finance and trading, an "Option Chain" is a table that displays various option contracts available for a specific financial instrument, like a stock or an index. It shows the different strike prices, expiration dates, option types (calls and puts), and related data for each contract.

Opstra:

Opstra: Opstra is a platform or service that likely provides financial data and analytics related to options trading. They may offer access to real-time or historical option chain data, volatility metrics, trading strategies, and other tools that help traders and investors make informed decisions in the options market.

integration of Selenium and Opstra:

When you use Selenium along with Opstra, you're combining the web automation capabilities of Selenium with the financial data provided by Opstra. This integration allows you to automate the process of retrieving option chain data from the Opstra platform, which can be useful for data analysis, backtesting trading strategies, or generating insights for options trading.

integration of Selenium and Opstra:

When you use Selenium along with Opstra, you're combining the web automation capabilities of Selenium with the financial data provided by Opstra. This integration allows you to automate the process of retrieving option chain data from the Opstra platform, which can be useful for data analysis, backtesting trading strategies, or generating insights for options trading.

Code sample for that

read the full code here

```
driver = webdriver.Chrome()
url = "https://sso.definedge.com/auth/realms/definedge/protocol/openid-connect/auth?response_type=code&client_id=opstra&redirect_uri=https://opstra.definedge.com/ssologin&state=e2cf559f-356c-425a-87e3-032097f643d0&login=true&scope=openid"

driver.get(url)
```

[54] Python

```
username = driver.find_element(By.ID, "username")
password = driver.find_element(By.ID, "password")
username.send_keys("-----@gmail.com")
password.send_keys("-----OPSTRA")

loginBtn = driver.find_element(By.ID, "kc-login")
loginBtn.click()

driver.get('https://opstra.definedge.com/strategy-builder')
time.sleep(.5)
expander = driver.find_element(By.CLASS_NAME, 'v-expansion-panel__header__icon')
expander.click()
time.sleep(.5)
```

[55] Python


thank
you